A constraint programming approach to simultaneous task allocation and motion scheduling for industrial dual-arm manipulation tasks

Behrens, Jan Kristof; Lange, Ralph; Mansouri, Masoumeh

DOI:
10.1109/ICRA.2019.8794022

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]

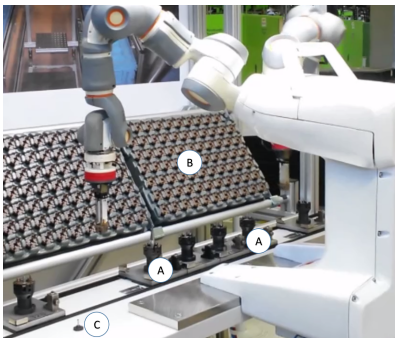
***Abstract*— Modern lightweight dual-arm robots bring the physical capabilities to quickly take over tasks at typical industrial workplaces designed for workers. Low setup times – including the instructing/specifying of new tasks – are crucial to stay competitive. We propose a constraint programming approach to simultaneous task allocation and motion scheduling for such industrial manipulation and assembly tasks. Our approach covers the robot as well as connected machines. The key concept are Ordered Visiting Constraints, a descriptive and extensible model to specify such tasks with their spatiotemporal requirements and 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. For large manipulation tasks with 200 objects, our solver implemented using Google's Operations Research tools requires less than a minute to compute usable plans. The proposed task model is robot-independent and can easily be deployed to other robotic platforms. This portability is validated through several simulation-based experiments.**

I. Introduction

Modern lightweight dual-arm robots such as the ABB YuMi or the KaWaDa Nextage are engineered in the style of a human torso to be easily applicable in industrial workplaces designed for workers. These robots are an answer to the demand for flexible, cost-effective 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 paths is not appropriate. With the capabilities of today's perception systems, which can detect and localize workpieces, boxes, and tools automatically, and a formalized goal or high-level task specification, the manual teach-in may be replaced by automated planning – in principle. Planning in these contexts involves three aspects: (a) *task planning* of the necessary steps and actions to achieve the overall task, (b) *scheduling* of these steps and actions, and (c) *motion planning* for each step and action.

For dual-arm robots, additionally (d) the *allocation* of task steps and actions to the individual arms must be decided upon. 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.



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 (A), picks an electric interface, supplied in a container (B) and places it on (A).

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), 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 (e.g., [10], [6], [16], [18], [7]), 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, i.e. there is an abstract plan with a number of unknowns and degrees of freedom in terms of scheduling, allocation and motion planning. Computing an optimal plan requires to treat the aspects scheduling, allocation and motion planning 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 motions of the manipulators 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

[1]Robert Bosch GmbH, Corporate Research, Renningen, Germany, behrens.jk@gmail.com, ralph.lange@de.bosch.com

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

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. In detail, our contributions are as follows:

1) For specifying the abstract task decomposition of a STAAMS problem, we propose a novel and intuitive model primitive named *Ordered Visiting Constraints* (OVC). It 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 be performed at given locations with ordering or temporal constraints in-between.
2) We propose an advanced CP concept named *Connection Variables* to link the OVC-based task model with a roadmap-based time-scalable motion series model into an unified STAAMS problem model. At the same time, we explain how this modularity allows to easily port a given OVC-based task model not only to different workspace layouts but even to different robots.
3) We present an adaptable solver, which allows for fine-grained user control over the different constraint optimization techniques to compute an almost-optimal plan for typical STAAMS problem sizes in few seconds.

The remainder of this paper is organized as follows: We present an analysis of typical industrial use-cases in Sec. II before we discuss related work in Sec. III. The STAAMS model with the OVC-based task model and the motion model as well as the corresponding solver, are presented in Sec. IV and Sec. V. Scalability and portability of the proposed system is evaluated in Sec. VI. The paper is concluded in Sec. VII.

II. Use-case Analysis and Problem Definition

We analyzed several industrial workplaces for characteristic properties and prevalent concepts serving as basis for the design of our STAAMS model. In this section, we report on these results, state our assumptions and then derive a problem definition for STAAMS.

Controlled environment. Industrial workplaces provide a controlled environment by design. We may assume that object locations and possible placements are known in advance, which allows for offline pre-calculation of motion roadmaps. Also, we 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 subset 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. [4]) 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 screw 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). 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. In the motor assembly use-case depicted in Fig. 1, the temporal dependencies are given by the assembly sequence for each motor. Also, in general, each arm can carry only one object at a time, i.e. the gripper is a reservable resource requiring 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 a molding machine, which is closed during processing. We generalize such stations and the robot arms together as *active components*.

Problem Definition. From the use-cases point-of-view, the problem of *STAAMS* can be defined as an integrated task and motion planning (ITAMP) problem limited to (1.) a given partially-ordered task decomposition with further spatiotemporal, combinatorial and ordering constraints, and (2.) predefined locations (6DoF poses) for small-sized objects so that motion planning can be reduced to motion scheduling along the shortest paths from precomputed roadmaps.

From a formal perspective, there is a close relation to multi-agent path finding (MAPF). More precisely, STAAMS can be formulated as the anonymous variant of MAPF, combined with target assignments (TAPF) given by partially-ordered sets of subtasks/actions with spatiotemporal, combinatorial and ordering constraints on predefined locations; and is NP-hard [21].

III. Related Work

In the last years, a number of techniques for ITAMP have been proposed, including extensions of classical task planning which iteratively query and take feedback from motion planning (e.g., [8], [25], [16], [3], [10], [7]), motion planning being guided by a symbolic layer (e.g., [6], [27]), and two-staged approaches (e.g., [18]).

From the listed works, only [6] and [25] consider multiple manipulators – multiple mobile single-arm robots or a mobile dual-arm robot, respectively. Both works unveil the complexity of ITAMP for multiple robots or manipulators. In contrast, STAAMS assumes a pre-existing abstract task decomposition effectively reducing

the task planning subproblem to task allocation and motion scheduling.

The state-of-the-art optimal TAPF method [20] cannot solve STAAMS problems in general, as it does not compute kinematically feasible motions for agents, nor can it be applied in cases requiring ordering decisions about task assignments. Online methods for multi-agent task assignment and scheduling algorithms have been developed for small-sized teams of agents, and very robust in the face of execution uncertainty [24]. Multi-robot task allocation with temporal/ordering constraints has been studied in the context of integrating auction-based methods with Simple Temporal Problems [22]. These methods, however, do not account for conflicting spatial interactions, as needed, for example, in dual-arm manipulations. The applications of CP to multi-robot task planning and scheduling often use a simplified robot motion model, and ignore the cost of spatial interaction among robots in the scheduling process [5].

The motion planning and scheduling subproblems of STAAMS can be seen as a multi-robot motion planning problem. LaValle [17] defined the following three classes: *Prioritized planning* assigns an order to the robots (arms) according to which their movements are planned (e.g., [15]). *Fixed-path planning* (also named *time-scaling*) only adjusts timings to prevent collisions (e.g., [23]). *Fixed-roadmap planning* considers paths in topological graphs . Our method falls in the third category.

Time-scaling problems are a special-case of STAAMS. In Sec. VI, we compare our approach against the time-scaling method by Kimmel et al. [13].

Regarding the task sequencing subproblem of STAAMS, we refer to the survey by Alatartsev et al. [1]. The survey, however, lacks the coverage for tasks that are applicable for multi-arm robots.

Very close to our work is the CP-based approach for dual-arm manipulation planning and cell layout optimization by Ejenstam et al. [9]. It uses a coarse discretization of the workspace. In contrast, our dense roadmaps enable the close coordination of the arms and thus allow more parallel movements. Kurosu et al. [15] describe a decoupled MILP-based approach to solve a STAAMS, where the motion planner is prone to fail due to simplified motion and cost models used in the one-shot MILP formulation. In contrast, our solver finds a mutually feasible solution for all subproblems.

In [2], we presented a toy use-case and solved it using MiniZinc. Without the OVC model and the tailored search strategy, this solution did not scale adequately.

IV. Modeling STAAMS Problems with OVCs

In this section, we present our formalism for specifying STAAMS problems as constraint programs. Our model consists of two submodels named *task model* and *motion model*. As illustrated in Figure 2, the task model is independent of any kinematic details and actual trajectories.

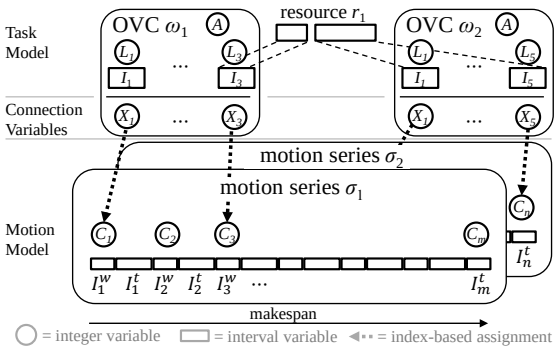


Fig. 2. Overview of our CP-based STAAMS model

Conversely, the motion model represents the trajectories of all active components independent of any task information. Both models are linked through *Connection Variables*, a special kind of CP variable. Next, we explain both submodels and then the Connection Variable mechanism. For readability, we write constants or values as lowercase Latin letters and constraint variables as capital letters. Compounds of constraint variables are denoted with small Greek letters.

A. Task Model

The first and most important element of the task model are OVCs, which can be considered as variable, constraint-based blueprints of OVAs. An OVC consists of four sets of CP variables modeling *primitive actions* (e.g. pick, place, drill, etc.) to be executed at certain *locations* in the workplace within certain *time intervals* by an *active component*. The locations $\mathbb{L}$ are a finite set of 6DoF poses of interest in the workplace – in particular possible object placements in containers and workpiece holders – in a common reference system.

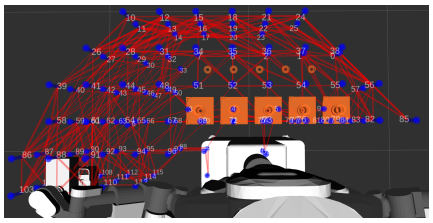
Def 1. Formally, an **OVC** is a tuple

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

The variables P_j represent the primitive actions, the variables $L_j \in \mathbb{L}$ describe the locations, and the variables I_j model the time intervals. The variable A represents the active component to be used. 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.

In C_{intra}, arbitrary constraints on and between these variables can be specified. In particular, each P_j can be constrained to a specific primitive action to be executed. Similarly, each L_j is typically constrained to one or few specific locations or specific location combinations for all location variables. Also, quantitative temporal constraints on the time intervals may be given.

The task model also allows for arbitrary constraints between OVCs, named *inter-OVC constraints* C_{inter}. Typical examples are temporal constraints between OVCs (e.g., for synchronization or ordering of OVCs or combinatorial constraints – e.g., to distribute m locations



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

amongst n OVCs). In the following, we refer to the set of all OVCs as Ω.

The second element of the task model are *resources* which describe abstract or physical objects such as tool, workpiece holders or robot grippers. A resource r 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.

B. Motion Model

The motion model represents the trajectories of all active components as *motion series* consisting of configuration variables – in the respective joint space of the active component – with time interval variables for the transition in-between. To be able to model the motion with CP, a roadmap-based approach is used (cf. for example [12]). The roadmap of an active component is a sufficiently dense sampling of the joint space, where the nodes are joint configurations and the edges represent short collision-free motions between them (cf. Fig. 3). In particular, the roadmap contains one or more nodes for each location $l \in \mathbb{L}$ in reach of the active component.

Def 2. Hence, the **motion series** σ_a of an active component a is a sequence of m configuration variables and a sequence of $2m-1$ interval variables

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

where the domain of the variables C_i are the nodes $\mathbb{C}$ of the active component's roadmap $r = (\mathbb{C}, \mathbb{E})$. 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}. For this purpose, the roadmap edges $\mathbb{E}$ denote the expected traveling time as edge weight. The roadmap thus yields information about paths between configurations and their durations to be used in the CP. Note that the roadmaps may also include multiple nodes for the same configuration – for example to consider the different collision geometries of the arm depending on the gripper state. The set of all motion series is named Σ. Collisions between any two active components a_i and a_j are prevented by a constraint requiring that pairs of conflicting joint configurations c_i and c_j, which are precomputed in a *collision table*, must not be assumed simultaneously.

C. Connection Variables

The task model and the motion model are linked in two ways: On the constants, a static *location mapping* λ per active component links the roadmap nodes $\mathbb{C}$ with the task locations $\mathbb{L}$. On the variables, *Connection Variables* are introduced to link the location variables of the OVCs with the configuration variables of the motion series. Such a connection denotes that the configuration variable has to be chosen such that the respective active component reaches the location given in the location variable. There exists exactly one such connection per location variable. Configuration variables not referenced by Connection Variables are used for evasive movements to avoid collisions and deadlocks.

These connections are also CP variables. They can be considered as meta variables as they specify to which configuration variable to point to. Formally, the domain of the Connection Variable $X_{\omega,j}$ for the location variable L_j of OVC ω 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 $C_i \in \lambda(L_j)$.

Connection Variables of ω always have to be strictly monotonic, i.e. $X_{\omega,j} < X_{\omega,j+1}$, since the locations $[L_1, \ldots, L_l]$ of ω have to be visited in this order. 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. Two Connection Variables must never reference the same configuration variable.

D. Example for Task Modeling with OVCs

As an example, we model a bi-manual pick-up of a workpiece container. Therefore, we create two OVCs ω_{L} and ω_{R}, each of length 2. We constrain the first location variables L_1 to assume either of the locations $\{l_{\mathrm{Grasp1}}, l_{\mathrm{Grasp2}}\}$. Through an inter-OVC constraint, we ensure that the combination of selected grasp poses yields a valid combination for a stable grasp. We constrain the second location variables L_2 to take either of the locations $\{l_{\mathrm{Place1}}, l_{\mathrm{Place2}}\}$ and an additional inter-OVC constraint ensures, that all 4 pick and place locations take compatible values. Temporal constraints ensure that the first intervals I_1 end together and the second intervals I_2 start together. Note, that the arms have to be controlled by a dedicated controller for the actual carrying. We assume, that the controller either provides information about occupied space over time, such that the solver can schedule possible other components to not interfere, or stays within a given subset of the workspace.

V. STAAMS Solver for OVCs

A constraint satisfaction solver usually interleaves backtracking search with constraint propagation. In the backtracking search, variables are selected according to a variable-ordering heuristic and variable values are chosen based on a value-ordering heuristic. In the following, we explain the horizon estimation for the motion series. Then we describe our choices for the search heuristics.

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. Therefore, the placement of the horizon variables in the variable-ordering heuristic is a crucial factor.

Variable ordering. Constraint information between the task and model model 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 and the location variables L_i. In our experiments, searching (1.) on the location variables, (2.) on the active component variables, (3.) on the Connection Variables, (4.) on the horizon variables, and then (5.) on the configurations variables of the motion series yielded reliably good solutions within a few seconds planning time.

Then, only the time interval variables of the resources, OVCs and motion series remain to be decided. As the resources' time interval variables are connected to the OVCs, which in turn are linked with the motion series, the solver thus determines the time-scaling of the motion series. More precisely, it decides about the waiting times $I_1^{\mathrm{w}}, \ldots, I_H^{\mathrm{w}}$. The time-scaling allows to prevent collisions, resolve resource conflicts, and satisfy any inter-OVC constraint (e.g. synchronization or ordering). In this process no superfluous waiting times should be added to optimize the makespan.

Value selection. For each variable, the solver has to assign a value from the variable's domain. For 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. To avoid long-lasting searches in the time-scaling step (i.e. in the 6th step) for disadvantageous decisions in the preceeding steps, we employ a Luby restart strategy (cf. [19]).

VI. Evaluation

We implemented our STAAMS model and solver in Python using the Google Operation Research Tools [11] and experimented with a KaWaDa Nextage dual-arm robot in a Gazebo simulation environment [14] on a HP zBook laptop. We implemented a wiper motor use-case (see Fig. 1) as well as a *sorting use-case* which resembles the experiment by Kimmel at al. for their dual-arm coordination algorithm [13]. In the latter use-case, 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.

In this section, we focus on the sorting use-case. First, we compare our results with [13]. Then, we show how our solver scales on instances of this use-case for up to 200 objects. Finally, we show the modularity of our STAAMS model by using one and the same task model on two robotic platforms. Code and setup details are available at `https://github.com/boschresearch/STAAMS-SOLVER`.

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 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. 4b, 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 s to 60 s. The solution quality for the first solution ranges approximately from 1.1 to 1.37 normalized makespan (solutions with a normalized makespan of approx. 2 can be constructed), which rapidly improves

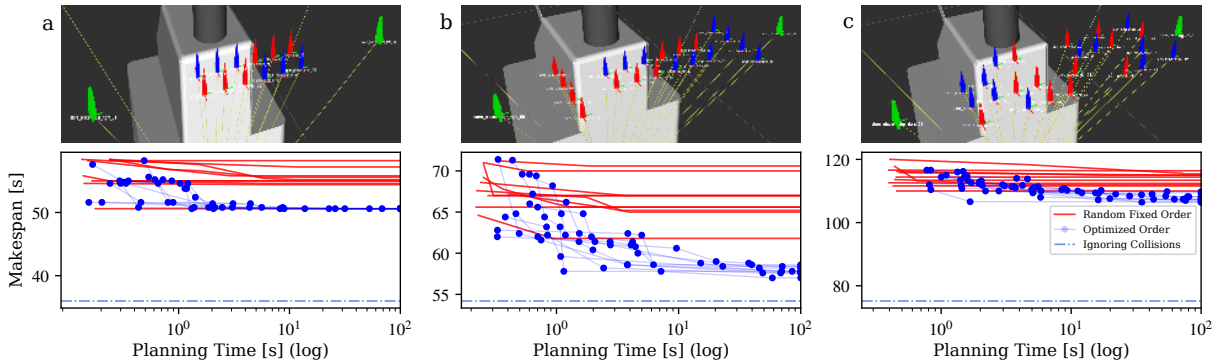


Fig. 4. Sorting scenarios (a)-(c) and makespan-vs-planning-time plots. Red lines show the makespan over planning time for a random fixed order of execution (cf. [13]). The blue lines depict the makespan, where we let the solver decide on the order. A lower bound for each problem – obtained by ignoring 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

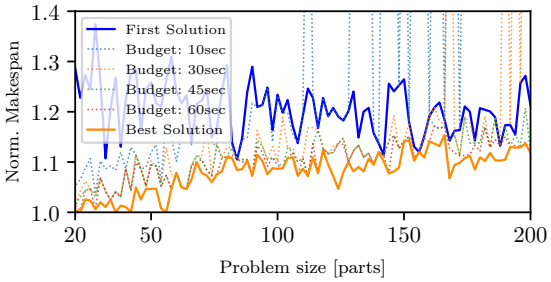
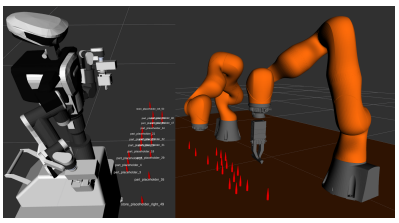


Fig. 5. Relative solution quality (i.e. makespan divided by the lower bound) for different problem sizes and stages in the search. The vertical lines in the right half of the figure indicate cases in which no solution was found within the given time budget.

with the following solutions to finally settle around 1.1 normalized makespan.

This 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 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 (see Fig. 6). 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 are identical. The differences are: The robot model (Moveit! [26] 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



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

automatically creates the roadmaps, collision tables, and name-location-configuration mappings.

VII. Conclusion and Future Work

In this paper, We have proposed a flexible model and solver for simultaneous task allocation and motion scheduling (STAAMS) for industrial manipulation and assembly tasks for dual-arm robots. Our STAAMS solver quickly completes and optimizes a given problem model instance – i.e. an abstract task specifications given as collection of Ordered Visiting Constraints 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. Also, Ordered Visiting Constraints allow us to transfer a given task model to another robot and/or workspace only by exchanging the relevant motion submodels. In order to broaden the applicability of this approach, we plan to include more task primitives 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.

Acknowledgments. 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. This work is partially supported by the Swedish Knowledge Foundation project "Semantic Robots".

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] Julien Bidot, Lars Karlsson, Fabien Lagriffoul, and Alessandro Saffiotti. Geometric backtracking for combined task and motion planning in robotic systems. *Artificial Intelligence*, 247:229 – 265, 2017.

[4] 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.

[5] Kyle E. C. Booth, Goldie Nejat, and J. C. Beck. A constraint programming approach to multi-robot task allocation and scheduling in retirement homes. In *Proc. of Principles and Practice of Constraint Programming (CP)*, 2016.

[6] 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.

[7] 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.

[8] Christian Dornhege, Patrick Eyerich, Thomas Keller, Sebastian Trüg, Michael Brenner, and Bernhard Nebel. Semantic attachments for domain-independent planning systems. In *Proc. of ICAPS '09*, pages 114–121, 2009.

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

[10] 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.

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

[12] 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.

[13] 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.

[14] 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)*, volume 3, September 2004.

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

[16] 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.

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

[18] 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.

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

[20] Hang Ma and Sven Koenig. Optimal target assignment and path finding for teams of agents. In *Proceedings of the International Conference on Autonomous Agents; Multiagent Systems*, AAMAS '16, pages 1144–1152, 2016.

[21] Hang Ma, Sven Koenig, Nora Ayanian, Liron Cohen, Wolfgang Hönig, TK Kumar, Tansel Uras, Hong Xu, Craig Tovey, and Guni Sharon. Overview: Generalizations of multi-agent path finding to real-world scenarios. In *Proceedings of the IJCAI-16 Workshop on Multi-Agent Path Finding*, 2016.

[22] E. Nunes, M. Manner, H. Mitiche, and M. Gini. A taxonomy for task allocation problems with temporal and ordering constraints. *Robotics and Autonomous Systems*, 2017.

[23] 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.

[24] Julie A. Shah, Patrick R. Conrad, and Brian Charles Williams. Fast distributed multi-agent plan execution with dynamic task assignment and scheduling. In *Proc of ICAPS '09*, 2009.

[25] S. Srivastava, E. Fang, L. Riano, R. Chitnis, S. Russell, and P. Abbeel. Combined task and motion planning through an extensible planner-independent interface layer. In *Proc. of ICRA*, pages 639–646, May 2014.

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

[27] Marc Toussaint. Logic-geometric programming: An optimization-based approach to combined task and motion planning. In *Proc. of 24th IJCAI*, pages 1930–1936, Buenos Aires, Argentina, jul 2015.