CHALMERS
UNIVERSITY OF TECHNOLOGY

Spatial–temporal load balancing and coordination of multi-robot stations

Citation for the original published paper (version of record):
Åblad, E., Spensieri, D., Bohlin, R. et al (2023). Spatial–temporal load balancing and coordination of multi-robot stations. IEEE Transactions on Automation Science and Engineering, 20(4): 2203-2214. http://dx.doi.org/10.1109/TASE.2022.3214567

(article starts on next page)

Spatial–Temporal Load Balancing and Coordination of Multi-Robot Stations

Edvin Åblad, Domenico Spensieri, Robert Bohlin, Johan S. Carlson, and Ann-Brith Strömberg

***Abstract*— Cycle time minimization in multi-robot manufacturing stations is computationally challenging. This is due to the many aspects that need to be accounted for, including assigning process tasks to robots, specifying robot configurations at tasks, sequencing, planning motions, and coordinating the robots to avoid collisions. Hence, to find good solutions, often some assumptions are made and/or the problem is divided into subproblems—often limiting the set of solutions with the risk of excluding the best ones. In this study, we generalize the completely disjoint solution method that challenges the so-called *shortest path* assumption, i.e., to let each robot use its shortest collision-free motion between any two configurations, regardless of the other robots. We devise a generalized method called *spatial–temporal load balancing and coordination*, which prevents robot–robot collisions by a sequence of disjoint solutions, guiding task assignments, sequences, and robot motions (path and velocity). We study both artificial and industrial instances. For some of them, our suggested method is superior to methods based on the shortest path assumption, with as much as a 28% reduction in cycle time. Moreover, for problem instances with no special structure, we establish that the shortest path assumption is often reasonable.**

***Note to Practitioners*—This work is motivated by a particular industrial problem instance of a spot-welding station with two robots and where welds are placed along the edge of a workpiece. Due to the special geometry of the instance one robot can only perform welds in the middle of the edge and the other only at the ends. As a result, if the robots use their shortest motions between welds, then waiting times are required to prevent collisions. Moreover, the tasks are too close to each other to allow for a completely disjoint solution. Hence, we suggest a method based on sequence of disjoint solutions.**

***Index Terms*— Robotic assembly, automotive manufacturing, robot programming, discrete optimization, generalized Voronoi diagram, path planning, coordination.**

Manuscript received 9 June 2022; revised 16 September 2022; accepted 10 October 2022. This article was recommended for publication by Associate Editor L. Bascetta and Editor C. Seatzu upon evaluation of the reviewers' comments. This work was supported in part by the Project Smart Assembly 4.0 through the Swedish Foundation for Strategic Research (SSF) under Project RIT15-0025 and in part by the Sustainable Production Initiative and the Production Area of Advance at Chalmers University of Technology. *(Corresponding author: Edvin Åblad.)*

Edvin Åblad, Domenico Spensieri, Robert Bohlin, and Johan S. Carlson are with the Fraunhofer-Chalmers Research Centre for Industrial Mathematics, Geometry and Motion Planning Group, 41288 Gothenburg, Sweden (e-mail: edvin.ablad@fcc.chalmers.se).

Ann-Brith Strömberg is with the Department of Mathematical Sciences, Chalmers University of Technology and University of Gothenburg, 412 58 Gothenburg, Sweden.

This article has supplementary material provided by the authors and color versions of one or more figures available at https://doi.org/10.1109/TASE.2022.3214567.

Digital Object Identifier 10.1109/TASE.2022.3214567

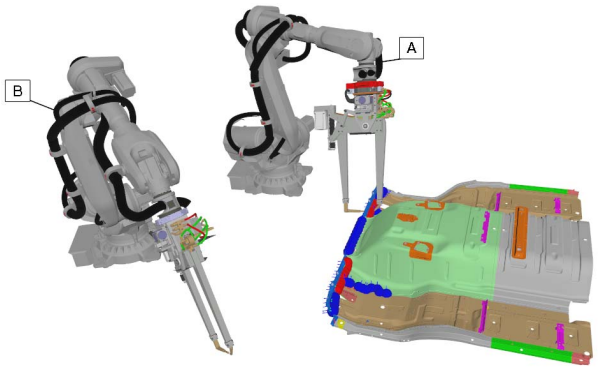


Fig. 1. Two robots (A and B) with spot weld tools stationed at a workpiece with spot weld tasks (blue/red markers). Robot B is configured at its, so-called, *home position*; Robot A is in a configuration valid for a specific spot weld task on the workpiece. Both robots reach the blue tasks, but only the right reaches the red ones. For clarity, the surrounding geometries (e.g., floors) are left out. Instance courtesy of Scania AB.

I. Introduction

TODAY'S manufacturing systems have an increasing demand for automatic programming and optimization support. Besides the classical demands on throughput and lead time, there is a recent trend towards a product individualization. By accounting for minor deviations in the parts to be assembled, the quality of the overall product can be increased by optimizing the assembly process for each individual product, see [1]. Such an individual process contrasts the established approach of mass production, in which the same process is used for a series of products and thus the parts to assemble must have a high quality, which is costly.

We consider the automatic optimization of a robotic cell, consisting of multiple robotic arms that need to perform a set of tasks on a workpiece. We will refer to this as the *load balancing* problem, and the finish time of the last robot (aka. makespan or cycle time) is to be minimized. In an automotive production line different kinds of tasks are of interest, for example inspection, sealing, and various types of welding operations; see Figure 1. However, many of the solution methods for the load balancing problem are rather independent of the type of task, e.g., even though [2] refers to this problem as the *welding cell problem* their main idea applies to other types of tasks. Some aspects of the tasks, such as tool positioning, must, however, be specified. For example, weld operations are free to rotate but need to be placed orthogonally to the workpiece surface, whereas inspection ones are allowed to deviate from a perfect orthogonal angle.

Another aspect concerns whether the robot is stationary during the task (e.g., inspection) or if the task involves a large robot motion (e.g., sealing). To simplify the presentation, we assume that for a robot to perform a specific task, there is a (possibly empty) discrete set of possible *configurations* of the robot's joints. Our method extends also to tasks involving tool motions.

A major complication of the load balancing problem consists of preventing pairs of robots from colliding. Thus, a solution to the load balancing problem includes (in addition to scheduling, i.e., the robot-task assignment, configurations choice at tasks, and task sequences) the robots' motions. A common assumption simplifying the problem is that the best motion for each robot is the shortest collision-free path between its configurations in the static environment (e.g., [2], [3], and [4]). To prevent robot–robot collisions, task sequences and motion velocities are then specified. However, a robot's detour from its shortest path may be beneficial in order to prevent robot–robot collisions. In some scenarios this is important due to two facts:

- If the shortest paths are used, then all solutions might include long waiting times for the robots.
- We often consider robots with six revolute joints, see Figure 1. The revolute velocity of each joint is limited, hence it is likely that there are many motions with (near) minimal duration, i.e., many shortest paths.

In this article, we challenge the use of the established *shortest path assumption* when scheduling, initially motivated by an industrial instance for which this assumption was unsuitable, see Figure 1. In this instance, robot A must perform (red) tasks at both ends of the workpiece's edge. Thus, when moving to the other end, it must use a detour to avoid robot B, which performs tasks in the middle of the workpiece's edge.

We suggest a generalization of our method [5], in which the robots are partitioned into disjoint zones of the workstation by use of a static partitioning surface; hence pairs of robots cannot collide. However, that method cannot be used on all problem instances since there is no guarantee that a static partition exists or that it allows for a good enough makespan. To resolve this issue, we suggest a method that enables a dynamic partition. This is motivated by the fact that if at all times the robots are free of collision, then there exists a dynamic partition of the workspace, and vice versa. Moreover, the dynamic partition and the robots' task sequences depends on each other. The method thus unifies the two computationally challenging aspects: scheduling and preventing robot–robot collisions.

As a consequence, the new method does not guarantee to find an optimal dynamic partition. Moreover, even though speculating that a more efficient implementation exists, we suspect that not utilizing the *shortest path* assumption will inevitably increase the required computational effort. As a result, we still believe that one should apply the shortest path assumption for most of the industrial encountered instances. Our new method is meant for challenging instances in which the shortest path assumption results in poor solutions.

The outline of the article is as follows. Related work is reviewed in Section II. In Section III the suggested method and its optimization models are presented and motivated. In Section IV we present the computational tests and results, which compare the method with a state-of-the-art software [6] on both (motivational) artificial and industrial instances.

II. Related Work

The general *load balancing problem* of a multi-robot station has lately received a lot of attention. However, not only solution methods differ, but also the problem formulation. First, we differentiate from the so-called line balancing problems (e.g., [7]): these often consider complete production/assembly lines, in which the robots can have completely different purposes. In such a setting the main complication is often represented by constraints, such as precedence, between tasks. However, the problem is often less detailed and does not include motion planning. An example presented in [8] assumes a constant duration between different groups of tasks and that robots can safely work simultaneously on these groups. Other examples that study the load balancing problem are presented in [9] and [10]: in the former the motion planning problem is disregarded and in the latter the workpiece is roughly a plane and thus the motions are easy to compute.

However, the robots' motions have a major impact on the makespan and on the feasibility of the problem. The motion planning problem is PSPACE-complete, and it is reasonable to assume that it has an exponential computational complexity with respect to the number of degrees of freedom (robot joints) (see [11]). Thus, planning a motion between two configurations in a static environment is a challenge, and proving a motion to be optimal (or that no motion exists) is not reasonable for a general instance. Therefore, sampling-based algorithms such as [12] or [13] are often used to compute the motions. On some instances (or for some robots) exact methods based on optimal-control such as [14] can be applied.

The complicating fact that the load balancing problem is influenced by the robots' motions needs to be handled. A common approach is to apply an algorithm that does not require planning all motions: it relies on an initial bounding of the motion duration and the assumption that no collisions occur. Then, the model is iteratively refined by planning a subset of motions: finding the actual durations and collisions. This can be done in different ways: in [3] a lower bounding problem is iteratively solved, the motions in the solution are planned and the lower bounding problem updated. In [2] a so-called *branch-and-price* method is used to detect which motions to plan and how to combine planned sequences into a solution. In [15] collision-free paths are precomputed and a mixed integer linear model is solved to minimize the cost of the station for a given makespan, where so-called *big-M* constraints model robot–robot collisions. In [4] a so-called *genetic algorithm* is purposed for both the assignment and sequencing of tasks as well as for the planning of the motions. All these, however, utilize the shortest path assumption.

In order to find a feasible solution in which robots do not collide, it is common to optimize the velocities of the robots and to fix the sequences and geometric paths of the motions; this optimization is often referred to as *coordination* (see [16]). A similar approach is to allow waiting in the optimization

problem that is solved to assign tasks and sequences, and to prohibit motions that cannot be used simultaneously (see, e.g., [2]). However, there are methods that prevent robot–robot collisions by modifying the geometric paths of the motions, e.g., the so-called *prioritized planning* approach [17], in which the motions are planned in a specific order of the robots, and are such that the already planned motions are avoided. The prioritized planning idea is extended in [18] that also coordinates previously planned motions simultaneously as the new robot path is planned. A further extension is to simultaneously plan motions for multiple robots as in [19] and [20]. In [21] also the fixed sequences are challenged by considering some robot–robot collisions during scheduling, the remaining robot–robot collisions are prevented by using multi-robot motion planning in a post processing step.

Another approach to avoid robot–robot collisions, suggested in [5], computes a static partition of the workstation, and each robot is always contained in a private zone. The partition is suggested to be a so-called *generalized Voronoi diagram* using the robots as so-called *generators*, see [22]. The use of a static partition is also used in [23]. To summarize, currently there are methods that coordinate robots by optimizing their velocities, i.e., *temporal coordination*, and methods that coordinate robots by separating them in space, i.e., *spatial coordination*, and some methods that exploit both aspects to plan motions. However, to the best of our knowledge, there is no method that exploits both the temporal and the spatial aspects for the load balancing problem considering a cluttered environment.

We apply a method similar to our previous work [3], in the sense that an optimization problem is iteratively solved to find task assignments (corresponding robot configurations) and sequences. In each iteration the model is updated, mainly by the new durations of the motions. In [3] the problem to solve is a so-called Min-Max Multiple Generalized Travelling Salesperson Problem, where min-max defines the makespan objective, multiple agents (i.e., robots) are considered, and generalized means that for each pair of task and agent there can be multiple configurations. However, to allow for the robots to be partitioned into disjoint zones, we also consider *set packing constraints* as in [5]. Our model is inspired by models of the generalized TSP, see e.g., [24].

III. Spatial–Temporal Load Balancing and Coordination

We generalize the concept of a workspace *partition* introduced in [5], in which the robots are separated into disjoint zones by a partitioning surface. The main motivation behind our former approach is to implicitly force the robots to not take shortcuts that cause needless waiting times. Our new approach includes a coarse time-indexing, in each time-period the workspace is partitioned such that consecutive partitions slightly overlap in time, to allow for the robots to safely transition between time-periods without necessarily waiting on another. To build these partitions and to optimize the corresponding robots' task sequences, we suggest an algorithm with two main stages and corresponding optimization models.

The algorithm is called *spatial–temporal* load balancing and coordination algorithm (ST-algorithm) to indicate that both the spatial and temporal domains are utilized; its flowchart is depicted in Figure 2. Additionally, algorithms based on the shortest path assumption will be referred to as SP-algorithms. Before the main stages of the ST-algorithm the initialization step consists of finding all colliding pairs of robot configurations at tasks, and for each robot to compute lower bounds on the durations of the collision-free motions between its configurations at tasks (see Section III-A for details and Figure 3 for an illustration of colliding configurations).

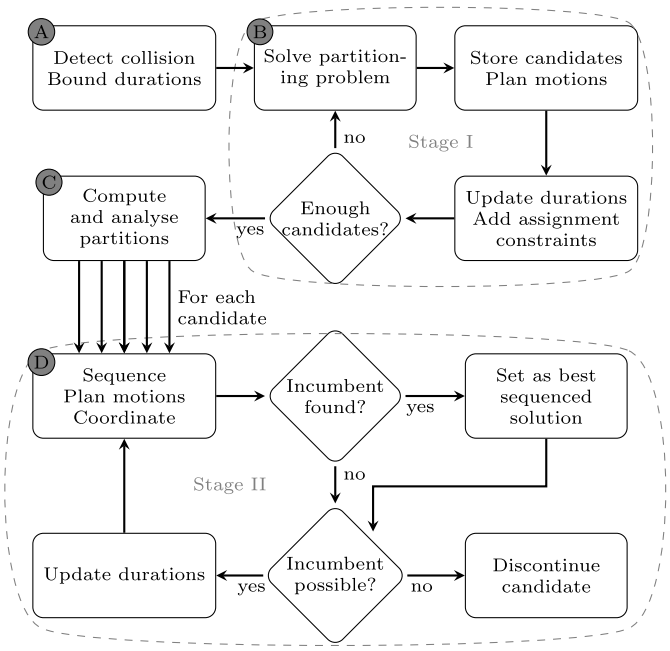


Fig. 2. Main components, corresponding subsections (A–D), and two stages of our *spatial–temporal* load balancing and coordination algorithm.

The purpose of stage I is to find multiple candidate (dynamic) partitions, of various numbers of time-periods. This is done by iteratively solving the stage I optimization model that selects a robot and its configuration for each task. Additionally, all robot configurations used in the same time-period need to be pairwise collision-free (the first configurations also being collision-free w.r.t. all configurations in the previous time-period). Moreover, in stage I the makespan is approximated by letting the duration between a robot's tasks to be the minimum duration between any configurations of the corresponding tasks. This simplification enables the stage I model to be efficiently solved with a reasonable accuracy. The blocks of stage I are iterated, resulting in many solutions to the stage I model by updating the lower bounds on the duration of motions and by prohibiting past solutions; the details are described in Section III-B. Each stage I solution generates a candidate (dynamic) partition that is static in each time-period.

In stage II of the ST-algorithm each candidate from stage I (and the corresponding static partitions) is evaluated. For a given time-period and partition only some of the robots' configurations do not collide with the partition; these configurations are guaranteed to be collision-free w.r.t. the configurations of the other robots. Stage II thus has the purpose to optimize the robots' task sequences given that the corresponding motions respect the given partitions. In this stage it is determined in which time-period and by which robot

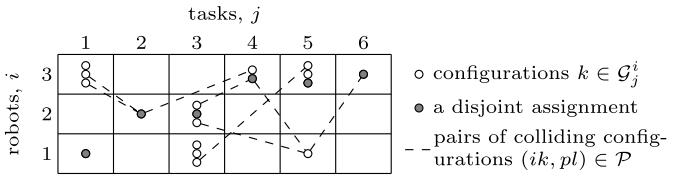


Fig. 3. Illustration of the discrete sets of robot configurations available for the tasks, an assignment that only requires a single time-period, and edges representing pairwise colliding configurations.

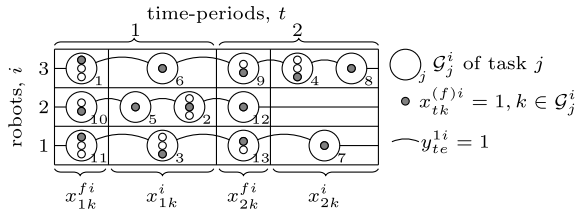


Fig. 4. Illustration of a solution to the stage I problem (1). The used sets G_j^i are visualized, but several robots can reach most tasks. The colliding configurations are not visualized (cf. Figure 3); a solution has no colliding configurations within a time-period and the start of the next time-period.

a task should be performed, w.r.t. the partitions. Moreover, in contrast to the stage I problem, each robot's task sequence is modelled as a path between specific configurations at tasks (instead of only specifying the tasks) making the resulting makespan more accurate. This is possible since the partitions prohibit robots from colliding and since only a subset of the robots' task-configurations are accepted by the partitions. Note that in stage II the fastest motion for a robot has been decoupled from the other robots' motions, thus motivating an accurate modelling of the makespan. The details of stage II are described in Section III-D.

A. Notation and Initialization

The initial step in the ST-algorithm (block A in Figure 2) is to detect collisions between pairs of robots' configurations at different tasks. We introduce $\mathcal{I}$ as the set of robots, $\mathcal{J}$ as the set of tasks, and $\mathcal{G}_j^i$ as the set of configurations available for robot $i \in \mathcal{I}$ at task $j \in \mathcal{J}$. In addition, the set $\mathcal{K}^i := \cup_{j\in\mathcal{J}} \mathcal{G}_j^i$, denotes the set of configurations associated with robot $i \in \mathcal{I}$.

The output of the initial step in the ST-algorithm is the set $\mathcal{P}$ of all pairwise colliding configurations (ik, rl) denoting robot i in configuration $k \in \mathcal{K}^i$ colliding with robot r in configuration $l \in \mathcal{K}^r$, see Figure 3. Moreover, let $\mathcal{C} \in \mathscr{C}$ be a set of cliques covering all edges in $\mathcal{P}$ (see Section III-B1 for details). Thanks to efficient collision queries [25], the set $\mathcal{P}$ of colliding configurations can be constructed in reasonable time, assuming that the number of configurations $|\mathcal{K}^i|$ for each robot $i \in \mathcal{I}$ is not too large; in our industrial instances these are in the order of thousands. Note that building the sets $\mathcal{K}^i$ is often a complication all of its own (see [26, §4.1]) and we note that apart from the initialization our method is quite insensitive to the size of $\mathcal{K}^i$, see also Section III-D3.

To introduce the main decision variables, we first let $\mathcal{T} = \{1, \dots, T\}$ to be the set of time-periods, each corresponding to a partition. Then let the decision variable $x_{tk}^i \in \{0, 1\}$ equals to one if robot $i \in \mathcal{I}$ uses configuration $k \in \mathcal{K}^i$ in time-period $t \in \mathcal{T}$, except for the *first* configuration in time-period $t \in \mathcal{T}$, which is represented by the variable $x_{tk}^{\mathrm{fi}} \in \{0, 1\}$. Let p_k^i denote the processing time of performing a task by robot i in configuration $k \in \mathcal{K}$. Moreover, we let z_t^i denote the time at which robot $i \in \mathcal{I}$ is finished with its work assigned in time-period $t \in \mathcal{T} \setminus \{T\}$ and can start the work in the next time-period. Finally, we let z denote the total makespan.

B. Generating Candidate Partitions (Stage I)

In order to state the partitioning problem (block B in Figure 2) as a mixed integer linear optimization problem (MILP), we need to introduce the path variables associated with stage I. Let $y_{te}^{1i} \in \{0, 1\}$ represent the decision whether to use the undirected edge $e = (j, q) \in \mathcal{E} \subset \mathcal{J} \times \mathcal{J}$, i.e., that robot $i \in \mathcal{I}$ moves from task j to task q in time-period t. To simplify the notation, interpret $[t + 1]$ as $(t \mod T) + 1$, e.g., the time-period following T is 1. Also, let $\delta(\mathcal{S}) := \{(j, q) \in \mathcal{E} \mid j \in \mathcal{S}, q \notin \mathcal{S}\}$ be the set edges to neighbour tasks of $\mathcal{S} \subset \mathcal{J}$.

The time c_e^{1i} of using edge y_{te}^{1i} is chosen as the minimum motion time for the robot between any two configurations in the respective two tasks of e. This time is a valid lower bound for the more accurate time that would use path variables between configurations instead of between tasks, see Section III-B2 for further motivation of this simplification. At this stage, the robots' motions are planned w.r.t. the static environment, which is quite computationally expensive, hence the motion durations are bounded from below by using the straight path. These bounds are used until the corresponding motions are planned and their durations known, cf. [3].

The stage I problem (see Figure 4) is then to

$$\text{minimize } z \tag{1a}$$

$$\text{s.t.} \sum_{k\in\mathcal{K}^i} x_{tk}^{\mathrm{fi}} = 1, \qquad i \in \mathcal{I}, \quad t \in \mathcal{T}, \tag{1b}$$

$$\sum_{t\in\mathcal{T}} \sum_{i\in\mathcal{I}} \sum_{k\in\mathcal{G}_j^i} (x_{tk}^{\mathrm{fi}} + x_{tk}^i) = 1, \qquad j \in \mathcal{J}, \tag{1c}$$

$$\sum_{k\in\mathcal{K}^i} p_k^i (x_{tk}^{\mathrm{fi}} + x_{tk}^i) + \sum_{e\in\mathcal{E}} c_e^{1i} y_{te}^{1i} \le z_t^i - z_{t-1}^i, \quad t \in \mathcal{T}, i \in \mathcal{I}, \tag{1d}$$

$$z_{t-1}^i + \sum_{k\in\mathcal{K}^i} p_k^i x_{tk}^{\mathrm{fi}} \ge z_{t-1}^r, \qquad i, r \in \mathcal{I}, t \in \mathcal{T}, \tag{1e}$$

$$\sum_{(ik)\in\mathcal{C}} (x_{tk}^{\mathrm{fi}} + x_{tk}^i + x_{[t+1]k}^{\mathrm{fi}}) \le 1, \qquad \mathcal{C} \in \mathscr{C}, t \in \mathcal{T}, \tag{1f}$$

$$\sum_{k\in\mathcal{G}_j^i} (x_{tk}^{\mathrm{fi}} + 2x_{tk}^i + x_{[t+1]k}^{\mathrm{fi}}) = \sum_{e\in\delta(j)} y_{te}^{1i}, \qquad t \in \mathcal{T}, i \in \mathcal{I}, j \in \mathcal{J}, \tag{1g}$$

$$\sum_{k\in\mathcal{G}_j^i} 2x_{tk}^i - \sum_{q\in\mathcal{S}\setminus\{j\}} \sum_{k\in\mathcal{G}_q^i} (x_{tk}^{\mathrm{fi}} + x_{[t+1]k}^{\mathrm{fi}}) \le \sum_{e\in\delta(\mathcal{S})} y_{te}^{1i}, \qquad t \in \mathcal{T}, i \in \mathcal{I}, j \in \mathcal{S} \subset \mathcal{J}, \tag{1h}$$

$$z_0^i = 0, \ z_T^i \le z, \qquad i \in \mathcal{I}, \tag{1i}$$

$$x_{tk}^{\mathrm{fi}}, x_{tk}^i, y_{te}^{1i} \in \{0, 1\}, \qquad t \in \mathcal{T}, k \in \mathcal{K}^i, i \in \mathcal{I}, e \in \mathcal{E}. \tag{1j}$$

The objective function (1a) expresses the makespan minimization, (1d) express the duration of each robot in each time-period, and (1e) enforce waiting times between time-periods so that robot i must wait if it finishes its first task before robot r enters the same time-period. Constraints (1b) ensure that each robot has a first task in each time-period and (1c)

enforce that each task is assigned. The set packing constraints (1f) ensure that colliding configurations cannot be assigned to the same time-period or be first in the following time-period. Constraints (1g) connect the symmetric path variables with the assigned tasks, expressing that $y^{1i}_{t\cdot}$ should form a path for robot $i \in \mathcal{I}$ over its assigned tasks in time-period $t \in \mathcal{T}$ with ends in the first tasks of time-periods t and $t+1$. Constraints (1h) (inspired by [24]) eliminate subtours by enforcing that if a path visits a task in a set $\mathcal{S}$ of tasks, then the path needs to enter and leave this set, unless the set contains the ends of the path. As usual with subtour elimination constraints, these are generated as needed by the solver. Finally, note that in [24] there are many families of valid inequalities that could be adapted to be used in (1). However, our preliminary tests did not find any additional valid inequalities that substantially accelerated the solution process.

1) Generating the Clique Cover: The goal is to compute a clique cover [27] that covers the edges in $\mathcal{P}$ with maximal cliques of nodes (sets of mutually exclusive configurations), i.e., colliding configurations or configurations of the same task. To achieve this, the algorithm from [27] may be used, but we used a simple algorithm that first greedily combines uncovered edges into cliques and when all edges are covered by cliques, the cliques are made maximal by including covered edges.

Preliminary tests showed that the resulting cover was better than the trivial cover with a pair of nodes for each edge. Moreover, we tested some other clique covers (e.g., [27]) and found no significant difference in solution time of model (1) for our instances. We suspect that this is due to the structure of the collision graph $\mathcal{P}$, in which two similar configurations are likely to collide with the same configurations of other robots. Moreover, it is often the robot's tool causing a collision; in that case all configurations of the same robot and task will have many colliding configurations in common.

2) Dominance Criterion: One main reason for using the path variables only between tasks is that a dominance criterion can be effectively applied, and for our industrial instances such criterion removes sufficiently many variables to make the model solvable in reasonable time. The key observation is that the only difference between two configurations $k, l \in \mathcal{G}^i_j$ for some robot $i \in \mathcal{I}$ and task $j \in \mathcal{J}$, are their processing times and their sets of colliding configurations, i.e., the set $\delta^{\mathrm{c}}(ik) := \{rl \mid (ik, rl) \in \mathcal{P}\}$. Hence, k dominates configuration l (i.e., l can be removed) if it holds that

$$p^i_k \leq p^i_l \quad \text{and} \quad \delta^{\mathrm{c}}(ik) \subseteq \delta^{\mathrm{c}}(il). \tag{2}$$

This dominance criterion is applied iteratively until no more variables can be removed from $\mathcal{G}^i_j$.

For our instances the criterion (2) is advantageous since the processing times are usually all equal. Moreover, for the geometry of our industrial instances, the set of configurations for each robot and task is often reduced to a singleton, i.e., $|\mathcal{G}^i_j| \lessapprox 1$. In contrast, if the motion durations between configurations (not between tasks) are considered then a direct generalization of (2) does almost never apply, the reason for this being that the motion times vary a lot, especially between different so-called inverse configurations.

The efficiency of dominance criterion (2) allows for the original set of configurations to be large, and, if needed, to closely represent a continuous set such as robot tool rotations or deviations. Recall Section III-A.

3) Decide the Number of Time-Periods and Generate Multiple Candidates: In model (1) the number of time-periods is constant, however, since the model requires a first task for each robot and time-period, it is beneficial to limit the number of time-periods. Moreover, the duration of each time-period as computed by (1d) is an underestimate, the true motions and their durations depend on the configurations and on the motions of other robots. Hence, the optimal objective value of model (1) is only a lower bound on the makespan and hence it is beneficial to find several different solutions with low objective values.

To choose the parameter T, we suggest solving model (1) for multiple values of T, e.g., increasing from $T = 1$ until the objective value grows too large. For our instances this often occurs for relatively few time-periods $T \lesssim 6$. To speed up the process, we suggest limiting the solution time, and then choosing the most promising value of T for a longer run.

Note that model (1) is quite pessimistic, in the sense that a robot may not continue in the next time-period until all other robots have reached the same time-period, cf. constraints (1e). Relaxing these constraints instead yields an optimistic model where the robots never wait, which may be feasible and not cause collisions. It is not clear which of these variants that yields the best solution to the complete load balancing problem when the waiting times are optimized, see Section III-D2. To resolve this, we suggest solving model (1) with and without the waiting times constraints (1e).

Moreover, we suggest solving the model multiple times, as indicated in Figure 2. In each iteration, we make the model more accurate by planning the motions of the robots for the decided tasks sequences and updating the corresponding durations. However, as the durations in (1d) are the minimum between all pairs of configurations between two tasks, this does not perturb the instance much. Thus, to exclude previous solutions, we suggest including the logical constraint

$$\sum_{t\in\mathcal{T}}\sum_{j\in\mathcal{J}}\sum_{k\in\mathcal{G}^{i^*(j)}_j} x^{fi}_{tk} + x^i_{tk} \leq |\mathcal{J}| - 1, \tag{3}$$

where $i^*(j)$ denotes the robot previously assigned to task j.

Constraint (3) is included in model (1) in all future iterations with equal numbers of time-periods. The constraint ensures that at least one task is assigned to another robot. Preliminary results showed that this constraint yields sufficiently diversified solutions, whereas excluding specific configurations or allowing a task to move to another time-period did not have much effect on the overall solution. Moreover, to limit the overall computation time the set of candidates need to be small, and for our instances we use roughly twenty candidate stage I solutions. We observed that the objective value of (1) does not increase much by iteratively including constraints of the type (3). Note that another (less restrictive) constraint can be used if more candidate solutions can be analysed in reasonable time.

4) Special Case $T = 1$: If the problem has a single time-period ($T = 1$), model (1) is substantially reduced to a model similar to the one presented in [5]. Since all configurations then need to be collision-free, any configuration that collides with all configurations of another task can then be removed.

Moreover, since there can be no waiting times the optimistic and pessimistic versions of Section III-B3 coincide. Further, the $\boldsymbol{y}$-variables will model a tour instead of a path, and the first task (x^{fi}_{tk}) of a time-period loses its meaning and only the assignment variables x^i_{tk} are needed; cf. [5].

5) SAT Model for Stage I: In order to generate more candidates, we utilize a crude model that is cheaper to solve and can be represented as a classic SAT (satisfiability) problem. The SAT model is derived from the two assumptions: (i) the processing times are much longer than the motion durations, hence sequencing becomes redundant; (ii) the processing times are all equal, hence the makespan objective reduces to assigning equally many tasks to each robot. Consequently, we enforce that each robot can perform maximum L tasks, evenly distributed over time-periods $t \in \mathcal{T} \setminus \{T\}$, and the remainder in time-period T. Letting L_t denote the number of tasks allowed in time-period $t \in \mathcal{T}$, we set $L_t := \lceil L/T \rceil$ for $t \in \mathcal{T} \setminus \{T\}$. For each value of T, L is increased from $\lceil |\mathcal{J}|/|\mathcal{R}| \rceil$ until the problem is satisfiable.

Using these assumptions, model (1) can be represented as a pure SAT problem with the following clauses:

$$\bigwedge_{i\in\mathcal{I}} \bigwedge_{t\in\mathcal{T}} \bigvee_{k\in\mathcal{K}^i} x^{\mathrm{fi}}_{tk}, \tag{4a}$$

$$\bigwedge_{j\in\mathcal{J}} \bigvee_{t\in\mathcal{T}} \bigvee_{i\in\mathcal{I}} \bigvee_{k\in\mathcal{G}^i_j} (x^{\mathrm{fi}}_{tk} \vee x^i_{tk}), \tag{4b}$$

$$\bigwedge_{t\in\mathcal{T}} \bigwedge_{i\in\mathcal{I}} \mathrm{Card}^{L_t-1}\left(\sum_{k\in\mathcal{K}^i} x^i_{tk}\right), \tag{4c}$$

$$\bigwedge_{t\in\mathcal{T}} \bigwedge_{\mathcal{C}\in\mathscr{C}} \mathrm{Card}^{1}\left(\sum_{(ik)\in\mathcal{C}} x^i_{tk} + x^{\mathrm{fi}}_{tk} + x^{\mathrm{fi}}_{[t+1]k}\right). \tag{4d}$$

The operator Card^n transforms a cardinality constraint (i.e., at most n elements can be true) into SAT clauses using a maximum cardinality network, see [28]. Note that Card^1 allows at most one element to be true, which has a more efficient SAT formulation (e.g., [29]). However, as the set of cliques $\mathscr{C}$ is generated from the set of colliding configurations $\mathcal{P}$, we found that a simple pairwise formulation was the most efficient for the SAT solver. For compactness of presentation we choose to present (4d), to avoid having a clause for all combinations of x^i_{tk}, x^{fi}_{tk}, and $x^{\mathrm{fi}}_{[t+1]k}$, e.g., $\neg x^i_{tk} \vee \neg x^r_{tl}$ for $(ik, rl) \in \mathcal{P}$. Note also that the constraint (3) that prohibits previous solutions can also be efficiently translated to a SAT constraint, which is done in [30, §3.5.2].

Formulation (4) is efficient enough to prove (in)feasibility of our industrial instances using the popular SAT solver called MiniSat [31]. Hence, by enumeration, the minimum L and the corresponding T can be acquired. Thus, in addition to the two variants of (1) we use the solution of (4) to produce candidate partitions to evaluate in the stage II of the ST-algorithm.

C. Compute and Analyse Partitions

Between stage I and stage II in the ST-algorithm (cf. Figure 2) for each candidate solution and time-period a generalized Voronoi diagram is built (as in [5]). This is a valid partition of the workspace (separating surfaces) such that each robot can use its assigned configurations in the corresponding time-period while remaining in its zone of the partition.

With these partitions in mind, the notation for the stage II problem is now introduced. For a given partition and corresponding time-period $t \in \mathcal{T}$ each robot can only use the subset $\bar{\mathcal{G}}^i_{tj} \subseteq \mathcal{G}^i_j$ of its configurations that are collision-free with respect to the partition. Moreover, the configuration of the first task in each time-period must be collision-free with respect to the partitions of both the previous and current time-periods, i.e., $\bar{\mathcal{G}}^{\mathrm{fi}}_{tj} := \bar{\mathcal{G}}^i_{tj} \cap \bar{\mathcal{G}}^i_{[t-1]j}$ for $t \in \mathcal{T}$.

By construction, we have that the solution to the stage I problem (1) uses configurations from the sets $\bar{\mathcal{G}}^i_{tj}$ and $\bar{\mathcal{G}}^{\mathrm{fi}}_{tj}$. Moreover, by only considering the configurations in these sets the packing constraint (1f) becomes redundant. As a consequence of these simplifications, the task sequences can be modelled in more detail. In particular, we introduce the path variables y^{2i}_{te}, where $e = (kl) \in \mathcal{E}^i_t \subseteq \mathcal{K}^i \times \mathcal{K}^i$ denote edges between pairs of configurations. The set of undirected edges $\mathcal{E}^i_t$ is the complete graph with the node set of configurations $\bar{\mathcal{K}}^i_t := \cup_{j\in J} \bar{\mathcal{G}}^i_{tj}$.

D. Evaluating Candidate Partitions (Stage II)

The main purpose of introducing the partitions is to decouple the motion planning problem among the robots. Within each time-period the robots are separated by the partitioning surface; hence, planning each motion with respect to the environment and the partitioning surface is sufficient to guarantee that the robots will not collide with each other.

Even though the path can now be planned for each robot separately, it is still computationally too expensive to plan every motion to find its duration. To resolve this, the procedure from stage I is applied, i.e., the duration along each edge $e = (kl) \in \mathcal{E}^i_t$ is initially bounded from below. This duration bound, denoted c^{2i}_{te}, is initialized from the stage I procedure as either the duration of the (possible colliding) straight motion between configurations k and l, or the duration of a previously planned motion that is collision-free w.r.t. the environment but might collide with the partition.

To find a sequence of configurations (block D in Figure 2) we consider the stage II problem to

$$\text{minimize } z \tag{5a}$$

$$\text{s.t. } \sum_{k\in\bar{\mathcal{K}}^i_t} x^{\mathrm{fi}}_{tk} = 1, \qquad i \in \mathcal{I}, t \in \mathcal{T}, \tag{5b}$$

$$\sum_{t\in\mathcal{T}} \sum_{i\in\mathcal{I}} \sum_{k\in\bar{\mathcal{G}}^i_{tj}} (x^{\mathrm{fi}}_{tk} + x^i_{tk}) = 1, \qquad j \in \mathcal{J}, \tag{5c}$$

$$\sum_{k\in\bar{\mathcal{K}}^i_t} p^i_k (x^{\mathrm{fi}}_{tk} + x^i_{tk}) + \sum_{e\in\mathcal{E}^i_t} c^{2i}_{te} y^{2i}_{te} \le z^i_t - z^i_{t-1}, \qquad t \in \mathcal{T}, i \in \mathcal{I}, \tag{5d}$$

$$z^i_{t-1} + \sum_{k\in\bar{\mathcal{K}}^i_t} p^i_k x^{\mathrm{fi}}_{tk} \ge z^r_{t-1}, \qquad i, r \in \mathcal{I}, t \in \mathcal{T}, \tag{5e}$$

$$x^{\mathrm{fi}}_{tk} + 2x^i_{tk} + x^{\mathrm{fi}}_{[t+1]k} = \sum_{e\in\delta(k)} y^{2i}_{te}, \qquad k \in \bar{\mathcal{K}}^i_t, t \in \mathcal{T}, i \in \mathcal{I}, \tag{5f}$$

$$\sum_{k\in\bar{\mathcal{G}}^i_{tj}} 2x^i_{tk} - \sum_{q\in\mathcal{S}\setminus\{j\}} \sum_{k\in\bar{\mathcal{G}}^i_{tj}} (x^{\mathrm{fi}}_{tk} + x^{\mathrm{fi}}_{[t+1]k}) \le \sum_{e\in\delta(\mathcal{S})} y^{2i}_{te}, \qquad t \in \mathcal{T}, i \in \mathcal{I}, j \in \mathcal{S} \subset J, \tag{5g}$$

$$z^i_0 = 0,\ z^i_T \le z, \qquad i \in \mathcal{I}, \tag{5h}$$

$$x^{\mathrm{fi}}_{tk} = 0, \qquad k \in \bar{\mathcal{K}}^i_t \setminus \bar{\mathcal{K}}^i_{[t-1]}, t \in \mathcal{T}, i \in \mathcal{I}, \quad \text{(5i)}$$

$$x^{\mathrm{fi}}_{tk}, x^i_{tk}, y^{2i}_{te} \in \{0,1\}, \quad k \in \bar{\mathcal{K}}^i_t, e \in \mathcal{E}^i_t, t \in \mathcal{T}, i \in \mathcal{I}. \quad \text{(5j)}$$

Model (5) differs from the stage I model (1) in that only the configurations allowed by the partitions ($\bar{\mathcal{G}}^i_{tj}$) can be used, the redundant set packing constraints (1f) are removed, and the path variables (y^{2i}_{te}) are between configurations rather than tasks (as in Figure 4). The last, difference is that different time-periods have different durations for the motions, which is a consequence of the motions respecting different partitions.

The solution of (5) is a sequence of configurations in each time-period. In between these configurations the robots' motions need to be planned with respect to the corresponding partitions. The resulting durations can be used to compute the actual makespan and to improve the lower bound provided by the model by updating the corresponding values of c^{2i}_{te}. In contrast to stage I, this is the only modification made to the stage II problem in each iteration. The computed makespan and the lower bound are used to determine if the current solution is the current best solution (i.e., the incumbent) and if the model cannot provide a new incumbent solution, respectively; cf. Figure 2.

The idea of iteratively solving the problem, planning the motions, and updating the motion durations originates from [3]. Indeed, for the special case of $T = 1$, the two approaches are equivalent (cf. [5]). For $T > 1$ we need to consider the sequencing problem (5), in which different configurations can be used in different time-periods.

1) Local Refinement of Stage I Solutions: The main change that allows for using the path variables y^{2i}_{te} (between pairs of configurations) in the stage II model is the smaller number of allowed configurations in each time-period. Using this observation, the quality of the stage I solutions can be improved if the number of allowed configurations is limited. We suggest applying a local refinement of each stage I solution, so that the resulting partitions more accurately account for the durations between configurations.

Given a solution to the stage I problem, define the set of available configurations to be $\bar{\mathcal{G}}^{i^*(j)}_{t^*(j)j} = \mathcal{G}^{i^*(j)}_j$ and $\bar{\mathcal{G}}^i_{tj} = \emptyset$ for $i \neq i^*(j)$ and $t \neq t^*(j)$, where $i^*(j)$ denotes the robot performing task j and $t^*(j)$ denotes the time-period the task j is assigned to. Then solve the stage II problem with the following modifications: relax the constraint (5i) and enforce the set packing constraints (1f). This can be seen as a local refinement since it allows the sequence of tasks to be rearranged within each time-period and (more importantly) it allows for better choices of configurations. For our type of robots, this refinement prevents unnecessary changes between different inverse configurations. At many of these configurations the robot occupies roughly the same volume even though some revolute joints are rotated, e.g., 180 degrees. For such configurations, the sets of conflicting configurations are approximately equal, but their locations in the configuration space are far apart.

The effect of using this local refinement is that new configurations, allowing for a lower makespan, will be chosen to generate the partitions for the stage II problem. The local refinement step reduces the risk of partitions hindering the usage of better configurations. Note that this is a step towards solving a more accurate version of the stage I problem (1) in which durations between configurations are accounted for.

2) Coordination Improvement: The models (1) and (5) are a bit pessimistic since they require the robot to remain in certain fixed disjoint partitions over a whole time-period. However, the true criteria are that the robots may not collide, i.e., at every given time point, their volumes should be disjoint, and a valid partition of the workspace should exist. An infinitely short time-period is, however, not feasible in the models (1) and (5). Moreover, the computational effort needed to prove a certain optimality gap typically increases with increasing numbers of time-periods. However, this issue can be partially resolved in a post-processing step, in which each robot uses its assigned task sequence and the configuration path of its planned motion. Instead, the velocities (and wait-times) are re-optimized to minimize the makespan subject to robot–robot collisions. This can be done efficiently using the algorithm suggested in [16]. This post-processing step can be seen as local search in a larger space than allowed by the models (1) and (5) and when solving our instances, it often results in a slightly improved makespan.

The coordination ensures that in each time point a valid partition of the workspace exists, i.e., each robot is assigned to a private volume. However, these time-varying surfaces are not built explicitly, but they are implicitly represented by the fact that the robots are always free from collision. Hence, unlike the partitions used (in the ST-algorithm in Figure 2) as collision objects in motion planning queries, these implicit partitions are only used to certify that the solution is safe from collisions.

3) Downsampling the Number of Alternatives: As noted in Section III-A, it can be useful to consider huge sets $\mathcal{K}^i$, which is problematic in the stage II model (5) since the number of sequence variables grows quadratically with the number of configurations. However, once the partitions have been computed it is possible to downsample the set of configurations for each robot independently, as in [32] and [26, §4.1]. Downsampling is problematic to do before stage I, since then key configurations (that do not collide with many other configurations) may be removed, which in some industrial cases leads to long waiting times due to coordination.

Note that the size of $\mathcal{K}^i$ is not an issue in the stage I model (1), since the sequence variables $\boldsymbol{y}^{1i}_{te}$ in that stage depend on the size of $\mathcal{J}$. Moreover, the dominance criterion (2) reduces the size of sets $\mathcal{K}^i$. However, to simplify the presentation and comparison we do not apply a downsampling procedure.

IV. Computational Results

The computations presented in this section were carried out on a computer with an AMD Ryzen 9 3900X 12-Core 3.79 GHz and 32GB of RAM. Concurrency is exploited when solving the MILP models and when planning motions. However, all presented computation times, including time-limits are so-called wall-clock times. The main focus of this study is not on computation time but rather on the potential of the ST-algorithm (Figure 2). Future improvements are suggested in Section V-A.

To evaluate and motivate the ST-algorithm suggested in Section III, we compare it with the state-of-the-art software IPS [6], in Section IV-A on artificial instances (for some of which the shortest path assumption is clearly violated), in Section IV-B on industrial instances (including the instance that initiated the research developed in this article).

To solve the MILP models we use the Gurobi [33] branch-and-cut optimizer using the lazy-callback routine that allows for verifying (and adding) subtour constraints on integer solutions. To solve the SAT models we use the MiniSat solver [31]. The remainder of the ST-algorithm, i.e., the dominance criterion and interfaces to the above solvers, are implemented in C++. For the artificial instances we used the programming language Julia [34] to plan the motions, to build the partitions, as the interface to the ST-algorithm, and to visualize the solutions. For the industrial instances we used the software IPS to plan the motions, to build the partitions, and to visualize the solutions. We used the scripting language LUA [35] as interface between the ST-algorithm and IPS.

The MILP models are not solved to optimality since the models are themselves approximations, and we believe that spending a lot of computational effort in proving optimality is not worth the effort. To this end we use a combination of two simple termination criteria for all MILP models: (i) a 10-minute time-limit and (ii) 1% optimality gap. Preliminary tests showed that after 10 minutes (on our largest instances), the objective value did no longer progress (roughly the same value as after one hour). The 1% optimality gap is often but not always achieved (e.g., for the large industrial instances when evaluating many time-periods). Moreover, we limit the number of candidate solutions generated in stage I to twenty and the maximum number of stage II iterations is ten per candidate solution.

The resulting run times varied from a *few hours* to nearly *a day* among the instances presented below. The majority of computational time is in stage II, and roughly 10% in stage I. In both stages it is the solution of the MILP models that spend nearly all the computational time. The SP-algorithm (implemented in IPS) converges after roughly *one hour* even for the large instances.

A. Artificial Instances

To illustrate a scenario in which the shortest path assumption is unsuitable, we consider a two-dimensional workspace: robots having two revolute joints each and straight links of equal length and radius, and tasks being points in the plane. For each task a robot can reach (and which is not located exactly at the robot's base or at maximum reach), the robot will have two configurations. Moreover, each robot $i \in \mathcal{I}$ has a corresponding home task $i \in \mathcal{J}$ and multiple corresponding configurations $k \in \mathcal{G}_i^i$, the home task has to be the robots first and last task in the solution and the processing time is zero, i.e., $p_k^i = 0, k \in \mathcal{G}_i^i, i \in \mathcal{I}$, whereas for all other tasks it is two seconds, i.e., $p_k^i = 2, k \in \mathcal{G}_j^i, j \in \mathcal{J} \setminus \mathcal{I}, i \in \mathcal{I}$.

In this setting, we first consider a small instance with six tasks; Figure 5 shows (a) the solution found by IPS (using the shortest path assumption), and (b)–(d) the best solutions found by our suggested algorithm for $T = 1, 2, 3$. Figure 5a reveals

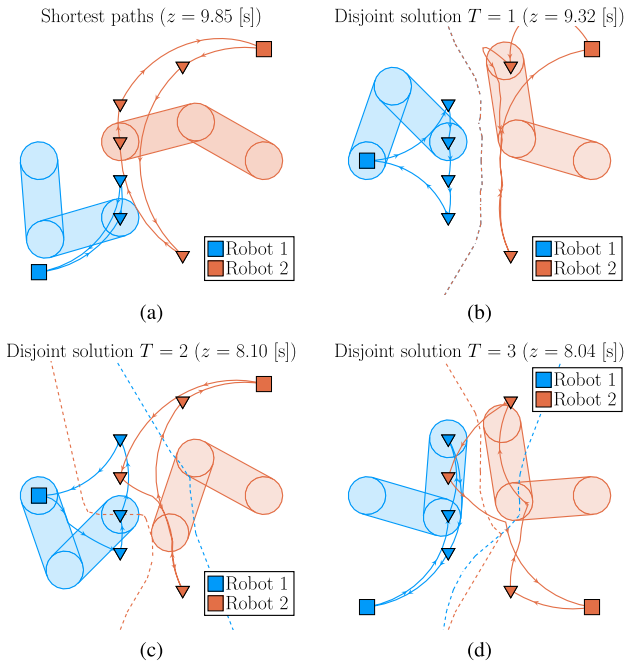


Fig. 5. Solutions of a small instance with six tasks (triangles) plus two home tasks (squares), snapshots of the robots (filled), traces of the robots' tool centres, and the partitions' boundaries (dashed lines) that the corresponding robot's motion is planned with respect to.

that the shortest path assumption is quite bad: indeed, the two right-most tasks (excluding home tasks) can only be reached by the right robot, moreover, the straight path (in the joint space) between these two tasks collides with the left robot at every other task. Thus, the right robot also performs some additional tasks on the way in order to avoid waiting.

Figure 5b illustrates the solution with a single time-period, i.e., the two robots are partitioned into disjoint zones. The resulting solution is slightly improved from that in (a), since the right-most robot performs two tasks and takes a detour instead of waiting for the left-most robot.

Figure 5c shows the solution with two time-periods. This solution is a bit harder to visualize since the coordination improvement step of Section III-D2 has been applied, as a result, the two robots are sometimes in different time-periods. Each robot has a corresponding partition that guides the robot's current motion (most apparent for the right robot). Figure 5d illustrates the solution for three time-periods, which also has the lowest makespan. However, increasing the number of time-periods is not always beneficial (since each robot performs at least one task in each period); e.g., $T = 4$ resulted in a makespan (z) of 8.64 s.

For the instance illustrated in Figure 5 detours are obviously beneficial, however, does this occur on larger instances? Or can other sequences resolve the collisions? To see if this is common, we generated ten instances with four robots and fifty randomly positioned tasks. One of them with the ST-algorithm's solution is illustrated in Figure 6. We found that detours are rarely generated and (for these instances) it is better to use the shortest paths. As compared with the SP-algorithm from IPS, the ST-algorithm produced (on average) 0.36% worse cycle times. However, an exact Wilcoxon signed rank test (cf. [36])

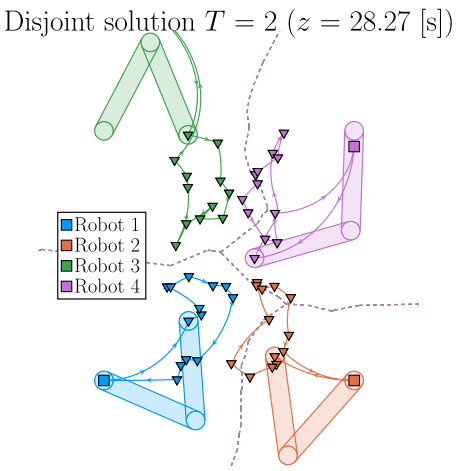


Fig. 6. Solutions of a large random instance with fifty tasks (triangles) plus four home tasks (squares), snapshots of the robots (filled), traces of the robots' tool centres, and the partitions' boundaries (dashed lines) that the corresponding robot's motion is planned with respect to.

TABLE I

NUMBER OF ROBOTS, TASKS, AND CONFIGURATIONS, AND THE IDENTICAL PROCESSING TIMES, FOR THE FOUR INDUSTRIAL INSTANCES

Instance	$\lvert\mathcal{I}\rvert$	$\lvert\mathcal{J}\rvert$	$\sum_{i\in\mathcal{I}}\lvert\mathcal{K}^i\rvert$	p_j
Spot	2	60	718	1.27
Spot+change	2	74	742	1.27
Inspection	4	50	789	1.0
Stud	4	47	1069	2.0

couldn't reject (with 95% confidence) that the logarithm of the cycle time ratios are zero-median distributed, i.e., the 0.36% are not significant and the solution qualities of the two algorithms cannot be distinguished by our results. Thus, we conclude that for most instances, the shortest path assumption should be used to yield a more tractable problem and that the solutions will be similar to those of the ST-algorithm. However, there are (large) instances for which there is a large gain in challenging the shortest path assumption, e.g., the industrial instances that are considered next.

B. Industrial Instances

We now turn to some industrial instances. First, we study two variations of the spot weld instance that motivated this research, recall Figure 1. Second, as for the artificial instances, we test the ST-algorithm on more common instances, including two other types of tasks: inspection and stud welding. Table I shows the number of robots, tasks, and task-configurations of these instances to indicate the problem sizes. As for the artificial instances each robot has a so-called home task in which its cycle starts and ends. For each instance, all tasks (except homes) have equal processing times, which is quite common. The processing times vary, however, among the instances. Note that larger processing times results in larger overlaps between time-periods in the ST-algorithm.

The spot-weld instance visualized in Figure 1 originally has a complication in the form of a tool change. We first ignore this complication by considering all tasks to be identical (Spot), and then we consider the real problem (Spot+change). The reason for the tool change is that a single task requires another spot-welding tool that the right-most robot can equip. We apply the simplification done in industry today, this single task is done first and then the robot will change tool (requiring 17.3 s) before starting to weld the other tasks. The tool change is represented by 14 tasks, each with a processing time of 1.236 s, the corresponding start of the sequence for this robot is then fixed.

TABLE II

THE MAKESPANS OF APPLYING AN SP-ALGORITHM (z^{SP}), THE ST-ALGORITHM (z^{ST}), AND THE DISJOINT ALGORITHM z^{D} FROM [5], AS WELL AS THEIR RELATIVE DIFFERENCE, FOR FOUR INDUSTRIAL INSTANCES WITH DIFFERENT KINDS OF TASKS

Instance	Spot	Spot+change	Inspection	Stud
z^{SP} [s]	70.87	80.36	23.61	33.79
z^{ST} [s]	50.53	62.83	24.62	33.84
z^{D} [s]	88.57	110.34	24.46	35.37
$\frac{z^{\mathrm{ST}}-z^{\mathrm{SP}}}{z^{\mathrm{SP}}}$ [%]	−28.70	−21.81	4.27	0.15
$\frac{z^{\mathrm{ST}}-z^{\mathrm{D}}}{z^{\mathrm{D}}}$ [%]	−42.95	−43.06	0.65	−4.33

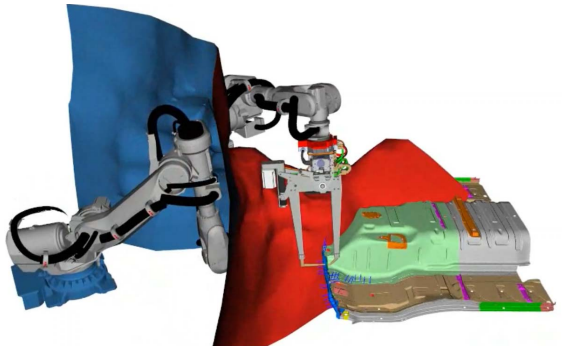
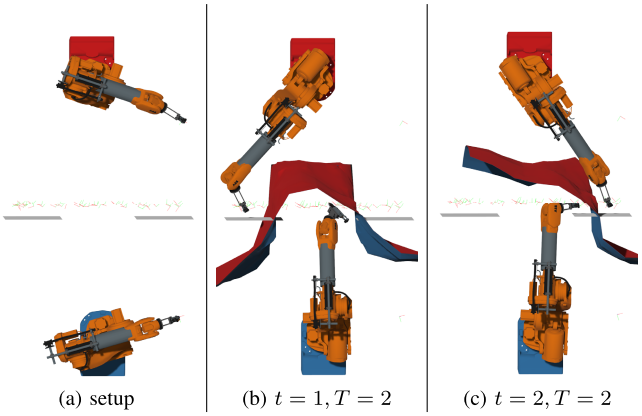


Fig. 7. A snapshot of the ST-algorithm's solution of the instance *Spot*, where the current partition allows for the robots to avoid collision.

Table II shows the results for the industrial instances, including the inspection and stud welding stations. For the spot weld instance (with or without tool change) that motivated this study, not assuming shortest paths is extremely beneficial. For this problem instance (cf. Figure 1) the tasks are positioned roughly on a single line. Further, due to the kinematics of the robots, at both ends of this line there are tasks that only the right robot can perform. Hence, if detours are not allowed, then the right robot (A) cannot access the leftmost tasks without waiting for the left robot (B) (regardless of task sequence). Thus, there are industrial instances in which it is useful to consider detours—a solution is illustrated in Figure 7.

A class of structured instances for which the ST-algorithm outperforms the SP-algorithm is exemplified in Figure 8. In this class the tasks are positioned roughly on a straight line (e.g., weld points on a workpiece's edge as for the spot weld station of Figure 1). Moreover, one of the two robots can only reach/perform tasks in the middle of this line. The tasks are also too densely positioned to allow for a disjoint solution ($T = 1$). For the instance in Figure 8 the SP-algorithm found a solution of $z = 68.5$ s. The ST-algorithm found a more balanced solution of $z = 55.4$ s, close to the lower bound (53.2) obtained by relaxing the robot–robot collisions requirements. This class of instances illustrates the use/power of the ST-algorithm and the benefits of allowing detours. It is likely that other classes of structured instances exist, for which the ST-algorithm outperforms the SP-algorithm.

Table II also shows the results for two conventional instances, where the tasks are more evenly spread over the workpiece. For these instances, the results are similar to those



(a) setup (b) $t = 1, T = 2$ (c) $t = 2, T = 2$

Fig. 8. A minimalistic problem instance where the tasks are positioned roughly on a line, the upper robot can reach all tasks, the lower can only reach tasks in the middle. (b) and (c) show snapshots of the ST-algorithm's solution, one in each time-period.

of the artificial instances: an SP-algorithm is often a better choice, and sometimes even a much better choice than the ST-algorithm. The fact that the SP-algorithm sometimes performs better has a fivefold explanation: (i) the stage I problem doesn't measure the accurate motion durations, (ii) the number of candidate solutions is quite limited, (iii) the model yields a pessimistic estimate of the makespan, (iv) the model is computationally demanding for a large T (number of time-periods), and (v) the models of both stages are not solved to optimality. Some of these aspects are discussed in Section V-A.

Lastly, Table II shows the general improvement of the generalization made in the ST-algorithm w.r.t. to the disjoint algorithm for load balancing of multi-robot station (see [5], called intersection-free therein) that searches for solutions with a single time-period $T = 1$. For the two versions of the spot weld instance, there is a very large gain, since the tasks are too densely positioned to allow for a disjoint partition; hence, the right-most robot does all the work. In contrast, for the stud weld instance, there exists a rather good disjoint solution, but the ST-algorithm finds a slightly better one. Finally, the inspection instance admits a completely disjoint solution, which is found by both the disjoint algorithm and the ST-algorithm.

V. Conclusion

In this study we devise a method for load balancing and coordination of a multi-robot station. Our main contribution is the solution of the problem in a general form, by relaxing the so-called shortest path assumption (i.e., that each robot uses its shortest collision-free motion between pairs of configurations regardless of the other robots). We suggest coordinating the robots by not only tuning their velocity, but also by allowing for detours to prevent robot–robot collisions. Our load balancing algorithm consists of two stages, each including an optimization model. Stage I uses a generalization of a model for a disjoint load balancing algorithm (where each robot receives its own disjoint zone). The generalization allows the solution to be disjoint during shorter time-periods. Stage II optimizes the sequences given the zones for each time-period.

We found that for certain instances our solutions have significantly shorter makespans than solutions from an algorithm using the shortest path assumption. Moreover, for both the artificial and industrial instances, we found that in terms of makespan, our algorithm is often comparable to the ones using the shortest path assumption. However, we believe that the computational effort will be lesser if the shortest path assumption is used, since fewer motions need to be planned. Consequently, we advise to first apply an SP-algorithm, and if the solution is poor (e.g., unbalanced or include long waiting times) apply our ST-algorithm.

A. Further Work

There are two major directions of further works, both aiming at solving the stage I and stage II models (or variants thereof) more efficiently. The first is to more rigorously evaluate different decomposition methods, valid inequalities, and heuristics to aid the solver (which doesn't necessarily need to be a MILP solver but, e.g., a constraint programming solver). Secondly, to disband the use of an exact solver for these models and develop a tailored heuristic (which typically perform well on related sequence problems).

Appendix Unsuccessful Modeling Ideas

Many aspects of our suggested modelling might seem unintuitive and that better options might be available. However, we preliminarily investigated several modelling approaches, which were clearly inferior. We briefly present these ideas. *Time-Indexed Models:* Instead of employing long time-periods spanning multiple tasks, we attempted to use a model with a fine time-indexing, where each task and motion spans multiple time-steps. In such a setting the number of time-steps is used to approximate the durations of tasks and motions. Hence, the makespan value becomes less accurate than that of our suggested model, but by increasing the number of time-steps this error becomes less severe. The argument for using this kind of model is that the robots only need to be separated in each given time point and not during an entire time-period.

First, we used a SAT formulation, with variables deciding in which time-step a robot starts a task and by which configuration. Constraints were generated to forbid the same robot to use the nearest time-step for other starts, depending on motion duration and task processing times. Similarly, colliding configurations were also constrained from being used. This idea is originally from [37] that models an airport's runway scheduling problem. They found it successful, however we did not, which is since our instances do not have the same kind of sparsity: a robot may process any task at any time, whereas an aeroplane needs to land in a narrow time-window.

Our second version of the time-indexed model we tested was based on using a MILP model, since much fewer linear constraints than SAT constraints are required to enforce sufficient time in-between any two consecutive tasks. However, the model was computationally intractable due to the constraints preventing robots from using colliding configurations simultaneously.

Our third version of the time-indexed model is a hybrid between our suggested model and one with finer time-steps. In this model each time-step corresponds to a task.

To forbid colliding configurations, constraints between adjacent time-steps are included, and robots are allowed to stay at tasks. The duration of a robot is, however, the sum of processing times, motion durations, and waiting times. Since the task order is decided by this model, no subtour constraints are needed. Despite this, we could not efficiently solve this model, mainly due to the constraints preventing colliding configurations, but also due to the sequence variables. We note that this model resembles our suggested model if we let each time-period comprise exactly one task. The difference is that our suggested model does not allow for starting the same task twice (to model waiting), which can be resolved by relaxing (1b) and (1c) to be inequality constraints. However, the resulting large number of time-periods makes the model hard to solve.

Our fourth version of the time-indexed model is similar to the first but is based on constraint programming. The configurations are represented by alternative intervals for the tasks with lengths corresponding to processing times. Disjunctive/no-overlap constraints with transition times between the intervals are used to model: (i) the motion duration and (ii) that two robots may not use colliding configurations too close in time (colliding motions were not accounted for as limitations of the formulation). We found that the IBM Ilog CP optimizer [38] often provided good solutions in reasonable time, but it was seldom able to prove optimality. Considering this a heuristic approach, which would be highly useful, however, we believe that a tailored heuristic would be more successful, see Section V-A. *Decomposition Methods:* Another family of methods tested was to solve models similar to (1) and (5) (with the coordination aspect disregarded, although the aim was to include it) using decompositions methods, such as Dantzig–Wolfe decomposition and (logic) Benders decomposition. The goal is to implicitly handle the sequencing variables by either Dantzig–Wolfe decomposition, which considers convex combinations of sequences of configurations (similar to [39]), rather than assignments of specific configurations, or by a Benders decomposition, in which the makespan objective is captured by Benders cuts generated by the Benders subproblem (that computes the best sequence of given configurations, inspired by [40]). Despite that both methods yield quite tractable problems, it seems that the vanilla branch-and-cut method using subtour elimination constraints is hard to outperform. We still think that a successful decomposition method can be constructed. Making the decomposition effective will, however, require the model to be slightly modified or some other property (that we have overlooked) be exploited. Moreover, the decomposition method needs to incorporate the coordination aspect (robots may not collide).

Regarding the Benders decomposition, we attempted to generate both classical Benders cuts by using a rich set of valid inequalities for the GTSP subproblems [24], and logical Benders cuts as in [40], as well as lifting the generated cuts as suggested by [41]. We also looked at [42] that surveys different methods for vehicle routing. One of them uses Benders decomposition on a so-called three-indexed formulation, containing binary variables for selecting the nodes of each vehicle. Regarding the Dantzig–Wolfe decomposition, we think that the sequencing is best contained in the subproblems; for the Benders decomposition approach this resulted in relatively small GTSP problems. However, since the Dantzig–Wolfe approach only prices and not constrains the subproblem, the subproblem become quite large and intractable.

REFERENCES

[1] R. Söderberg, K. Wärmefjord, J. S. Carlson, and L. Lindkvist, "Toward a digital twin for real-time geometry assurance in individualized production," *CIRP Ann.*, vol. 66, no. 1, pp. 137–140, 2017, doi: 10.1016/j.cirp.2017.04.038.

[2] D. Hömberg, C. Landry, M. Skutella, and W. A. Welz, "Automatic reconfiguration of robotic welding cells," in *Math for the Digital Factory*. Cham, Switzerland: Springer, 2017, pp. 183–203, doi: 10/dvjz.

[3] D. Spensieri, J. S. Carlson, F. Ekstedt, and R. Bohlin, "An iterative approach for collision free routing and scheduling in multirobot stations," *IEEE Trans. Autom. Sci. Eng.*, vol. 13, no. 2, pp. 950–962, Apr. 2016, doi: 10.1109/TASE.2015.2432746.

[4] 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 Robot. Automat. Lett.*, vol. 6, no. 2, pp. 1335–1342, Apr. 2021, doi: 10/fxqj.

[5] E. Åblad, D. Spensieri, R. Bohlin, and J. S. Carlson, "Intersection-free geometrical partitioning of multirobot stations for cycle time optimization," *IEEE Trans. Autom. Sci. Eng.*, vol. 15, no. 2, pp. 842–851, Oct. 2017, doi: 10/gc2bhc.

[6] (May 2021). *Industrial Path Solutions*. [Online]. Available: https://industrialpathsolutions.com/ips-robot-optimization

[7] O. Battaïa and A. Dolgui, "A taxonomy of line balancing problems and their solution approaches," *Int. J. Prod. Econ.*, vol. 142, no. 2, pp. 259–277, 2013, doi: 10.1016/j.ijpe.2012.10.020.

[8] T. C. Lopes, C. G. S. Sikora, R. G. Molina, D. Schibelbain, L. C. A. Rodrigues, and L. Magatão, "Balancing a robotic spot welding manufacturing line: An industrial case study," *Eur. J. Oper. Res.*, vol. 263, no. 3, pp. 1033–1048, Dec. 2017, doi: 10.1016/j.ejor.2017.06.001.

[9] J. Xin, C. Meng, F. Schulte, J. Peng, Y. Liu, and R. R. Negenborn, "A time-space network model for collision-free routing of planar motions in a multirobot station," *IEEE Trans. Ind. Informat.*, vol. 16, no. 10, pp. 6413–6422, Oct. 2020, doi: 10.1109/TII.2020.2968099.

[10] 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 Robot. Autom. Lett.*, vol. 4, no. 4, pp. 3844–3851, Oct. 2019, doi: 10/gmpvcq.

[11] S. M. LaValle, "Hierarchical methods," in *Planning Algorithms*, L. Cowles, Ed. Cambridge, U.K.: Cambridge Univ. Press, 2006, pp. 210–212, doi: 10.1017/CBO9780511546877.

[12] R. Bohlin and L. E. Kavraki, "Path planning using lazy PRM," in *Proc. Millennium Conf. IEEE Int. Conf. Robot. Automat. Symposia (ICRA)*, San Francisco, CA, USA, vol. 1, Apr. 2000, pp. 521–528, doi: 10.1109/ROBOT.2000.844107.

[13] S. Karaman and E. Frazzoli, "Optimal kinodynamic motion planning using incremental sampling-based methods," in *Proc. 49th IEEE Conf. Decis. Control (CDC)*, Atlanta, GA, USA, Dec. 2010, pp. 7681–7687, doi: 10.1109/CDC.2010.5717430.

[14] C. Landry, R. Henrion, D. Homberg, M. Skutella, and W. Welz, "Task assignment, sequencing and path-planning in robotic welding cells," in *Proc. 18th Int. Conf. Methods Models Autom. Robot. (MMAR)*, Miedzyzdroje, Poland, Aug. 2013, pp. 252–257, doi: 10.1109/MMAR.2013.6669915.

[15] 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," *Robot. Comput.-Integr. Manuf.*, vol. 44, pp. 97–116, Apr. 2017, doi: 10/ggh369.

[16] D. Spensieri, R. Bohlin, and J. S. Carlson, "Coordination of robot paths for cycle time minimization," in *Proc. IEEE Int. Conf. Autom. Sci. Eng. (CASE)*, Aug. 2013, pp. 522–527, doi: 10.1109/CoASE.2013.6654032.

[17] M. Erdmann and T. Lozano-Pérez, "On multiple moving objects," *Algorithmica*, vol. 2, nos. 1–4, pp. 477–521, Nov. 1987, doi: 10/dv9stf.

[18] M. Saha and P. Isto, "Multi-robot motion planning by incremental coordination," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Oct. 2006, pp. 5960–5963, doi: 10/d9rpwx.

[19] R. Shome, K. Solovey, A. Dobson, D. Halperin, and K. E. Bekris, "DRRT*: Scalable and informed asymptotically-optimal multi-robot motion planning," *Auto. Robots*, vol. 44, nos. 3–4, pp. 443–467, Mar. 2020, doi: 10/gn669q.

[20] H. Ha, J. Xu, and S. Song, "Learning a decentralized multi-arm motion planner," presented at the 4th Conf. Robot Learn. (CoRL), in Proceedings of Machine Learning Research, vol. 155, J. Kober, F. Ramos, and C. Tomlin, Eds. Cambridge MA, USA: Journal of Machine Learning Research, 2021, pp. 103–114. [Online]. Available: https://proceedings.mlr.press/v155/ha21a.html

[21] J. Chen et al., "Cooperative task and motion planning for multi-arm assembly systems," 2022, *arXiv:2203.02475*, doi: 10/hj2r.

[22] J. Edwards, E. Daniel, V. Pascucci, and C. Bajaj, "Approximating the generalized Voronoi diagram of closely spaced objects," *Comput. Graph. Forum*, vol. 34, no. 2, pp. 299–309, May 2015, doi: 10.1111/cgf.12561.

[23] 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," *Robot. Comput.-Integr. Manuf.*, vol. 73, Feb. 2022, Art. no. 102197, doi: 10/hjc3.

[24] M. Fischetti, J.-J. Salazar-González, and P. Toth, "The generalized traveling salesman and orienteering problems," in *The Traveling Salesman Problem and Its Variations*. Boston, MA, USA: Springer, 2007, pp. 609–662, doi: 10/c2zshz.

[25] E. Larsen, S. Gottschalk, M. C. Lin, and D. Manocha, "Fast distance queries with rectangular swept sphere volumes," in *Proc. Millennium Conf. IEEE Int. Conf. Robot. Automat. Symposia (ICRA)*, vol. 4, Apr. 2000, pp. 3719–3726, doi: 10/db2qkp.

[26] E. Åblad, "Mathematical modelling for load balancing and minimization of coordination losses in multirobot stations," M.S. thesis, Dept. Math. Sci., Chalmers Univ. Technol., Gothenburg, Sweden, 2020. [Online]. Available: https://research.chalmers.se/publication/515684

[27] E. Tomita, A. Tanaka, and H. Takahashi, "The worst-case time complexity for generating all maximal cliques and computational experiments," *Theor. Comput. Sci.*, vol. 363, no. 1, pp. 28–42, 2006, doi: 10.1016/j.tcs.2006.06.015.

[28] R. Asín, R. Nieuwenhuis, A. Oliveras, and E. Rodríguez-Carbonell, "Cardinality networks: A theoretical and empirical study," *Constraints*, vol. 16, no. 2, pp. 195–221, Apr. 2011, doi: 10/c2b6dr.

[29] J. Marques-Silva and I. Lynce, "Towards robust CNF encodings of cardinality constraints," in *Proc. Int. Conf. Princ. Pract. Constraint Program.*, C. Bessière, Ed. Berlin, Germany: Springer, 2007, pp. 483–497, doi: 10/fp5m43.

[30] E. Åblad, "Intersection-free load balancing for industrial robots," M.S. thesis, Dept. Math. Sci., Chalmers Univ. Technol., Gothenburg, Sweden, 2016.

[31] N. Eén and N. Sörensson, "An extensible SAT-solver," in *Theory and Applications of Satisfiability Testing*, E. Giunchiglia and A. Tacchella, Eds. Berlin, Germany: Springer, 2004, pp. 502–518, doi: 10/dqxrqm.

[32] J. S. Carlson, D. Spensieri, K. Wärmefjord, J. Segeborn, and R. Söderberg, "Minimizing dimensional variation and robot traveling time in welding stations," *Proc. CIRP*, vol. 23, pp. 77–82, Jan. 2014, doi: 10.1016/j.procir.2014.03.199.

[33] Gurobi Optimization, LLC. (2019). *Gurobi Optimizer Reference Manual*. [Online]. Available: http://www.gurobi.com

[34] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah, "Julia: A fresh approach to numerical computing," *SIAM Rev.*, vol. 59, no. 1, pp. 65–98, 2017, doi: 10/f9wkpj.

[35] R. Ierusalimschy, *Programming in Lua*, 4th ed. 2016.

[36] T. Berthold, "Heuristic algorithms in global MINLP solvers," Ph.D. dissertation, Dept. Math. Natural Sci., Technische Universität Berlin, Berlin, Germany, 2014.

[37] P. Avella, M. Boccia, C. Mannino, and I. Vasilyev, "Time-indexed formulations for the runway scheduling problem," *Transp. Sci.*, vol. 51, no. 4, pp. 1196–1209, Nov. 2017, doi: 10/gcmvnb.

[38] P. Laborie, J. Rogerie, P. Shaw, and P. Vilím, "IBM ILOG CP optimizer for scheduling," *Constraints*, vol. 23, no. 2, pp. 210–250, 2018, doi: 10/gdg3b4.

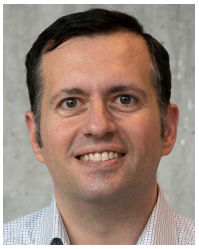
[39] W. A. Welz, "Robot tour planning with high determination costs: Routing under uncertainty," M.S. thesis, Technische Universität Berlin, Fakultät II-Mathematik und Naturwissenschaften, Berlin, Berlin, Germany, 2014, doi: 10/gf57.

[40] S. Riazi, C. Seatzu, O. Wigstrom, and B. Lennartson, "Benders/gossip methods for heterogeneous multi-vehicle routing problems," in *Proc. IEEE 18th Conf. Emerg. Technol. Factory Autom. (ETFA)*, Sep. 2013, pp. 1–6, doi: 10/dzpc.

[41] T. L. Magnanti and R. T. Wong, "Accelerating benders decomposition: Algorithmic enhancement and model selection criteria," *Oper. Res.*, vol. 29, no. 3, pp. 464–484, Jun. 1981, doi: 10/bs956m.

[42] G. Laporte, "The vehicle routing problem: An overview of exact and approximate algorithms," *Eur. J. Oper. Res.*, vol. 59, no. 3, pp. 345–358, 1992, doi: 10/b8mckf.

Edvin Åblad was born in 1991. He received the Ph.D. degree in applied mathematics and mathematical statistics in Chalmers in 2022. He is currently a Researcher at the Fraunhofer-Chalmers Research Centre for Industrial Mathematics, Geometry and Motion Planning Department. His bachelor's thesis treated optimization of power consumption in smart home appliances and his M.S. thesis was on intersection-free load balancing for industrial robots. His current research interests include computational geometry, multi-agent modeling, and optimization algorithms.



Domenico Spensieri was born in 1978. He received the M.Sc. degree in computer engineering from the University of Pisa with major in automation, robotics and control systems, in 2003, and the Ph.D. degree in product and production development from Chalmers University in 2021. After a brief period in industry, in 2004, he joined the Fraunhofer-Chalmers Centre (FCC) for Industrial Mathematics, Gothenburg, as a Software Engineer. He is currently an Applied Researcher at FCC, working on robotics, and modeling, simulation and optimization of multi-agent systems, including industrial robots and human–robot applications.

Robert Bohlin was born in 1972. He received the Ph.D. degree in mathematics on robot path planning from the Chalmers University of Technology in 2002. He is currently a Researcher and the Project Manager at the Fraunhofer-Chalmers Research Centre for Industrial Mathematics—FCC, Geometry and Motion Planning Department. His research interests include methods, algorithms and tools for virtual product realization and in particular automatic path planning, collision detection, simulation of flexible material, optimization, and kinematics.

Johan S. Carlson was born in 1972. He received the Ph.D. degree in mathematical statistics on how to reduce geometrical variation in assembled products from the Chalmers University of Technology in 2000.

He is currently the Director of the Fraunhofer-Chalmers Research Centre for Industrial Mathematics—FCC, and is heading the Department of Geometry and Motion Planning. His research interests include methods, algorithms, and tools for virtual product realization and in particular geometry simulation and assurance. He has over 20 years of experience with industrial development and implementation related to mathematics as a leading edge in virtual product realization and many of the results have been transferred into commercial software products and working procedures.

Ann-Brith Strömberg was born in 1961. She received the Ph.D. degree from Linköping University in 1997. She is currently a Professor of Mathematical Optimization with the Department of Mathematical Sciences, Chalmers University of Technology, and the University of Gothenburg. Her research interests include mathematical modeling and solution of optimization problems, including discrete, convex non-smooth, simulation-based, and multiobjective optimization. Much of her research is carried out in cooperation with academy and industry and includes scheduling of production and maintenance, load balancing, integration of variable electricity generation in the energy systems, and transport planning.