Simon Hagemann, Rainer Stark

An optimal algorithm for the robotic assembly system design problem: An industrial case study

Journal article | **Accepted manuscript (Postprint)**

This version is available at <https://doi.org/10.14279/depositonce-10862>

Hagemann, S., & Stark, R. (2020). An optimal algorithm for the robotic assembly system design problem: An industrial case study. CIRP Journal of Manufacturing Science and Technology.
<https://doi.org/10.1016/j.cirpj.2020.08.002>

Terms of Use

Copyright applies. A non-exclusive, non-transferable and limited right to use is granted. This document is intended solely for personal, non-commercial use.

WISSEN IM ZENTRUM
UNIVERSITÄTSBIBLIOTHEK

Technische
Universität
Berlin

An Optimal Algorithm for the Robotic Assembly System Design Problem: An Industrial Case Study

Simon Hagemann (Corresponding Author)

Technische Universität Berlin – (Industrial Information Technology)

Mercedes-Benz AG – (Information Technology Digital Factory)

simon.hagemann@daimler.com

Rainer Stark

Technische Universität Berlin – (Industrial Information Technology)

Fraunhofer IPK – (Virtual Product Creation)

Abstract

The design process of flow-oriented assembly systems is characterized by being both highly complex and time consuming. Especially those design processes categorized into robotic and multi variant encountered in the automotive body-in-white production stages. Unlike established manual and template-based assembly system design models, which are currently applied in industry, the here presented novel approach uses a knowledge-based search algorithm and automatically generates optimal assembly system configurations. The algorithm has been implemented in a software prototype and the results have been validated against different large-size industrial scenarios in the automotive field of body-in-white production.

Keywords: robotic assembly system design, discrete optimization, mixed linear integer programming, branch and bound, automotive, body-in-white, assembly line balancing, Orasda

1. Introduction

In the early phases, the robotic assembly system design (RASD) comprises the objective to determine the optimal assembly system (AS) configuration. (Michalos et al. 2015) However, the solution space to configure those is of such large extent, that even experts cannot design or even consider all feasible solutions. Assembly systems, which do not correspond to the optimal design solution, primarily lead to cost increases and other drawbacks, such as lower efficiency, availability, etc. Analyses have shown that production planners spend most of their capacity on non-value adding processes such as the gathering, analyzing and preparation of data. Less than 20% of their time is spent on the actual RASD process (Bracht et al. 2011).

Digitalization and automation in the early RASD phases show a high potential for solving this problem, which has been stated in previous studies (Michalos et al. 2015, S. 69; Bracht et al. 2011; Hagemann und Stark 2018, S. 192; Niggemann et al. 2014, S. 173; Jasperneite und Niggemann 2012, S. 36; Raimung Menges 2005, S. 14). Automated data analysis and the initial configuration of assembly systems can decrease the workload of production planners significantly and give them the opportunity to focus on work which requires their core competencies and creativity (Bauernhansl 2014, S. 17–18). For this reason, the future role of production planners will develop from the current procedural knowledge owner towards the declarative decision maker (Jasperneite und Niggemann 2012).

Established manual and template-based RASD approaches require: (i) the analysis of the virtual product model, including the corresponding meta-data, (ii) the transfer of the gathered information to a virtual assembly system under given project premises, (iii) the evaluation of the assembly system configuration against possible design alternatives. (Hagemann und Stark 2018, S. 192; Hu et al. 2011, S. 715; Rekiek und Delchambre 2006) Since the number of alternatives is high, the planner frequently relies on heuristics and decides intuitively which of the numerous design alternatives is the closest to the technical and economical optimum. (Schmigalla 1995) The planners' procedural knowledge to design assembly systems is hard to formalize and highly dependent on the planners' experience (Michalos et al. 2015).

The automated design of AS has been regarded insufficiently by research and industry. Current prototypes and commercial software systems do provide the functionality to manually design assembly

systems and have already improved the RASD process significantly. Nevertheless, the mentioned software tools do not adequately support the production planner in data preparation and decision-making. The paper tackles exactly this research gap. In the following sections a novel algorithm is presented to automatically design optimal assembly system configurations. It will be referred to as Orasda (Optimal Robotic Assembly System Design Algorithm). The algorithm has been implemented in a prototype and tested for the automotive field of body-in-white (BIW).

2. Literature Review and Research Gap

The deficient investigation of the RASD process automation has led to an increasing gap between different phases of the product creation process in manufacturing companies. Computer-aided design (CAD) and computer aided production planning (CAPP) tools are long established and accepted in industry, whereas the computational support for RASD is highly underdeveloped (Klauke 2002). Hagemann and Stark (Hagemann und Stark 2018) outline in their findings the complexity and highly creative process of RASD. Nevertheless, numerous authors emphasize the great potential for process digitalization and automation in this field. Niggemann et al. (Niggemann et al. 2014, S. 173) see potential in different areas of RASD by the use of intelligent software assistance systems. In the authors' opinion, an urgent need for RASD assistance is required due to the significantly increasing complexity of current and future assembly system solutions. Bracht et al. (Bracht et al., S. 63) suggest partial automation of RASD, such as the automated welding gun assignment, even though describing the RASD process as highly creative and therefore hard to formalize. In addition, the authors emphasize the importance of a consistent and continuous database. Michalos et al. (Michalos et al. 2015, S. 1) describe the large size and complexity of current BIW assembly lines and call for methods assisting those areas of RASD, in which stations are allocated in sequential order. The authors do not primarily see the main potential in reducing the work dedicated to the design, but rather in the achievement of higher documentation quality and more optimal assembly system configurations, which engineers might not even have considered. Boysen et al. (Boysen et al. 2007) underline the high-requirement capital expenditures for designing and installing assembly systems. Accordingly, the optimality of RASD configurations is of the utmost importance. In order to ease the design of optimal solutions, suitable optimization models – tailored for industrial challenges – are required.

In general, the RASD problem resembles the assembly line balancing (ALB) problem. The balancing of assembly lines has the objective to assign tasks to workstations and their resources in the most balanced way (Naveen Kumar & Dalgobind Mahto 2013). Both the ALB and RASD problem are considered to be NP-hard (Michalos et al. 2015, S. 69; Lopes et al. 2017, S. 1033; Gao et al. 2009, S. 1065). Classic balancing scenarios have been the subject of a vast number of publications, such as (Rekiek und Delchambre 2006; Klauke 2002; Boysen et al. 2007; Lopes et al. 2017, S. 1033; Sivasankaran und Shahabudeen 2014; Ghosh und GAGNON 1989). Boysen has classified balancing problems into four categories (Boysen et al. 2007). Category I aggregates the assignment of tasks under a given cycle time, targeting the minimization of workstations. Category II comprises the assignment of tasks to a given number of stations and resources in order to reduce the cycle time. However, Category E embraces the scenario in which the number of stations and cycle time are both flexible, in order to reduce the idle times. Contrarily, in category F both cycle time and the number of stations depict fixed values. Since all ALB problems create large solution spaces, simplifications have been assumed in prior publications, leading to the term 'simple assembly line balancing' (SALB). Especially for those ALB problems, which consider numerous degrees of freedom, simplifications have been indispensable to enable initial possible approaches. Hence, there is still a gap between current methods and industrial requirements (Boysen et al. 2007). The robotic assembly line balancing (RALB), firstly introduced by Rubinovitz and Levitin (Rubinovitz und Bukchin 1991; Levitin et al. 2006), focuses on assembly lines solely composed of robots. Later on, several authors, such as (Lopes et al. 2017; Michels et al. 2018, S. 320; Daoud et al. 2012, S. 1353; Kim und Park 1995, S. 2311) have proposed further solutions to solve the RALB problem. Despite the similarity between the RALB and the RASD problem, existing RALB approaches are not capable of solving the RASD problem. Thus, Robotic Assembly Line Design (RALD) aims at

determining, under a given cycle time, the number of stations and affiliated robots required to carry out the operational tasks. Similarly, Michalos et al. (Michalos et al. 2015) focused on the robotic assembly line design in the automotive field of body-in-white. The authors describe a method which configures a body-in-white robotic assembly line against multiple-user input information. The method is based on a rule-based, formula-driven algorithm and has been validated against an industrial BIW case study. The research results have shown the great potential for automation in the field of automated body-in-white RALD. However, the algorithm is not generally applicable, since only sequential assembly lines are considered. Michels et al. (Michels et al. 2018) present a method which applies a mixed-integer linear programming (MILP) formulation and enables the creation of parallel stations. The algorithm has been tested against several industrial scenarios and, for most of them, the algorithm has successfully delivered feasible results. In addition, it has been proven that the solutions provided by the algorithm are of a better quality than manually designed lines.

Initial methods solving the RALD have been developed and published, yet the presented solutions omit essential industrial requirements to solve the RASD problem holistically. The presented methods focus mainly on assembly lines in an sequential order, even though Michels et al. (Michels et al. 2018) consider the paralleling of stations. In the early stages, assembly system structures can assume deviating structures and handling processes occupy serious shares of the cycle time. To ease a generic and automatic design of robotic assembly systems, all of the following design requirements need to be covered:

(i) assembly systems of divergent (non-sequential) structures	(iv) joining process design:
	a. different joining technologies
(ii) station design:	b. fixture- and stationary-based joining
a. multiple robots per station (for handling and joining)	(v) handling process design:
b. geometry stations and respot stations	a. station loading
(iii) product variants	b. part collection
	c. unloading

It has been shown that none of the here presented scenarios equal the RASD problem tackled in this paper. Hence, in previous studies of (Hagemann und Stark 2018), (Hagemann et al. 2019) existing methods of ruled-based systems, artificial intelligence (AI) and mathematical optimization have been analyzed and evaluated. The evaluation has revealed that discrete mathematical optimization techniques, as already applied for numerous SALB, RALB and RALD problems, suit the RASD problem best due to the generation of optimal solutions. Whereas methods of artificial intelligence, like the application of deep learning and neural networks, revealed advantages in its inductive characteristics. However, the dependency on either large amounts of high-quality data or the manual training by production planners denote major drawbacks. The optimality and the lack of transparency of AI systems or even the feasibility of their results cannot be ensured. (Hagemann et al. 2019) (Damaševičius 2008) Hence, in this paper, an algorithm, which covers all of the above-listed requirements (i-v), is introduced. A discrete optimization formalization will be applied which automatically generates solutions for robotic assembly systems, specialized on the automotive body-in-white production stages.

3. Formalization of procedural design knowledge

Conventional, robotic BIW assembly systems are organized in workstations (Kiefer 2007). Those stations in their entirety are arranged sequentially or in tree structures, parallel stations are an exception. Each of the stations, in turn, is equipped with a certain number of robots and the corresponding equipment such as tools, grippers, fixtures, etc. (Biesinger et al. 2018) Both indicators - the station structure and the equipment of each station - define the assembly system configuration. (Boysen und Fliedner 2008) The configuration affects versatile assembly system evaluation indicators such as costs, availability, capacity, flexibility, etc. In the early phases of planning the engineer is particularly interested in finding the cheapest solution, satisfy the product requirements and fulfill the project

premises. The following subchapters expound the manual assembly system configuration process and aim at formalizing the production planning approach.

Assembly sequence

The assembly sequence determines the order in which BIW parts are joined. Exemplifying figure 1 and figure 10 show assembly sequences (AS). In industrial practice, it is common that the assembly sequence of the whole vehicle is subdivided into the AS of assembly groups. The overall parent node of the tree structure corresponds to the target assembly of the visualized assembly group. Child nodes otherwise can be differentiated into single parts (circle) and joining steps (box) (Burr 2008). By mapping the joining sequence structure to the product model, each joining step (JS) may be enriched with product information such as joining technologies and the corresponding number of joining elements, shown in the box in figure 1. In comparison to the product structure, which is known by the term “engineering bill of materials” (EBOM), the assembly sequence or manufacturing bill of materials (MBOM) aims at describing the order in which parts will be joined in production (Kretschmer et al. 2017; Kiefer 2007).

Rough assembly system structure design

The assembly sequence enables the production planner to derive a rough and initial station structure, indicated in figure 1. Each joining step requires one or various stations to be processed. Automotive BIW production stations are distinguished into the so-called geometry stations (GS) and respot stations (RS) (Hsieh et al. 1997, S. 39). GS serve for placing the first joining elements and guarantee an initial structure of each assembly. After ensuring the initial structure of the (sub-) assembly, remaining respot joining elements may be placed in following RS to reinforce the assembly structure. The order of the stations belonging to one joining step is usually sequential, mainly due to traceability reasons. The number of required stations is correlated to multiple factors, such as the cycle time, the number of parts in the assembly, the applied joining technologies (JT) and the number of corresponding joining elements.

Figure 1: Assembly sequence and station structure

Stations design

BIW stations consist of various resource types such as fixtures, robots, tools, grippers and additional equipment, if required (Biesinger et al.). In early phases of planning, not the exact resource assignment takes place, but the number of resource types has to be determined (Kusiak). Table 1 (left up) visualizes a standardized assembly station and its layout. The corresponding and exemplifying balancing chart in Table 1 (right up) illustrates the designed station process, in which the light green handling robot loads the fixture (process 1+2) and afterwards collects the parts from the load carriers for the next cycle (process 3+4). The dark green robot unloads the assembly and transfers it to the next station, after the joining process is terminated (last process). The blue robots represent joining robots with the joining processes (5-8). The grey centered object portrays a fixture. All tasks inside one station are limited by

the cycle time t_c . In automotive green-field projects, the cycle time is a given premise (Michalos et al. 2015; Michels et al. 2018).

In particular, the number of stations and robot modules, including their predecessor and successor relationship, define the systems' structure and set the assembly system capacity. A robot module consists of the robot itself, a platform, a control unit and one to several tools or grippers (Kusiak). Depending on tools or grippers attached to robots, handling or joining capabilities are obtained. Joining processes can be designed as the following scenarios show: (i) scenario j1: tool directly attached to the robot and the joining process takes place in a fixture – compare table 1. It depicts the conventional way of joining parts. (ii) hj1 and (iii) hj2, in which the joining process is combined to handling processes (will be introduced with the handling scenarios). (Michels et al. 2018) Besides the combined scenarios two handling scenarios are shown in table 1. (i) Scenario h1: the conventional way of handling parts inside a station. The left green robot is responsible for the handling of the parts or (sub-) assemblies inside the station. Parallel to the joining process of the blue robots, the handling robot collects the parts that have to be handled inside the station. As soon as the joining process is finished and as soon as the joined parts have been taken out of the fixture, the new parts can be sequentially positioned inside the fixture. Due to the high amount of time required for this handling scenario, especially for a high number of parts, scenario h2 brings up an attractive alternative. In scenario h2 first all parts are collected, then temporarily stored on a table, and only afterwards transferred all at once to the following station. As mentioned before hj1 combines handling and joining capabilities with the application of double equipment. Double equipment allows to join and to handle with one robot in sequential order. This type of equipment is usually allocated to those robots which load fixtures (scenario hj1). Since the time required for the loading process is empirically considered low compared to the cycle time, hj2 avoids idle times and the handling robot supports the joining process with attached tools. Equally, the unloading robots can spend idle times for joining at a stationary tool or even use the following cycle time for joining (scenario hj2).

Table 1
Assembly scenarios

Assembly scenario	Resource design	Process design
Handling scenario: h1 Joining scenario: j1		
Handling scenario: h2 Joining scenario: j1		

Handling
scenario:
hj1, hj2

Joining
scenario: **j1**

4. Algorithm

This chapter is dedicated to the description of the algorithm solving the RASD problem. Prior to the better understanding of the algorithm, the following table 1 lists all notations used in the mathematical model.

Table 2

Table of notations

Notation	Description	Notation	Description
Resource Types			
DE	Double equipment	ST	Stationary tool
FX	Fixture	ULR	Unload robot
JR	Joining robot		
LC	Load carrier		
LR	Load robot		
Variables and attributes			
a	Resource availability		
c	Cycle	C	Set of cycles
d	auxiliary variable for double equipment		
e	auxiliary variable for stationary tool		
j	Joining element	J	Job – set of joining elements
j_g	Joining element geometry	L	Set of jobs
f	Joining step	F	Set of joining steps
i	Costs (investment)		
l_w	Limit max. number of robots inside one station		
l_a	Limit min. availability of one station		
l_i	Limit costs of best current solution		
m	Part	M	Set of parts
$n(j1)$	Number of joining elements set with j1		
p	Process	P	Set of processes
r	Resource	R	Set of resources
s	Station	S	Set of stations
s_g	Geostation	S_g	Set of geo stations
t_c	Cycle time		
$t_{h,l}$	Handling process time, loading		
$t_{h,c}$	Handling process time, collecting		
$t_{h,u}$	Handling process time, unloading		
t_j	Process time for j	t_j	Process time for J
t_ε	Residual time of robot		

t_0	Process start time		
t_1	Process end time		
w	Robot (worker)	W	Set of robots
$x_{j,r}$	Assignment variable of j to r	$X_{j,r}$	Set of assignments

4.1 Input Data and Output Data Model

The algorithm relies on input data primarily consisting of the assembly sequence tree structure and attached joining elements. Figure 2 (left box) visualizes the input data and aims at defining a conceivable data structure. Joining steps have one successor, but can consist of several predecessors. Each step consists of minimum two parts to be joined, each part in turn, owns the attribute material. In addition, every joining step has one up to many joining elements, which connect the parts to each other. The joining elements hold the following attributes, (i) geometrical relevance, with possible values geo- or respot joining, (ii) joining technology (JT), for instance spot welding. Each of those joining elements can join up to three parts. Besides these product and assembly sequence related data, the project premise cycle time t_c , the number of max. joining robots l_w and the availability limit l_a serve as important input variables. (Hagemann und Stark 2018)

The expected output data – the assembly system description – is based on process and resource-related data, both affiliated to production stations. Resources can, depending on the hierarchy level, consist of 1...* child-nodes. Each station consists of a minimum of one resource. Equally, every station consists of a minimum of one process. As introduced in chapter 2, the resource type classifies the resources into different categories. Resources exhibit different process capabilities referring to processes p such as welding, handling, etc. The resource costs c_r and availability a_r complete the resource attributes. Process objects carry two attributes to describe the start and end time of the process in between the cycle time t_c . The type depicts the kind of process conducted, for instance welding, clinching or even handling, whereas the bool variable geoflag determines whether the process serves as geometry joining or respot joining.

Figure 2: Data structure of input and output

4.2 Formalization of RASD Knowledge

In this section, several technical constraints will be defined to limit the solution space of the algorithm. The constraints are based on the above-described formalized RASD knowledge.

Cycle time and station: the cycle time t_c is the time for each station to process the aligned tasks. After the expiration of the cycle time t_c , parts must be ready for handover to the following assembly station. The following equation (1) ensures that the given cycle is complied. Let $t_{c,0}$ be the time, when the first process starts, which is aligned to the cycle c . Similar, the end of the last process aligned to cycle c determines $t_{c,1}$.

$$|t_{c,1} - t_{c,0}| \leq t_c \quad \forall c \in C \quad (1)$$

Joining elements: Each joining element must be assigned to exactly one resource. Let $x \in \{0,1\}$ be the variable, which decides whether joining element j is assigned to resource r . Equation (2) enforces that all joining elements are assigned to exactly one resource. Therefore, for all joining elements the sum of all assignments x over all resources must equal the value 1.

$$\sum_{r \in R} x_{j,r} = 1 \quad \forall j \in J \quad (2)$$

Geometry joining elements: geo elements have to be processed in a geo fixture. Respot elements can be set in the same or following stations under consideration of equation (5). In addition, the geo station s_g depicts the first station of all stations belonging to one joining step. In the equation (3) the assignment variable x ensures that all geo joining elements J_g are assigned to resources r which belong to a geo station.

$$\sum_{r \in R_{s,g}} x_{j,r} = 1 \quad \forall j \in J_g \quad (3)$$

Tool capability: joining elements are only allowed to be assigned to tools which carry the capabilities for the required joining process at hand, which the following equation (4) ensures. Let p be the process required to process j .

$$\sum_{r \in R_p} x_{j,r} = 1 \quad \forall j \in J_p \quad (4)$$

Joining sequence and accessibility: for each joining step, accessibility simulations are performed that validate the accessibility of joining elements with corresponding tools for the given assembly sequence. The tool accessibility for deviant joining sequences is not ensured. Hence, each joining step f has to be processed in stations, which are only used for joining elements of exactly this joining step. Inequation (5) shows that for resources belonging to stations which process joining step f , the assignment variable $x_{j,r}$ has to be higher (in this case the sum must be 1) than the sum for resources of deviant joining steps f' (in this case the sum must be 0).

$$\sum_{r \in S_f} x_{j,r} = 1 \wedge \sum_{r \in S_{f'}} x_{j,r} = 0 \quad \forall j \in J \quad (5)$$

Availability: the availability of each station a_s must be greater or equal than a given availability limit, which is shown in equation (6). The station availability is determined by the product over all resource availabilities of this station and shown in equation (7).

$$a_s \geq l_a \quad \forall s \in S \quad (6)$$

$$a_s = \prod_{r \in R_s} a_r \quad (7)$$

Number of robots and resources: the station availability limits the maximum number of resources inside each station. In addition, the maximum number of robots is limited by a constant l_w in order to enable a collision-free execution of the joining process and is ensured by equation (8). The value of l_w can vary between the single joining steps and has to be determined manually.

$$|R_{w,s}| \leq l_w \quad \forall s \in S \quad (8)$$

Joining process: joining robots can start processing as soon as handling processes have been collected and loaded for the joining process. Let $t_{c,l,1}$ be the time of cycle c , when the last loading process ends and $t_{c,j,0}$ the time, when the first joining process starts. Equation (9) ensures that in all cycles the joining process only starts after parts are loaded. Similar, the first unloading process $t_{c,u,0}$ can only start after the last joining processes $t_{c,j,1}$ has been finished, see equation (10).

$$t_{c,l,1} \leq t_{c,j,0} \quad \forall c \in C, \forall l \in L, \forall j \in J \quad (9)$$

$$t_{c,j,1} \leq t_{c,u,0} \quad \forall c \in C, \forall j \in J, \forall u \in U \quad (10)$$

4.3 Automation Framework

The automation framework comprises four sub-steps. Sub-step (i) analysis of the given joining sequence and product data. Sub-step (ii) adds every single joining step to a queue for the methodology to process. Sub-step (iii) calculates the most optimal station structure and design for each single joining step. Sub-step (iv) transfers the calculated assembly system solutions of each single joining step back into the overall assembly system structure under consideration of the assembly graph. In the following paragraphs, the single sub-steps are introduced:

(i) Joining Sequence and Product Data Analysis

The analysis of product and joining sequence comprises different aspects and denotes the preprocessing of the here presented algorithm. The first aspect is the analysis of different product variants. Each joining step can differ in its composition, depending on the variant: (sub-) assemblies and parts, joining technologies and the number of joining elements. Figure 6 portrays a composition of variants. In the figure, the colored frames indicate different variants referring to assemblies and single parts. The assembly system capabilities and capacities need to cover the requirements of all variants. Therefore, in the first phase, the algorithm calculates for each joining step the maximum number of parts which have to be handled in one cycle. Secondly, joining elements need to be grouped to jobs. Jobs are defined as sets of joining elements, which carry similar joining parameters and therefore can be conducted with the same tool. Each variant can hold different kind of jobs or equal jobs of deviant sizes. The algorithm needs to ensure, that for all given jobs the necessary equipment is available, which holds the corresponding capabilities. This means that same jobs in different variants can be merged. However, deviant jobs need to be considered separately. In addition, the order in which the jobs are processed has to be determined. In general, a sequential or parallel procedure is possible. The order depends on different aspects. Adhesive curves, for instance, need to be set on the parts before other joining technologies are applied (if the parts are already joined, it is impossible to add the glue between the single parts). Therefore, gluing is the only joining technology that is allowed to be placed before the geo station. Different mechanical joining processes can take place simultaneously. However, joining technologies such as laser welding processes require certain welding protection stations and hemming processes require special fixtures and should not be combined with other joining processes. For the comprehensive decision-making concerning the order of processes, the algorithm accesses a knowledge base containing a priority matrix for each joining technology combination.

(ii) Joining Step Queue

Due to equation (5), the affiliation of joining elements to their joining steps has to be respected. For this reason, the further calculation of optimal assembly system structures is based on each joining step individually. A queue of all single joining steps will serve as a task list for the algorithm to process.

(iii) Station Structure and Design

The algorithm's core component calculates the optimal system structure and the station design for each single joining step. In the early phases of RASD the optimal solution is the one with the lowest costs, taking into consideration the introduced constraints. It is evident that there is a large number of planning

alternatives in both the assembly system configuration and the balancing of each station. Still, a key prerequisite of the method is to find the optimal AS configuration, since deviations would lead to high costs in industrial practice. The hereby presented method is based on a branch and bound algorithm (BB). The branch and bound approach is a well-known method in combinatorial optimization. The BB method *branches* the optimization problem into several combinatorial sub-problems. *Bound* on the other hand points to the use of boundaries. The solution space, which is usually described in tree structures, enables the combination of all possible solutions for each subproblem. (Brusco und Stahl 2005) The method has major advantages in firstly the limitation of the solution space and secondly the memory occupation.

Branch and Bound Framework

In the BB framework each path equals one feasible station solution, to solve the RASD problem – the feasible region. The solution space comprises several search-trees, each of them for one feasible single production station. The search-trees are dynamically created depending on which jobs are in the joining step and on which resources have already been assigned to the assembly system. Figure 3 exemplifies a path for one small joining step with the following properties: (i) the maximum number of joining robots l_w in each production station is three and (ii) the number of jobs $|J|$ equals two. In the figure, the displayed search tree path shows the solution for one geo station and one respot station. Unfortunately, due to visualization reasons, it is not possible to show the whole solution space for this scenario. On the highest level, the above-introduced handling scenarios $h1$ and $h2$ are available for the algorithm to select. Beneath, marked with the light blue box called job 1, possible design alternatives to fulfill the requirements of job 1 are considered. In its first row, the possibility is given, whether to equip the handling robot (loading) with double equipment, which has been introduced as scenario $hj1$. Similarly, on the next level the other stations' handling robot, which is responsible for unloading the fixture, can be equipped with a stationary tool according to scenario $hj2$. The following row serves for selecting the number of conventional joining robots for job 1.

Figure 3: Branch and Bound Framework

The algorithm repeats the procedure for job 2. The design result for this search path is displayed on the right and includes the joining robots, the double equipment, and the stationary tool. Since the geo station design is finished at that point, the unloading robot has been added. Similarly, to the first station's calculation procedure, the following stations are calculated if necessary, based on each already calculated station solution. The following stations are respot stations, which means no additional parts are added and therefore no additional handling robot is required. Thus, the unloading robot of the predecessor station unloads the assembly and loads it in the following stations' fixture. In this example, the calculation terminates at that point, since all joining elements are assigned to resources.

While calculating, each node needs to be validated against the given feasibility constraints (1) - (10). Equation (1) demands to finish all processes within the cycle time. For this, the variable $t_{h,l}$ is the time required to load the number of parts $|M|$ inside a fixture. The standard process time for loading $t_{p,l}$ describes the duration for handling one part from the gripper into the fixture. Equation (11) leads to the total time for loading the station. Handling scenario $h2$ first collects the parts in one cycle time and transfers them all at once inside the fixture in the following cycle. The corresponding time $t_{h,l}$ equals $t_{p,l}$, since the number of parts equals one. During the remaining cycle time and parallel to the

joining and unloading processes, the parts are collected from the load carriers. The duration of $t_{h,c}$ is equally calculated with equation (12).

$$t_{h,l} = |M| * t_{p,l} \quad (11)$$

$$t_{h,c} = |M| * t_{p,c} \quad (12)$$

At the end of the cycle time the handling robot unloads the joined (sub-) assembly of the fixture and moves it towards the next station. Equation (13) calculates the required time for unloading the fixture. Since the parts are already attached to each other, $|M|$ assumes the value one. The following inequalities (14) and (15) examine the compliance with constraint (1), in which it is checked if the collection and loading process is smaller or equal to the cycle time and if the sum of loading and unloading times are smaller or equal to the cycle time.

$$t_{h,u} = |M| * t_{p,u} \quad (13)$$

$$t_{h,l} + t_{h,u} \leq t_c \quad (14)$$

$$t_{h,l} + t_{h,c} \leq t_c \quad (15)$$

Subsequently, equation (16) calculates the remaining time left in the cycle for conventional joining processes $t_{j,c}$ by subtracting the loading and unloading time from the cycle time. Logically, it is the time, in which the fixture is equipped with the parts and joining can proceed.

$$t_{j,c} = t_c - (t_{h,l} + t_{h,u}) \quad (16)$$

Constraint (6) demands to exceed a minimum availability criterion for each station. Equation (7) calculates the availability of each station based on the availability of every single resource and inequality (6) checks the compliance of the constraint. The number of resources is limited by this constraint.

Constraint (3) requires that all geo joining elements, affiliated to one joining step, are set in the geo fixture. Equation (17) calculates the number of geo joining elements $n_j(j1)$ which can be assigned to one conventional joining robot or a stationary tool ($hj2$), depending on the job. Equally, equation (18) calculates the number of joining elements $n_j(hj1)$ which can be assigned to a double equipment robot. Inequality (19) checks the compliance of constraint (3). For all jobs, the sum of assigned geo joining elements needs to equal the number of given geo joining elements.

$$n_{j,g,c}(j1) = \frac{t_{j,c}}{t_j} \quad (17)$$

$$n_{j,g,c}(hj1) = \frac{(t_{j,c} - t_{h,c})}{t_j} \quad (18)$$

$$n_{j,g,c} = \sum_{R_w} n_{j,g,c}(j1) + n_{j,g,c}(hj1) \quad \forall J \in JL \quad (19)$$

Lastly, equation (20) checks whether all joining elements have been assigned. The entirety of all joining elements are the sum of assigned geo and respot elements of all jobs over all stations and robots. The calculation of the number of respot joining elements equals the calculation of the number of geometry joining elements. At that point, no further equipment is required and represents the termination criterion of the algorithm.

$$n_j = \sum_c (n_{j,g,c} + n_{j,c}) \quad (20)$$

Station Design: Dynamic Solution Space and Search Strategy

Search strategies have a high impact on the efficiency of algorithms. In general, search strategies can be differentiated in uninformed and heuristic search strategies (Ertel 2017). Most popular uninformed strategies are breadth-first search and depth-first search. Heuristic strategies, on the other hand, use heuristics based on problem-specific knowledge. In this application, the search strategy will be based on a Branch and Bound backtracking algorithm. The following assertions explain the decision: (i) Completeness and exactness: in theory, the depth-first search is not finite and therefore neither complete nor optimal. In the here presented use case each joining robot node subtracts at least one joining element. Thus, all paths are finite and the depth-first search strategy is complete and optimal. (ii) Branch and Bound: depth-first search enables the comparison of the current path to already existing solutions. While calculating, the algorithm can neglect solutions that cause higher costs. The Branch and Bound procedure can be used to decrease the number of combinations significantly. (iii) Space complexity: compared to other search strategies, for the depth-first search, there is only the necessity to keep the current path in memory. Depth-first-search signifies that in each step the deepest node is extended. In backtracking the current node will not even be extended completely, rather only one of the nodes' children will be further analyzed (Russell und Norvig 2012).

The implementation of Orasda is based on nested recursive functions for adding stations and for adding resources. The algorithm's logic entails: (i) expansion station-wise (depth) through the (re-) call of the function *BeginnNewStation*, (ii) expansion job-wise (go through jobs and add robots; depth) through the (re-) call of the function *EquipRespotCell*, (iii) expansion robot-wise (increase n_r and add for current job; width). As soon as a path is fulfilled, the *BestSolution* object will be replaced, if the solution exceeds the cost value of the previous best solution. As soon as all possible paths have been calculated, the algorithm stops and provides the best solution.

The function *BeginNewStation* decides whether to generate a geometry station or a respot station and initiates the station calculation process. The following pseudo-code describes the logic of that function. This function will be (re-) called as long as there are joining elements to assign. The joining elements are listed in the *JobList*. The *JobList* involves all jobs with the corresponding joining element quantity—categorized in geometry and respot joining elements through the geo-flag. In addition, the *StationList* and the *BestSolution* are passed to the function. The *StationList* acts as temporary cache in which the current resource assignment is stored. In the first step the function checks whether there are joining elements with the geo-flag to assign (compare line 2). According to constraint 3, all geo joining elements need to be set in a geo station. For geo station all input parts/sub-assemblies of the joining step need to be collected – in contrast to respot stations. Therefore, for geo stations either scenario *h1* or alternatively scenario *h2* are worth considering. By applying equation (11) and (12), the compliance of constraint (1) can be confirmed (compare line 4). The function creates a new station, compare line 5. Subsequently, the function *EquipGeoStation* is called for adding joining robots to the station in all feasible and meaningful combinations (see pseudo-code function 2). A similar procedure is applied for scenario *h2*, in which all parts are collected by an up-streamed handling station, compare line 8-10. The second station depicts the actual geo station, line 12-14. For each feasible geo station combination for which respot joining elements are remaining, the function *BeginNewStation* is recalled, since for valid assembly system solutions further respot stations are required.

Function 1 of RASD Algorithm

```

1: function BeginNewStation(StationList, BestSolution, JobList)
2:   if geoflag == true then
3:     foreach HandlingScenario in HandlingScenarios
4:       if h1 == true && h1 == feasible then //equation 11, 12
5:         new station  $s_g$ 
6:          $s.add(r_{LR}, r_{ULR}, r_{LC}, r_{FX})$ 
7:         EquipGeoStation( $s$ , StationList, BestSolution, JobList)
8:       elseif h2 == true && h2 == feasible then //equation 11, 12
9:         new station  $s$ 

```

```

10:      s.add(rLR, rULR, rTBL, rLC)
11:      StationList.add(s)
12:      new station s
13:      s.add(rULR, rFX)
14:      EquipGeoStation(s, StationList, BestSolution, JobList)
15:    else
16:      print „handling process > cycle time“
17:    endif
18:  elseif geoflag == false then
19:    new station
20:    s.add(rURL, rFX)
21:    EquipRespotStation(s, StationList, BestSolution, JobList)
22:  endif
23: end function

```

Geometry Stations

The following pseudo-code describes the logic, which is applied to generate all feasible solutions for equipping the geo station. The function's main input is the geo station initiated in the *BeginnNewStation* function. The *EquipGeoStation* function initiates the nested recursive backtracking procedure, fills the geo station with robots, and checks the compliance to the constraints. Likewise, the recursive function call of function *BeginnNewStation* (see line 25) serves for the continuous creation of new stations and the call of *EquipGeoStation* (see line 27) for adding new robots for the different jobs to the respot station.

Every time the algorithm calls the function *EquipGeoStation*, it decides whether to select double equipment for the loading robot and/or a stationary tool for the unloading robot. The decision is made based on the auxiliary variables $d \in \{0,1\}$ and $e \in \{0,1\}$. Double equipment and stationary tools can be assigned for a job in a station only once, which is ensured by the variables $d_s \in 0,1$ and $e_s \in 0,1$. If d_s or e_s equals one, the double equipment or stationary tool has already been chosen for another job in the station, and for this reason is not available anymore (compare line 2+3). The number of joining elements that the double equipment can set is calculated by equation (18). The number that one r_{ULR} can set with a stationary tool is calculated by equation (17). In line 12 and 16 the BB-bounding takes place. As soon as there is an already better calculated solution, the algorithm discards the current path and goes back to the previous incomplete expanded node. The for-loop in line 13 adds conventional robots n_r for the current job until the limit l_w is reached. The next time the function is called, J is increased by one. According to constraint (3), all J_g have to be set in the fixture of the GS, for which the compliance is checked in line 23. If not, the function adds robots until the constraint criteria is fulfilled. If the maximum allowed number of robots l_w is reached and still not all geo elements can be set, there is no valid solution for this assembly step. For each valid solution, the residual time t_ε of each robot is calculated. The remaining capacity of robots can be further used for respot joining elements, if existing – see line 18 & 19. The robots are then added to the station. The existence of geo joining elements of further jobs, leads either to the termination of the loop or the recall of the current function for adding resources.

Function 2 of RASD Algorithm

```

1:  function EquipGeoStation(s, StationList, BestSolution, JobList)
2:    for d=0 to 1
3:      for e=0 to 1
4:        if (ds == 0 && d == 1) then
5:          rLR.add(rDE)
6:          AssignJoiningElements(rDE) //equation (18)
7:        endif
8:        if (es == 0 && e == 1)
9:          rULR.add(rST)
10:         AssignJoiningElements(rST) //equation (18)
11:        endif
12:        if i > li // inequality(6) == false then break //constraint (6)
13:        for n = 1 to (lw - |RJR|) //constraint (8)
14:          if inequality (18) == true then //constraint (3)
15:            s.add(n*rJR)

```



```

16:         if  $i > l_i$  // inequality(6) == false then break //constraint (6)
17:         AssignJoiningElements( $R_{JR}$ ) //equation (17)
18:     if  $t_e \geq t_j$  then
19:         AssignJoiningElements( $R_{JR}$ ) //equation (17)
20:     endif
21:     J++
22:     if J! = none then
23:         if J.geoflag == 0 then //constraint (3)
24:             StationList.add(s)
25:             BeginNewStation(StationList, BestSolution, JobList)
26:         else
27:             EquipGeoStation(s, StationList, BestSolution, JobList)
28:         endif
29:     else
30:         StationList.add(s)
31:         BestSolution=StationList
32:     endif
33: endif
34: endfor
35: endfor
36: endfor
37: endfor
38: end function

```

Respot stations

Based on each geo station solution, all feasible RS combinations are calculated. Similar to the logic of geo stations the function has a recursive characteristic, which iteratively (re-) calls the function *BeginNewStation* to create new stations. Likewise, the function calls itself for continuously adding robots for the different jobs as long as there are elements to assign in the *JobList* considering l_w and l_a . Different to the geo stations, each node in respot stations denotes a feasible station solution, since constraint (3) only refers to geo stations. As long as there are still joining elements to assign, the algorithm expands the solution space based on the backtracking approach. In addition, the double equipment solution is not a meaningful solution for respot stations, since there are no separated loading robots for the loading process. The unloading robot of the predecessor station processes the loading.

Function 3 of RASD Algorithm

```

1  function EquipRespotStation(s, StationList, BestSolution, JobList)
2      for e=0 to 1
3          if ( $e_s == 0$  &&  $e == 1$ ) then
4               $r_{ULR}.add(r_{ST})$ ;
5              AssignJoiningElements( $r_{ST}$ ) //equation (18)
6              if  $i > l_i$  // inequality(6) == false then break; //constraint (6)
7          endif
8          for n = 0 to ( $l_w - |R_{JR}|$ )
9              s.add( $n * r_{JR}$ )
10             AssignJoiningElements( $n * r_{JR}$ ) //equation (20)
11             if  $i > l_i$  // inequality7 == false then break; //constraint (6)
12             StationList.add( $s_{rs}$ )
13             if |J| > 0 then
14                 BeginNewStation(JobList, StationList, SolutionList)
15             endif
16             J++
17             if J! = none then
18                 BeginNewStation(StationList, BestSolution, JobList)
19                 EquipRespotStation( $s_{rs}$ , StationList, BestSolution, JobList)
20             else
21                 BestSolution=StationList
22             endif
23         endfor
24     endfor
25 end function

```

4.4 Overall Assembly System Structure

The algorithm has calculated the optimal assembly system design for each single joining step. At that time, the individual solutions of the assembly system must be reassembled. Therefore, successor attributes are added to each station. For those stations calculated by the search algorithm, the successor station results from the search tree path. All other stations' successors are derived from the assembly sequence structure, as introduced in chapter 3 (compare figure 1). Since the last station of each joining step is the predecessor station of the first station of the successor joining step, the station with no successor is the last station and depicts the root node of the assembly system tree-structure. For enabling the handling of finished assemblies to the next station, the algorithm adds a table in between the stations of different joining steps (compare figure 7).

5. Software Prototype and Industrial Scenario

In order to conduct tests on real industrial scenarios, the presented algorithm has been implemented in a prototype software tool. The prototype is file-based and written in the programming language c#. The import of joining sequences and product data is processed in the data format AutomationML (AML). Before applying MILP algorithms, it is of the utmost importance to analyze which extent the calculation can assume in industrial use-cases. The computational expense of MILP algorithms is high and increases exponentially, depending upon the input variables and their values to be combined. Hence, their efficiency is a decisive factor. Processing times, which are often used as evaluation criteria, highly depend on the applied computers and are therefore hard to compare, rather it has been proven meaningful to utilize the number of expanded nodes as evaluation criteria. For existing approaches, which have been introduced in chapter 2, either the calculability for industrial scenarios has not been evaluated or the approaches were limited to selected scenarios. For that reason, the evaluation in this paper focuses on the calculability of industrial scenarios since it is the prerequisite for any industrial application. Therefore, an analysis inside a European automotive manufacturer has been conducted and revealed the maximum values the algorithm's input variables can assume. The study examines all joining step characteristics of five car models. Figure 4 shows the value range that joining steps can assume: the range of the cycle time is between 45 and 150 s. Independent of the number of different jobs and the cycle time, the maximum number of joining elements does not exceed 200. If there is only one job in the joining step, the maximum number of joining elements found equals the maximum number of joining elements of 200. Joining steps of that extent usually contain welding jobs, since welding is still a fast and cheap way of joining. Alternative joining technologies, such as riveting, require longer standard process times and are therefore significantly more expensive and time-consuming, but allow the joining of multi-material assemblies. Joining steps containing joining technologies used for material mixes, which require process times higher than 3.5 seconds, do not contain more than 150 points. If there is a second job in the same assembly step, the second job has maximum 50 joining elements. In rare cases, there is a third job in one assembly step. In the third job, there are maximum ten more joining elements. Lastly, the study has revealed, that maximum four parts or subassemblies are to be joined in one joining step.

Cycle time: 45...150 s		
Job_1 Process time: 2.5...3.5 s; J : 1...200 Process time: 3.5...4.5 s; J : 1...150		
yes	jobs>1	no
Job_2 Process time: 2.5...4.5 s J : 1...50		
yes	jobs>2	no
Job_3 Process time: 2.5...4.5 s J : 1...10		
parts: 1...4		

Figure 4: Range of input data

The theoretical calculation of the number of combinations, for the here-presented algorithm, is complex. Hence, computational experiments have been conducted. Results are shown in figure 5, which confirms the importance of the presented study. The 3D plot shows the computational effort for the calculation of assembly system configurations and is based on 400 calculation experiments. The green surface shows the calculation effort for joining steps containing one spot welding job over the cycle time and the number of joining elements. Since the number of geometry joining elements is limited by the capacity of the first station (compare constraint 3), the experiments are limited to the respot elements. The number of expanded nodes in the search algorithm determines the computational expense. It is evident that the calculation expense increases exponentially. The black box demonstrates the industrial value range conducted by the study and confirms that the calculation of such scenarios is of a low extent. The blue graph in turn exhibits the number of combinations without the application of the bounding logic in the algorithm and underlines its high benefit: Already for the mentioned scenario with one job the number of combinations would highly increase and would be hard to calculate in an appropriate time.

Figure 5: Computational expense of the algorithm

It is not obvious which of the total possible inputs leads to the maximum number of expanded nodes. The following table 3 aims at determining the most challenging scenarios. The six test cases have been designed by fixing those variables for which the worst value is obvious – such as the cycle time. The shorter the cycle time, the higher the number of combinations. Hence, the cycle time is set to 45 s. The standard process time has been set to either 3.5 s or alternatively to 4.5 s and represents the worst value for the two cases of the maximum number of joining elements. Lastly, the number of joining elements for one, two and three jobs has always been set to the maximum. Only the ratio of joining elements has not been fully tested, due to the high calculation effort and the large number of possible ratio-combinations. Samples have shown that the ratio has an impact on the number of expanded nodes, but the number of nodes has never exceeded the amount of 10^9 . That proves that the number of existing combinations is easily calculable in between minutes on normal, commercially available computers.

Table 3
Test Cases

Test Case	Job1	Job2	Job3	no. expanded nodes
1	$t_{sp} = 3.5 \text{ s} \mid JE: 200$	-	-	$6,7 * 10^4$
2	$t_{sp} = 4.5 \text{ s} \mid JE: 150$	-	-	$6,6 * 10^5$
3	$t_{sp} = 3.5 \text{ s} \mid JE: 150$	$t_{sp} = 4.5 \text{ s} \mid JE: 50$	-	$3,64 * 10^7$
4	$t_{sp} = 4.5 \text{ s} \mid JE: 100$	$t_{sp} = 4.5 \text{ s} \mid JE: 50$	-	$1,74 * 10^8$
5	$t_{sp} = 3.5 \text{ s} \mid JE: 150$	$t_{sp} = 4.5 \text{ s} \mid JE: 40$	$t_{sp} = 4.5 \text{ s} \mid JE: 10$	$2,72 * 10^8$
6	$t_{sp} = 4.5 \text{ s} \mid JE: 100$	$t_{sp} = 4.5 \text{ s} \mid JE: 40$	$t_{sp} = 4.5 \text{ s} \mid JE: 10$	$9,89 * 10^8$

The algorithm has been tested with a high number of real industrial scenarios. One of the assembly system solutions obtained by the algorithm will be examined and discussed below regarding its feasibility, optimality and further improvement potential. The selected scenario depicts the last part of the rear car assembly and is subject to the cycle time of 60 s. The corresponding assembly sequence is shown in figure 6. The assembly sequence contains four joining steps with 1-2 jobs each. The t_j or spot welding is 2.5 s and for riveting is 4.5 s. For geometry joining elements, which are written in brackets behind the respot joining elements, the standard process time is one second longer for both joining technologies. The two different vehicle variants considered are differentiated by the colors orange and green. Jobs conducted by the preprocessor are shown in the black boxes. For joining step I $6.1 * 10^4$ nodes have been expanded, for joining step II 95 nodes, for joining step III 55 nodes and for joining step IV 189 nodes. The total number of nodes amounts to 61985. Without the application of the bounds in the algorithm, the costs of the solutions are distributed so strongly that the most expensive solution is 193 % more costly compared to the cheapest solution. The high cost distribution confirms once again the importance of considering the holistic solution space.

Figure 6: Industrial Case Study

Figure 7 shows the calculated assembly system and a table with evaluation indicators, comparing the generated system with the real assembly system. The frames around the stations indicate the affiliation to the joining steps. The assembly system has 20 workstations. 15 handling robots enable the handling in between those stations. Three of those handling robots carry double equipment and nine of them are equipped with stationary tools. In total, 31 joining robots have been assigned to the stations, five of them for riveting and all others for spot welding. The robots have a utilization rate of 72.14 %. 68.05 % of this time is applied for joining processes (value adding processes) and 36.67 % for handling processes (non-value adding processes). Idle times represent 27.86 %. Discussions with assembly system design experts have revealed several positive and negative aspects concerning this calculated assembly system solution. Positive to mention is the confirmation of the high potential of the automated design of assembly systems. The RASD experts confirmed the feasibility of the calculated system. In addition, the experts confirmed the high planning quality of the calculated solution and the tremendous potential of cost- and time-saving in the early RASD phases, even though a holistic evaluation of the algorithm's solutions, from a RASD view, is still pending. First point to mention is the use of the same fixtures and grippers for different variants. Despite nowadays flexible grippers and fixtures allow for the processing of several product variants, they are still required for parts which vary significantly in their geometry. The decision whether to use one or several fixtures for different variants depends on the distance of clamping positions. The automated analysis of those distances might be a future research topic. In addition, the application of stationary tools and double grippers has been discussed. In general, the application of those applications has a lot of potential to increase the utilization of handling robots. Nevertheless, the application for large parts can lead to collisions and therefore requires at least a rough estimation of a RASD expert. For the further improvement of the algorithm, experts have suggested to examine the handling of symmetrical parts, such as the single parts of all joining steps in this example. Symmetrical parts or even parts that do not have contact surfaces to each other, might be handled in one process step in a parallel way, which in turn leads to time saving.

The comparison with the real production system enables a more quantitative examination of the generated assembly system. Nevertheless, caution is advised when comparing the rough planning solution (by ORASDA) with a real assembly system. The planning of a real assembly system takes numerous additional boundary conditions into account, such as the building geometry, robotic simulations, material flow simulations, buffers and decoupling concepts, etc. In addition, the impact on the solution highly depends on the responsible planner. However, the comparison of the indicators shown in the table below indicates that the generated plant approximates the manually planned solution.

	Orasda Assembly System Initial Production System Configuration	Existing Assembly System Planned, Optimized and Evaluated System
Cost (fictitious values)	4.76 MUSD	5.22 MUSD
Balancing efficiency	72,14 %	67,18 %
Throughput time	1191 s	1262 s
No. of stations	20	22
No. of robot systems	43	46
- Welding	23 (+10)	26 (+6)
- Riveting	5	4
- Handling	15	18

Figure 7: Industrial scenario front car

6. Conclusion and future research

The design of robotic assembly systems, as applied in the automotive body-in-white production stages, is both a complex and time-consuming task. In early design phases, engineers are confronted with numerous design alternatives. The objective to design the assembly system in the most cost-efficient way is based on the individual experience, intuition and creativity of the designers. It has been stated that feasible assembly system design alternatives vary extremely in costs. Current software tools enable a digital planning process and an integrated data concept. Nevertheless, none of them supports the assembly system design in the actual creative, elaborative design process. Exactly this research gap has been tackled in this paper. Therefore, a novel algorithm has been introduced that automatically generates assembly system alternatives based on given assembly sequences, product data and project premises. The algorithm is not limited to purely sequential assembly lines, instead capable to design assembly systems of deviating structures, including the therefore necessary design of complex joining and handling scenarios.

Orasda bases on a MILP formulation, for which the computational effort increases exponentially depending on the given input data. Computational experiments have been conducted to confirm the calculability for industrial scenarios within reasonable times. Likewise, the evaluation of the assembly system solutions has revealed that results are feasible and of high quality. Still, the evaluation has unveiled further optimization potential. Future research will majorly focus on two aspects. Firstly, the consideration of additional input information, gathered from automated and assembly-oriented 3D product analysis. Secondly, on the addition of further scenarios to combine, for instance varying

alternative joining sequences. This extension of the algorithm will lead to an increased solution space and might require more precise bounding conditions, the application of advanced search strategies or even the application of heuristics.

References

- Bauernhansl, Thomas (2014): Die Vierte Industrielle Revolution – Der Weg in ein wertschaffendes Produktionsparadigma. In: Thomas Bauernhansl, Michael ten Hompel und Birgit Vogel-Heuser (Hg.): Industrie 4.0 in Produktion, Automatisierung und Logistik. Anwendung, Technologien, Migration. Wiesbaden: Springer Vieweg (SpringerLink), S. 5–24.
- Biesinger, Florian; Meike, Davis; Krass, Benedikt; Weyrich, Michael: A Case Study for a Digital Twin of Body-in-White Production Systems General Concept for Automated Updating of Planning Projects in the Digital Factory. In: Conference Proceedings IEEE Conference on Emerging Technologies and Factory Automation, Bd. 23, S. 19–26.
- Biesinger, Florian; Meike, Davis; Kraß, Benedikt; Weyrich, Michael (2018): A digital twin for production planning based on cyber-physical systems: A Case Study for a Cyber-Physical System-Based Creation of a Digital Twin. A Case Study for a Cyber-Physical System based Creation of a Digital Twin. In: Conference Proceedings CIRP Conference on Intelligent Computation in Manufacturing, Bd. 79, S. 355–360.
- Boysen, Nils; Fliedner, Malte (2008): A versatile algorithm for assembly line balancing. In: *European Journal of Operational Research* 184 (1), S. 39–56. DOI: 10.1016/j.ejor.2006.11.006.
- Boysen, Nils; Fliedner, Malte; Scholl, Armin (2007): A classification of assembly line balancing problems. In: *European Journal of Operational Research* 183 (2), S. 674–693. DOI: 10.1016/j.ejor.2006.10.010.
- Bracht, Uwe; Geckler, Dieter; Wenzel, Sigrid (2011): Digitale Fabrik. Methoden und Praxisbeispiele. Berlin u.a.: Springer (VDI-Buch).
- Brusco, Michael J.; Stahl, Stephanie (2005): Branch-and-Bound Applications in Combinatorial Data Analysis. New York: Springer-Verlag (Statistics and Computing).
- Burr, Holger (2008): Informationsmanagement an der Schnittstelle zwischen Entwicklung und Produktionsplanung im Karosserierohbau. Dissertation. Universität des Saarlandes, Saarbrücken. Lehrstuhl für Konstruktionstechnik/CAD.
- Damaševičius, Robertas (2008): Structural Analysis of Promoter Sequences Using Grammar Inference and Support Vector Machine. In: Ignac Lovrek (Hg.): Knowledge-based intelligent information and engineering systems. 12th international conference : proceedings, Bd. 5177. Berlin: Springer (Lecture notes in computer science, 5177-5179), S. 98–105.
- Daoud, Slim; Amodeo, Lionel; Yalaoui, Farouk; Chehade, Hicham; Duperray, Philippe (2012): New mathematical model to solve robotic assembly lines balancing. In: *IFAC Proceedings Volumes* 45 (6), S. 1353–1358. DOI: 10.3182/20120523-3-RO-2023.00183.
- Ertel, Wolfgang (2017): Introduction to Artificial Intelligence. Second edition. Cham: Springer International Publishing (Undergraduate Topics in Computer Science).
- Gao, Jie; Sun, Linyan; Wang, Lihua; Gen, Mitsuo (2009): An efficient approach for type II robotic assembly line balancing problems. In: *Computers & Industrial Engineering* 56 (3), S. 1065–1080. DOI: 10.1016/j.cie.2008.09.027.

- Ghosh, Soumen; GAGNON, ROGER J. (1989): A comprehensive literature review and analysis of the design, balancing and scheduling of assembly systems. In: *International Journal of Production Research* 27 (4), S. 637–670. DOI: 10.1080/00207548908942574.
- Hagemann, Simon; Stark, Rainer (2018): Automated Body-in-White Production System Design. Data-Based Generation of Production System Configurations. In: Proceedings of the 4th International Conference on Frontiers of Educational Technologies. New York NY: ACM, S. 192–196.
- Hagemann, Simon; Sünnetcioglu, Atakan; Stark, Rainer (2019): Hybrid Artificial Intelligence System for the Design of Highly-Automated Production Systems. In: *Procedia Manufacturing* 28, S. 160–166. DOI: 10.1016/j.promfg.2018.12.026.
- Hsieh, L.-H.; Schmahls, T.; Seliger, G. (1997): Assembly Automation in Europe - Past Experience and Future Trends. In: Koichi Shimokawa, Ulrich Jürgens und Takahiro Fujimoto (Hg.): Transforming Automobile Assembly. Experience in Automation and Work Organization. Berlin, Heidelberg: Springer Berlin Heidelberg, S. 19–37.
- Hu, S. J.; Ko, J.; Weyand, L.; ElMaraghy, H. A.; Lien, T. K.; Koren, Y. et al. (2011): Assembly system design and operations for product variety. In: *CIRP Annals* 60 (2), S. 715–733. DOI: 10.1016/j.cirp.2011.05.004.
- Jasperneite, Jürgen; Niggemann, Oliver (2012): Systemkomplexität in der Automation beherrschen. In: *atp edition*, S. 36–44.
- Kiefer, Jens (2007): Mechatronikorientierte Planung automatisierter Fertigungszellen im Bereich Karosseriebau. Dissertation. Universität des Saarlandes, Saarbrücken. Lehrstuhl für Fertigungstechnik/CAM.
- Kim, H.; Park, S. (1995): A strong cutting plane algorithm for the robotic assembly line balancing problem. In: *International Journal of Production Research* 33 (8), S. 2311–2323. DOI: 10.1080/00207549508904817.
- Klauke, Sabine (2002): Methoden und Datenmodell der "Offenen Virtuellen Fabrik" zur Optimierung simultaner Produktionsprozesse. Als Ms. gedr. Düsseldorf: VDI-Verl. (Fortschritt-Berichte VDI Reihe 20, Rechnerunterstützte Verfahren, 360).
- Kretschmer, R.; Pfouga, A.; Rulhoff, S.; Stjepandić, J. (2017): Knowledge-based design for assembly in agile manufacturing by using Data Mining methods. In: *Advanced Engineering Informatics* 33, S. 285–299. DOI: 10.1016/j.aei.2016.12.006.
- Kusiak, Andrew: The production equipment requirements problem. In: *International Journal of Production Research*, Bd. 25, S. 319–325.
- Levitin, Gregory; Rubinovitz, Jacob; Shnits, Boris (2006): A genetic algorithm for robotic assembly line balancing. In: *European Journal of Operational Research* 168 (3), S. 811–825. DOI: 10.1016/j.ejor.2004.07.030.
- Lopes, Thiago Cantos; Sikora, C.G.S.; Molina, Rafael Gobbi; Schibelbain, Daniel; Rodrigues, L.C.A.; Magatão, Leandro (2017): Balancing a robotic spot welding manufacturing line: An industrial case study. An industrial case study. In: *European Journal of Operational Research* 263 (3), S. 1033–1048. DOI: 10.1016/j.ejor.2017.06.001.
- Michalos, G.; Fysikopoulos, A.; Makris, S.; Mourtzis, D.; Chryssolouris, G. (2015): Multi criteria assembly line design and configuration – An automotive case study. In: *CIRP Journal of Manufacturing Science and Technology* 9, S. 69–87. DOI: 10.1016/j.cirpj.2015.01.002.
- Michels, Adalberto Sato; Lopes, Thiago Cantos; Sikora, Celso Gustavo Stall; Magatão, Leandro (2018): The Robotic Assembly Line Design (RALD) problem: Model and case studies with practical extensions. In: *Computers and Industrial Engineering*, Bd. 120, S. 320–333.

- Naveen Kumar & Dalgobind Mahto (2013): Assembly Line Balancing: A Review of Developments and Trends in Approach to Industrial Application. In: *Global Journal of Researches in Engineering and Industrial Engineering*, S. 1–23.
- Niggemann, Oliver; Jasperneite, Jürgen; Vodencarevic, Asmir (2014): Konzepte und Anwendungsfälle für die intelligente Fabrik. In: Thomas Bauernhansl, Michael ten Hompel und Birgit Vogel-Heuser (Hg.): *Industrie 4.0 in Produktion, Automatisierung und Logistik. Anwendung, Technologien, Migration*. Wiesbaden: Springer Vieweg (SpringerLink), S. 173–190.
- Raimung Menges (2005): Maßstab für die Fertigung im 21. Jahrhundert. Alle Phasen des Produktlebenszyklus in die Unternehmensstrategie einbezogen. In: *Intelligenter Produzieren* 2005 (1), S. 14–16.
- Rekiek, Brahim; Delchambre, A. (2006): Assembly line design. The balancing of mixed-model hybrid assembly lines with genetic algorithms. London: Springer (Springer series in advanced manufacturing).
- Rubinovitz, j.; Bukchin, J. (1991): Design and balancing of robotic assembly lines. In: *Proceedings of the Fourth World Conference on Robotics Research*.
- Russell, Stuart J.; Norvig, Peter (2012): *Künstliche Intelligenz. Ein moderner Ansatz*. 3., aktualisierte Aufl. München: Pearson (Always learning).
- Schmigalla, Hans (1995): *Fabrikplanung. Begriffe und Zusammenhänge*. 1. Aufl. München: Hanser (REFA-Fachbuchreihe Betriebsorganisation).
- Sivasankaran, P.; Shahabudeen, P. (2014): Literature review of assembly line balancing problems. In: *Int J Adv Manuf Technol* 73 (9-12), S. 1665–1694. DOI: 10.1007/s00170-014-5944-y.