A Constraint Programming Approach to Simultaneous Task Allocation and Motion Scheduling for Industrial Dual-Arm Manipulation Tasks

Jan Kristof Behrens[1] Ralph Lange[1] Masoumeh Mansouri[2]

***Abstract*— Modern lightweight dual-arm robots bring the physical capabilities to quickly take over tasks at typical industrial workplaces designed for workers. In times of mass-customization, low setup times including the instructing/specifying of new tasks are crucial to stay competitive. We propose a constraint programming approach to simultaneous task allocation and motion scheduling for such industrial manipulation and assembly tasks. The proposed approach covers dual-arm and even multi-arm robots as well as connected systems or machines. The key concept are Ordered Visiting Constraints, a descriptive and extendable model to specify such tasks with their spatiotemporal requirements and task-specific combinatorial or ordering constraints. Our solver integrates such task models and robot motion models into constraint optimization problems and solves them efficiently using various heuristics to produce makespan-optimized robot programs. The proposed task description model is robot independent and thus can easily be deployed to other robotic platforms. Flexibility and portability of our proposed model is validated through several experiments on different simulated robot platforms. For large manipulation tasks with 200 objects, our solver implemented using Google's Operations Research tools and ROS requires less than a minute to compute usable plans.**

I. Introduction

Modern lightweight dual-arm robots such as the ABB YuMi or the KaWaDa Nextage are designed in the style of a human torso to be easily applicable in industrial workplaces designed for workers. These types of robots are an answer to the demand for flexible, cost-efficient production of customer-driven product variants and small lot sizes.

Such flexible production requires fast methods to specify new tasks for these robots. Classical teach-in by means of fixed poses and paths is not appropriate. With the capabilities of today's perception systems, which can detect and localize workpieces, boxes, and tools automatically in typical workplaces, and a formalized goal or high-level task specification, the manual teach-in may be replaced by automated planning – in principle. Optimal planning involves three aspects: (a) *task planning* of the necessary steps and actions to achieve the overall goal/task, (b) *scheduling* of these steps and actions, and (c) *motion planning* for each step and action.

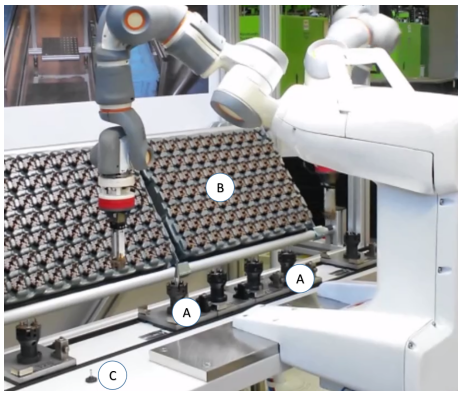

Fig. 1. Assembling of wiper motors with a dual-arm robot. The robot picks a tool from (C), places it on the shaft of the rotor of an electric motor in the workpiece holder (B), picks an electric interface, supplied in a container (B) and places it on the motor shaft.

Dual-arm manipulation further requires to decide about (d) the *allocation* of task steps and actions to the individual arms. Moreover, the complexity of scheduling and motion planning is increased heavily, due to the necessity to closely coordinate the manipulators to prevent self-collisions of the robot.

All four aspects – task planning, scheduling, allocation and motion planning – are closely interrelated. Ideally, to achieve optimal plans with regard to the makespan (production time) or similar objectives, they have to be considered in one coherent formalism and planning algorithm. In the last years, significant progress has been made to closely couple task planning with motion planning by passing feedback from motion planning to task planning (e.g., [7], [4], [14], [16], [5]), but research is still far from an ideal solution.

In many industrial use-cases, task planning is not required as the necessary steps and actions to process and assemble a workpiece are already given in digital form. That is, we already have an abstract plan, but with a number of unknowns and degrees of freedom in terms of the three aspects scheduling, allocation and motion planning. Computing an optimal, executable plan again requires to treat these aspects in a highly integrated and coherent manner, which we refer to as *simultaneous task allocation and motion scheduling* (STAAMS). An optimal plan depends not only on the motion of the manipulators

[1]Robert Bosch GmbH, Corporate Sector Research and Advance Engineering, Renningen, Germany, jan.behrens@de.bosch.com, ralph.lange@de.bosch.com

[2]Örebro University, Sweden, masoumeh.mansouri@oru.se

but also on the order in which a workpiece is assembled, the order in which the components are taken from boxes or conveyor belts, in which they are processed by other machines, etc. – in particular if connected systems or machines impose temporal constraints. The number of actions to be scheduled can be very high which results in big combinatorial complexity. Moreover, a suitable STAAMS solver has to consider different assignments of subtasks to arms, while taking the individual working ranges into account as well as task steps in which the arms have to cooperate.

In this paper, we propose a flexible model and solver for STAAMS for multi-arm robots in industrial use-cases. The proposed model and solver are based on constraint programming (CP) and constraint optimization, respectively. The key concept of the model to specify an abstract task plan is named *Ordered Visiting Constraints* (OVC). The OVC concept is developed out of the observation that many production steps can be described concisely by sequences of actions (e.g. drilling, picking, welding or joining) to perform with one of the robot arms at given locations, with temporal constraints and dependencies between them.

Our approach utilizes the fact that many industrial workplaces provide a controlled and unobstructed environment in which motion planning can be performed using precomputed roadmaps. For the robot motions, we propose a model of *time-scalable motion series* that can be directly integrated with constraint-based scheduling. By the concept of *Connection Variables*, we link the two models – the task model and the motion model – flexibly and seamlessly into an unified STAAMS problem model. The proposed solver tightly integrates value selection heuristics, scheduling and Luby restarts to compute an optimal plan for a given model instance.

An important feature of our approach is the portability of an abstract model to a different workplace layout and robot. This portability is achieved by clearly separated submodels for the robot's kinematics and for the mappings from symbolic locations to geometric poses in the workplace.

The remainder of this paper is organized as follows: We discuss related work in Sec. II before we present an analysis of typical industrial use-cases in Sec. III. Our main contributions, the STAAMS model with the OVC-based task model and the motion model as well as the corresponding planning system, are presented in the Sections IV and V, respectively. We show the scalability and portability of the proposed system and compare it to pure time-scaling in Sec. VI. The paper is concluded in Sec. VII.

II. Related Work

We can see the motion planning and scheduling subproblems of STAAMS as a multi-robot motion planning problem, where the objective is to find a path for each robot leading it from start to final configuration without colliding with other robots and obstacles. State-of-art approaches tackling this problem often do not address the task/goal allocation problem, i.e., goals/tasks are assumed to be given.

LaValle [15] formulates the motion scheduling problem in a joint configuration space derived from the Cartesian product of the configuration spaces of all robots. Another view to solving this problem is to employ a decoupled approach including prioritized planning, fixed-path planning, or fixed-roadmap planning: *Prioritized planning* assigns an order to the robots (arms) according to which their movements are planned [13]. In *fixed-path planning* – also referred to as *time-scaling* – every robot follows a given path and the planner only adjusts the timings along the paths to prevent collisions [18]. In *fixed-roadmap planning*, topological graphs for the configuration spaces of each robot are used to plan the paths and the timings together.

Our proposed method falls in the third category. A fixed-roadmap is particularly suitable for industrial settings, as the environments are not often subject to change. Kimmel et al. [10] employ a time-scaling approach via an incremental search over coordination diagrams to schedule two given sequences of pick-and-place tasks. Time-scaling problems can be modeled easily as a special-case with our STAAMS model. Our approach performs such time-scaling in its last step. In Sec. VI, we compare our simultaneous task allocation and motion scheduling approach with pure time-scaling using an experimental setup in the style of the one used by Kimmel et al.

Alatartsev et al. [1] present a survey about the task sequencing problem for industrial robots, where sources for execution variants are systematically identified for a given task specification (e.g., multiple inverse kinematic solutions, partial ordering) and optimized based on various cost functions. The survey, however, lacks the coverage for tasks that are applicable for multi-arm robots.

In the context of constraint optimization for task scheduling in industrial applications, Kolakowska et al. [12] plan paint strokes to account for paint quality with respect to the stroke order and direction. The scheduling considers only the movements to the beginning of the stroke, not the stroke itself, thus ignoring the dependency between the stroke motion and the order. This approach is not generalizable to multi-robot scenarios, as this dependency cannot be ignored due to robot-robot collisions.

In our approach, we employ CP to model the abstract task specification and the robot motion. Similarly, Ejenstam et al. [6] use CP to solve the problem of dual-arm manipulation planning and cell layout optimization. They, however, discretize the workspace in a coarse level, e.g., a node in their roadmap can resemble a larger workspace partition, and disallow two arms being in the same node to avoid collisions. Conversely, we create dense

roadmaps to enable the close coordination of arms, thus allow more parallel movements of arms.

Kurosu et al. [13] describe a decoupled MILP-based approach to solve a STAAMS, where the allocation and order of the tasks decided by a MILP solver are given to a motion planner to find collision-free motions. However, if the motion planner fails to find a feasible solution mainly due to the simplified motion and cost model used in the MILP formation, a new MILP solution will not be generated. This is not the case for us, as a single CP solver finds a mutually feasible solution for all sub-problems.

In our previous work [2], we hand-coded the full requirements of robot and workspace in the MiniZinc language. In this paper, we introduce a coherent formalism which allows to model the robot and workspace as well as an the abstract task plan and its invariants. We propose OVCs as task model primitives and time-scalable motion series as motion model primitives. Also, we provide automated procedures to create the data objects such as the roadmaps for motion planning, which may be created automatically from 3D sensor data and cover the full workspace of a real robot. Most important, our planning system uses carefully chosen variable ordering and value selection heuristics for efficient planning. For the implementation, we used the Google Operation Research tools [8], which (in contrast to MiniZinc solvers) allows fine-grained definition of the search strategy.

III. Use-case Analysis

In this section, we describe three typical industrial use-cases for dual-arm robots, followed by an analysis of characteristic properties and prevalent concepts. These properties and the concept of Ordered Visiting Actions serves as basis for the design of our STAAMS model in Section IV.

A. Use-Cases

Assembly of wiper motors. At the workstation depicted in Figure 1, the rotors are already inserted into workpiece holders (A) on a conveyor system, arriving in groups of five. The stators with the brushes and the electric interfaces are supplied in transport containers (B). Mounting a stator on a rotor requires to place a cone-shaped tool on the motor shaft temporarily. (C) marks the home position of this tool.

Sorting objects. The robot has to pick up colored objects from the table and place them depending on their color in one of two containers. All parts on the table are reachable by both arms. The containers are only reachable by either of the arms (see Fig. 4) so that an object's color defines the arm that has to pick this object. This use-case inspired by Kimmel et al. [10] will serve as a reference use-case for the evaluation.

Injection molding. Parts have to be taken from a source container and inserted into an injection molding machine. When the molding process is finished, the parts have to be taken from the machine and placed under a camera for visual inspection and hold into a fixture for an electrical check. The latter requires to press a button simultaneously to start the check. While the molding machine may process two parts simultaneously, the visual and electrical checks can only process one part at a time. Finally, the finished parts are placed in another container.

B. Analysis

These industrial use-cases illustrate several characteristic properties:

Controlled environment. Industrial workplaces provide by design a controlled and unobstructed environment. Therefore, we assume that all object locations and possible placements are known in advance, which allows for offline pre-calculation of motion roadmaps and collision tables. Furthermore, we may assume the absence of external interferences such as humans.

Unobstructed workspace. We assume that relevant objects never obstruct each other. This implies that there exists a collision-free subspace of the workspace that does not alter over time and allows to reach all relevant object locations with at least one robot arm. For example this applies to drilling, riveting, welding, glueing, and assembling of small parts. As a consequence, we do not require a complex scene graph (cf. [3]) that tracks geometric relations between all objects in the workspace.

Ordered Visiting Actions. Suitable plans for these use-cases may be specified as a series of motions (per arm) to visit relevant locations in the workspace. At each location, the manipulator may perform local actions such as turning in a screw or picking an object from a container, which – for our scheduling purposes – can be abstracted as constraint on the visiting duration at that location. While the overall order of actions may be changed, some actions like pick-and-place are subject to a partial ordering and are therefore considered as an entity. We refer to such entities as *Ordered Visiting Actions* (OVA) in the following. OVAs may be used to model many advanced tasks such as joining, welding, sorting, inspecting, drilling, and milling.

Temporal dependencies. Often, there are additional temporal dependencies between OVAs. For example, molding has to precede the visual and electrical checks in the molding use-case and in the motor assembly use-case the temporal dependencies are given by the assembly sequence for each motor. In the sorting use-case, there are no temporal dependencies between the OVAs per se. Yet, each arm can transport only a single object at a time, i.e. the gripper is a reservable resource, which requires to schedule the OVAs per arm.

Active components. Another important observation is that processing stations in the workspace may also take on different configurations, just as the robot arms. An example is the door of the molding machine in the third

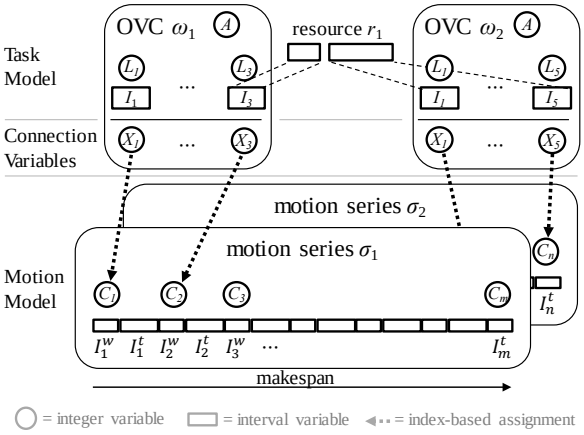

Fig. 2. Overview of CP-based STAAMS model

use-case. We refer to such stations and all active robot components together as *active components*.

The key concept of the following STAAMS model are Ordered Visiting Constraints (OVC), which are constraint-based blueprints for OVAs. A set of OVCs and the temporal dependencies specify the essence of the overall task, but leaves open many allocation and ordering decisions as they are subject to optimization by our solver. This task-centered approach allows to include the experts intuition about the solution as constraints or heuristics into the solving process.

IV. CP-Based STAAMS Model

In this section, we define our STAAMS model, which is based on constraint programming (CP). The core elements of such a program are variables and logical or arithmetic constraints between those variables.

Figure 2 shows the key concepts of our model. First, we explain the variables and constraints that model the robot motion. Subsequently, we present our task model with the key concept Ordered Visiting Constraints. For readability, we write constants or values as lowercase Latin letters, constraint variables as capital letters, and compounds of constraint variables as small Greek letters.

A. Motion Variables and Constraints (Motion Model)

As mentioned before, a plan for a robotic system is a timed motion series per arm. Here, we describe the variables making up the motion model and other auxiliary components.

Def 1. A **location** $l \in SO(3)$ denotes a pose of interest in the workspace (in particular possible object placements in containers and workpiece holders) in a common reference system. The set of all locations is $\mathbb{L}$.

Def 2. An **active component** A is a unit of a robotic system (e.g. a manipulator) or another machine under the control of our system. The set of all active components is $\mathbb{A}$.

Def 3. A **roadmap** $r = (\mathbb{C}, \mathbb{E})$ is a discretized, graph-based representation of the configuration space (cf. for example [9]) of an active component a. The nodes $\mathbb{C}$ are configurations of a, e.g. manipulator joint angles. In particular, there are one or more nodes for all locations that can be reached by a. For better connectivity, further traveling nodes are sampled in free space. The edges $\mathbb{E}$ are inserted between neighboring configurations, if a short motion plan for a collision free traversal between them exists. The edges have the minimal traveling time as weight. The roadmap thus yields information about paths between configurations and their lengths to be used in the CP (cf. Fig. 3). $\mathbb{C}$ may also include multiple nodes for the same configuration – e.g., to consider the different collision geometries of the arm depending on the gripper state.

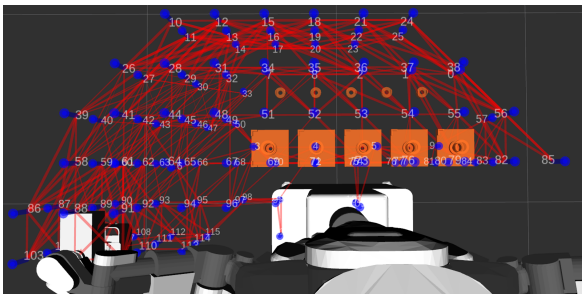

Fig. 3. A roadmap for the left arm of a KaWaDa Nextage robot.

Def 4. The **location mapping** λ holds for every location $l \in \mathbb{L}$ the set of roadmap nodes (i.e. robot configurations) that reach to l.

Def 5. A **collision table** $ct_{i,j}$ is defined for two roadmaps r_i and r_j and lists all pairs of conflicting nodes.

Def 6. An **interval** I is a tuple $(T_\mathrm{s}, T_\mathrm{e}, D)$ of CP variables with T_s as start time variable, T_e as end time variable and D as duration variable.

Def 7. A **motion series** σ is a sequence of m configuration variables and a sequence of $2m-1$ interval variables (cf. Fig. 2)

$$\sigma = ([C_1, \ldots, C_m], [I_1^\mathrm{w}, I_1^\mathrm{t}, I_2^\mathrm{w}, I_2^\mathrm{t}, \ldots, I_m^\mathrm{w}]).$$

An interval variable I_i^w models the time spent at configuration C_i whereas I_i^t denotes the traveling time between the configurations C_i and C_{i+1}. The set of all motion series – one for each active component – is named Σ.

During scheduling, the shortest traveling times between any two configurations is computed from the roadmap to set the duration of I_i^t once C_i and C_{i+1} are bound.

To prevent collisions between active components, we add a constraint for each pair of active components a_i and a_j. If a_i and a_j assume configurations that are in conflict according to $ct_{i,j}$, this constraint requires the corresponding intervals to be temporally disjunctive.

With the defined variables, we can describe a time-scalable motion series for each active component. In the next subsection, we introduce our task model with its Ordered Visiting Constraints concept.

B. Ordered Visiting Constraints (Task Model)

In Section III-B, we elaborated that Ordered Visiting Actions (OVAs) are a natural and comfortable description

of behavior for assembly robots. Here, we introduce our constraint-based formalization of OVAs.

Def 8. An **Ordered Visiting Constraint** (OVC) represents an OVA and is defined by a tuple

$$\omega = (A, [P_1, ..., P_l], [L_1, ..., L_l], [I_1, ..., I_l], C_{\text{intra}}).$$

Variable A represents an active component and thus has domain $\mathbb{A}$. The variables P_j represent the *primitive actions* (e.g. pick, place, drill, etc.) to be executed. The variables L_j describe locations in the domain $\mathbb{L}$. The variables I_j model time intervals. A triple P_j, L_j and I_j denotes that active component A shall perform action P_j during time I_j at location L_j (cf. Fig. 2). More precisely, during I_j, it shall be in a configuration C with $C \in \lambda(L_j)$ to perform P_j. We refer to the set of all OVCs as Ω.

C_{intra} is a set of constraints on these variables. Examples for such constraints are the selection of certain active components, a set of locations to visit, specific primitive actions to be performed, or a minimum waiting time at a certain location.

Def 9. Inter-OVC constraints C_{inter} may be used to define task-specific constraints between two or more OVCs. Typical examples are temporal constraints between OVCs (e.g., by causality, for synchronization, or for mutual exclusion) or combinatorial constraints (e.g., to distribute m source locations amongst n OVCs).

Def 10. A **resource** r is an abstract or physical object (e.g., a tool, a workpiece holder, a robot gripper, etc.) that can be reserved exclusively for arbitrary time intervals. Typically, reservations are defined the by referencing start or end variables of interval variables of those OVCs that require this resource (cf. Fig. 2).

C. Examples for Task Modeling with OVCs

Programming with OVCs is conceptually more like arranging a plan than imperative programming. The OVC variables we create resemble the OVAs to be carried out. A simple pick-and-place action can be implemented straightforwardly by a single OVC with two location variables L_1 and L_2 constrained to start and goal location. If no constraint is given on A, any arm may perform the action.

The sorting use-case may be implemented by one such OVC per object. The solver may then decide about the partitioning of the OVCs to the two arms and the order per arm.

Alternatively, the sorting use-case may be implemented by OVCs without intra-OVC constraints on L_1 but with an inter-OVC constraint enforcing different location values for all L_1 variables of all OVCs. L_2 has to be constrained depending on the object color, given by the solver's decision for L_1.

For the electrical check in the third use-case, two actions have to be coordinated. This task is modeled by two OVCs: ω_{fixture} for holding the part into the fixture and ω_{push} to push the button for starting the check. ω_{push} has one location variable constrained to the button location (i.e. $L_1 = l_{\text{Button}}$). ω_{fixture} has three location variables constrained to the position for the visual check, the fixture location, and the destination container. To synchronize the two OVCs, we constrain the push interval (I_1 of ω_{push}) to be during the fixture interval (I_2 of ω_{fixture}).

After creating the appropriate number of OVCs, we generally strive to constrain the OVC variables as far as possible to prevent symmetric solutions, but without excluding the optimal solution. For example, in case of the second alternative for the sorting use-case, we would fix the order of the OVCs by an inter-OVC constraint to prevent that the assignment of the L_1 variables is permutated just as the ordering of the OVCs.

V. STAAMS Solver for OVCs

In the following, we first explain how the motion model and the task model are integrated by *Connection Variables* into a STAAMS model. The Connection Variables ensure that assignments in the task model are propagated to the motion model and vice-versa (cf. Fig. 2). Then, we explain the horizon estimation for the number of configurations in each motion series. Finally, we present our solver's search strategy, defined by the variable ordering and the value selection heuristics.

A. Connection Variables

The Connection Variables link the task model and the motion model by an index-based mechanism. We create such a variable $X_{\omega,j}$ for each location variable L_j of every OVC ω. The domain of $X_{\omega,j}$ is the index of the configurations $[C_1, \ldots, C_m]$ of the motion series σ of the active component A of ω. An assignment $X_{\omega,j} = i$ states that the ith configuration variable C_i of the motion series σ of A has to reach to the jth location of ω, formally $C_i \in \lambda(L_j)$. In this way, the Connection Variables establish the execution order for the OVCs assigned to an active component. Initially, if A of ω is unconstrained, it is not defined which motion series the domain of $X_{\omega,j}$ refers to.

Connection Variables have the following properties and are subject to the following constraints:

1) It always is $X_{\omega,j} < X_{\omega,j+1}$ since the locations $[L_1, \ldots, L_l]$ of ω have to be visited in this order.
2) Yet, two OVCs for the same active component may be interleaved (e.g., $X_{\omega_1,1} = 3$, $X_{\omega_2,1} = 4$, $X_{\omega_2,2} = 5$, and $X_{\omega_1,2} = 6$) if there is no conflicting inter-OVC constraint or resource-constraint.
3) Two Connection Variables must never reference the same configuration variable.

The index-based mechanism of the Connection Variables couples the task model and the motion model in a very flexible manner.

B. Horizon estimation

Initially, we do not know the optimal horizon m for every active component, i.e. the number of configuration

variables. Therefore, we integrate an iterative deepening approach directly in our model.

For each active component a, we create a constraint variable H named *horizon*. We prevent any movements after the Hth configuration in the motion series of a by constraining all configurations $C_{i>H}$ to C_H.

Small horizon values generally render the problem unsatisfiable, while large values bloat the search space unnecessarily and may cause superfluous motions. A lower bound for H is the number of all location variables of the OVCs assigned to the corresponding active components.

By means of the motion series Σ, the OVCs Ω, the resources with their interval variables, the Connection Variables, and the horizon variables, we described all variables of the CP-based STAAMS model – together with the corresponding generic and task-specific constraints. Next, we discuss how to efficiently search solutions for a given model instance.

C. Search strategy

A constraint satisfaction solver computes one or more variable assignments that each satisfy all constraints. Such solvers usually interleave a backtracking search with constraint propagation. In the backtracking search, variables are selected according to a variable-ordering heuristic, and values for the variables are chosen based on a value-ordering heuristic. Then, it propagates this decision by checking every constraint involving the selected variable for an effect on other variables and applies that effect on the possible values of the affected variables, i.e. it reduces the individual domains of the affected variables. Once a decision leads to an empty domain, a previous decision must have rendered the partial assignment infeasible, so the solver backtracks to the previous decision.

Variable ordering. The Connection Variables constitute a special case in our model. Due to their index-based mechanism, the constraint information from the motion model to the task model and vice-versa cannot be propagated until decisions on the involved Connection Variables have been made. The Connection Variables, again, require to first decide on the active component variables A. These are best decided upon, when the location variables L_i of the OVCs are bound. Therefore, we use the following variable ordering: (1.) Location variables, (2.) active component variables, (3.) Connection Variables, (4.) horizon variables, and then (5.) configurations variables of the motion series. At this stage, only the the time interval variables of the resources, OVCs and motion series remain to be decided. As the time interval variables of the resources are connected to the OVCs, which in turn are linked with the motion series by the Connection Variables, the planning system has to decide about the time-scaling of the motion series. More precisely, the planning system has to decide about the waiting times $I_1^{\mathrm{w}}, \ldots, I_H^{\mathrm{w}}$ of each motion series. The time-scaling has to prevent collisions, resolve resource conflicts, and satisfy any inter-OVC constraint (e.g. synchronization or ordering). Also, no superfluous waiting times should be added to optimize the makespan. By solving this time-scaling problem as final step, we obtain the timed motion plans for all active components.

Value selection. For each selected variable, the solver has to assign a value from the variable's domain. In case of the Connection Variables (4) and horizon variables (3), we use a minimum value heuristic to foster short motion series. For (1), (2), and (5), we use a random value selection heuristic as there is no clear preference for these variables. In case of a good value selection, the remaining search process involves only few backtracks. We employ a Luby restart strategy (cf. [17]) to avoid long-lasting searches in the time-scaling step (i.e. in the 6th step). This is especially useful when poor value selections have been made in steps (1) to (5).

VI. Evaluation

We implemented our STAAMS model and solver in Python using the Google Operation Research Tools [8] and experimented with a KaWaDa Nextage dual-arm robot in a Gazebo simulation environment [11] on a HP zBook laptop. We implemented the first and second use-case given in Sec. III-A. Here, we focus on the sorting use-case, which resembles the experiment by Kimmel at al. for their dual-arm coordination algorithm [10], and compare the results. Afterwards, we show how our solver scales on instances of this use-case for up to 200 objects. The modularity of our STAAMS model (cf. Fig. 2) enables the re-use of tasks expressed as *Ordered Visiting Constraints*. To showcase this, we deployed an example task (taking all objects from a table) on two robotic platforms by reusing the task model.

Comparison with pure time-scaling. We modeled three instances of the sorting use case with increasing number of objects from 12 to 24 and varying degree of conflict between the two arms (see Fig. 4). Then, we compared our approach against the theoretical lower bound obtained by ignoring collisions between the manipulators as well as against the method by Kimmel et al., which time-scales the trajectories of both manipulators to prevent collisions. We mimic their solver by using a randomized but fixed order of collecting the objects and leave only the scheduling to our solver. The results are visualized in Fig. 4. The diagrams show plots of the makespan (as quality measure) over the time spent to solve the instance (stopped after 100 s) for ten different fixed order runs (red) and ten runs with order optimization (dark-blue). Our solver produces the first solutions sometimes as fast as in 0.1 s and usually converges within 3 s on the instances shown. By optimizing the order, our solver consistently outperforms the fixed order runs – or reaches the same performance in the rare case that by chance a very good order is selected. Since both approaches utilize some random decisions, the plotted

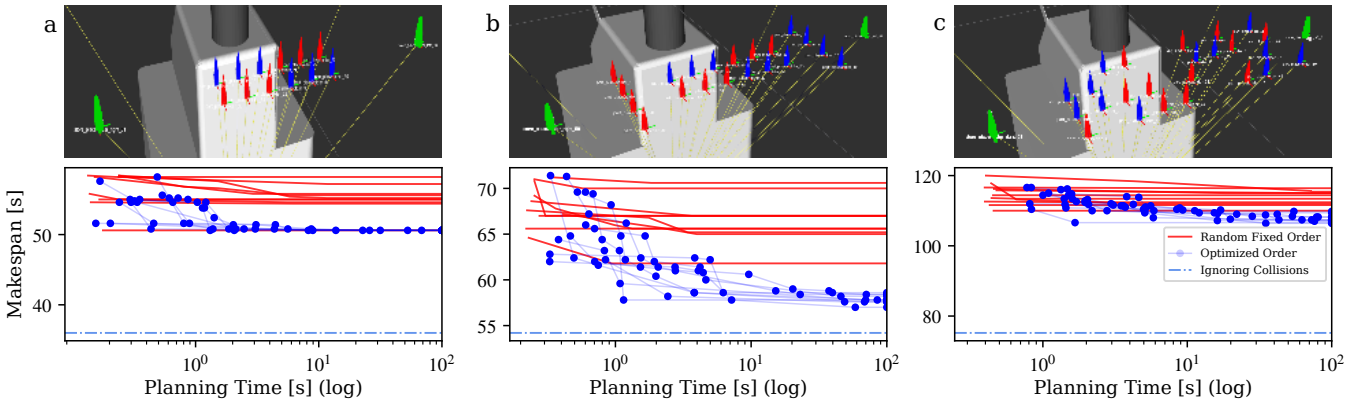

Fig. 4. Sorting scenarios (a)-(c) and makespan-vs-planning-time plots. Red lines show the makespan over planning time for a random fixed order of execution (cf. [10]). The blue lines depict the makespan, when we let the solver decide on the order. A lower bound for each problem – obtained by ignoring collisions (relaxation of the problem) – is plotted in light blue, Blue Objects are dropped into a container by the left arm at the left destination (green), and vice versa for the red objects. (a) 12 objects with high conflict potential, (b) as (a) but with eight uncritical objects more to allow for efficient scheduling. (c) A randomly chosen instance with 24 objects and much interaction

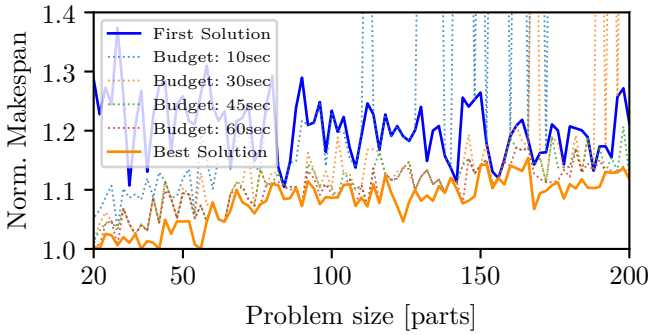

Fig. 5. In this diagram, the solution quality (makespan) divided by the lower bound (which ignores collisions between the manipulators) is shown for different problem sizes and different stages in the search. The vertical lines in the right half of the figure indicate cases in which no solution was found within the budgets of 10 or rather 30 sec.

outcomes visualize a distribution. With this in mind, it becomes very clear that our STAAMS solver provides much more consistent and higher-quality results. In scenario (b), it gets very close to the theoretical lower bound (light blue). Interestingly, it takes only 7 s more to handle eight additional objects in (b) compared to (a).

Scalability. To evaluate the scaling properties of our approach, we ran a series of 80 experiments similar to scenario (b) with a time limit of 180 s. Starting from the twenty parts depicted in Fig. 4 b, we added for each experiment two extra parts to the scene – one for each arm – up to 200 parts in total. In Fig. 5, the normalized makespan, i.e. the makespan divided by the theoretical lower bound, is plotted over the problem size for the first solutions, the best solutions, and computing time budgets from 10 to 60 s. The solution quality for the first solution ranges approximately from 1.1 to 1.37 normalized makespan,[1] which rapidly improves with the following solutions to finally settle around 1.1 normalized makespan. Fig. 6 shows for the same set of experiments the computing time over the problem size for the first solution and the time budgets needed to reach the best solution quality plus 15 or rather 7.5%. We find the first solutions for the largest instances in about 25 s, while our approach converges for normal sized instances in about 1 to 8 s (cf. Fig. 4 b).

Our solver computes high-quality solutions even for large problem instances in a few seconds or minutes. Please note that the high scalability compared to ITAMP planners stems from two facts: First, STAAMS solving does not require to decide about the actions to be executed but rather to complete and optimize a given abstract plan (here modeled by OVCs) only. Second, in our motion model we limit the motions to stick to predefined roadmaps.

Portability. Flexibility and portability of our modeling language are validated through several experiments on different simulated robot platforms (seeFig. 7). The task models, i.e. the sets of OVCs, that have been used to perform the pallet emptying on the KaWaDa Nextage and KUKA LBR iiwa platforms[2] are identical. The differences are[3]: The robot model (Moveit! [19] robot configuration to access motion planning, kinematic calculations, and collision checking); a seed robot configuration (as required by the Inverse kinematics (ik) solvers); a ”tuck” robot configuration (in which the arms do not obstruct each others workspaces); the names of kinematic chains, end-effectors, and the base frame; the static scene collision layout (represented in meshes or primitive shapes; and the locations of the workpieces. With this information and scripts in place, our system automatically creates the roadmaps, collision tables, and

[1] A solution with a normalized makespan of approximately 2 can always be constructed by only moving one arm at a time.

[2] We like to thank the researchers of the Robotic perception group (ROP) at the Czech Institute of Informatics, Robotics and Cybernetics (CIIRC CTU) for providing the KUKA simulation environment.

[3] The code and the setup details will be released on Github until the 31st of October 2018 at `https://github.com/boschresearch/STAAMS-SOLVER`

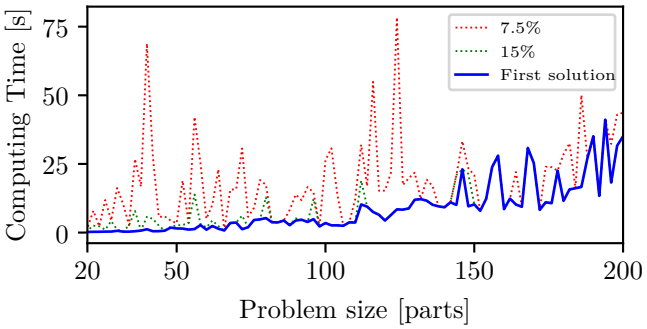

Fig. 6. The diagram shows the computing time depending on the problem size for scenario (b). The first solution is depicted in blue. The dashed lines show the time budget needed to find solutions 15 and 7.5 percent worse than the best solution.

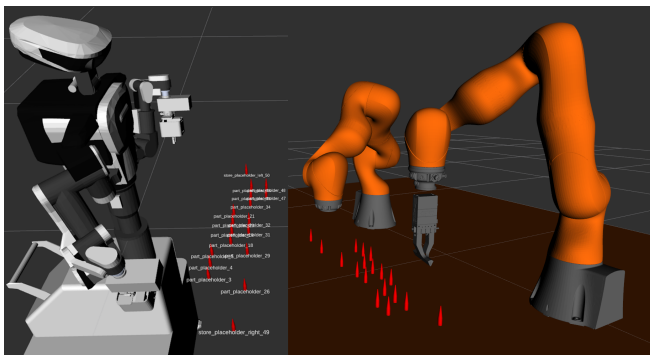

Fig. 7. A task – cleaning up the table – deployed on a KaWaDa Nextage robot (left) and a pair of KUKA LBR iiwa robots (right).

name-location-configuration mappings.

VII. Conclusion and Future Work

In this paper, we proposed a flexible model and solver for simultaneous task allocation and motion scheduling (STAAMS) based on constraint programming (CP) and constraint optimization for industrial manipulation and assembly tasks for dual-arm robots[4]. The core modeling concepts are Ordered Visiting Constraints and time-scalable motion series, which are linked by meta CP variables named Connection Variables. In our evaluation, we showed that our STAAMS solver quickly completes and optimizes a given problem model instance – i.e. an abstract task specifications given as collection of OVCs for a robot motion model – and delivers high-quality, executable motion plans. We demonstrated the scalability of our approach on large problem instances with up to 200 actions, which were solved in less than 180 s. We also showed, that the OVC concept allows to transfer a given task model to another robot and/or workspace by exchanging the relevant motion submodels only. We assume that our task-centric robot programming approach is suited not only for textual specification but also for multi-modal input variants. Therefore, we want to explore robot programming by natural language and demonstrations.

To broaden the applicability of this approach, we plan to include more task primitives (additionally to reaching goals) like trajectories for welding. We will also investigate the extension of our approach to include action models with safe approximations, when the actual space occupancy and duration are not known, e.g., when employing force-position control.

[4]The link to the extended version of our video will be available at `https://github.com/boschresearch/STAAMS-SOLVER`

References

[1] Sergey Alatartsev, Sebastian Stellmacher, and Frank Ortmeier. Robotic Task Sequencing Problem: A Survey. *J Intell Robot Syst*, 80(2):279–298, November 2015.

[2] Jan Kristof Behrens, Ralph Lange, and Michael Beetz. CSP-Based integrated Task & Motion Planning for Assembly Robots. In *Proc. of the Workshop on AI Planning and Robotics at ICRA '17*, Singapore, May 2017.

[3] S. Blumenthal, H. Bruyninckx, W. Nowak, and E. Prassler. A scene graph based shared 3D world model for robotic applications. In *Proc. of ICRA '13*, pages 453–460, Karlsruhe, Germany, May 2013.

[4] S. Cambon, R. Alami, and F. Gravot. A Hybrid Approach to Intricate Motion, Manipulation and Task Planning. *Int'l Journal of Robotics Research*, 28(1):104–126, January 2009.

[5] Neil T Dantam, Zachary K Kingston, Swarat Chaudhuri, and Lydia E Kavraki. An incremental constraint-based framework for task and motion planning. *The International Journal of Robotics Research*, March 2018.

[6] Joakim Ejenstam. Implementing a Time Optimal Task Sequence For Robot Assembly Using Constraint. Master's thesis, Uppsala Universitet, Sweden, 2014.

[7] Caelan Reed Garrett, Tomas Lozano-Perez, and Leslie Pack Kaelbling. FFRob: Leveraging Symbolic Planning for Efficient Task and Motion Planning. *The International Journal of Robotics Research*, 37(1):104–136, January 2018. arXiv: 1608.01335.

[8] Google Inc. Google Optimization Tools. Retrieved February 28, 2018, from `https://github.com/google/or-tools`.

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

[10] A. Kimmel and K. E. Bekris. Scheduling Pick-and-Place Tasks for Dual-arm Manipulators using Incremental Search on Coordination Diagrams. In *Proc. of PlanRob Workshop at ICAPS '16*, London, UK, June 2016.

[11] N. Koenig and A. Howard. Design and use paradigms for Gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, volume 3, pages 2149–2154 vol.3, September 2004.

[12] Ewa Kolakowska, Stephen F. Smith, and Morten Kristiansen. Constraint optimization model of a scheduling problem for a robotic arm in automatic systems. *Robotics and Autonomous Systems*, 62(2):267–280, February 2014.

[13] Jun Kurosu, Ayanori Yorozu, and Masaki Takahashi. Simultaneous Dual-Arm Motion Planning for Minimizing Operation Time. *Applied Sciences*, 7(12):1210, November 2017.

[14] F. Lagriffoul and B. Andres. Combining task and motion planning: A culprit detection problem. *Int'l Journal of Robotics Research*, 35(8):890–927, July 2016.

[15] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006.

[16] T. Lozano-Pérez and L. P. Kaelbling. A constraint-based method for solving sequential manipulation planning problems. In *Proc. of IROS '14*, pages 3684–3691, September 2014.

[17] M. Luby, A. Sinclair, and D. Zuckerman. Optimal speedup of Las Vegas algorithms. *Information Processing Letters*, 47(4):173–180, September 1993.

[18] P. A. O'Donnell and T. Lozano-Perez. Deadlock-free and collision-free coordination of two robot manipulators. In *Proc. of ICRA '89*, pages 484–489, Tsukuba, Japan, May 1989.

[19] Ioan A. Sucan and Sachin Chitta. Moveit! Retrieved September 08, 2018, from `http://moveit.ros.org`.