THESIS FOR THE DEGREE OF DOCTOR OF TECHNOLOGY

Mathematical Modelling and Methods for Load Balancing and Coordination of Multi-Robot Stations

EDVIN ÅBLAD

CHALMERS

Division of Applied Mathematics and Statistics
Department of Mathematical Sciences
CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg, Sweden 2022

Mathematical Modelling and Methods for Load Balancing and Coordination of Multi-Robot Stations
EDVIN ÅBLAD
ISBN 978-91-7905-661-2

Doktorsavhandlingar vid Chalmers tekniska högskola
Ny serie nr 5127
ISBN 0346-718X

Department of Mathematical Sciences
Chalmers University of Technology and University of Gothenburg
SE-412 96 Göteborg, Sweden
Phone: +46 (0)31 772 1000

Edvin Åblad is employed by the
Fraunhofer-Chalmers Research Centre for Industrial Mathematics
Chalmers Science Park
SE-412 88 Göteborg, Sweden
Phone: +46 (0)31 772 4275

Online version: `https://research.chalmers.se/person/edvind`

Cover image: Stud welding station and Voronoi Diagrams

Typeset with LaTeX.
Printed by Chalmers Reproservice
Göteborg, Sweden 2022

Mathematical Modelling and Methods for Load Balancing and Coordination of Multi-Robot Stations

EDVIN ÅBLAD
Division of Applied Mathematics and Statistics
Department of Mathematical Sciences
Chalmers University of Technology and University of Gothenburg

Abstract

The automotive industry is moving from mass production towards an individualized production, individualizing parts aims to improve product quality and to reduce costs and material waste. This thesis concerns aspects of load balancing and coordination of multi-robot stations in the automotive manufacturing industry, considering efficient algorithms required by an individualized production. The goal of the load balancing problem is to improve the equipment utilization. Several approaches for solving the load balancing problem are suggested along with details on mathematical tools and subroutines employed.

Our contributions to the solution of the load balancing problem are fourfold. First, to circumvent robot coordination we construct disjoint robot programs, which require no coordination schemes, are flexible, admit competitive cycle times for several industrial instances, and may be preferred in an individualized production. Second, since solving the task assignment problem for generating the disjoint robot programs was found to be unreasonably time-consuming, we model it as a generalized unrelated parallel machine problem with set packing constraints and suggest a tailored Lagrangian-based branch–and–bound algorithm. Third, a continuous collision detection method needs to determine whether the sweeps of multiple moving robots are disjoint. We suggest using the maximum velocity of each robot along with distance computations at certain robot configurations to derive a function that provides lower bounds on the minimum distance between the sweeps. The lower bounding function is iteratively minimized and updated with new distance information; our method is substantially faster than previously developed methods. Fourth, to allow for load balancing of complex multi-robot stations we generalize the disjoint robot programs into sequences of such; for some instances this procedure provides a significant equipment utilization improvement in comparison with previous automated methods.

Keywords: automotive manufacturing, Smart Assembly 4.0, makespan minimization, motion planning, Voronoi diagram, set packing, continuous collision detection, decomposition, mathematical modelling, vehicle routing

List of publications

This thesis is based on the following appended papers:

Paper I E. Åblad, D. Spensieri, R. Bohlin, and J. S. Carlson.
Intersection-free geometrical partitioning of multirobot stations for cycle time optimization.
IEEE Transactions on Automation Science and Engineering 15(2) (2018) 842–851 doi:10.1109/TASE.2017.2761180

Paper II E. Åblad, A.-B. Strömberg, and D. Spensieri.
Exact makespan minimization of unrelated parallel machines.
Open Journal of Mathematical Optimization, 2 (2021) art. no. 2, 15 p. doi:10.5802/ojmo.4.

Paper III E. Åblad, D. Spensieri, R. Bohlin, and A.-B. Strömberg.
Continuous collision detection of pairs of robot motions under velocity uncertainty.
IEEE Transactions on Robotics, 37(5) (2021) 1780–1791
doi:10.1109/TRO.2021.3050011.

Paper IV E. Åblad, D. Spensieri, R. Bohlin, J. S. Carlson, and A.-B. Strömberg.
Spatial–temporal load balancing and coordination of multi-robot stations.
Submitted for journal publication (2022)

Paper V E. Åblad, A.-B. Strömberg, and D. Spensieri.
A Lagrangian-based method for makespan minimization of parallel machines with set packing constraints.
Manuscript (2022)

Specification of my contribution to the appended papers:

Paper I: I developed the main algorithm, task assignment model, and the approximation algorithm for the Generalized Voronoi diagram. I implemented the code, ran the experiments and wrote the manuscript. The co-authors contributed ideas regarding algorithms, the software IPS, and structuring and formulating the final manuscript.

Paper II: I suggested and analysed the auxiliary variables. I implemented and tuned the Lagrangian based branch–and–bound algorithm. The co-authors aided with ideas regarding algorithms, implementations, and formulation of the manuscript.

Paper III: I formalized the clearance bound and suggested the sampling technique. I developed, implemented, and evaluated the optimization routine of the clearance bound function, as well as the octree approximation. D. Spensieri and R. Bohlin provided knowledge of the existing clearance routine and sensitivity analysis of robot paths, respectively. I produced the final manuscript with consultations from all co-authors.

Paper IV: I did all preliminary tests that converged to the two algorithmic stages and the formulation of the corresponding models. I implemented the interfacing scripts, and for the two-dimensional instances also the motion planning and partitioning routines. I did all the computational experiments and the complete initial draft of the manuscript. The co-authors aided with fruitful discussions on modelling and algorithms, and with feedback on the formulations and illustrations in the final manuscript.

Paper V: I did the initial tests resulting in the Lagrangian relaxation. I implemented the concurrent branch–and–bound algorithm, including heuristics, Lagrange dual maximization, and variable fixing. The co-authors aided with ideas regarding algorithms, implementations, and formulation of the manuscript.

Other relevant publications co-authored by Edvin Åblad:

- D. Spensieri, E. Åblad, R. Bohlin, J. S. Carlson, and R. Söderberg.
  Modeling and optimization of implementation aspects in industrial robot coordination.
  Robotics and Computer-Integrated Manufacturing 69 (2021) 102097 doi:10.1016/j.rcim.2020.102097.

- D. Spensieri, E. Åblad, J. Kressin, J. S. Carlson, and A. Andersson.
  Collision-free robot coordination and visualization tools for robust cycle time optimization.
  ASME Journal of Computing and Information Science in Engineering 21(4) (2021) 041011 doi:10.1115/1.4050047.

Acknowledgements

First and foremost, I would like to thank my industrial supervisors Dr. Robert Bohlin and Dr. Domenico Spensieri at the Fraunhofer–Chalmers Centre (FCC), for all the support given in terms of algorithmic expertise, implementation aspects, and in terms of encouragement. I would also like to thank my co-workers at FCC and the PhD students of the optimization group for creating an inspirational and pleasant environment.

I would like to especially thank my main supervisor Prof. Ann-Brith Strömberg at the Department of Mathematical Sciences at Chalmers University of Technology and Gothenburg University, for her engagement in this work, and especially for her high-quality peer-reviews and valued feedback.

Moreover, I would like to give a special thanks to the funding project Swedish Foundation for Strategic Research, project no. RIT15-0025, Smart Assembly 4.0.

And last, but not least, thanks to my friends and my loving wife for all their encouraging support, and also to my newborn dotter for setting a deadline.

Edvin Åblad
Gothenburg, March 7, 2022

Contents

1 Introduction

This thesis is motivated by the need to improve equipment utilization and throughput in automotive manufacturing systems; see [9, 85]. Our focus is on production lines (in particular, assembly lines), in which a workpiece moves through a sequence of stations (e.g., assembly cells) in each of which a set of tasks are performed; see Figure 1. The tasks could be carried out by humans or by robots (which is the focus of the thesis), and the types of tasks include welding, sealing, gluing, mounting, and other operations.

Figure 1: An assembly cell performing welding tasks on a workpiece that transitions through the assembly line.

The problem of distributing tasks among agents (robots or humans) in a production line is well-studied and is known as *line balancing*, where the term *balancing* refers to the objective that the agents should be equally occupied. This objective directly translates into *cycle time* (also known as makespan), i.e., the time it takes to complete the tasks in a station. The *cycle time* also relates to the production line throughput, and thus to the overall equipment utilization. Our work considers both single and multiple stations, but rarely an entire production line. Hence, we refer to the problem as *load balancing*. This also to emphasize that a more detailed solution is desired as compared to what is often used in *line balancing*; see Section 1.2.

1.1 Stations in the automotive industry

A typical station consists of several industrial robotic arms (or *industrial robots*) mounted around the workpiece location. These robots can be of varying types depending on the tasks; as an example, the robot in Figure 2 comprises six revolute joints, enabling it to position its tool according to each task. A robot *configuration* is a specification for each of these joints (e.g., angles of revolute joints), and the set of all configurations is called the *configuration space*. A certain position (and rotation) of the tool corresponds to multiple configurations, which

are referred to as *inverse configurations* of that tool position. Moreover, a *robot motion* is defined by a *path* in the configuration space and a velocity profile along that path. Thus, if the velocity of the robot motion is not emphasized, we refer to it as a robot path. See [83] for details on kinematics.

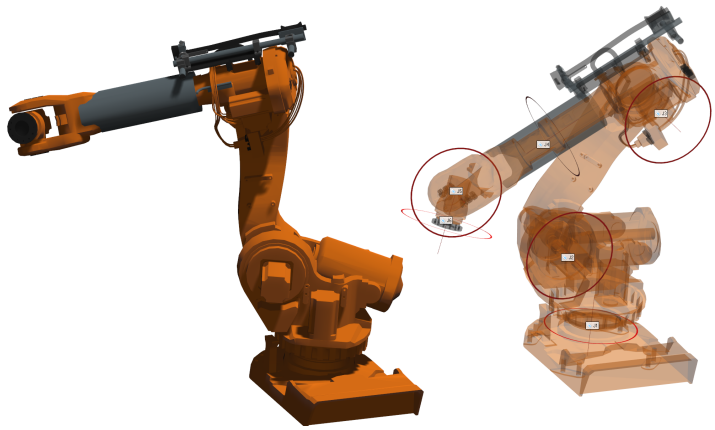

Figure 2: Two configurations of a typical industrial robot with six revolute joints. In the right illustration, the rotational axes are highlighted by red circles.

A task to be performed in a station can be of various types, but they all reduce to the same definition. For a task to be completed the robot's tool is required to follow a motion. The motion of the tool can be specified—e.g., such that the tool is required to be in a specific position—or not completely specified—e.g., that a stud weld must be placed normal to a plane, but its rotation in that plane is unspecified. Moreover, as every tool position corresponds to several robot *inverse configurations*, each task can be performed by a (possibly infinite) number of *alternative* motions, each with start and end configurations of the robot.

The robot's motion between pairs of tasks needs to be collision-free w.r.t. the environment and other robots. The motion is achieved by a *controller* that is specific for each type of robot. The controller is provided with a list of instructions, mainly consisting of configurations on the paths between the tasks and details on how to move between them. For example, the joints can be instructed to have a constant velocity, or the tool be instructed to follow a linear motion (in the workspace). The robot can be instructed to stop at a configuration or to pass by it at any feasible speed. Moreover, to verify that the motion is collision-free w.r.t. the workpiece, it must be analysed in a simulation tool.

Another important type of instruction coordinates multiple robots in order to prevent robot–robot collisions. When a robot enters a workspace shared with

another robot, it sends a *wait* instruction to a *programmable logic controller* (PLC), which responds whether the workspace is free or occupied. When the robot leaves the shared workspace, a *free* instruction is sent to the PLC to allow other robots to enter. This system of signals can be seen as a way to modify the velocity of the robot motions to ensure a collision-free program, but more importantly as a safeguard against collisions due to unexpected robot motion failures. Note that there are alternative systems and instructions enabling robots to collaborate in a shared workspace, e.g., ABB's multimove. See [2] for examples of robot instructions.

1.2 The load balancing problem in line production

In general, given one or several stations the *load balancing* problem is to divide the tasks among the robots, decide a task sequence and a corresponding motion for each robot, such that the cycle time is minimized and without collisions with the environment or among robots. To model this problem, it is common to use a simulation software such as RobotStudio [1] or Industrial Path Solutions (IPS) [58], in which, e.g., the geometry and the robot motions can be represented; see Figure 3.

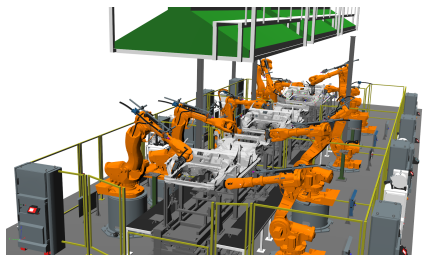

Figure 3: An assembly line modelled in IPS. Courtesy of Volvo Cars.

There are many variations of the load balancing problem, e.g., replacing robots with humans or to minimize the energy consumption rather than the cycle time. This thesis will consider some variations in Section 2, and also some limitations in Section 1.5.

1.3 Towards an individualized production

We denote a solution to the load balancing problem to be an *offline solution*, if it has been constructed by engineers, possibly aided by simulation and optimization software. Such solutions are currently used in industry, since the load balancing problem is composed of several tightly connected problems.

Moreover, it is common that a model excludes some details, e.g., simulating flexible components and their interaction with the robots, or using a simplified controller. Hence, any solution computed will need to be analysed and possibly modified. Therefore, a robot program is generally time-consuming to create. On the one hand, if this robot program is intended to produce many identical products (mass production), then the time usage is not a crucial issue. On the other hand, if programs need to be tailored to smaller production series, then the time to create the robot programs can be problematic.

The mass production concept is challenged by the project Smart Assembly 4.0 (SA4.0), where one key point is that the product quality can be substantially increased (or its production cost be reduced) by considering every product as an individual. The idea is that the physical production system, together with an accurate digital copy, a *digital twin*, enables a simulation and/or optimization to be conducted for individual products; see [104, 97]. A concept of an individualized production is illustrated in Figure 4. A consequence of an individualized production is that offline solutions cannot be used in such systems.

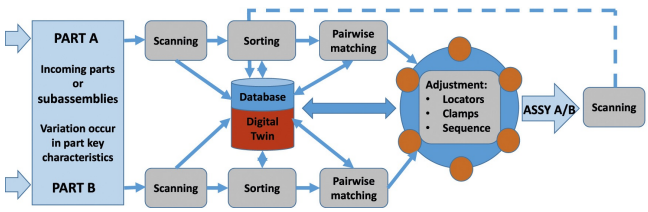

Figure 4: SA4.0 setup where the input to the assembly cell (before scanning) has been optimized using the digital twin. E.g., locators have been modified, thus placing the workpiece in a slightly different position. See [104] for details on the SA4.0 setup.

One enabler for an individualized production is the ability to automatically generate solutions to the load balancing problem, i.e., *online solutions*. This means that the load balancing problem needs to be solved for every individual product, using models with enough detail to produce complete robot programs. So, in contrast to an *offline solution* no verification or modification by an engineer can be permitted, since such a process would be too time-consuming to be conducted for every product individual.

1.4 Objectives

The main goal of this thesis is the minimization of losses (i.e., increments of cycle time) caused by the robot coordination. Improving the equipment utilization by minimizing the cycle time is a related and important aspect to be considered in the thesis.

Motivated by the need for *online solutions* arising from an individualized production, we contribute to the efficient generation of robot programs. This need could also be met by generating *robust* robot programs, i.e., programs that could be reused or modified to solve similar problem instances.

The need for automatically computed online solutions also motivates a generalization of existing models and algorithms for load balancing and coordination of multi-robot stations to handle instances requiring the robots to collaborate in a cluttered workspace. This relates to coordination losses since a central complication in the load balancing problem is preventing robot–robot collisions, which affects robot–task assignments, task sequences, and the robots' motions.

1.5 Limitations

We consider applications and instances from the automotive industry. However, many of the methods and ideas will, after suitable adaptations, apply to other types of production lines in which robots operate in a shared workspace.

We will consider the load balancing problem for robots only, and not for human agents. Moreover, we will assume that the cycle time is to be minimized.

We will only briefly consider precedence constraints between tasks (see Section 2.5), which enforce the tasks to be executed in a specific order due to logical constraints or to improve product quality.

We do not cover the topic of robot controllers but assume that the robot possesses constraints on the velocities of, e.g., joints or the tool. This is compatible with the goal of creating robot programs online and is accomplished by applying a preprocessing algorithm.

A related topic is the dynamic effects caused by the weight of a robot and, similarly, the simulation of deformable parts of the robot (e.g., cables), which is left out of this thesis in order to simplify the presentation.

1.6 Outline

This thesis surveys the load balancing problem for industrial robots. In Section 2, we review approaches to solve the load balancing problem considering different models and assumptions. In Section 3, we give a mathematical background to the methods used in the appended papers and in routines used for load balancing. In Section 4, we present details of the subproblems and routines presented in Section 2.1. The appended papers are summarized in Section 5 while in Section 6 we state the main conclusions of this thesis and pointers to future work.

2 Approaches to the load balancing problem

We here cover the variations of the load balancing problem that is specified in Section 1.2, with the limitations mentioned in Section 1.5. Note that these approaches prevent that pairs of robots collide (robot–robot collisions) using different methods, and since they study different applications also the motion planning problems have varying complexity. As a result, not all of these approaches are interchangeable or even solve the same problem. This section is concluded by two subsections regarding the complexity of the motion planning problem, and the commonly used *shortest-path assumption* that decouples the planning of motions of different robots; see Sections 2.7 and 2.8, respectively.

2.1 Lazy load balancing

The *lazy load balancing* algorithm by Spensieri et al. [106] is composed of a few complex steps in a loop; see Figure 5. The idea is to decompose the load balancing problem into two parts, the combinatorial problem of assigning and sequencing the tasks and the geometrical problem of planning the robots' motions.

The initial step aims to find a small set of alternative ways to perform each task without colliding with the environment. Each alternative is associated with collision-free start and end configurations and a collision-free motion for the robot (where the tool follows a given motion). The resulting set consists of samples from the (possibly infinite) set of feasible alternatives. The idea employed in lazy load balancing is to use the concept of inverse configurations, and to partition the set of alternatives into connected subsets, in each of which the same inverse configuration apply, see Section 4.1 for details.

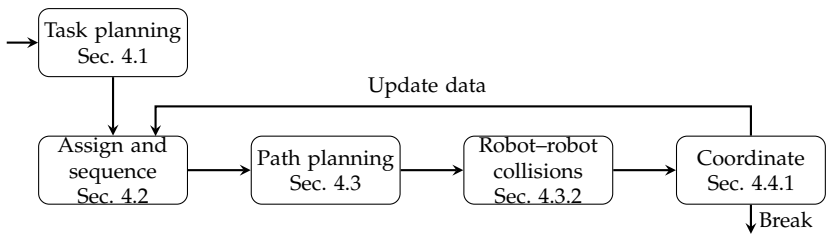

Figure 5: The iterative lazy load balancing approach described in [106].

The first step in the loop is to solve the problem of assigning each task to a robot, selecting an alternative for each task, and deciding an ordering of the tasks, in a collision-free way with the objective to minimize the cycle time. The loop follows a so-called lazy approach, where the word lazy—borrowed from

lazy path planning, see [20]—means that initially no robot motions are computed, and thus no collisions are known, while the motion times are bounded from below. The three following steps in the loop are: path planning between the configurations chosen in the first step (see Section 4.3); detection of robot–robot collisions among these paths, i.e., which might cause a collision (see Section 4.3.2); and coordination of the robots' motions, i.e., tuning their velocities so that collisions are prevented (see Section 4.4.1).

Before the loop continues to the next iteration, the data in the task assignment and sequencing model is updated, by including the planned motion times and colliding pairs of task assignments or robot paths that cannot be used simultaneously. The algorithm terminates when no new paths need to be planned, or when, e.g., a computation time limit is reached. In the first case, the solution is optimal if all steps are solved optimally (which is typically not the case; see the respective sections) and the shortest robot paths are also the optimal ones when also considering robot–robot collisions; see Section 2.7.

2.2 Line balancing

The relation between line balancing and load balancing can (generally) be seen as follows. The line balancing problem is to assign tasks to stations, while the load balancing problem is to determine in detail how to perform the assigned tasks in the station. However, there exist some attempts to perform load balancing with multiple stations (see, e.g., [106]) and—vice versa—some line balancing models that incorporate some details about how to perform the tasks within the stations (including robots, sequences, motions, collisions, etc.).

As an example, Cantos Lopes et al. [23] describe an assembly line by a mixed integer linear programming (MILP) model (see Section 3.1). The authors assume that the robot motion times between certain groups of tasks are constant. They then ensure that collisions are avoided by enforcing the robots to work on disjoint task groups, thus making this approach applicable only to a subset of instances. Performing the division into groups can be hard, since checking if two robots can never collide when working in different groups is non-trivial for a general problem instance. Moreover, it is also a strong assumption that the robot motion time between two groups is constant, since a collision-free motion might not even exist. As a consequence, this MILP model cannot be used in a truly online scenario, since the solution might require additional touch-ups from an engineer in-order to produce feasible programs.

The term line balancing is broad and often refers to a larger scale than the load balancing problem does; see Battaïa and Dolgui [12]. The line balancing problem typically comprises a complete machining, assembly, or disassembly line. In some simple versions the robots, or machines, are assumed to be identi-

cal and the order of the tasks negligible. The line balancing problem typically includes sequence constraints on the jobs, so-called precedence constraints. A typical case for line balancing is to study the robustness of a schedule, and to minimize the throughput loss due to unexpected failures; see Müller et al. [82]. Another example of line balancing is Michels et al. [78], which optimizes the design of the assembly line by minimizing its purchase cost.

2.3 Disjoint load balancing

In the appended Paper I (cf. [3]) we present an alternative to lazy load balancing by introducing an additional constraint enforcing the robots to work in disjoint workspaces. Thus, no robot–robot collisions can occur, and no velocity-tuning or time coordination will be needed to prevent such collisions.

In short, the task assignment is done without considering the sequence ordering but instead ensuring that all pairs of tasks assigned to different robots are collision-free; see Section 4.2.2 for details. Using the resulting task assignment, a spatial partition (i.e., disjoint workspaces) is constructed as the medial surface called *Voronoi diagram*; see Section 4.4.3 for details. The lazy load balancing loop is applied with the medial surface being a part of the environment and without the robot–robot collisions and coordination steps. The procedure is then repeated by searching for a new task assignment, until a termination criterion is met; see Paper I for details.

The method is proven to be quite effective on some industrial instances, which is due to a number of reasons; this despite that an additional constraint has been introduced, and thus we can't expect the cycle time to decrease. However, when partitioned into disjoint workspaces the task assignment and sequencing problem becomes easier to solve, and hence the heuristic algorithms are likely to perform better. Moreover, and more importantly, there is no increase in cycle time due to the velocity-tuning or to the *wait signals* introduced in the coordination (see Section 4.4.1 for details). As a result, we even observed that the disjoint load balancing decreased the cycle time for some problem instances. A last, but not least, important note is that there exist industrial instances for which the disjoint load balancing problem is even infeasible.

Note that Voronoi diagrams are often used in mobile multi-robot systems. For example, Kim and Son [62] utilize Voronoi diagrams to balance the load of and to prevent collisions among robots, applying pesticide to orchards. Moreover, Voronoi diagrams are frequently used in the path planning problem to navigate in a cluttered environment, cf. [18].

Recently Zhou et al. [122] solved the load balancing problem using a simpler spatial partitioning by projecting the workpiece onto a plane that is partitioned into a rectangular region for each robot. The authors allow robots to process

tasks on the boundary of these regions. As a consequence, a coordination step similar to that in the lazy load balancing approach is needed to prevent robot–robot collisions. A similar simple spatial partition is used also by Hagebring et al. [52] to prevent robot–robot collisions.

2.4 Predefined makespan

Skutella and Welz [103] study a load balancing problem in which the makespan is predefined and the total path length is minimized. Moreover, they assume that tasks can only be performed by a single alternative, thus studying a somewhat simpler problem. They use column generation (see Section 3.5.2) to solve it. Their column generation master problem ensures that each job is performed once, the subproblem is to find the least penalized tour respecting the makespan, and on top of this they use branch–and–bound to resolve integrality as well as robot–robot collisions.

The approach has, however, some drawbacks and limitations. First, the motion time between tasks is assumed to be known, which is unreasonable since not all pairs of paths can be planned in reasonable time (cf. Section 4.3). Second, since every tour found within the course of the branch–and–bound algorithm needs to be checked if it is collision-free, thus very many paths need to be planned. Third, the approach does not comprise that a typical weld task can be completed using several robot configurations.

Landry et al. [66] address many of the issues regarding the requirement of known robot paths, using an approach similar to that in [106], by planning paths and checking collisions only for tours that are optimal in a current approximation of the motion times. Hömberg et al. [55] refine the procedure by warm starting the solution process of finding the optimal tour from previous computations, incorporated in a branch–and–price scheme.

Pellegrinelli et al. [88] study a load balancing problem with predefined makespan. In their version they design the robot station (decide robot type, its positions, the workpiece position, and the tools used) with the aim to minimize the costs of the robot, tool, and fixtures. They do an extensive preprocessing step, computing a so-called probabilistic roadmap (cf. Section 4.3) for each combination of robot type, robot position, workpiece position, and tool type. This roadmap is utilized to find collision-free paths between the tasks and to define a MILP model optimizing the task sequence. The reported results and stated limitations suggest, however, that this method applies only to small instances with a single robot and roughly 20 tasks.

2.5 Precedence constraints

Some applications require certain tasks to be done in a specific order. Then, so-called *precedence constraints* must be taken into account in the load balancing problem. E.g., precedence constraints are necessary for applications in which the weld sequence has a large impact on the product quality; cf. [119, 109]. There is, however, little work done to include these constraints in the load balancing problem. If the path planning and collision part of the load balancing problem is removed or assumed to be trivially solved, then the remaining problem becomes a generalization of the well-known travelling salesperson problem, cf. [100, 99], or of the job-shop scheduling problem, cf. [41, 96]; see Section 4.2.1 for details.

Wang et al. [118] solve a problem related to the load balancing problem. In their problem, a robot has a single alternative for each task, two robots are considered, and the product quality is optimized. A case is studied where the robots are to perform around fifty weld tasks, and the heat generated from the welds are considered; to ensure a high enough product quality, two adjacent weld tasks cannot be consequently performed due to excess heat. Particle swarm optimization (PSO) is applied to solve the problem, they consider a fitness function that involves both makespan and product quality.

Chen et al. [26] recently suggested a fascinating solution method for the load balancing problem with precedence constraints and each task has a single robot and alternative, the precedence constraints ensure that parts are assembled in the correct order (e.g., bricks placed on top of each other). Their solution method is based on a huge precomputed roadmap (possible robot motions) for each robot, and on an extensive pre-computation of all potential collisions. They then solve a MILP optimization problem that is a slight relaxation of the original problem where robot–robot collisions are allowed during motions (but not when processing tasks). Any resulting robot–robot collisions are resolved by simultaneously planning both robots' motions, which generally is very computationally expensive (see Section 2.7). Note however that simultaneous planning of multiple robots is an active research topic, see e.g., [87].

2.6 Applications where robot motions are computationally cheap

When the robots' motions are expensive to compute, an efficient algorithm will need to reduce the number of motions that needs to be computed, as in Sections 2.1 and Section 2.4; see Section 2.7 for details on when robot motions are computationally expensive. However, when the robot motions are computationally cheap, e.g., a specific type of robot or environment, other algorithms apply. Here, we list a few such applications.

Laser cutting problems usually comprise a single robot. The motions of this robot are computationally cheap, but some tasks need to be performed in a

certain order, i.e., precedence constraints are present. Dewil et al. [34] tackled this problem using a local search heuristic.

Kovács [64] studied the remote laser welding problem. In this work, a single robot is considered, but the tasks can be performed from a continuous set of configurations. The motion planning of the robots is, however, performed (only) as a post-processing step. Erdős et al. [37] also studied the remote laser welding problem with a single robot, in their work they first determine a task sequence, then a path for the tool, and finally they compute a corresponding path for the robot.

The laser sharing source problem is a generalization of the load balancing problem where the robots share laser power sources. Hence, a robot can perform a weld task only when a laser power source is available. Rambau and Schwarz [91] study this problem and assume the robot paths to be known, and each weld task can be formed by a single alternative. The problem is solved using a tailored branch–and–bound algorithm.

Tereshchuk et al. [110] considered a multi-robot station in aircraft assembly. In this setting the tasks are on a relatively flat surface, the motion planning problem is assumed trivial, and the motion durations are approximated by increasing the task processing times. A heuristic algorithm is used to find a balanced solution without robot–robot collisions.

Sometimes the load balancing problem as described in Section 2.1 is simplified and then solved. E.g., Xin et al. [121] assume that the degrees of freedom in the configuration space of the robots are large enough to position the tool centre point at a desired location in a plane. Furthermore, they assume that if the tool centre points do not collide (using a non-zero tolerance), then robots can be configured to not collide. With these assumptions, they suggest a time-indexed network flow model with some side constraints to solve the load balancing problem, and they solve the model using a genetic algorithm (GA).

Touzani et al. [111] also suggest using GA, their algorithm comprises three sequential steps, first optimize the task sequences for the robots (using GA), second to optimize the robot configurations at the tasks, and lastly to plan collision-free paths. Touzani et al. [112] improve the method presented in [111] by incorporating the steps in a feedback loop that diversifies the search and attempts to find new sequences and configurations to minimize the cycle time, including waiting times needed to prevent robot–robot collisions. Still, similarly to [106] and [55], they plan the motions of all robots separately to reduce the complexity of the path planning problem.

2.7 Path planning—the curse of dimensionality

To conclude this section, we emphasize and clarify the main differences between the methods mentioned above and where they apply. The main difference is how hard the path planning step is considered to be. In [106, 55] the path planning step is considered as the bottleneck, while in [23], it is not. It all depends on what kind of robot(s) and environment are considered. First, if the robot is low-dimensional, e.g., having two or three degrees of freedom—typically representing its position in a Euclidean space—then the path planning problem becomes computationally quite simple. Second, if the environment is simple, i.e., if straight paths between pairs of configurations are likely to be collision-free, then the path planning problem can be solved for very high-dimensional robots; see, e.g., [76].

It is known that the path planning problem is PSPACE-complete, which implies that it cannot be solved in polynomial time unless P = NP; cf. [21, 93]. Moreover, in a fixed dimension the path planning problem is polynomially solvable ([22]). As a consequence, it appears that any general path planning algorithm possesses a running time that is exponential w.r.t. the dimension; see [72]. This "curse of dimensionality" does not only appear in obvious cases such as sampling approaches (n samples in each of the d dimensions yields n^d combinations), but also for seemingly simple problems, such as locating the nearest neighbour in a collection of points (see [57]), which is used in several path planning algorithms; see Section 4.3.1. Thus, in practice, path planning problems are typically solved only approximately. From Section 4.3, it will become clear that if the problem is high-dimensional and does not admit a crude approximation, it will generally require a long computation time.

2.8 Shortest path assumption

The complexity of the path planning problem motivates the assumption that the robots do not collide along their paths, i.e., each robot uses its shortest collision-free path regardless of the other robots; we denote this as the *shortest path assumption*. Without this assumption, the path planning algorithm would need to consider all the robots simultaneously, effectively multiplying the number of dimensions in the path planning problem by the number of robots in the load balancing problem. With the *shortest path assumption*, the shortest collision-free path between two robot configurations becomes independent of the other robots' paths, and thus the robot's paths are well-defined regardless of the task sequence of any robot involved. Note however, that if the surroundings are not cluttered with obstacles, then the path planning problem becomes simpler, allowing for multiple robots to be simultaneously planned (cf. [26]).

The method used to ensure that the robots do not collide in the final solution

varies between the existing methods, and is known as the *coordination* of the robots. The most well-known method is to retain the robot paths while tuning their velocities; cf. [105, 55, 88, 111]. Disjoint robot paths can also be achieved by planning the paths subject to disjoint workspaces, as in [3]; see Section 4.4. There are more general methods that also modify the robot paths; Liu et al. [75] first decide the robots' task sequences and then plan all robot motions by *prioritized motion planning*, i.e., they sequentially plan the robots' motions and consider robots of earlier iterations as moving obstacles; see Section 4.4.2.

3 Mathematical modelling and optimization

This section provides a mathematical background to modelling and optimization methods that are utilized to solve different parts of the load balancing problem. While the section is written to be self-contained, the methods are given a focus roughly proportional to their use in Section 4.

3.1 Mixed integer linear programming (MILP)

A mixed-integer linear program is an optimization problem with affine objective and constraint functions, where some variables are restricted to be integral. Any MILP can thus be written as

$$\begin{aligned} z^* := \text{minimum} \quad & c^\top x, && \text{(1a)} \\ \text{such that} \quad & Ax \geq b, && \text{(1b)} \\ & x \in \mathbb{Z}^{n_1} \times \mathbb{R}^{n_2}, && \text{(1c)} \end{aligned}$$

where $n = n_1 + n_2$ is the dimension of the variable space, and m is the number of inequality constraints (c and b are vectors and A is a matrix), also let $\mathcal{I} := \{1, \ldots, n_1\}$ and z^*_{LP} be the optimal value of the corresponding LP relaxation of (1), i.e., relax (1c) and let $x \in \mathbb{R}^n$.

The MILP formulation (1) may seem restrictive, but by exploiting the integrality requirement many combinatorial optimization problems can be modelled as MILPs. Hence, it is reasonable that MILP is NP-hard[1] (see, e.g., [29, Ch. 1.3]). Moreover, the question "Is the feasible set non-empty?" is in [30] shown to be NP-complete, whereas its negation "Is the feasible set empty?" is NP-hard and conjectured not to be in NP.

An example of a problem that can be expressed as a MILP is the travelling salesperson problem (TSP). It is defined on a directed graph $G = (\mathcal{V}, \mathcal{A})$, where the nodes $v \in \mathcal{V}$ and arcs $a \in \mathcal{A}$ represent cities and roads, respectively. Each arc $a \in \mathcal{A}$ is associated with an arc cost c_a representing the travelling time between the two nodes; if $c_{(ij)} = c_{(ji)}$, $(ij) \in \mathcal{A}$, then the problem is called a symmetric TSP (STSP) and otherwise it is an asymmetric TSP (ATSP). The solution to the ATSP (assuming that the arc costs satisfy the triangle inequality) equals the shortest tour that visits each city exactly once, i.e., the least weighted Hamiltonian tour. The ATSP can be modelled as a MILP in many ways; an early

[1] A decision problem is NP if the answer "yes" can be verified in polynomial time. A decision problem is NP-hard if any NP problem can be reduced to it in polynomial time. A decision problem is NP-complete if it is NP and NP-hard; see [29, Ch. 1.3].

and simple formulation is given by Miller, Tucker, and Zemlin [79], and is to

$$\begin{aligned}
\underset{x,u}{\text{minimize}} \quad & \sum_{a\in\mathcal{A}} c_a x_a, && && \text{(2a)}\\
\text{such that} \quad & \sum_{a\in\delta^+(i)} x_a = 1, && i\in\mathcal{V}, && \text{(2b)}\\
& \sum_{a\in\delta^-(i)} x_a = 1, && i\in\mathcal{V}, && \text{(2c)}\\
& u_i - u_j + (n-1)x_{(ij)} \le n-2, && (ij)\in\mathcal{A} \mid i,j\neq s, && \text{(2d)}\\
& u_i \in [1, n-1], && i\in\mathcal{V}\setminus\{s\}, && \text{(2e)}\\
& x\in\mathbb{B}^m. && && \text{(2f)}
\end{aligned}$$

Here, the variables $u_i \in \mathbb{R}$ denote the order in which the nodes are being visited, i.e., $u_i = 3$ implies that node $i \in \mathcal{V}$ is the third node visited after the arbitrarily chosen source $s \in \mathcal{V}$. Moreover, the constraints (2c) and (2b) ensure that each node has one entering and one leaving arc, respectively, where $\delta^-(i)$ and $\delta^+(i)$ denote the sets of arcs entering and leaving node i, respectively. The reason that the variables u_i and the constraints (2d) are needed is to prevent so-called *subtours*, i.e., to ensure that the solution admits only one connected tour, and not multiple disjoint tours.

3.2 The branch–and–bound algorithm

Branch–and–bound (B&B) is a general algorithm that always includes three components: (i) a partition of the feasible set, (ii) a relaxation of the minimization problem that can be solved to optimality, and (iii) an incumbent solution (i.e., the best known feasible solution). The algorithm partitions the problem into several problems with smaller feasible sets, this is known as *branching*. This gives rise to the so-called B&B tree, in which each node represents a feasible set, and its children a partition of this set. In each iteration, the algorithm selects a node i in the B&B tree, computes the *lower bound* ($\underline{z}_i$) by the relaxation applied to node i and use the *upper bound* $\overline{z}$ given by the incumbent solution; branching is applied to node i if $\underline{z}_i < \overline{z}$, creating new (child) nodes of the B&B tree. A new incumbent solution is retrieved when the relaxed solution turns out to be feasible. It is also common to apply heuristics to facilitate the search for incumbent solutions.

When applied to solve a MILP the most common specialization is to relax the integrality restriction (a so-called LP-relaxation) to receive the lower bound z^*_{LP} on z^* and to form the partitions by rounding a fractional value up or down. Formally, let $\underline{x}^i$ be an LP solution in node i; if $\underline{x}^i$ is feasible in (1) the incumbent

is updated and no further partitioning is needed, otherwise $\exists j \in \mathcal{I} : \underline{x}_j^i \notin \mathbb{Z}$ and two new nodes are created with the additional constraints $x_j \leq \lfloor \underline{x}_j^i \rfloor$ and $x_j \geq \lceil \underline{x}_j^i \rceil$, respectively.

There are many details that together determine whether an implementation of B&B is efficient or not, such as choosing the next node i in the B&B tree, or choosing the fractional variable to use for the next branching; See [29, Ch. 9.2] for such details. Regardless of such details, B&B is also very dependent on the MILP formulation, or more precisely the size of the *integrality* gap (the difference between the optimal objective value of the MILP and its LP lower bound). In fact, when the integrality gap is large, it is likely that very many partitions are needed before $\underline{z}_i \geq \overline{z}$ can be verified.

For example, consider replacing the constraints (2d) with the so-called subtour elimination constraints

$$\sum_{a \in \delta^+(\mathcal{S})} x_a \geq 1, \quad \emptyset \subset \mathcal{S} \subset \mathcal{V}, \tag{SEC}$$

which was suggested by Dantzig, Fulkerson, and Johnson [33]. The SEC constraints state that every proper subset of nodes needs at least one outgoing arc. Using the SEC constraints results in a better lower bound for the ATSP as compared to using the constraints (2d), which is a classical result; cf. [68]. For example, with $|\mathcal{V}| = 50$ nodes represented by uniformly distributed points in the unit square and the arc costs defined as the Euclidean distances between them; by using the SEC constraints instead of (2d), the integrality gap is reduced from roughly 15% to 1%. Thus, B&B would be much more effective by using (SEC), if not for the issue that these constraints are exponentially many. This topic will be addressed in Section 3.4.

3.3 Polyhedral theory, convex hulls, and tight formulations

A natural question that arises from the (SEC) formulation example is if there exists an alternative MILP formulation of a general MILP (1) with no integrality gap, i.e., a so-called *perfect* formulation. This is indeed true, and it follows from two observations. First, relax the feasible set of (1) to its convex hull, i.e., consider the set $X_{\texttt{conv}} := \text{conv}\{\, x \in \mathbb{Z}^{n_1} \times \mathbb{R}^{n_2} \mid Ax \geq b \,\}$. Moreover, the objective function is affine, hence, the optimal objective value remains unchanged. Second, by Meyer's theorem ([77]) $X_{\texttt{conv}}$ is a polyhedral set and can be expressed on the form of (1) (if $X_{\texttt{conv}}$ is bounded this follows from the representation theorem by Minkowski [80]).

Hence, any MILP can be solved as a regular LP if a perfect formulation is known, which does not contradict the fact that MILP is NP-hard and LP is solvable in polynomial time. For example, the ATSP formulation using (SEC) is

not perfect, but it still has an exponential number of constraints. This indicates that even though a perfect formulation exists for a general MILP, it might contain an exponential number of constraints. Hence, any MILP with a perfect formulation of polynomially many constraints can be solved in polynomial time. On the other hand, there exist polynomially solvable problems, e.g., the minimum spanning tree problem, that has an exponentially large perfect formulation (see [29, Ch. 4.10] for details on sizes of perfect formulations).

The quality of a constraint can also be classified w.r.t. the perfect formulation. An inequality $a^\top x \leq \delta$ satisfying all points in $X_{\texttt{conv}}$ is called a *valid* inequality (VI). Moreover, a VI defines a *face* $F := X_{\texttt{conv}} \cap \{x \in \mathbb{R}^n \mid a^\top x = \delta\}$ if F is non-empty. The dimension of F yields a quality measure of the VI, where the highest dimensional faces (one dimension less than the dimension of $X_{\texttt{conv}}$) are called *facets*. A perfect formulation consists of every facet-defining VI of $X_{\texttt{conv}}$. Similarly, for a polyhedron P, a VI $a^\top x \leq \delta$ *dominates* another VI $\tilde{a}^\top x \leq \delta'$ if $\{x \in P \mid a^\top x \leq \delta\} \subset \{x \in P \mid \tilde{a}^\top x \leq \delta'\}$. Note that a facet-defining VI cannot be dominated. The SEC constraints are examples of facet-defining inequalities, whereas (2d) are not.

Each MILP formulation of a certain problem will have different VIs, thus each problem formulation needs to be analysed in order to find VIs, facets, and in the best case, its perfect formulation. For instance, in Paper V a MILP formulation, containing so-called set packing constraints, is investigated. These constraints define a feasible set of the form $\{x \in \mathbb{B}^{|\mathcal{V}|} \mid x_i + x_j \leq 1, (ij) \in \mathcal{E}\}$, which can be represented as choosing non-adjacent vertices on the undirected graph $G = (\mathcal{V}, \mathcal{E})$ and an edge in $\mathcal{E}$ represents a pair of mutually exclusive variable assignments. The undirected graph $G = (\mathcal{V}, \mathcal{E})$ can be used to derive facet-defining VIs; for instance, each *clique*[2] $\mathcal{C} \subset \mathcal{V}$ in G induces the facet-defining VI: $\sum_{i \in \mathcal{C}} x_i \leq 1$, cf. [84, Cor. 3.5].

3.4 The branch–and–cut algorithm

When no perfect formulation is known, one can instead rely on generating *cuts*, which are VIs that cut off a current LP optimal solution. One example is Gomory's [47] mixed integer inequalities; see, e.g., [29, Ch. 5.3] for implementation details. These cuts have the property that as long as the LP solution is fractional, a cut can be generated, yielding a tighter representation of $X_{\texttt{conv}}$; continuing this process iteratively leads to a *cutting plane* algorithm for solving (1).

Moreover, given a (perfect) formulation that is too large to handle, the cutting plane algorithm can be applied by solving a so-called *separation problem* that aims to generate cuts on a given form. For example, finding a SEC constraint

[2]A clique is an inclusion-wise maximal subset $\mathcal{S} \subset \mathcal{V}$ such that every pair of nodes in S is connected by an edge $e \in \mathcal{E}$.

cutting off an LP solution $\underline{x}$, is equivalent to finding a non-empty set $\mathcal{S} \subset \mathcal{V}$ minimizing the sum $\sum_{a \in \delta^+(\mathcal{S})} \underline{x}_a$. This is the *minimum capacity cut* problem, which can be solved efficiently, see [108].

To this end, most state-of-the-art MILP solvers use an algorithm called branch–and–cut (B&C), which is a combination of B&B and the cutting plane algorithm. The difference from B&B is that in each iteration a choice is made whether the LP bound should be strengthened by branching or by cutting.

To illustrate that B&C relies on a tight model formulation, we again consider the ATSP formulation using (2d) versus using (SEC), and the same random instance as in Section 3.2. In both cases the Gurobi MILP solver [50] is used which can generate several general cuts, such as Gomory cuts. Also, the SEC constraints are initially relaxed and added whenever a violating integral solution is found. This is a so-called lazy-constraint; another option would be to add the SEC constraints violating the LP solution as previously described. The result is that the formulation (2) required 30960 nodes and 615378 simplex iterations, whereas using (SEC) only used 684 nodes and 5863 simplex iterations.

In load balancing, B&C is often used when a MILP is to be solved to optimality, which is the case of the task assignment (described in Section 4.2) and line balancing (see Section 4.2.3). However, for some harder MILP instances, either some inexact method or a decomposition method needs to be applied.

3.5 Decomposition and reformulations

To decompose an optimization problem is to split the problem into multiple smaller problems, where the most straightforward decomposition comes from the observation that two parts of a problem are mutually independent. For example, if an ATSP is the model of a weld sequence to be done by an industrial robot, then, since the robot motions do not depend on the sequence, the motions can either be computed before the sequence, or a lazy approach can be used, as described in Section 2.1. However, if instead multiple robots are considered and collisions are to be avoided, then a robot motion becomes dependent on the motions (and sequences) of other robots. To overcome such issues, it is very common to use some type of suboptimization (Section 3.5.3), where some parts of the problem are simply assumed to be independent, which can lead to suboptimal or even infeasible solutions that need to be handled.

When the problem cannot be naturally decomposed, or there is no reasonable assumption that allows it to be decomposed one could instead employ a mathematical reformulation that allows for an efficient decomposition. This section will briefly cover the three most common ones: Lagrangian relaxation, Dantzig–Wolfe decomposition, and Benders decomposition. The first twp rely on a relaxation that enables the decomposition and then an iterative process

towards feasibility; Benders decomposition instead relies on a restriction of the problem (fixing some variables) that enables the decomposition and then an iterative process towards optimality.

3.5.1 Lagrangian relaxation

A Lagrangian relaxation (LR) is, as the name indicates, a relaxation of an optimization problem; and although it is applicable for non-linear optimization problems, see [13, Ch. 6] for details, we restrict our presentation to MILP problems. Consider the MILP formulation (1) and let A and b be partitioned into $(A_1^\top, A_2^\top)^\top$ and $(b_1^\top, b_2^\top)^\top$, respectively. Letting $S := \{x \in \mathbb{Z}^{n_1} \times \mathbb{R}^{n_2} \mid A_2 x \geq b_2\}$, for any vector of so-called Lagrangian multipliers $u \in \mathbb{R}^{m_1}_+$ we have the following relaxation

$$z_{\text{LR}}(u) := \min_{x \in S} \left(c^\top x + u^\top (b_1 - A_1 x)\right) \quad \leq \quad z^* := \min_{x \in S} \quad c^\top x, \quad \text{s.t.} \quad A_1 x \geq b_1. \tag{3}$$

The inequality in (3) holds since any x satisfying $A_1 x \geq b_1$ is negatively penalized. Hence, $z_{\text{LR}}(u)$ is a *lower bound* on z^*, and the best possible Lagrangian bound is retrieved by solving the so-called Lagrangian dual problem $z_{\text{LD}} := \max_{u \geq 0} z_{\text{LR}}(u)$, where z_{LD} is called the Lagrangian dual bound. The function $z_{\text{LR}} : \mathbb{R}^{m_1} \mapsto \mathbb{R}$ is non-smooth and concave and can be maximized using subgradient optimization; see, e.g., [71], Paper III, and Paper V for details.

The Lagrangian dual bound is also always not lower than that of the linear relaxation, i.e., $z_{\text{LD}} \geq z_{\text{LP}}$, where equality holds if (A_2, b_2) constitutes a *perfect formulation* of S. This can be seen by posing the Lagrangian dual as an LP problem by using a perfect formulation of S. Using the LP dual one can then show that $z_{\text{LD}} = \min_{x \in \text{conv}\, S}\{c^\top x \mid A_1 x \geq b_1\}$ and observe that $\text{conv}\, S \cap \{x \in \mathbb{R}^n \mid A_1 x \geq b_1\} \subseteq \{x \in \mathbb{R}^n \mid Ax \geq b\}$ yields that $z_{\text{LD}} \geq z_{\text{LP}}$. Moreover, LP duality theory directly relates LR and LP, let u^*_{LP} be the optimal LP dual variables of (1) corresponding to the constraints $A_1 x \geq b_1$, it holds that $z_{\text{LD}} \geq z_{\text{LR}}(u^*_{\text{LP}}) \geq z_{\text{LP}}$; cf. [44]. It is therefore common to use u^*_{LP} as initial Lagrangian multipliers in the subgradient optimization method.

As an example, consider the ATSP formulation (2) employing the SEC constraints and Lagrangian relax the constraints (2b). The corresponding Lagrangian relaxed problem becomes a shortest spanning $r-$arborescence problem, which is the directed version of the *minimum spanning tree* problem. Without going into much detail, it can be shown that the resulting bound is stronger than the LP bound. For an analogous relaxation of the STSP—where the Lagrangian relaxed problem becomes a minimum spanning tree problem—the Lagrangian dual bound equals the LP bound.

3.5.2 Dantzig–Wolfe decomposition

The Dantzig–Wolfe decomposition is closely connected to the Lagrangian relaxation, and it attempts to solve the problem $z_{\text{LD}} = \min_{x \in \operatorname{conv} S}\{c^\top x \mid A_1 x \geq b_1\}$. To simplify the presentation, we assume that S is a bounded set so that $\operatorname{conv} S$ can be described by the convex combination of its extreme points x^q, $q \in \mathcal{Q}$. The so-called master problem is thus

$$z_{\text{LD}} = \underset{\lambda}{\text{minimum}} \sum_{q \in \mathcal{Q}} c^\top x^q \lambda_q, \tag{4a}$$

$$\text{such that} \sum_{q \in \mathcal{Q}} A_1 x^q \lambda_q \geq b_1, \tag{4b}$$

$$\sum_{q \in \mathcal{Q}} \lambda_q = 1, \tag{4c}$$

$$\lambda_q \geq 0, \quad q \in \mathcal{Q}. \tag{4d}$$

The master problem (4) is solved by a technique called *column generation* that uses the concept of *reduced costs* to generate a sufficiently large subset of $\mathcal{Q}$. In short, a subset $\bar{\mathcal{Q}} \subseteq \mathcal{Q}$ is sufficient to solve (4) if all reduced costs of λ_q, $q \in \mathcal{Q} \setminus \bar{\mathcal{Q}}$ are non-negative, otherwise the subset $\bar{\mathcal{Q}}$ is extended by some q violating this condition. Moreover, a subset $\bar{\mathcal{Q}}$ that constitutes a feasible solution to the version of (4) restricted to the set $\bar{\mathcal{Q}}$ would give rise to optimal dual variable values u and v, corresponding to (4b) and (4c), respectively. With these values the minimum reduced cost can be computed as $z_{\text{LR}}(u) - ub_1 - v$. Hence, there is a one-to-one relation between the Dantzig–Wolfe decomposition and the Lagrangian relaxation.

A typical case for applying Lagrangian relaxation or Dantzig–Wolfe decomposition is when the subproblem in (3), i.e., to compute $z_{\text{LR}}(u)$, decomposes into several subproblems. One example is the *generalized assignment problem* that assigns jobs to machines and each machine has a maximum capacity. By Lagrangian relaxing the constraint modelling that each job should be done, the corresponding Lagrangian problem will decompose into one *binary knapsack problem* per machine. A decomposition into several the subproblems is often computationally tractable, however, it is not required, e.g., the Dantzig–Wolfe reformulation for load balancing suggested by Skutella and Welz [103] does not decompose.

As a last remark, a crucial difference between the Dantzig–Wolfe decomposition and the Lagrangian relaxation is that the Dantzig–Wolfe decomposition allows for a direct generalization of B&B, called branch–and-price (B&P). When (4) has an optimal solution λ^*, but $\underline{x} := \sum_{q \in \bar{\mathcal{Q}}} x^q \lambda_q^* \notin S$ (thus $\exists j \in \mathcal{I} : \underline{x}_j \notin \mathbb{Z}$), then branching occurs by adding the constraints $x_j \leq \lfloor \underline{x}_j \rfloor$ and $x_j \geq \lceil \underline{x}_i \rceil$ to

the respective branches. Similar to B&B a more advanced partitioning can also be used, but regardless, the new constraints must either enter the master problem, by extending A_1, b_1, or the subproblem, by modifying the set S; the most common is to modify S if the subproblem remains computationally tractable.

3.5.3 Suboptimization

Suboptimization is a technique that heuristically fixes the values of some variables of the problem and then solves the remaining problem. This is often used when a problem becomes too computationally expensive. One example of this is the task planning step (see Section 4.1), the goal of which is to discretize a continuous set of alternatives that allows a robot to complete a task. The gain is that the remaining load balancing problem can be solved, but since some alternatives to complete a task are no longer available the resulting solution cannot be verified to be optimal.

A similar idea is used to find good feasible solutions during a B&C process, where one example is the efficient algorithm called *relaxation induced neighbourhood search* (RINS) [32]. RINS is an improvement heuristic that fixes all variables that have the same values in a feasible solution as in an LP solution, the remaining problem is then solved with B&C. Since some variables are fixed heuristically, an optimal solution cannot be guaranteed, but a better feasible solution might be found.

The risk of ending up at suboptimization becomes larger if the variables are poorly fixed (due to a too crude approximation). However, this can be resolved using some kind of feedback loop and iteratively adjusting the fixing of the variables until a satisfactory solution is received. A feedback loop is used in Paper I, in which a surrogate model is used to fix the values of some variables; in each iteration the surrogate model is solved with the additional constraints that all previously found optimal solutions are infeasible. This is equivalent to finding the k best solutions to this surrogate model, and even though the surrogate model provides a lower bound that could be used to determine a sufficiently small value k it is for most instances too large. As a remedy, some other termination criterion, such as a time limit, can be of greater practical use.

3.5.4 Benders decomposition

For MILP problems (1) where the continuous part is considered to be "easy" or even decompose whenever the integral variables are fixed, Benders decomposition [17] can be used. Initially, the master problem is a relaxation of (1) where some continuous variables are removed (or fixed) and the constraints involving these variables are relaxed. Given an optimal solution to the master problem, the remaining problem (subproblem) is linear, and its optimal dual solution

yields a constraint that cuts off the previous solution to the master problem. This constraint is a conic combination of constraints from the original problem where LP duality is used to bound the relaxed continuous variables.

Hence, Benders decomposition is an iterative approach that in each iteration solves a relaxation of the original problem that is also projected onto the space of the integral variables. Then it either provides a certificate that the resulting solution is optimal in the original problem or a cut that separates the resulting solution from the original (projected) feasible set. See [29, Ch. 8.3] for a more elaborate presentation of the Benders decomposition.

3.5.5 Logic-based Benders decomposition

Logic-based Benders is a generalization of the classical Benders decomposition that does not require the subproblem to be linear. Again, the master problem fixes the values of some variables of the problem, and a subproblem involving these fixed variables generates a cut whenever the fixed variables are non-optimal. Depending on the properties of the subproblem the cut will be derived differently. E.g., if the subproblem is an LP then the classical Benders decomposition is retrieved, see [56] for details.

As an example on how general the logic-based Benders framework is constructed we describe the lazy load balancing (see Section 2.1) as a special case of it. In the master problem the cost, time to move a robot between two configurations, is variable. Moreover, the master problem solves the task assignment and sequencing problem (see Section 4.2) using the cheapest available costs. The subproblems are then to find the shortest path for each robot to travel along each selected robot path, while the cuts generate new lower bounds for the costs of using these paths.

This illustrates the main drawback of logic-based Bender, and even though it is a very general procedure, in order to work efficiently good cuts need to be derived on a per-problem basis.

4 Detailed steps and subroutines used within load balancing

As described in Section 2, there are different approaches for solving the load balancing problem and also some variations of the problem itself; nonetheless, these approaches typically share some subproblems and/or subroutines. In this section these subroutines are detailed to an extent roughly proportional to their importance in this thesis.

4.1 Task planning

There are many types of tasks that can be performed by an industrial robot in the automotive industry: spot welding, stud welding, inspection, and sealing, to name a few. These all have different requirements on the position of the robot's tool. For instance, a stud weld needs to be orthogonal to the workpiece, but its rotation is free to vary. A sealing task, on the other hand, requires that the tool traverses continuously along a path. Thus, a task can be performed by possibly infinitely many alternatives.

However, all approaches to load balancing outlined in Section 2.1 assume that each task can be performed by a finite (and quite limited) set of alternatives. Many approaches even consider a single alternative for each task. The task planning step aims to fill this gap by assuming that there exists a small set of representative alternatives to perform each task, a short explanation can be found in [24]. Thus, the task planning step will introduce an error or approximation, since possible ways of performing a task are excluded; however, this error can be controlled, see the following example.

To exemplify how to find such samples we consider a stud weld task. Let $\alpha \in [0, 2\pi]$ be the rotational degree of freedom. Then for any given $\hat{\alpha} \in [0, 2\pi]$ there exist a representative interval $[\underline{\alpha}, \overline{\alpha}] \ni \hat{\alpha}$ in which the possible inverse configurations (see Section 1.1) corresponding to $\hat{\alpha}$ remain feasible w.r.t. joint and collision constraints. The task planning step finds these intervals and determines suitable samples in each interval. By construction there exists a collision-free path between any two configurations in each interval; this path can also be made sufficiently short. Hence, the error introduced by using these samples, instead of every possible $\alpha \in [0, 2\pi]$, can be made arbitrarily small.

Note that this analysis relies on the critical (and common) assumption that robot–robot collisions are absent. But there are counterexamples for which each task has a unique "correct" alternative, since all other alternatives give rise to robot–robot collisions (or long waiting times). The risk of missing such "correct" alternatives can be reduced by increasing the number of representative alternatives for each task, which also increases the computation complexity of the load

balancing problem. Hence, it is crucial to identify promising alternatives in the load balancing problem. This complication is partially tackled in Paper IV and is described in Section 4.5.

A question to raise here is whether this discretization is really required. The answer is that it depends on the complexity of the robot and the collision geometry, which determines the difficulty of the path planning problem; see, [43, 64, 115] for examples on continuous tasks.

4.2 Task assignment and sequencing

Here the task planning step is done, one must assign an order in which to complete these tasks, and—if there are multiple robots—make a task assignment for each robot. Moreover, depending on the approach for planning the robot paths and preventing robot–robot collisions, different types of optimization problems arise, which will be summarized in this section.

4.2.1 The travelling salesperson problem (TSP) with variants

The travelling salesperson problem is a very well-studied problem; see, e.g., [69, 11]. There are also a lot of variations and generalizations of the TSP. Here, we will survey those that often occur in the context of the load balancing problem.

In the simplest case, when there is a single robot and each task can be performed by a single alternative, the STSP or the ATSP is retrieved, depending on the type of task. However, single alternatives are rare and, typically, there are several robots involved as well as multiple alternatives for each task and robot. This generalization is poorly studied, and we next review some related problems.

If multiple robots are considered—and thus load balancing is needed—the problem becomes an mTSP, or as more commonly denoted a vehicle routing problem (VRP) without limited vehicle capacities, see [15, 16]. Moreover, since the robots are located at different positions, it is rather a so-called multi-depot VRP (MDVRP); see [81] for an overview. In addition, with the objective to minimize the cycle time (i.e., makespan) the problem is a so-called makespan MDVRP. Note that each task has a single alternative and the robots are assumed to be identical, e.g., automated guided vehicles (AGVs). Makespan MDVRPs are studied from different perspectives; see [10, 25] for a polyhedral approach, [42] for a heuristic MILP approach, [117] for a TSP based approach, and [113, 28, 120] for robotic applications.

The makespan objective is not commonly used for the VRP, but it is more common for the closely related flexible job-shop problem; cf. [14]. Hence, there

is also a wide range of literature related to the load balancing problem. For example, in [96] so-called *transition times* between activities are considered, these are used to model the travel time between robot poses. These times are identical among the robots, which is not the case for robotic applications where the robots are positioned at different locations in the room.

A serious attempt to solve the load balancing problem of a multi-robot station is due to Hömberg et al. [55], they use column generation to solve an MDVRP with vehicle capacities and a heterogeneous fleet, which they refer to as the welding cell problem. The capacities can be used to model a maximum allowed makespan, which is sufficient for many industrial robotic applications. This approach lacks, however, the option of multiple alternatives in which a robot can perform a task. Another approach to solve the makespan VRP with a heterogeneous fleet is the B&B procedure described in [90], where each B&B node corresponds to a partial task assignment and for each robot (i.e., vehicle) a TSP is solved. In [94] a promising logic-based Benders decomposition approach is suggested for the makespan MDVRP with a heterogeneous fleet, as in [55] a single alternative in which a robot can perform a task is assumed.

There are works that address the issue of multiple alternatives when a single robot is considered. The resulting sequencing problem is often referred to as a generalized TSP (GTSP), for which there are exact approaches, such as [40] (using B&C), [99] (using B&B), and heuristic approaches, e.g., [60]. See [7] for an overview of the single robot case.

There are, to the best of our knowledge, only some literature covering the problem with multiple heterogeneous robots, multiple alternatives, and asymmetric travelling times (mGTSP) with a makespan objective, here denoted makespan mGTSP. This problem is considered in, e.g., [106]. The solutions to real sized problems are, however, often disclosed within commercial optimization packages, such as *Industrial Path Solutions* [58] and *IBM ILOG CP Optimizer* [65]. These packages also include constraints forbidding paths and task assignments that would imply a robot–robot collision, which is also considered in [106] and [55]. Recently, [111] suggests solving the makespan mGTSP with a genetic algorithm due to the computational complexity.

There are other interesting variants of the problem. One example is a TSP with moving tasks, as described in [49]. Another is when the robot and alternative is given for each task, but there are precedence constraints among tasks—this may arise during assembly of a product—in [26] the authors use MILP models and a sophisticated path planner to solve the problem.

4.2.2 Makespan minimization of unrelated parallel machine problem with set packing constraints

In a simpler model the task sequence is decided in a later stage and the problem is assigning tasks to robots. In general, the unrelated parallel machine problem is to assign n tasks to m machines (robots), the processing time depending on both the machine and task (i.e., unrelated times), and minimizing the makespan (aka cycle time). By the three-field notation from scheduling theory [48] this problem is denoted as $R||C_{\max}$, where R denotes the unrelated processing times and $C_{\max}$ that the objective is minimizing the makespan. To model robot–robot collisions a twofold generalization of $R||C_{\max}$ is made by introducing: (i) sets of alternatives for the machine–task assignments and (ii) set packing constraints prohibiting certain pairs of alternatives of different tasks and machines. By the three-field notation this generalization is denoted as $R|SP|C_{\max}$, where SP denotes that there are set packing constraints among the tasks.

In Paper I, we present a decomposition method that uses $R|SP|C_{\max}$ as a surrogate model for assigning tasks to robots such that there exists a fixed spatial partition of the multi-robot station. If the problem consists of a single workstation, the remaining sequencing problem is to solve a GTSP for each robot. However, if the decomposition method is applied to multiple stations, then tasks can be swapped between robots at different workstations and the sequencing problem is thus still a makespan mGTSP (with fewer alternatives for the tasks). Moreover, the surrogate model $R|SP|C_{\max}$ can be computationally challenging. As a remedy, a MILP model is formulated and a tailored Lagrangian-based B&B algorithm is developed in Paper V.

Note that, even for two machines, the $R||C_{\max}$ (without set packing constraints) is NP-hard and for an arbitrary number of machines there exists no polynomial $\frac{3}{2}$-approximation algorithm[3] (unless P=NP), see [74]. Note that, for a fixed number of machines in $R||C_{\max}$, there exist fully polynomial-time $(1+\varepsilon)$-approximation algorithms, e.g., with a running time of $n(m/\varepsilon)^{\mathcal{O}(m)}$ according to [59]. In Paper II a Lagrangian-relaxation based method is suggested for the $R||C_{\max}$, the relaxation is very strong resulting in an effective method. In fact, for the special case of two machines there is no duality gap (the Lagrange dual bound coincides with the primal optimal objective value).

If the machines are assumed to be identical in $R||C_{\max}$, then the problem is denoted $P||C_{\max}$, and for $P||C_{\max}$ there is a $(1+\varepsilon)$-approximation algorithm with running time of $\mathcal{O}((n/\varepsilon)^{\varepsilon^{-2}})$, cf. [54]. As a side note, the $P||C_{\max}$ is a special case of the so-called flexible job-shop scheduling problem, where each task consists of several *activities* that should be executed in a specified order; the

[3] An τ-approximation algorithm is constructed to find a solution with an objective value less than τ times the optimal objective value.

$P||C_{\max}$ is retrieved when each task consists of a single activity. The $R||C_{\max}$ is, however, not a special case of the flexible job-shop scheduling problem, unless the flexible job-shop problem is slightly generalized to let the processing times be unrelated, cf. [8].

4.2.3 The assembly line balancing problem

As noted in Section 2.2, line balancing concerns the case when an assembly line contains multiple workstations. It is, however, often simplified and assumes, e.g., some fixed processing time for a task. Thus, this resembles the $R||C_{\max}$ approach in Section 4.2.2, and sequencing is instead done at a later stage. When considering an entire assembly line it is more common that some tasks are subjected to precedence constraints (see, e.g., [51]), which falls outside the scope of this thesis.

4.3 Path planning

When a task-sequence for each robot is given, the corresponding collision-free paths need to be computed. In this thesis we concentrate on the case where the path planning problem—called motion planning problem if time-dependent constraints are present—has a high computational complexity, i.e., with a high-dimensional robot in a cluttered environment. Moreover, we here assume that the robots do not collide with each other; see Section 4.4 on how robot–robot collisions are prevented. Note also, if the path planning problem is relatively low-dimensional, then there are attempts to solve the sequencing problem as a multi-goal path planning problem; see [38, 115]. This is sometimes called the multi-robot patrolling problem; see [89] for details.

Our path planning problem can be understood as follows (see [72] for details): navigate a robot from a start to an end configuration, in a cluttered 3D *world space*. The robot is controlled by the continuous (possibly dependent) parameters in the *configuration space* $\mathcal{C}$. The configuration space is partitioned into $\mathcal{C}_{\texttt{free}}$ and $\mathcal{C}_{\texttt{obs}}$, corresponding to the collision-free configurations and those that collide with an obstacle, respectively. Each path in $\mathcal{C}_{\texttt{free}}$ is associated with a length that is given by constraints on the robot, such as maximum joint and tool velocities. The path planning problem is to find a shortest continuous path in $\mathcal{C}_{\texttt{free}}$ from the start ($q_{\texttt{init}}$) to the goal configuration ($q_{\texttt{goal}}$).

Note that the configuration space is dependent on the robot type, but also that the same robot can have multiple representations of its configuration space. For instance, a quad-copter is typically assumed to be able to follow any path in $\mathbb{R}^3$, hence one representation is the world space. At the same time, the configuration space could be represented by how the quad-copter is controlled,

e.g., the thrust of each engine. However, for an industrial robot this is not the case, i.e., a certain position and rotation of the tool may correspond to several (as well as no) robot configurations. Thus, the configuration space of an industrial robot is most commonly represented by the angles of its revolute joints and values defining the positions of other types of joints (such as prismatic). In the most common case the robot has six revolute joints.

4.3.1 Algorithms for path planning

There are two well-known types of path planning algorithms, so-called *combinatorial* and so-called *sampling-based*. Combinatorial path planning algorithms are complete and report a correct solution within finite time, but suffer from long computing times and are thus unable to solve "industrial-grade" path planning problems, see [72, p. 80]. Recall that path planning is known to be PSPACE-complete (cf. [21]). Thus, we focus on sampling-based path planning algorithms that are generally efficient but may fail to find the shortest path within finite time.

Rapidly-exploring random tree (RRT) is one of the most popular sampling-based method for solving the path planning problem, or rather the path planning feasibility problem in which any collision-free path is desired. RRT is a *probabilistically complete*[4] algorithm and has an exponential decay of failure probability in terms of number of samples, see [73]. In short, the RRT algorithm is initialized with the tree $G = (\mathcal{V}, \mathcal{E})$ with vertices $\mathcal{V} = \{q_{\texttt{init}}\}$ and edges $\mathcal{E} = \emptyset$; then iteratively does the following: randomize a state $q_{\texttt{rand}} \in \mathcal{C}$, find the closest state $q_{\texttt{near}} \in \mathcal{V}$ to $q_{\texttt{rand}}$; from $q_{\texttt{near}}$, take a small step towards $q_{\texttt{rand}}$ to generate the new sample $q_{\texttt{new}} \in \mathcal{C}_{\texttt{free}}$; extend G according to $\mathcal{V} := \mathcal{V} \cup \{q_{\texttt{new}}\}$, $\mathcal{E} := \mathcal{E} \cup \{(q_{\texttt{near}}, q_{\texttt{new}})\}$. The algorithm terminates when $q_{\texttt{goal}}$ is included in G. There are many variations of the RRT that specify the details of the above description, e.g., how to generate new samples, how to find the closest state $q_{\texttt{near}}$, and how to take a step towards a given point. Moreover, since both the start and the goal configurations are known, a common generalization is to simultaneously grow two trees, one from each configuration, see [116].

A drawback of the RRT-based approach is that it is designed for a single query. However, it is very common to conduct multiple queries, in which case a sampling-based roadmap can be used, which is also known as a *probabilistic roadmap* (PRM), see [61]. A PRM builds an initial graph $G = (\mathcal{V}, \mathcal{E})$ (i.e., a roadmap) of $\mathcal{C}_{\texttt{free}} \supset \mathcal{V}$ and then use this roadmap to find the final path. Here, G is constructed by generating many random (uniformly distributed) configurations in $\mathcal{C}_{\texttt{free}}$ and a so-called *local planner* to connect samples in small neighbourhoods. There is a trade-off between how computationally expensive

[4] An algorithm is probabilistically complete if it converges within an infinite amount of time.

the local planner should be and how many samples can be generated and connected. Most commonly the local planner consists of checking if the straight line in configuration space is collision-free, thus enabling the set $\mathcal{V}$ to be large.

In order for a path planner to be competitive, the algorithm must have a strategy for so-called *corridors* or *narrow passages*, which typically arises when a robot needs to navigate through the door opening of a car workpiece. One example is the *iterative diffusion* algorithm presented in [39] and that generalizes the RRT. It works as follows: beginning with a low (or even negative) threshold for collision in order to construct an initial RRT, then by iteratively increasing the threshold for collision and by excluding violated edges the RRT splits into multiple RRT's. To reconnect the RRT's the coming iterations generate new samples near excluded edges, since these edges are likely to correspond to narrow passages. Note that this requires a sophisticated distance query that is able to compute negative distances (penetrations depths), see Section 4.3.2.

Another issue is that while building the RRT as well as the roadmap, a huge number of paths must be checked for collision by the local planner. Bohlin and Kavraki [20] tackle this issue by using a so-called lazy approach, including a strategy for narrow passages. In their approach, the PRM is initialized as if $\mathcal{C}_{\texttt{obs}} = \emptyset$ and then only nodes and edges participating in the shortest path are checked for collision. When a collision is identified, the corresponding node or edge is deleted and a new shortest path is found. If the PRM becomes too sparse (or disconnected) then new random configurations are generated close to some nodes $v \in \mathcal{V}$, with the goal to reconnect the components of the graph.

As a final remark, we mention a path planning algorithm based on optimal control, see [67, 55]. The optimal control problem is stated in terms of control parameters, i.e., differences in the configuration parameters; the problem is thus infinite dimensional, which is handled by a time discretization. The constraints of the optimal control models are of three types. First, boundary constraints ensuring that $q_{\texttt{init}}$ and $q_{\texttt{goal}}$ correspond to the first and last configuration, respectively. Second, physical constraints modelling the robot's movements according to the control inputs of each time step; some authors include dynamic properties here. Third, collision preventive constraints, which ensure that each state is free of collision; see Section 4.3.2. The resulting problem is typically solved using a *sequential quadratic programming* SQP solver [46], despite the presence of the non-smooth collision constraints functions. The objective function in SQP is a local approximation of the Lagrange function, its Hessian matrix is not well-defined due to the non-smooth constraint functions. In [55] this issue is addressed by using the *BFGS* update ([13, Ch- 8]) formulas for approximating the Hessian matrix and finite differences to approximate the subgradient of the constraint function.

4.3.2 Continuous collision detection

Every path planning algorithm relies on *continuous collision detection* (CCD) in order to detect collisions during a robot motion. Three well-known classes of algorithms that check whether a path is contained in $\mathcal{C}_{\texttt{free}}$ or not, are *conservative advancement* (CA), *sweep volume approximation*, and *bounding volume hierarchy* (BVH). The most common as well as the oldest algorithm is CA (see, e.g., [31]), in which the (smallest) distance from the robot to the environment is checked in discrete sample configurations and then an assumption on the robot's maximum velocity is used to determine if the samples are dense enough. Sweep volume approximation (see, e.g., Herman [53]) uses primitive shapes (such as cylinders) to approximate the robot; then by assuming a specific type of robot (e.g., only revolute joints) the sweep volumes of these shapes are computed to a desired accuracy. BVH is a hierarchy of outer approximations of the robot's geometry, and the BVH can be used to model the sweep volume; in order to receive a better approximation, either the path is divided into subpaths or the robot is divided into subparts; see, e.g., [92].

Despite its age, CA is still competitive and is frequently used. One reason for this is that it only depends on an analysis of the robot's velocity, and its distance to the environment at specific configurations. The distance computation depends on how the geometry of the robot and the environment is represented. First, if the geometrical representation is done by triangulated bodies, then a popular distance computation uses a *proximity query package* (PQP; see [70]) where sets of triangles are contained in a bounding volume created by a rectangle sweep sphere (RSS); a BVH is then created by recursively splitting sets of triangles into smaller subsets. Second, if the geometrical representation is a union of convex polyhedra then there are efficient routines to compute the distance (e.g., [45]), and to compute the *signed distance* (see [63]). A signed distance equals the Euclidean distance if it is positive, and otherwise it equals some measure of the penetration depth. This is highly useful in motion planning algorithms, since when a collision is detected (i.e., a negative distance) then a set of colliding configurations can be derived; this idea is used in the optimal control approach described in [55].

The main issue of CA is to determine in which configurations the distance should be computed. The most common idea (cf. [101]) is a binary search strategy which measures the distance in the end points of the path. If the path is too long to be determined collision-free, its middle point is also measured, creating two shorter paths for which the procedure is repeated recursively. A key for efficiency is to use PQP by utilizing the fact that in most distance computations a lower bound is sufficient. This query is much cheaper to compute within the PQP framework, compared to an exact distance computation.

Paper III formalizes the idea of CA for the case when there are multiple

robots that are to be checked for collision. For the special case when only one robot is present, the test resembles that of a binary search. In Paper III each distance computation is done in the middle of a path; then the path is split into three parts: one of them is collision-free and two (possibly empty) may contain collisions.

4.3.3 Lazy load balancing—which paths to plan?

In Section 2.1, we introduced the concept of lazy load balancing, that is, to initially bound the length of each path from below. Then, iteratively, update these lengths by planning the paths used in the solution to the task assignment and sequencing problem (recall Section 4.2). This is done in, e.g., [106, 112] and it can be shown (recall Section 3.5.5) that an optimal solution to the load balancing problem is retrieved. However, the argument requires two major properties to hold: first that the path planner is able to find the shortest path and second that the final schedule has no collisions among robots. Unfortunately, these properties do not hold in practice: as discussed in Section 4.3.1 the path planning problem is often solved by a sampling approach. Moreover, in confined and shared workspaces robot–robot collisions need to be prevented by coordination.

4.4 Coordination

So far in this section the robots are assumed not to collide. Hence, their motions need to be made disjoint w.r.t. other robots' motions. This is often referred to as coordination.

This section presents several approaches to coordinate the robots in order to prevent collision. But first note that all of these approaches need to check if any two robots' motions collide or not. The collision check is a generalization of CCD (described in Section 4.3.2) and a common solution approach is to generalize CA (see Paper III and [55]). In Paper III, the collision check is done by iteratively measuring the robot–robot distance at configurations corresponding to a minimum of a function bounding the robot–robot distance from below.

Note also that coordination does not only seek for a collision-free solution but also for a robust solution, in the sense that the robots should not collide even if one (or several) of them breaks (unexpectedly) and does not follow its planned motion velocity.

4.4.1 Time and signal optimization

One of the most obvious remedies to prevent robot–robot collisions is to pause one of the robots until the shared workspace is free from the other robots. This is also reflected in the way industrial robots are typically programmed: to utilize

wait signals that tell other robots that a specific region is currently occupied. Spensieri et al. [105] propose the use of a MILP model of the problem of finding optimal wait signals, by analysing a so-called coordination diagram (see Figure 6). Moreover, a wait signal causes a robot to stop, which increases the robot wear and adds cycle time from the dynamical effects. Hence, there is also a need to minimize the number of signals (see [107]).

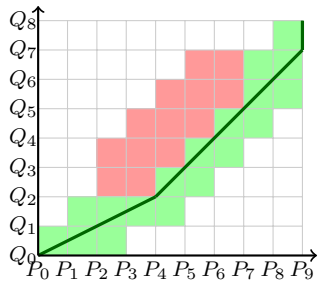

Figure 6: A coordination diagram for two robots (P and Q) with discrete positions that allow for wait instructions. Each axis corresponds to the time parametrization of a robot motion, a cell is coloured green if the corresponding subpaths are disjoint, red if a collision is found, and white if its state is not determined. The black line corresponds to the fastest collision-free (velocity tuned) motions.

Another approach to prevent robot–robot collisions is presented in [55], where all detected pairs of colliding motions give rise to new constraints in the task assignment and sequencing problem. Hence, it does not consider the benefit of using and optimizing wait signals, but the solution is feasible. The approach is useful since it highlights the important fact that the need for coordination can be partially prevented by modifying the order of the tasks. This fact is also utilized in [106].

Historically, it has been more common to consider each robot's workspace and to assign tasks to each robot according to its workspace, and that wait signals are introduced only when the robot is required to leave its workspace. In this spirit Segeborn et al. [102] attempt to cluster the tasks in order to reduce the need for coordination.

4.4.2 Prioritized motion planning

Another method for coordinating robots without considering a high-dimensional motion planning problem is to use a *prioritized motion planning* approach, see [36, 114, 27]. The idea is quite straightforward and is a suboptimization technique. The assumption is that a priority can be given to each robot, and that

planning and fixing the motions for a higher prioritized robot will not negatively impact the motions for lower prioritized robots. Thus, for a given task sequence (recall Section 4.2) the corresponding motions of the highest prioritized robot is computed. This process is repeated for the next prioritized robot, and so on. The main difficulty with this approach is that a motion planning algorithm (not a path planning algorithm) is required, since the planned robot motions constitute time-dependent obstacles, cf. [101].

If the priority assumption described above (almost) holds, a prioritized motion planning may result in high-quality solutions, since both waiting and detours are allowed in order to avoid collisions. This quality can be slightly improved by generalizing the prioritized motion planning algorithm as in [98]. When planning the motion of a robot, the previously planned robots' paths are fixed and their velocities can be tuned. This resulting in a (harder) path planning problem in the product space of the current robots' degrees of freedoms and the progress along each previously planned path.

4.4.3 Voronoi diagrams

A Voronoi diagram is a partition of a space created by so-called *generators*. The Voronoi diagram is always equally distant from the two closest generators. A Voronoi cell is the part of the space that is closest to a specific generator. In its most minimalistic form, the generators are assumed to be points and the distance measure is Euclidean. In our application we will assume that robots or rather unions of triangles are the generators. Hence, a generalized Voronoi diagram (GVD) is used; see [35, 86] for details about Voronoi diagrams, and Figure 7 for an illustration.

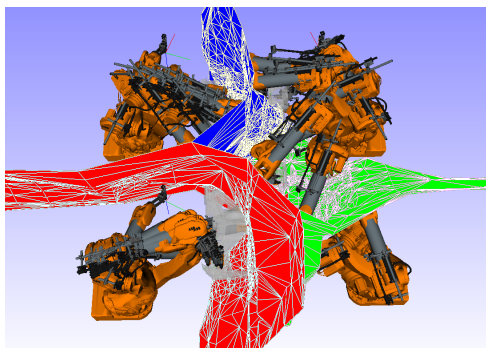

Figure 7: Illustration of a GVD approximation, where the generators constitute the union of the robot positions from the disjoint task assignment computed using the algorithm described in Section 4.2.2.

In Paper I we suggest a disjoint load balanced solution, which is a coordination approach that does not rely on wait instructions. Instead, a partition of the space is made based on a generalized Voronoi diagram. First, a task assignment without any pairs of colliding assignments is found (recall Section 4.2.2). Then, using the geometry of the robots positioned at every assigned task as generators of the Voronoi diagram. Thus, each robot is assigned a private workspace, and every path that is planned within this workspace will trivially be collision-free w.r.t. any path of any other workspace. Hence, the need for wait signals is removed. Even though our objective is to minimize the cycle time, the energy consumption becomes computationally easier to minimize since the velocities of the robots can be modified without risk of causing robot–robot collisions [19, 95].

The obvious drawback with this approach is that an additional constraint has been introduced. This may cause unbalanced solutions (i.e., some robots are mostly idle) as well as infeasible solutions. Due to this drawback, the disjoint solution approach cannot always be used, hence it can be seen as an alternative approach to other coordination approaches.

4.5 Combined load balancing and coordination

As noted, the issue of preventing robot–robot collisions cannot (directly) be reduced to a high-dimensional path planning problem, since robot–robot collisions can occur whenever at least one of the robots is processing a task. Hence, a first step is to prevent such collisions; this needs to be considered when assigning tasks to robots, when choosing alternative robot configurations, and when deciding the task sequences. This is done by, e.g., Hömberg et al. [55]. However, to ensure that robot–robot collisions do not occur they finally tune the velocities of the robots' motions.

In Paper IV we suggest a generalization of the disjoint solution approach which unifies the load balancing, path planning, and coordination problems. The idea is to consider a sequence of disjoint (partial) solutions, which limits the drawbacks of the completely disjoint solution approach. Moreover, unlike the prioritized planning approach, we do not introduce any time-dependent constraints in the motion planning problem, but by construction all robot motions are planned w.r.t. the partitioning surface of the corresponding (partial) solution.

The approach suggested in [26] targets a slight modification of the load balancing problem in which each task is assigned to a specific robot and robot configuration, but there are also precedence constraints among tasks. These modifications make the load balancing problem much smaller w.r.t. the number of task alternatives and possible task sequences. As a result, a MILP model con-

taining many so-called big-M constraints (which are typically computationally challenging) is successfully applied. The MILP model prohibits robot–robot collisions during the processing of tasks (similarly as in Paper IV). However, to resolve the remaining collisions they use a sophisticated motion planner that simultaneously plan the motions of robots for which collisions are observed. We have attempted a similar MILP model for the load balancing problem, but only very small problem instances could be solved within reasonable computing time.

5 Summary of the appended papers

This section summarizes the appended papers and highlights their use for load balancing of industrial robots in a production line. The papers are numbered in a chronological order and their interrelations are visualized in Figure 8. Briefly summarized, Paper I introduces the concept of disjoint load balancing, Paper II and Paper V investigate $R||C_{\max}$ and $R|SP|C_{\max}$, respectively, to enable the solution of computationally harder disjoint load balancing instances. Paper III accelerates a continuous collision detection routine used in the coordination of robots and thus in Paper IV. In Paper IV the idea of Spatial–temporal load balancing is introduced, which generalizes the disjoint load balancing approach.

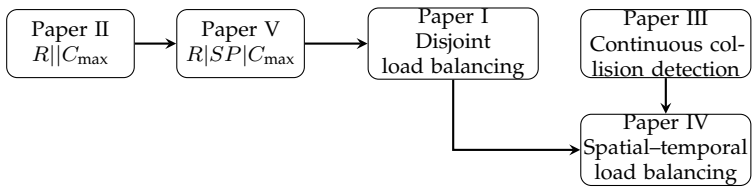

Figure 8: Illustration of the relation between the appended papers. Directions indicate that an idea/method in the origin paper is generalized or applied in the destination paper.

5.1 Paper I: Intersection-free Geometrical Partitioning of multi-robot Stations for Cycle Time Optimization

We suggest an approach to coordinate the robots' motions using the $R|SP|C_{\max}$ problem, as described in Section 4.2.2, and using the generalized Voronoi diagram (GVD), as described in Section 4.4.3. This approach eliminates the need to coordinate the robots' motions in time using a wait signal scheme. The motivation for this is to reduce the wear and cycle time losses that are associated with wait signals; see Section 4.4.1.

The proposed algorithm is visualized in Figure 9 and the steps are summarized as follows. By assuming that the motions times between tasks are negligible, the load balancing problem reduces to the $R||C_{\max}$. Then, by introducing set packing constraints that exclude all pairs of task assignments corresponding to colliding robots (regardless of time), the $R|SP|C_{\max}$ is retrieved. The resulting task assignment is then used to construct the Voronoi generators, each as the union of a robot volume at its assigned configurations. Note that the set packing constraints ensure that these generators are disjoint and thus that the GVD is well-defined. After introducing the GVD (see Figure

7) the software Industrial Path Solutions (IPS) is used to compute a sequence and the corresponding motions (possibly using other task alternatives) for each robot. Note that load balancing of tasks is allowed here whenever multiple workstations are present. The feedback loop modifies $R|SP|C_{\max}$ in one of two ways. First, if the GVD prohibits every path to a selected task, then new set packing constraints are introduced, which correspond to task alternatives that make the robot intersect with the sweep of a path from the home configuration to the inaccessible task. Second, if a feasible solution is found, this particular task assignment is excluded from the feasible set of the $R|SP|C_{\max}$. Two termination criteria are suggested. First, since the $R|SP|C_{\max}$ provides a lower bound on the load balancing problem, the algorithm stops whenever this lower bound exceeds the current best solution. Second, since this lower bound is very weak a time limit is also used.

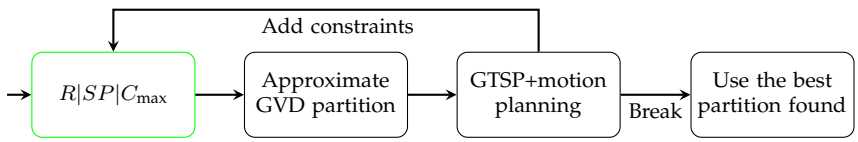

Figure 9: Illustration of the disjoint load balancing algorithm.

The results are somewhat surprising: Despite the additional constraint (spatial partition), for many industrial instances a solution is computed with little or no increase in the robots' cycle times compared to solutions of the conventional approach of coordinating the robots' motions in time. Moreover, when also the dynamic effects are estimated there seems to be a reduction in cycle time for some instances. The strongest result is retrieved when the algorithm is applied to a production line containing multiple workstations, since even when a partition of the workspace is enforced, most tasks are accessible by multiple robots in different workstations. It is likely that each partitioning suggested by the algorithm permits a solution in which the load is fairly well-balanced. We also believe that, since the remaining optimization problem (when the workspace has been partitioned into sub-spaces) is much smaller than the original makespan mGTSP instance (many variables are then fixed to zero), the task assignment and sequencing algorithm within IPS performs better. As a result, we were able to find solutions with lower cycle times by utilizing these partitions than without them. However, we also constructed counterexamples (that can occur in industrial instances) that didn't permit even a feasible solution. This means that the disjoint load balancing approach is a useful option only for some load balancing instances.

The paper is published in IEEE Transaction on Automation Science and Engineering (see [3]) and the algorithm is incorporated in the software IPS [58].

5.2 Paper II: Exact makespan minimization of unrelated parallel machines

In this paper we investigate the makespan minimization of unrelated parallel machines ($R||C_{\max}$). The $R||C_{\max}$ is studied as opposed to the complete $R|SP|C_{\max}$ from Paper I, since we found the problem to be complex even without set packing constraints. Many ideas from Paper II are generalized in Paper V, which targets the $R|SP|C_{\max}$ for the disjoint load balancing algorithm of Paper I.

In Paper II we apply a Lagrangian relaxation to solve the $R||C_{\max}$ and combine it with a B&B algorithm, a local search heuristic, and a *deflected subgradient* optimization routine. The *deflected subgradient* routine reduces the well-known zigzagging phenomenon of subgradient optimization routines, by determining the next step direction as a combination of the current subgradient and the previous step direction. A main contribution of the paper and key ingredient in the B&B algorithm, is that our Lagrangian subproblem allows for an efficient computation of strong so-called *residual costs*, which are used to phantom branch–and–bound nodes. As the second main contribution we show that for the case of two machines in the $R||C_{\max}$ the Lagrangian relaxation results in a zero duality gap, i.e., the optimal primal value of $R||C_{\max}$ and the optimal value of the Lagrange dual problem coincide. For this case, the bounding step in the B&B algorithm is very effective.

To evaluate our suggested method we compared it with the use of a state-of-the-art MILP solver when applied to several MILP formulations and algorithm parameter settings. Among these formulations we found that a formulation with auxiliary variables is highly effective and that it can be used in combination with a state-of-the-art cutting plane method for the $R||C_{\max}$. However, all of these MILP formulations were outperformed by our strong Lagrangian-based B&B algorithm. The algorithm solved almost every problem instance within the time limit, whereas the MILP-based algorithms could not prove optimality for the hardest instances.

The paper is published in Open Journal of Mathematical Optimization (see [5]) and the algorithm implementation as well as additional computational results are publicly available on Github (`https://github.com/Fraunhofer-Chalmers-Centre/rcmax/`).

5.3 Paper III: Efficient collision analysis of pairs of robot paths

The main objective of Paper III is to efficiently construct a coordination diagram, as illustrated in Figure 6. In this work, the disjointness of pairs of robot paths are to be checked, i.e., whether the robots' sweep volumes do *not* intersect. By generalizing the ideas of conservative advancement (see Section 4.3.2), in which

a maximum velocity, or rather a so-called *unit-velocity* parametrization of each robot path is assumed, the robot's paths can be expressed in terms of *maximum displacement*. Hence, each distance measurement will yield a lower bound on the clearance (i.e., the robot–robot distance); the lower bounding function is illustrated in Figure 10.

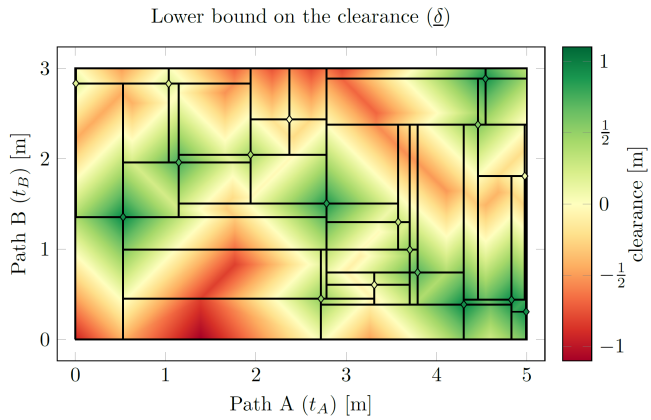

Figure 10: The domain representing the two robots' motions is decomposed into rectangular regions such that the piecewise linear lower bounding function is convex within each region. The black lines denote the region boundaries and the rhombus shaped markers represent collision-free points at which the distance between the two robots is computed.

The hypothesis of the paper is that computing the clearance at the current minimum of the lower bounding function is an efficient strategy for: (i) finding a (eventual) collision early in the process, (ii) gaining new clearance information since the current minimum is often far from previous samples. One enabler for this idea is that the lower bound can be minimized effectively (in polylogarithmic time), by dividing the domain into subdomains in which the lower bounding function is convex; see Figure 10. The minimum in each such domain can be computed analytically (in constant time). An additional feature of this minimization algorithm is that it can be useful for other applications, since only a bivariate function admitting a unit L^1 Lipschitz constant is assumed.

A generalization is also considered in which the domain corresponds to paths for entire task sequences, for which each pair of subpaths needs to be determined disjoint. The idea is then not to lose information on the boundary of the domain. This generalization is shown to be highly effective, especially when each path consists of many short subpaths.

Our suggested methods are compared with the method of first building

approximations of the sweep volumes and then checking their intersections (see e.g., [6]). An interesting difference between these two approaches is that our methods scale linearly with the number of pairs of paths whereas the sweep volume method scales linearly with the number of paths. Hence, for a large enough number of paths, the sweep volume approach should still be more efficient. We found that for our industrial instances, the number of paths typically need to be very large before it is computational preferable to use sweep volumes.

The paper is published in IEEE Transactions on Robotics (see [4]) and the algorithm is incorporated in the software IPS (see [58]).

5.4 Paper IV: Spatial–temporal load balancing and coordination of multi-robot stations

In this paper we generalize the disjoint load balancing method developed in Paper I. We consider a sequence of disjoint (partial) solutions, time periods, and a dynamic partition that changes between time periods. As a consequence, we introduce two new mathematical models, both ensuring that the assigned tasks are disjoint within each time period. Moreover, these models also include aspects such as task sequences, waiting times, and makespan minimization. The models are somewhat pessimistic since they assume robots to synchronously move between time periods. These needless synchronizations are partially prevented by a succeeding optimization, in which the velocities of the robots are tuned in order to minimize the cycle time as well as to avoid robot–robot collisions, but unbound to the concept of the time periods. Moreover, the algorithm comprises two stages: the first generates several promising candidates for (dynamic) partitions and the second stage minimizes the cycle time for each such partition, e.g., by tuning the robots' velocities.

The method is called *spatial–temporal* (ST) since both domains are exploited in order to optimize the multi-robot station. In particular, robot–robot collisions are prevented by assigning the tasks in such a way that collision avoidance is necessary only during the robots' motions. This property is achieved by the dynamic partition. The method is called temporal, since also the robot velocities are optimized and since the partitions are dynamic. Figure 11 illustrates a solution computed by the ST-algorithm, in which both domains need to be exploited; in fact, this multi-robot station does not allow for a completely disjoint solution. More importantly, in this multi-robot station, the commonly used shortest path assumption (see Section 2.8) results in very long waiting times; hence, the robot motions need to be spatially modified in order to avoid robot–robot collisions. The ST-algorithm produces a high-quality solution for this problem instance, since (i) the workspace partition prevents robot–robot

collisions, (ii) the multiple time periods allow for minimizing the cycle time, and (iii) the robots' velocities can be tuned in the post-processing step in order to overcome the synchronous transaction between time-periods.

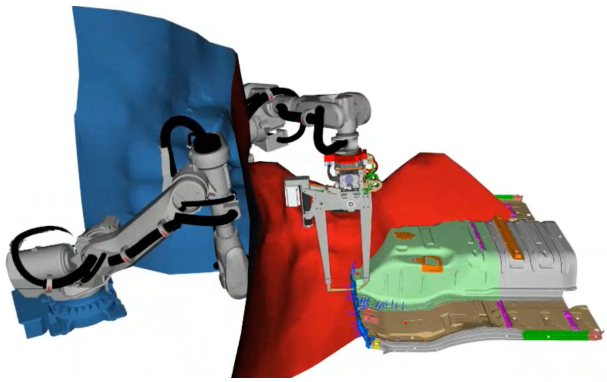

Figure 11: A snapshot of a solution generated by the ST-algorithm, in which the right-most robot is guided to reach above the other robot. The visualized partition is only one of many partitions used in the program; in a later time-period the right-most robot works closer to its base.

The paper is submitted for journal publication and the main ideas have been presented at the 34th Conference of the European Chapter on Combinatorial Optimization (ECCO). Madrid, Spain (2021).

5.5 Paper V: A Lagrangian-based method for makespan minimization of parallel machines with set packing constraints

In this paper we return to study the $R|SP|C_{\max}$ problem utilized in the disjoint load balancing solution approach, in order to enable an efficient solution of computationally more demanding problem instances and to enable the disjoint load balancing algorithm being used on complex multi-robot stations. Note that the $R|SP|C_{\max}$ generalizes the $R||C_{\max}$ in two ways. It introduces the set packing constraints which, in turn, motivate the consideration of multiple alternatives for each robot to perform each task.

The Lagrangian relaxation made in Paper II is generalized to include the set packing aspect and the multiple task alternatives. This affects how the subproblems are solved, how to compute the strong variable fixing, how the heuristics should provide feasible solutions, and how the Lagrange dual is maximized. We suggest multiple techniques: (i) using a so-called *clique cover* to model the set packing constraints, which are then penalized in the Lagrangian

dual function; (ii) using the LP relaxation to find an initial primal solution and initial values of the Lagrangian multipliers; (ii) accelerating the resulting maximization in the large dual space by using so-called bundle methods, which combine several subgradients (from recent iterations) to identify a better search direction; (iii) making use of logical reduction techniques to remove variables that do not contribute to an optimal solution; (iv) developing several heuristics to retrieve feasible solutions from the LP solution and from solutions to the Lagrangian relaxation.

Our Lagrangian-based branch–and–bound (LBB) method is evaluated on a large set of problem instances, both industrial (from the disjoint load balancing algorithm), synthetic structured instances resembling the industrial ones, and unstructured instances in order to evaluate how well the method generalize. Figure 12 shows how the method compares with using a state-of-the-art MILP solver. It also shows that for reasonable computation times (above 0.1 s) our suggested LBB method is able to solve a higher fraction of problem instances to optimality than the MILP solver. The figure also shows that the variable fixing (VF) accelerates the solution process.

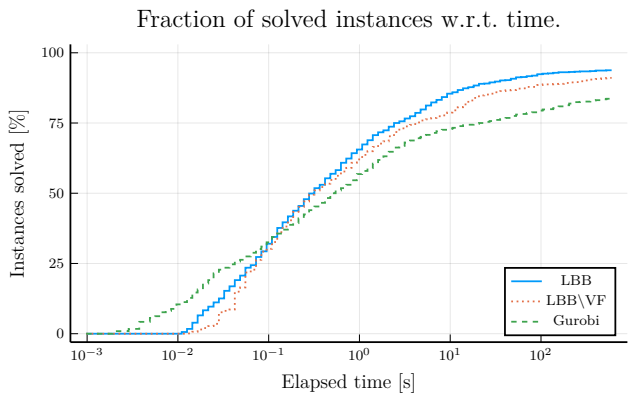

Figure 12: Fraction of instances solved by our LBB algorithm (LBB), without the variable fixing (LBB\VF) and the Gurobi optimizer.

This manuscript is neither yet presented nor published.

6 Conclusion and further work

To summarize, there are many variations of both the problem of balancing the workload of industrial robots and of the solution approaches. Moreover, it seems that the path planning problem is at the core of the load balancing problem and that different assumptions on the difficulty of the path planning problem leads to completely different solution approaches. Another important aspect is how to prevent robot–robot collisions, for which there are currently several remedies: (i) adjust the velocity of the robots' motions, (ii) use a prioritized planning approach, (iii) enforce the robots to be in disjoint workspaces, and (iv) construct a sequence of disjoint (partial) solutions.

The main contributions of this thesis to the solution of the problem of load balancing multi-robot stations are the following:

- the idea of generating a complex partition of the workspace in order to (partially) prevent the need for coordination (Papers I and IV);
- the idea of generating a sequence of such partitions to generate high-quality solutions for complex multi-robot stations (Paper IV);
- the idea of using a strong Lagrangian relaxation to solve $R||C_{\max}$ and $R|SP|C_{\max}$ that allows for the effective variable fixing (Papers II and V);
- a new sampling technique for continuous collision detection in two dimensions (Paper III).

These contributions target the three stated objectives of (i) reducing the cycle time, (ii) reducing the computational effort required for the load balancing problem (to allow for product individualization), and (iii) automating more complex multi-robot stations by generalizing existing methods to provide high-quality solutions. Paper I targets objectives (i), (ii), and partially (iii). Paper IV targets mainly objective (iii) but also objectives (i) and (ii). Papers II, III, and V targets mainly objective (ii) by enhancing the results of papers I and IV.

For the most complex industrial versions of the load balancing problem, there seems to be a great need for future research. Indeed, regarding how to solve the task assignment and sequencing problem—which currently can be solved only to some extent by commercial software—there are only a few (rudimentary) methods published in the (open) academic literature. Some related complex aspects of the problem concern how to model and prevent robot–robot collisions, how to choose from the (continuous) sets of task-alternatives, and how to compute and choose among alternative robot motions between configurations.

Many ideas, such as continuous collision detection being directly applicable to other problems, are not explored in this thesis. Similarly, the load balancing problem can be considered for other types of production cells and lines. However, some tailoring of solution methods is often required. For instance, there may be precedence constraints between the tasks, but the main ideas—of how robot–robot collisions can be prevented by a coordination in either time or space—still apply.

References

[1] ABB. Robot Stuido, 2022. URL https://new.abb.com/products/robotics/sv/robotstudio.

[2] ABB AB, Robotics. Technical reference manual RAPID instructions, functions and data, 2018. URL https://library.abb.com. Doc. ID: 9AKK107046A8697.

[3] E. Åblad, D. Spensieri, R. Bohlin, and J. S. Carlson. Intersection-free geometrical partitioning of multirobot stations for cycle time optimization. *IEEE Transactions on Automation Science and Engineering*, 15(2):842–851, 2018. doi:10/gc2bhc.

[4] E. Åblad, D. Spensieri, R. Bohlin, and A.-B. Strömberg. Continuous collision detection of pairs of robot motions under velocity uncertainty. *IEEE Transactions on Robotics*, 37(5):1780–1791, 2021. doi:10/gh5mq6.

[5] E. Åblad, A.-B. Strömberg, and D. Spensieri. Exact makespan minimization of unrelated parallel machines. *Open Journal of Mathematical Optimization*, 2:art. no. 2, 2021. doi:10/gjzkjv.

[6] N. Ahuja, R. T. Chien, R. Yen, and N. Bridwell. Interference detection and collision avoidance among three dimensional objects. In *Proceedings of the First AAAI Conference on Artificial Intelligence*, pages 44–48. AAAI Press, 1980.

[7] S. Alatartsev, S. Stellmacher, and F. Ortmeier. Robotic task sequencing problem: A survey. *Journal of Intelligent & Robotic Systems*, 80(2):279–298, 2015.

[8] A. Allahverdi, J. N. Gupta, and T. Aldowaisan. A review of scheduling research involving setup considerations. *Omega*, 27(2):219–239, 1999. doi:10/b9d8w5.

[9] P. Almström, C. Andersson, A. Mohammed, and M. Winroth. Achieving sustainable production through increased utilization of production resources. In *Proceedings of 4th Swedish Prod. Symp.*, pages 398–406, Lund, Sweden, 2011.

[10] D. Applegate, W. Cook, S. Dash, and A. Rohe. Solution of a min-max vehicle routing problem. *INFORMS Journal on Computing*, 14(2):132–143, 2002. doi:10/cxc3k2.

[11] D. L. Applegate, R. E. Bixby, V. Chvatál, and W. J. Cook. History of TSP computation. In *The Traveling Salesman Problem: A Computational Study*, chapter 4, pages 93–128. Princeton University Press, Princeton, NJ, USA, 2006.

[12] O. Battaïa and A. Dolgui. A taxonomy of line balancing problems and their solution approaches. *International Journal of Production Economics*, 142(2):259–277, 2013. doi:10/f4r7p5.

[13] M. S. Bazaraa, H. D. Sherali, and C. Shetty. *Nonlinear Programming: Theory and Algorithms*. Wiley, Hoboken, NJ, USA, 2006. doi:10/cwps3z.

[14] J. C. Beck, P. Prosser, and E. Selensky. Vehicle routing and job shop scheduling: What's the difference? In E. Giunchiglia, N. Muscettola, and D. S. Nau, editors, *Proceedings of the 13th International Conference on Automated Planning and Scheduling*, pages 267–276, Trento, Italy, 2003. AAAI Press.

[15] T. Bektas. The multiple traveling salesman problem: an overview of formulations and solution procedures. *Omega*, 34(3):209–219, 2006.

[16] E. Benavent and A. Martínez. Multi-depot multiple TSP: a polyhedral study and computational results. *Annals of Operations Research*, 207(1): 7–25, 2013. doi:10/cdfw4s.

[17] J. F. Benders. Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4:238–252, 1962. doi:10/d9prkz.

[18] P. Bhattacharya and M. L. Gavrilova. Voronoi diagram in optimal path planning. In C. Gold, editor, *4th International Symposium on Voronoi Diagrams in Science and Engineering (ISVD 2007)*, pages 38–47, Glamorgan, UK, 2007. IEEE. doi:10/c69p64.

[19] S. Björkenstam, D. Gleeson, R. Bohlin, J. S. Carlson, and B. Lennartson. Energy efficient and collision free motion of industrial robots using optimal control. In *2013 IEEE International Conference on Automation Science and Engineering (CASE)*, pages 510–515. IEEE, 2013. doi:10/f2zvpw.

[20] R. Bohlin and L. E. Kavraki. Path planning using lazy PRM. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings*, volume 1, pages 521–528, San Francisco, CA, USA, 2000. IEEE. doi:10/fb7gks.

[21] J. Canny. *The Complexity of Robot Motion Planning*. MIT Press, Cambridge, MA, USA, 1988.

[22] J. Canny. Computing roadmaps of general semi-algebraic sets. *The Computer Journal*, 36(5):504–514, 1993. doi:10/c73rkc.

[23] T. Cantos Lopes, C. G. S. Sikora, R. Gobbi Molina, D. Schibelbain, L. C. A. Rodrigues, and L. Magatão. Balancing a robotic spot welding manufacturing line: An industrial case study. *European Journal of Operational Research*, 263(3):1033–1048, 2017. doi:10/gnsnf3.

[24] J. S. Carlson, D. Spensieri, K. Wärmefjord, J. Segeborn, and R. Söderberg. Minimizing dimensional variation and robot traveling time in welding stations. *Procedia CIRP*, 23:77–82, 2014. doi:10/f3m33m.

[25] J. Carlsson, D. Ge, A. Subramaniam, and Y. Ye. Solving min-max multi-depot vehicle routing problem. In P. M. Pardalos and T. F. Coleman, editors, *Lectures on Global Optimization*, volume 55 of *Fields Institute Communications*, pages 31–46. AMS, Providence, RI, USA, 2009. doi:10/gnsnf7.

[26] J. Chen, J. Li, Y. Huang, C. Garrett, D. Sun, C. Fan, A. Hofmann, C. Mueller, S. Koenig, and B. C. Williams. Cooperative task and motion planning for multi-arm assembly systems, 2022. preprint, doi:10/hj2r.

[27] M. Chen, S. Bansal, J. F. Fisac, and C. J. Tomlin. Robust sequential trajectory planning under disturbances and adversarial intruder. *IEEE Transactions on Control Systems Technology*, 27(4):1566–1582, 2018. doi:10/gnsndk.

[28] X. Chen, P. Zhang, G. Du, and F. Li. Ant colony optimization based memetic algorithm to solve bi-objective multiple traveling salesmen problem for multi-robot systems. *IEEE Access*, 6:21745–21757, 2018. doi:10/hfk2.

[29] M. Conforti, G. Cornuejols, and G. Zambelli. *Integer Programming*. Springer, Cham, 2014. doi:10/ggmgxb.

[30] S. A. Cook. The complexity of theorem-proving procedures. In M. A. Harrison, R. B. Banerji, and J. D. Ullman, editors, *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, pages 151–158, New York, NY, USA, 1971. ACM. doi:10/fstktp.

[31] R. Culley and K. Kempf. A collision detection algorithm based on velocity and distance bounds. In *Proceedings. 1986 IEEE International Conference on Robotics and Automation*, volume 3, pages 1064–1069, San Francisco, CA, USA, 1986. IEEE. doi:10/dbprjr.

[32] E. Danna, E. Rothberg, and C. Le Pape. Exploring relaxation induced neighborhoods to improve MIP solutions. *Mathematical Programming*, 102 (1):71–90, 2005. doi:10/drtgp9.

[33] G. Dantzig, R. Fulkerson, and S. Johnson. Solution of a large-scale traveling-salesman problem. *Journal of the Operations Research Society of America*, 2(4):393–410, 1954. doi:10/bdjpbx.

[34] R. Dewil, P. Vansteenwegen, D. Cattrysse, M. Laguna, and T. Vossen. An improvement heuristic framework for the laser cutting tool path problem. *International Journal of Production Research*, 53(6):1761–1776, 2015. doi:10/gm36kg.

[35] J. Edwards, E. Daniel, V. Pascucci, and C. Baja. Approximating the generalized Voronoi diagram of closely spaced objects. *Computer Graphics Forum*, 34(2):299–309, 2015. doi:10/f7mbjr.

[36] M. Erdmann and T. Lozano-Perez. On multiple moving objects. *Algorithmica*, 2(1):477–521, 1987. doi:10/dv9stf.

[37] G. Erdős, C. Kardos, Z. Kemény, A. Kovács, and J. Váncza. Process planning and offline programming for robotic remote laser welding systems. *International Journal of Computer Integrated Manufacturing*, 29(12): 1287–1306, 2016. doi:10/hjdc.

[38] J. Faigl and L. Přeučil. Self-organizing map for the multi-goal path planning with polygonal goals. In T. Honkela, W. Duch, M. Girolami, and S. Kaski, editors, *International Conference on Artificial Neural Networks*, pages 85–92, Berlin, Heidelberg, 2011. Springer. doi:10/cq49w4.

[39] E. Ferre and J.-P. Laumond. An iterative diffusion algorithm for part disassembly. In *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004*, volume 3, pages 3149–3154, New Orleans, LA, USA, 2004. IEEE. doi:10/cv8np5.

[40] M. Fischetti, J.-J. Salazar-González, and P. Toth. The generalized traveling salesman and orienteering problems. In G. Gutin and A. P. Punnen, editors, *The Traveling Salesman Problem and Its Variations*, volume 12 of *Combinatorial Optimization*, pages 609–662. Springer, Boston, MA, 2007. doi:10/c2zshz.

[41] F. Focacci, P. Laborie, and W. Nuijten. Solving scheduling problems with setup times and alternative resources. In *Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems*, pages 92–101, Breckenridge, CO, USA, 2000. AAAI Press.

[42] M. Franceschelli, D. Rosa, C. Seatzu, and F. Bullo. Gossip algorithms for heterogeneous multi-vehicle routing problems. *Nonlinear Analysis: Hybrid Systems*, 10:156–174, 2013. doi:10/f48vdc.

[43] I. Gentilini, F. Margot, and K. Shimada. The travelling salesman problem with neighbourhoods: MINLP solution. *Optimization Methods and Software*, 28(2):364–378, 2013. doi:10/fxmcjw.

[44] A. M. Geoffrion. Lagrangean relaxation for integer programming. In M. L. Balinski, editor, *Approaches to Integer Programming*, volume 2 of *Mathematical Programming Studies*, pages 82–114, Berlin, Heidelberg, 1974. Springer. doi:10/g9f8.

[45] E. G. Gilbert, D. W. Johnson, and S. S. Keerthi. A fast procedure for computing the distance between complex objects in three-dimensional space. *IEEE Journal on Robotics and Automation*, 4(2):193–203, 1988. doi:10/d7q7k3.

[46] P. E. Gill, W. Murray, and M. A. Saunders. SNOPT: An SQP algorithm for large-scale constrained optimization. *SIAM Review*, 47(1):99–131, 2005. doi:10/fm2h6m.

[47] R. E. Gomory. An algorithm for the mixed integer problem. Technical Report RM-2597-PR, The Rand Corporation, Santa Monica, CA, 1960.

[48] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. R. Kan. Optimization and approximation in deterministic sequencing and scheduling: A survey. In P. Hammer, E. Johnson, and B. Korte, editors, *Discrete Optimization II*, volume 5 of *Annals of Discrete Mathematics*, pages 287–326. Elsevier, 1979. doi:10/bn7sj2.

[49] L. B. Gueta, R. Chiba, J. Ota, T. Ueyama, and T. Arai. Coordinated motion control of a robot arm and a positioning table with arrangement of multiple goals. In *2008 IEEE International Conference on Robotics and Automation*, pages 2252–2258, Pasadena, CA, USA, 2008. IEEE. doi:10/fnsv65.

[50] Gurobi Optimization, LLC. Gurobi optimizer reference manual, 2022. URL www.gurobi.com.

[51] A. L. Gutjahr and G. L. Nemhauser. An algorithm for the line balancing problem. *Management Science*, 11(2):308–315, 1964. doi:10/fd5djw.

[52] F. Hagebring, A. Farooqui, M. Fabian, and B. Lennartson. On optimization of automation systems: Integrating modular learning and optimization.

IEEE Transactions on Automation Science and Engineering, 2022. doi:10/hmb8. Early Access.

[53] M. Herman. Fast, three-dimensional, collision-free motion planning. In *Proceedings. 1986 IEEE International Conference on Robotics and Automation*, volume 3, pages 1056–1063, San Francisco, CA, USA, 1986. doi:10/bf3hcs.

[54] D. S. Hochbaum and D. B. Shmoys. Using dual approximation algorithms for scheduling problems: Theoretical and practical results. *Journal of the ACM*, 34(1):144–162, 1987. doi:10/ckst6p.

[55] D. Hömberg, C. Landry, M. Skutella, and W. A. Welz. Automatic reconfiguration of robotic welding cells. In L. Ghezzi, D. Hömberg, and C. Landry, editors, *Math for the Digital Factory*, volume 27 of *Mathematics in Industry*, pages 183–203. Springer, Cham, 2017. doi:10/dvjz.

[56] J. N. Hooker. Logic-based Benders decomposition for large-scale optimization. In J. M. Velásquez-Bermúdez, M. Khakifirooz, and M. Fathi, editors, *Large Scale Optimization in Supply Chains and Smart Manufacturing: Theory and Applications*, volume 149 of *Springer Optimization and Its Applications*. Springer, Cham, 2019. doi:10/gnsndg.

[57] P. Indyk and R. Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In J. Vitter, editor, *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, pages 604–613, New York, NY, USA, 1998. ACM. doi:10/c65pxc.

[58] IPS AB. Industrial Path Solution, 2022. URL http://www.fcc.chalmers.se/software/ips/.

[59] K. Jansen and L. Porkolab. Improved approximation schemes for scheduling unrelated parallel machines. *Mathematics of Operations Research*, 26(2):324–338, 2001. doi:10/ccdbjw.

[60] D. Karapetyan and G. Gutin. Efficient local search algorithms for known and new neighborhoods for the generalized traveling salesman problem. *European Journal of Operational Research*, 219(2):234–251, 2012. doi:10/fx5xwv.

[61] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996. doi:10/fsgth3.

[62] J. Kim and H. I. Son. A Voronoi diagram-based workspace partition for weak cooperation of multi-robot system in orchard. *IEEE Access*, 8: 20676–20686, 2020. doi:10/g84p.

[63] Y. J. Kim, M. C. Lin, and D. Manocha. DEEP: Dual-space expansion for estimating penetration depth between convex polytopes. In *Proceedings 2002 IEEE International Conference on Robotics and Automation*, volume 1, pages 921–926, Washington, DC, USA, 2002. IEEE. doi:10/cqzshc.

[64] A. Kovács. Integrated task sequencing and path planning for robotic remote laser welding. *International Journal of Production Research*, 54(4): 1210–1224, 2016. doi:10/gnsnf6.

[65] P. Laborie, J. Rogerie, P. Shaw, and P. Vilím. IBM ILOG CP optimizer for scheduling. 20+ years of scheduling with constraints at IBM/ILOG. *Constraints*, 23(2):210–250, 2018. doi:10/gdg3b4.

[66] C. Landry, R. Henrion, D. Hömberg, M. Skutella, and W. Welz. Task assignment, sequencing and path-planning in robotic welding cells. In *18th International Conference on Methods and Models in Automation and Robotics (MMAR)*, pages 252–257, Miedzyzdroje, Poland, 2013. IEEE. doi:10/gnsndd.

[67] C. Landry, M. Gerdts, R. Henrion, D. Hömberg, and W. Welz. Collision-free path planning of welding robots. In M. Fontes, M. Günther, and N. Marheineke, editors, *Progress in Industrial Mathematics at ECMI 2012*, volume 19 of *Mathematics in Industry*, pages 251–256. Springer, Cham, 2014. doi:10/gnsndf.

[68] A. Langevin, F. Soumis, and J. Desrosiers. Classification of travelling salesman problem formulations. *Operations Research Letters*, 9(2):127–132, 1990. doi:10/fw4cms.

[69] G. Laporte. The traveling salesman problem: An overview of exact and approximate algorithms. *European Journal of Operational Research*, 59(2): 231–247, 1992. doi:10/bqx7tw.

[70] E. Larsen, S. Gottschalk, M. C. Lin, and D. Manocha. Fast distance queries with rectangular swept sphere volumes. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings*, volume 4, pages 3719–3726, San Francisco, CA, USA, 2000. IEEE. doi:10/db2qkp.

[71] T. Larsson, M. Patriksson, and A.-B. Strömberg. Conditional subgradient optimization—Theory and applications. *European Journal of Operational Research*, 88(2):382–403, 1996. doi:10/fdw7nv.

[72] S. M. LaValle. Motion planning. In *Planning Algorithms*, pages 63–356. Cambridge University Press, 2006. doi:10/fv5hmg.

[73] S. M. LaValle and J. J. Kuffner Jr. Randomized kinodynamic planning. *The International Journal of Robotics Research*, 20(5):378–400, 2001. doi:10/bnxbnz.

[74] J. K. Lenstra, D. B. Shmoys, and É. Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Mathematical Programming*, 46 (1–3):259–271, 1990. doi:10/fntb9s.

[75] Y. Liu, W. Zhao, T. Lutz, and X. Yue. Task allocation and coordinated motion planning for autonomous multi-robot optical inspection systems. *Journal of Intelligent Manufacturing*, 2021. doi:10/g8vc. online only.

[76] R. Luna, M. Moll, J. Badger, and L. E. Kavraki. A scalable motion planner for high-dimensional kinematic systems. *The International Journal of Robotics Research*, 2019. doi:10/gm25w8.

[77] R. R. Meyer. On the existence of optimal solutions to integer and mixed-integer programming problems. *Mathematical Programming*, 7:223–235, 1974. doi:10/drg9ds.

[78] A. S. Michels, T. C. Lopes, C. G. S. Sikora, and L. Magatão. The robotic assembly line design (RALD) problem: Model and case studies with practical extensions. *Computers & Industrial Engineering*, 120:320–333, 2018. doi:10/gdtvtd.

[79] C. E. Miller, A. W. Tucker, and R. A. Zemlin. Integer programming formulation of traveling salesman problems. *Journal of the ACM*, 7(4): 326–329, 1960. doi:10/fs8qpn.

[80] H. Minkowski. *Geometrie der Zahlen*. Lieferun, Leipzig, 1st edition, 1896.

[81] J. R. Montoya-Torres, J. López Franco, S. Nieto Isaza, H. Felizzola Jiménez, and N. Herazo-Padilla. A literature review on the vehicle routing problem with multiple depots. *Computers & Industrial Engineering*, 79:115–129, 2015. doi:10/gfpnrb.

[82] C. Müller, M. Grunewald, and T. S. Spengler. Redundant configuration of robotic assembly lines with stochastic failures. *International Journal of Production Research*, 56(10):3662–3682, 2018. doi:10/gnsnf4.

[83] R. M. Murray, Z. Li, and S. S. Sastry. *A Mathematical Introduction to Robotic Manipulation*. CRC press, 1994. doi:10/ggsjf7.

[84] G. Nemhauser and L. Trotter Jr. Properties of vertex packing and independence system polyhedra. *Mathematical Programming*, 6(1):48–61, 1974. doi:10/cc7738.

[85] J. M. Nilakantan, G. Q. Huang, and S. Ponnambalam. An investigation on minimizing cycle time and total energy consumption in robotic assembly line systems. *Journal of Cleaner Production*, 90:311–325, 2015. doi:10/f64d3b.

[86] A. Okabe. *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams*. Wiley, Chichester, England, 2nd edition, 2000.

[87] K. Okumura and X. Défago. Quick multi-robot motion planning by combining sampling and search, 2022.

[88] S. Pellegrinelli, N. Pedrocchi, L. M. Tosatti, A. Fischer, and T. Tolio. Multi-robot spot-welding cells for car-body assembly: Design and motion planning. *Robotics and Computer-Integrated Manufacturing*, 44:97–116, 2017. doi:10/ggh369.

[89] D. Portugal and R. Rocha. A survey on multi-robot patrolling algorithms. In L. M. Camarinha-Matos, editor, *Technological Innovation for Sustainability*, volume 349 of *Information and Communication Technology*, pages 139–146, Berlin, Heidelberg, 2011. Springer. doi:10/dbz5vf.

[90] J. Rambau and C. Schwarz. On the benefits of using NP-hard problems in Branch & Bound. In B. Fleischmann, K.-H. Borgwardt, R. Klein, and A. Tuma, editors, *Operations Research Proceedings 2008*, pages 463–468. Springer, Berlin, Heidelberg, 2009. doi:10/cdw8bf.

[91] J. Rambau and C. Schwarz. Solving a vehicle routing problem with resource conflicts and makespan objective with an application in car body manufacturing. *Optimization Methods and Software*, 29(2):353–375, 2014. doi:10/gnsnf5.

[92] S. Redon, M. C. Lin, D. Manocha, and Y. J. Kim. Fast continuous collision detection for articulated models. *Journal of Computing and Information Science in Engineering*, 5(2):126–137, 2005. doi:10/dzdw6p.

[93] J. H. Reif. Complexity of the mover's problem and generalizations. In *20th Annual Symposium on Foundations of Computer Science*, pages 421–427, San Juan, PR, 1979. IEEE. doi:10/fvd7ff.

[94] S. Riazi, C. Seatzu, O. Wigström, and B. Lennartson. Benders/gossip methods for heterogeneous multi-vehicle routing problems. In *2013 IEEE*

18th Conference on Emerging Technologies & Factory Automation (ETFA), pages 1–6. IEEE, 2013. doi:10/dzpc.

[95] S. Riazi, K. Bengtsson, O. Wigström, E. Vidarsson, and B. Lennartson. Energy optimization of multi-robot systems. In *2015 IEEE International Conference on Automation Science and Engineering (CASE)*, pages 1345–1350. IEEE, 2015. doi:10/f3ng8g.

[96] A. Rossi and G. Dini. Flexible job-shop scheduling with routing flexibility and separable setup times using ant colony optimisation method. *Robotics and Computer-Integrated Manufacturing*, 23(5):503–516, 2007. doi:10/b5rmpg.

[97] R. Sadeghi Tabar, L. Lindkvist, K. Wärmefjord, and R. Söderberg. Efficient joining sequence variation analysis of stochastic batch assemblies. *Journal of Computing and Information Science in Engineering*, art. art. no. 21-1311, 2021. doi:10/hmb9.

[98] M. Saha and P. Isto. Multi-robot motion planning by incremental coordination. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5960–5963. IEEE, 2006. doi:10/d9rpwx.

[99] R. Salman. *Optimizing and Approximating Algorithms for the Single and Multiple Agent Precedence Constrained Generalized Traveling Salesman Problem*. Licentiate thesis, Department of Mathematical Sciences, Chalmers University of Technology, 2017. URL https://research.chalmers.se/en/publication/252879.

[100] R. Salman, J. S. Carlson, F. Ekstedt, D. Spensieri, J. Torstensson, and R. Söderberg. An industrially validated CMM inspection process with sequence constraints. *Procedia CIRP*, 44:138–143, 2016. doi:10/f3rtbx.

[101] F. Schwarzer, M. Saha, and J.-C. Latombe. Exact collision checking of robot paths. In J.-D. Boissonnat, J. Burdick, K. Goldberg, and S. Hutchinson, editors, *Algorithmic Foundations of Robotics V*, volume 7 of *Springer Tracts in Advanced Robotics*, pages 25–41. Springer, Berlin, Heidelberg, 2004. doi:10/fpwz.

[102] J. Segeborn, D. Segerdahl, F. Ekstedt, J. S. Carlson, M. Andersson, A. Carlsson, and R. Söderberg. An industrially validated method for weld load balancing in multi station sheet metal assembly lines. *Journal of Manufacturing Science and Engineering*, 136(1), 2014. doi:10/f24hxx.

[103] M. Skutella and W. Welz. Route planning for robot systems. In B. Hu, K. Morasch, S. Pickl, and M. Siegle, editors, *Operations Research Proceedings*

2010, Operations Research Proceedings, pages 307–312. Springer, Berlin, Heidelberg, 2011. doi:10/bcxsc9.

[104] R. Söderberg, K. Wärmefjord, J. S. Carlson, and L. Lindkvist. Toward a Digital Twin for real-time geometry assurance in individualized production. *CIRP Annals*, 66(1):137–140, 2017. doi:10/gbn5xp.

[105] D. Spensieri, R. Bohlin, and J. S. Carlson. Coordination of robot paths for cycle time minimization. In *2013 IEEE International Conference on Automation Science and Engineering (CASE)*, pages 522–527, Madison, WI, USA, 2013. IEEE. doi:10/f22vrk.

[106] D. Spensieri, J. S. Carlson, F. Ekstedt, and R. Bohlin. An iterative approach for collision free routing and scheduling in multirobot stations. *IEEE Transactions on Automation Science and Engineering*, 13(2):950–962, 2016. doi:10/f3p244.

[107] D. Spensieri, E. Åblad, R. Bohlin, J. S. Carlson, and R. Söderberg. Modeling and optimization of implementation aspects in industrial robot coordination. *Robotics and Computer-Integrated Manufacturing*, 69:art. no. 102097, 2021. doi:10/gh9tmk.

[108] M. Stoer and F. Wagner. A simple min-cut algorithm. *Journal of the ACM*, 44(4):585–591, 1997. doi:10/ds2gz2.

[109] R. S. Tabar, K. Wärmefjord, and R. Söderberg. A new surrogate model–based method for individualized spot welding sequence optimization with respect to geometrical quality. *The International Journal of Advanced Manufacturing Technology*, 106:2333–2346, 2020. doi:10/gh85nw.

[110] V. Tereshchuk, J. Stewart, N. Bykov, S. Pedigo, S. Devasia, and A. G. Banerjee. An efficient scheduling algorithm for multi-robot task allocation in assembling aircraft structures. *IEEE Robotics and Automation Letters*, 4 (4):3844–3851, 2019. doi:10/gmpvcq.

[111] H. Touzani, H. Hadj-Abdelkader, N. Séguy, and S. Bouchafa. Multi-robot task sequencing & automatic path planning for cycle time optimization: Application for car production line. *IEEE Robotics and Automation Letters*, 6(2):1335–1342, 2021. doi:10/fxqj.

[112] H. Touzani, N. Séguy, H. Hadj-Abdelkader, R. Suarez, J. Rosell, L. Palomo-Avellaneda, and S. Bouchafa. Efficient industrial solution for robotic task sequencing problem with mutual collision avoidance & cycle time optimization. *IEEE Robotics and Automation Letters*, 7(2):2597–2604, 2022. doi:10/hcqw.

[113] M. Turpin, N. Michael, and V. Kumar. An approximation algorithm for time optimal multi-robot routing. In H. L. Akin, N. M. Amato, V. Isler, and A. F. van der Stappen, editors, *Algorithmic Foundations of Robotics XI*, volume 107 of *Springer Tracts in Advanced Robotics*, pages 627–640. Springer, Cham, 2015. doi:10/gnsndj.

[114] J. P. van den Berg and M. H. Overmars. Prioritized motion planning for multiple robots. In *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 430–435, Edmonton, AB. Canada, 2005. IEEE. doi:10/cvm5qc.

[115] K. Vicencio, B. Davis, and I. Gentilini. Multi-goal path planning based on the generalized traveling salesman problem with neighborhoods. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2985–2990, Chicago, IL, USA, 2014. IEEE. doi:10/gnsndh.

[116] J. Wang, W. Chi, C. Li, and M. Q.-H. Meng. Efficient robot motion planning using bidirectional-unidirectional RRT extend function. *IEEE Transactions on Automation Science and Engineering*, pages 1–10, 2021. doi:10/g84r.

[117] X. Wang, B. Golden, and E. Wasil. The min-max multi-depot vehicle routing problem: heuristics and computational results. *Journal of the Operational Research Society*, 66(9):1430–1441, 2015. doi:10/f7qqbr.

[118] X. Wang, Y. Shi, Y. Yan, and X. Gu. Intelligent welding robot path optimization based on discrete elite PSO. *Soft Computing*, 21(20):5869–5881, 2017. doi:10/gb23f4.

[119] K. Wärmefjord, R. Söderberg, and L. Lindkvist. Strategies for optimization of spot welding sequence with respect to geometrical variation in sheet metal assemblies. In *Proceedings of the ASME 2010 International Mechanical Engineering Congress and Exposition*, volume 3 of *Design and Manufacturing, Parts A and B*, pages 569–577, Vancouver, British Columbia, Canada, 2010. ASME. doi:10/f3rqsw.

[120] C. Wei, Z. Ji, and B. Cai. Particle swarm optimization for cooperative multi-robot task allocation: A multi-objective approach. *IEEE Robotics and Automation Letters*, 5(2):2530–2537, 2020. doi:10/g84n.

[121] J. Xin, C. Meng, F. Schulte, J. Peng, Y. Liu, and R. Negenborn. A time-space network model for collision-free routing of planar motions in a multi-robot station. *IEEE Transactions on Industrial Informatics*, 2020. doi:10/gnsndm.

[122] B. Zhou, R. Zhou, Y. Gan, F. Fang, and Y. Mao. Multi-robot multi-station cooperative spot welding task allocation based on stepwise optimization: An industrial case study. *Robotics and Computer-Integrated Manufacturing*, 73:art. no. 102197, 2022. doi:10/hjc3.