<http://www.diva-portal.org>

Postprint

This is the accepted version of a paper presented at *International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research, Online, 21-24 Sep 2020*.

Citation for the original published paper:

Wessén, J L., Schulte, C., Carlsson, M. (2020)

Scheduling of Dual-Arm Multi-Tool Assembly Robots and Workspace Layout Optimization

In: *Proceedings of 17th International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research*,

https://doi.org/10.1007/978-3-030-58942-4_33

N.B. When citing this work, cite the original published paper.

Permanent link to this version:

<http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-281417>

Scheduling of Dual-Arm Multi-Tool Assembly Robots and Workspace Layout Optimization

Johan Wessén^{1,2} ^[0000–0003–4126–6879], Mats Carlsson³^[0000–0003–3079–8095],
and Christian Schulte²^[0000–0002–6283–7004]

¹ ABB Corporate Research, Västerås, Sweden

`johan.wessen@se.abb.com` ,

² KTH Royal Institute of Technology, Stockholm, Sweden,

`cschulte@kth.se` ,

³ RISE Research Institutes of Sweden, Uppsala, Sweden

`mats.carlsson@ri.se`

Abstract. The profitability of any assembly robot installation depends on the production throughput, and to an even greater extent on incurred costs. Most of the cost comes from manually designing the layout and programming the robot as well as production downtime. With ever smaller production series, fewer products share this cost. In this work, we present the dual arm assembly program as an integrated routing and scheduling problem with complex arm-to-arm collision avoidance. We also present a set of high-level layout decisions, and we propose a unified CP model to solve the joint problem. The model is evaluated on realistic instances and real data. The model finds high-quality solutions in short time, and proves optimality for all evaluated problem instances, which demonstrates the potential of the approach.

Keywords: Assembly manufacturing · Constraint programming
· Robot planning and scheduling

1 Introduction

Consumer products are being manufactured in ever smaller series, often using assembly manufacturing. This means that complex sub-parts, such as cameras and circuit boards, are put together, akin to a 3D puzzle. These products are mainly manufactured by hand, in countries with competitive labor cost.

At the same time, lightweight industrial robots have been introduced, such as YuMi by ABB [1] and Sawyer by Rethink Robotics [17]. Many of these robots are designed to take over repetitive and tedious tasks from humans, occupying similar floor area and have dual arms with similar arm reach. Some even target small parts assembly manufacturing [1].

However, robot based assembly manufacturing is at a price point out of reach for many products. It is important to obtain good enough throughput of the robots, as it determines how much value the robots create. However, cost has the greatest impact on net value of the robot installation, and the cost is

largely determined by the time it takes to deploying the robot, as it often costs significantly more than the robot itself, plus the cost of production downtime. The inspiration for this work comes from a real world installation of a YuMi, where a highly skilled programmer and researcher from ABB spent six weeks to design the layout and program the robot. At deployment, unforeseen obstacles were present in the workspace, and the layout design and program had to be changed, which took another week. This work builds on [4] and [10].

Deploying industrial robots is a time-consuming, iterative process, and requires trained developers. The process entails **A)** layout design of the robot workspace, and robot programming. Layout design means deciding where to put locations of interest, such as pick-locations, as well as deciding what apparatus to use when multiple are available, based on their locations. These decisions greatly affect the throughput of the robots. The robot programming can be subdivided into **B)** task allocation and **C)** task sequencing, **D)** (inter-arm) scheduling, all while efficiently utilizing the multiple tools of each robot hand. It is slow and error-prone, this is especially true for multiple arms, where the programmer must coordinate the arms such that they do not collide during **E)** execution. This coordination greatly adds to the complexity of the problem. The two-dimensional case is NP-hard [19], and is an active field of research, with CP solutions at the forefront [11]. All sub-problems **A)**-**E)** are tightly interconnected.

Each task, such as pick or press, is in itself a small robot program. However, for the level of abstraction of this and related work, each task can be abstracted as a constraint on the duration of said task. Similarly, travel between task locations is a small program, which, in this and most of the related work, is abstracted as a constraint on the travel time between task pairs.

The AI community has a long history of addressing model-based robot programming, both Automated Ground Vehicles (AGV:s) using planning [6] and first-order logic [3] to achieve **C)**. AI for robotics is a long-standing topic [16], and robotics is an active part of the AI community [15]. Task planning, including temporal planning, is a field of active research [8], and there is a sub-field called integrated task and motion planning [12, 18, 7, 14, 9, 5, 13] addressing **C)** while also generating the travel motion between tasks. These works address single arms, while one recent CP approach includes a second arm [2], thus addressing **B)**, **D)**, **E)**. For that work, however, **C)** was given.

To our knowledge, no published work dealing with **A)** in this context exists, and no work approaching **B)**, **C)**, **D)** **E)** in unison. Some prior works include motion generation in their problem formulation, however due to our approach to arm coordination, these motions can be generated separately, under weak assumptions, and still result in collision-free robot programs. Thus, this work takes a larger scope on the robot deployment process and addresses all aspects **A)**-**E)**. We present the following contributions:

1. The first optimization-based model integrating **A)** layout optimization, **B)** task allocation, **C)** task sequencing, **D)** scheduling and **E)** collision avoidance for a dual-arm robot, to the best of our knowledge. The model has

features from vehicle routing and flexible job-shop, with variable locations and travel time that depends on location and performing agent.

2. An evaluation showing that the model is efficient, quickly delivering good solutions as well as reaching optimality for all problem instances.
3. An efficient way of handling arm-to-arm collisions while generating robot programs, based on dividing the workspace into two sides, and the preliminary observation that this does not affect the makespan.

2 The Assembly - Application, Cell and Robot Program

Assembly manufacturing is a type of production, where several *components* are combined into an *assembly*. Components could be of any kind such as electronic boards, motors, and sensors. Finished assemblies are forwarded to further processing steps. The assembly process is then repeated. In industrial robotics, a *robot workspace* is the space used by one or more robots and the equipment relating to the application, if the robot is included it is called a *robot cell*, and if it is used for assembly manufacturing it is called a *assembly cell*. Figure 1 shows the assembly cell used in this paper.

Cell & Application: In our example, there are five *tray locations* for holding *component trays*. Each component tray holds multiple components of one particular component type. At most one component tray can be put at any of the tray locations. Each component tray is paired with a camera facing the tray, to keep track of available components. However, the focal point of the cameras is located between the camera and the tray. The reason for this is explained later.

Fig. 1. An assembly cell with an ABB YuMi, two fixtures, five tray locations and a camera facing each tray location

In the assembly cell there are two *fixtures*. These are constructions in front of the robot, and are designed to hold one specific component each. When that component is placed, the next component is mounted on top of the first one, and so on, until we finish that fixture's *sub-assembly*. Subsequently, one sub-assembly is picked up from its fixture, and merged with the other sub-assembly, to form the final assembly, using each component type once. The final assembly is then picked up from its fixture and put on a output tray for further processing, and the assembly process starts over. The output tray is placed at the right side of the robot, not shown in Figure 1.

One should note that it is possible to use one fixture to finish the entire assembly, and this is probably how humans would do it. However, since the robot arms work in parallel, and only one robot hand can work at each fixture at a time, a second fixture is often used to shorten the makespan. So, in this study we use two fixtures. It is also worth noting that there could be scenarios where

the component mounting orders are not fixed, however this study is restricted to cases where the ordering *is* fixed.

The YuMi Robot: Our particular assembly cell contains a YuMi [1], a robot from ABB with two *arms* having one *hand* each. Each hand, shown in Figure 2, features one *gripper* tool and two *suction cup* tools. These tools are designed to *pick*, move and then *place* objects. An object being a component, or a (sub-)assembly. The (empty) gripper tool is also used to *press* components after placing, to ensure good fit. A gripper tool *picks* objects with its two parallel “fingers”. A suction cup tool picks objects by adhering to a flat surface. The object, the application and the robot programmer determine what tool to use for each object. When holding an object with the gripper, the position and orientation, or *pose*, are sufficiently well known to place it. This is not true for the suction cup tool, however the object pose can be determined by processing an image of the object in the tool. Thus, when using suction cup tool, a *camera* task must be performed at any of the five camera focal point, introduced previously.

Note that one tool can hold one object, independently of the other tools. This means that one robot hand can hold up to three objects simultaneously. This fact is often used by experienced programmers to improve the makespan.

Robot programming. The software controlling the robot and cameras is called the *robot program*. It consists of two sequences of tasks, one per arm. Each arm program alternates between performing a task and moving to the next task. The tasks mentioned thus far are *pick*, *camera*, *place*, *press*. Each task is itself a program, provided by the robot programmer. However, for the abstraction level of this work, a task is a constraint on the duration of said task. Similarly, travel between task locations is abstracted as a constraint on the travel time between said locations.

In summary, the robot program must obey task orders at the fixtures. The program must be realizable. That is, no more than one component per tool, and perform pick-(camera)-place with the same arm, in that order. Finally, the program must avoid arm-to-arm collisions. A robot programmer often spends considerable effort to keep arms apart.

To allow the arms to work independently of each other, the optimization model of this work relies on the following assumptions on tasks and travel. (1) Parallel work: the model assumes that any task at one fixture, performed by one arm, does not interfere with any task at the other fixture, performed by the other arm. (2) Elbows out: if the left robot hand is located to the left of the right robot hand, then no part of the arms are colliding. (3) No zig-zag: travel between fixtures and other locations does not pass through volumes to the left (right) of that location when using the right (left) arm.

There is a small (implicit) volume around each fixture, dedicated to the arm working at the fixture. When a task is done, the robot arm leaves this volume, *as a part of the task implementation* (and task duration). This makes sure no collisions occur when tasks are executed by different arms at the same fixture.

Fig. 2. A robot hand with one gripper tool and two suction cup tools

3 Modeling

Our CP model captures the dual robot arm program for manufacturing one assembly. It captures variable task locations, thus optimizing tray layout and which camera to use while avoiding arm collisions. It is a scheduling model that borrows from vehicle routing models, augmented with variable locations.

The model features a set $\mathbb{T} = \{0, \dots, T-1\}$ of tasks, a set $\mathbb{L} = \{0, \dots, L-1\}$ of locations, a set $\mathbb{W} = \{0, \dots, W-1\}$ of time slots, and a set $\mathbb{H} = \{\text{Gripper}, \text{SuctionCup}\}$ of robot hand tools.

For task t , we have decision variables $\text{arm}_t \in \{0, 1\}$, the arm performing t ; $\text{succ}_t \in \mathbb{T}$, the successor task of t ; $\text{location}_t \in \mathbb{L}$, the location of t ; $\text{load}_t^\tau \in \mathbb{N}$, the load of tool τ when arriving at t ; $\text{travel_cost}_t \in \mathbb{W}$, the travel time from t to its successor; $\text{duration}_t \in \mathbb{W}$, the duration of processing t ; $\text{arrival}_t \in \mathbb{W}$, the time at which the arm is ready to perform t ; $\text{start}_t \in \mathbb{W}$, the time at which t starts processing; and $\text{end}_t \in \mathbb{W}$, the time at which t finishes processing. For many tasks, the location is fixed up front. There can be a waiting period between the arrival time and the start time.

We use successor variables to capture task order, i.e., $\text{succ}_i = j$ means that task j is the successor of task i . The model includes $T-4$ actual tasks and four dummy tasks, with the convention that $T-4$ and $T-2$ are the dummy start and end tasks of arm 0, $T-3$ and $T-1$ are the dummy start and end tasks of arm 1, the start task of arm 1 is the successor of the task of arm 0, and vice versa. The total set of tasks thus form a Hamiltonian circuit. Let $\mathbb{T}_{\text{start}} = \{T-4, T-3\}$ and $\mathbb{T}_{\text{end}} = \{T-2, T-1\}$ denote the start and end tasks, respectively. Let $\mathbb{T}_{\text{TrayPick}} \subset \mathbb{T}$ denote the set of tasks that occur at a tray location.

The application requires both robot arms to return to the original position to be ready for the next assembly, so we must account for the travel time from the last actual task back to the original position. To achieve this, we constrain the locations of the end task, the start task, and the first actual task to coincide.

The objective is to minimize the makespan, i.e., the maximal completion time of the two arms. Instance data are encoded in part by a set of functions:

$$\begin{aligned}
W_L(a, s, g) &\in \mathbb{Z} = \text{travel time from source location } s \text{ to target location } g \\
&\quad \text{using arm } a, \text{ or } -1 \text{ if infeasible} \\
W_D(a, t) &\in \mathbb{N} = \text{duration of task } t \text{ for arm } a \\
P_L(l) &\in \mathbb{Z} = \text{relative position of location } l \text{ in robot workspace, left to right,} \\
&\quad \geq 0 \text{ indicating relative positions, } -1 \text{ indicating shared space} \\
O(t, t') &= \begin{cases} 2, & \text{if task } t \text{ must precede task } t' \text{ using the same arm,} \\ 1, & \text{if task } t \text{ must precede task } t', \text{ but } t, t' \text{ may use different arms,} \\ 0, & \text{otherwise} \end{cases} \\
\Delta(t, \tau) &= \begin{cases} 1, & \text{if task } t \text{ uses tool } \tau \text{ to pick a component or (sub)-assembly,} \\ -1, & \text{if task } t \text{ uses tool } \tau \text{ to place it,} \\ 0, & \text{otherwise} \end{cases} \\
\text{Cap}(t, \tau) &\in \mathbb{N} = \text{the max possible load for tool } \tau \text{ when arriving to task } t
\end{aligned}$$

The model's constraints are grouped as follows:

Successors. Recall that the tasks should form a Hamiltonian circuit, which can be seen as arm 0's task sequence followed by arm 1's task sequence, linked via the dummy tasks. Of course, the same arm must be used within the given task sequence. Finally, the robot arms must return to their original position. This all is encoded by the following constraints:

$$\begin{aligned}
& \mathbf{Circuit}([succ_t \mid t \in \mathbb{T}]), \\
& arm_t = arm_{succ_t} \quad \forall t \in \mathbb{T} \setminus \mathbb{T}_{\text{end}}, \\
& succ_{T-2} = T-3, succ_{T-1} = T-4, \\
& arm_{T-4} = arm_{T-2} = 0, arm_{T-3} = arm_{T-1} = 1, \\
& location_{succ_{T-4}} = location_{T-4} = location_{T-2}, \\
& location_{succ_{T-3}} = location_{T-3} = location_{T-1}
\end{aligned} \tag{1}$$

Time Constraints. Travel costs depend on locations and arm, and durations depend on task and arm:

$$\begin{aligned}
travel_cost_t &= W_L(arm_t, location_t, location_{succ_t}) \quad \forall t \in \mathbb{T} \setminus \mathbb{T}_{\text{start}} \setminus \mathbb{T}_{\text{end}}, \\
duration_t &= W_D(arm_t, t) \quad \forall t \in \mathbb{T} \setminus \mathbb{T}_{\text{start}} \setminus \mathbb{T}_{\text{end}}
\end{aligned} \tag{2}$$

The following relates each actual and dummy task's time variables. Note that this forms the link between the routing and scheduling parts of the model. Together with the routing constraints, this prevents tasks, including the arms' travel time, from overlapping:

$$\begin{aligned}
& arrival_t \leq start_t \quad \forall t \in \mathbb{T}, \\
& start_t + duration_t = end_t \quad \forall t \in \mathbb{T}, \\
& end_t + travel_cost_t = arrival_{succ_t} \quad \forall t \in \mathbb{T}, \\
& duration_t = travel_cost_t = 0 \quad \forall t \in \mathbb{T}_{\text{start}} \cup \mathbb{T}_{\text{end}}, \\
& arrival_t = start_t = end_t = arrival_{succ_t} \quad \forall t \in \mathbb{T}_{\text{start}} \cup \mathbb{T}_{\text{end}}, \\
& arrival_t = 0 \quad \forall t \in \mathbb{T}_{\text{start}}
\end{aligned} \tag{3}$$

Precedence Constraints. Precedences at fixtures are imposed by constraints on the time intervals of tasks. Precedences among pick-(camera)-(other)-place tasks are handled similarly. Note that the same arm is required for each such task, and so the inequation can be stricter and include all time spent on a task:

$$\begin{aligned}
& end_t \leq start_{t'} \quad \forall t, t' \in \mathbb{T} \text{ where } O(t, t') = 1, \\
& arrival_{succ_t} \leq arrival_{t'} \wedge arm_t = arm_{t'} \quad \forall t, t' \in \mathbb{T} \text{ where } O(t, t') = 2
\end{aligned} \tag{4}$$

Capacity Constraints. The $load_t^\tau$ variables are related via the $\Delta(t, \tau)$ values:

$$\begin{aligned}
& load_t^\tau = 0 \quad \forall \tau \in \mathbb{H} \quad \forall t \in \mathbb{T}_{\text{start}} \cup \mathbb{T}_{\text{end}}, \\
& load_{succ_t}^\tau = load_t^\tau + \Delta(t, \tau) \quad \forall t \in \mathbb{T} \quad \forall \tau \in \mathbb{H}
\end{aligned} \tag{5}$$

Tools cannot exceed their physical capacity, and for some tasks, they must not be occupied:

$$load_t^\tau \leq Cap(t, \tau) \quad \forall t \in \mathbb{T} \quad \forall \tau \in \mathbb{H} \quad (6)$$

Location Constraints. Each $location_t$ is initialized with a given subset of $\mathbb{L}$. For this model, all task locations are fixed, except tray locations and camera locations. There is only room for one tray at each tray location:

$$\text{AllDifferent}([location_t \mid t \in \mathbb{T}_{\text{TrayPick}}]) \quad (7)$$

Finally, to prevent the arms from colliding, we split the robot workspace into non-shareable zones and shareable zones. For every pair of tasks that use different arms and placed in non-shareable zones, we require the the left arm (arm 0) to be left of the right arm (arm 1):

$$\begin{aligned} arm_p = 0 \wedge arm_q = 1 &\implies \\ P_L(location_p) = -1 \vee P_L(location_q) = -1 &\vee \\ P_L(location_p) < P_L(location_q) &\quad \forall p, q \in \mathbb{T} \end{aligned} \quad (8)$$

4 Evaluation

We evaluate dual arm, multi tool, assembly programs using real world data. The real world data consists of the possible locations of all task types, the travel times of manually generated paths between all locations, for a workspace of a real case, as well as task durations for each arm. The problem instances are solved by the model presented in Section 3. All instances are solved using single thread Gecode 6.2.0 on Intel Core i7-6850 3.6 GHz CPU. The primary decisions of our search are load balanced task assignment to arms, using a mix of static and dynamic variable order, and a mix of static and random value selection.

The number of tasks of a problem instance is mainly dependent on a) number of components, and b) whether a component is picked using suction cup or gripper. We evaluated instances of 3, 4 and 5 components, using suction cup vs. gripper for different components, and equipping the robot with 1 or 2 suction cup tools per arm. This results in 11 problem instances of 17 to 26 tasks.

For each problem instance we generated 8 variants with nominal travel times randomly perturbed by $\pm 25\%$, each variant solved 10 times by using different randomization seeds for value selection in the search. The problem instances are named e.g. 5C4S-2 meaning 5 components, out of which 4 are handled by suction cup with 2 suction cups per arm.

We present the runtime distribution solving each instance to optimality as a box plot extending from the first and third quartile ($Q1, Q3$), with a line indicating the median. The whiskers extends to at most the range $[Q3 + 1.5 \cdot (Q3 - Q1), Q1 - 1.5 \cdot (Q3 - Q1)]$. Data outside this range are shown as outliers.

As a complementary perspective on the optimization progress, we report runtime distributions to reach optimality gaps 0.05 and 0.10 as well, i.e., the

time at which the makespan is 5% and 10% away from the optimal value. This information can be used for selecting timeouts in a real-world scenario.

Most runs finish within seconds to a few minutes, with the notable exception of the case this work was inspired by, and also the largest case: 5C4S-2. Having two suction cup tools per arm gives more flexibility and opens up for better throughput, however, inspecting instances *-1 vs *-2 in Fig. 3 it is clear that this incurs an increased optimization time.

Fig. 3. Optimization time and time to reach optimality gap 0.05 and 0.10.

We also repeated all the experiments relaxing constraint (8), thus ignoring arm-to-arm collisions. Very much against the working hypothesis, for every problem instance the optimal makespan was unaffected by whether or not constraint (8) was active. Note that, if some or all locations become fixed, this is definitely not the case.

5 Conclusions and Future Work

The contributions of this work include: a first optimization-based model that integrates all aspects of the problem, an evaluation that shows that the model delivers good solutions quickly and is able to solve all evaluated instances to optimality, and an approach, based on spatial separation, to handling arm-to-arm collisions without any observed degradation of solution quality.

Current results are promising. Future work includes more than two robot arms, and explore how this affects arm-to-arm collision avoidance. It would be interesting to involve other characteristics, scale up to larger problem instances and optimize over the number of fixtures used.

Acknowledgements. This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP).

References

1. ABB AB: The ABB YuMi industrial robot, <http://new.abb.com/products/robotics/industrial-robots/yumi>
2. Behrens, J.K., Lange, R., Mansouri, M.: A constraint programming approach to simultaneous task allocation and motion scheduling for industrial dual-arm manipulation tasks. *International Conference on Robotics and Automation (ICRA)* (2019). <https://doi.org/10.1109/ICRA.2019.8794022>
3. Burgard, W., Cremers, A.B., Fox, D., Hähnel, D., Lakemeyer, G., Schulz, D., Steiner, W., Thrun, S.: The interactive museum tour-guide robot p. 1118 (1998). <https://doi.org/10.5555/295240.295249>
4. Ejenstam, J.: Implementing a Time Optimal Task Sequence For Robot Assembly Using Constraint Programming. Master's thesis, Department of Information Technology, Uppsala University, Sweden (2014), <http://urn.kb.se/resolve?urn=urn:nbn:se:uu:diva-229581>
5. F, L., Dimitrov, D., Saffiotti, A., Karlsson, L.: Constraint propagation on interval bounds for dealing with geometric backtracking. In *Proc. of IEEE/RSJ Int Conf. on Intelligent Robots and Systems* (2012). <https://doi.org/10.1109/IROS.2012.6385972>
6. Fikes, R.E., Nilsson, N.J.: Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* **2**(3), 189 – 208 (1971). [https://doi.org/10.1016/0004-3702\(71\)90010-5](https://doi.org/10.1016/0004-3702(71)90010-5)
7. Garrett, C.R., Lozano-Pérez, T., Kaelbling, L.P.: Ffrob: Leveraging symbolic planning for efficient task and motion planning. *CoRR* **abs/1608.01335** (2016). <https://doi.org/10.1177/0278364917739114>
8. ICAPS: Icaps competitions, <http://icaps-conference.org/index.php/Main/Competitions>
9. Kaelbling, L.P., Lozano-Perez, T.: Hierarchical task and motion planning in the now pp. 1470–1477 (2011). <https://doi.org/10.1109/ICRA.2011.5980391>
10. Knutar, F.: Automatic Generation of Assembly Schedules for a Dual-Arm Robot Using Constraint Programming. Master's thesis, Department of Information Technology, Uppsala University, Sweden (2015), <http://urn.kb.se/resolve?urn=urn:nbn:se:uu:diva-276340>
11. Li, J., Harabor, D., Stuckey, P., Ma, H., Koenig, S.: Symmetry-breaking constraints for grid-based multi-agent path finding. *Proceedings of the AAAI Conference on Artificial Intelligence* **33**, 6087–6095 (07 2019). <https://doi.org/10.1609/aaai.v33i01.33016087>
12. Lozano-Pérez, T., Kaelbling, L.P.: A constraint-based method for solving sequential manipulation planning problems. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems* (2014). <https://doi.org/10.1109/IROS.2014.6943079>
13. Mösenlechner, L., Beetz, M.: Parameterizing actions to have the appropriate effects. In *Proc. of IEEE/RSJ Intl Conf. on Intelligent Robots and Systems* (2011). <https://doi.org/10.1109/IROS.2011.6048523>
14. Mösenlechner, L., Beetz, M.: Fast temporal projection using accurate physics-based geometric reasoning. In *Proc. of IEEE Int Conf. on Robotics and Automation* (2013). <https://doi.org/10.1109/ICRA.2013.6630817>
15. Pecora, F., Mansouri, M., Hawes, N., Kunze, L.: Special issue on reintegrating artificial intelligence and robotics. *Knstliche Intelligenz* **33**(4), 315–317 (2019), <https://doi.org/10.1007/s13218-019-00625-x>
16. Pecora, F.: Is model-based robot programming a mirage? a brief survey of AI reasoning in robotics. *Knstliche Intelligenz* **28**(4), 255–261 (2014). <https://doi.org/10.1007/s13218-014-0325-0>

17. Rethink Robotics: The Rethink Robotics Sawyer robot, <https://www.rethinkrobotics.com/sawyer/>
18. Toussaint, M.: Logic-geometric programming: An optimization-based approach to combined task and motion planning. In: In Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI) (2015), <https://dl.acm.org/doi/10.5555/2832415.2832517>
19. Yu, J., LaValle, S.M.: Structure and intractability of optimal multi-robot path planning on graphs. Proceedings of the AAAI Conference on Artificial Intelligence (2013). <https://doi.org/10.1609/aaai.v33i01.33016087>