Optimizing Modular Robot Composition: A Lexicographic Genetic Algorithm Approach

Jonathan Külz and Matthias Althoff

Abstract— **Industrial robots are designed as general-purpose hardware with limited ability to adapt to changing task requirements or environments. Modular robots, on the other hand, offer flexibility and can be easily customized to suit diverse needs. The morphology, i.e., the form and structure of a robot, significantly impacts the primary performance metrics acquisition cost, cycle time, and energy efficiency. However, identifying an optimal module composition for a specific task remains an open problem, presenting a substantial hurdle in developing task-tailored modular robots. Previous approaches either lack adequate exploration of the design space or the possibility to adapt to complex tasks. We propose combining a genetic algorithm with a lexicographic evaluation of solution candidates to overcome this problem and navigate search spaces exceeding those in prior work by magnitudes in the number of possible compositions. We demonstrate that our approach outperforms a state-of-the-art baseline and is able to synthesize modular robots for industrial tasks in cluttered environments.**

I. Introduction

Modular robots (MRs) are an intuitive solution to an increasing need for mass customization and small-scale manufacturing [1]–[3]. Thanks to their versatility and robustness by design, modular manipulators promise significant technological advances in industrial automation [4], [5]. As shown in Fig. 1, diverse tasks require different compositions of robot modules. One of the main challenges with MRs is determining a robot composition, i.e., selecting and arranging modules that suit the given requirements. Due to the exponential number of possible permutations of modules, an exhaustive search for the optimal modular composition is usually infeasible. Meta-heuristics such as deep reinforcement learning or genetic algorithms (GAs) have been successfully adapted to facilitate MR design optimization [6], [7]. However, these approaches require evaluating an immense number of robot designs, rendering their straightforward application to complex tasks impossible and resulting in long run times.

A. Contribution

We propose a novel GA-based approach for MR composition optimization on heterogeneous modules driven by a lexicographic fitness function. By hierarchically assessing the fitness of solution candidates, we combine the exploratory capabilities of GAs with the computational efficiency of hierarchical elimination methods while enhancing interpretability compared to a single scalar fitness value. As a result, in contrast to prior approaches, our GA (a) considers computationally expensive metrics, such as trajectory cycle time, during optimization, (b) works in arbitrary environments with any number of workspace targets, and (c) imposes no constraints on the robot structure except that a serial kinematic has to be assembled. We shot that our approach finds module compositions and trajectories in cluttered and complex environments tailored to task-specific requirements and outperforms a state-of-the-art benchmark regarding robot complexity and cycle time. All tasks and solutions discussed are available in the CoBRA benchmark [8], together with several interactive visualizations[1]. The project's website and additional videos can be found at `https://lexicographic-ga.cps.cit.tum.de`.

B. Related Work

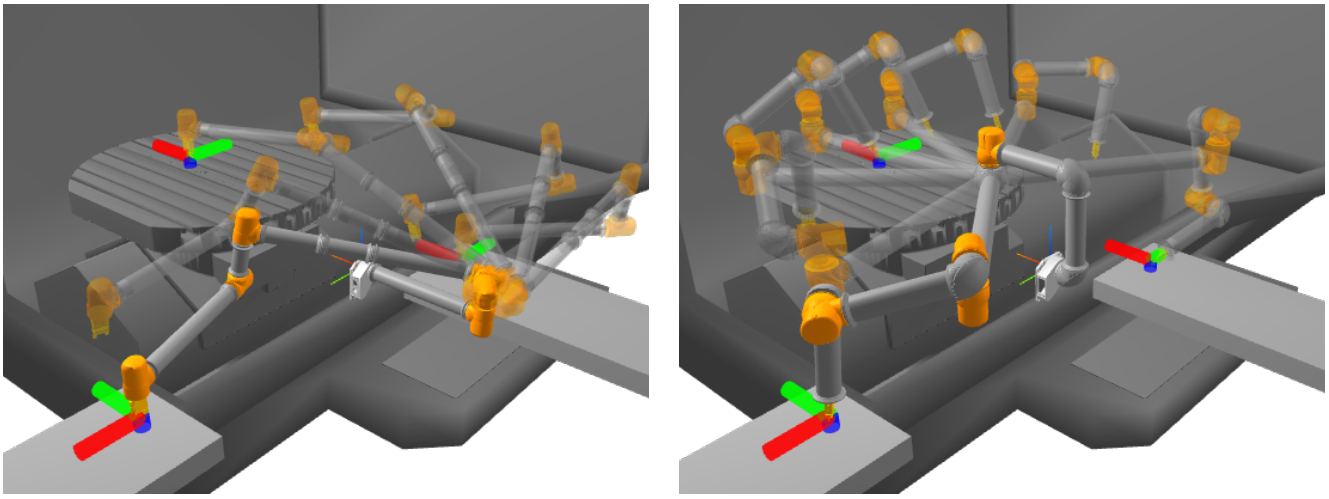
Modular robots were introduced decades ago, starting in the 1980s [9]–[11]. While the automatic generation of kinematic and dynamic models [12], [13] and their self-programming [5] has incentivized the introduction of new robot hardware [14]–[16], the optimization of their modular composition still poses a considerable challenge.

As exhaustive approaches usually fail, one potential solution to finding optimized compositions is limiting the search space, for example, by focusing on morphologies commonly seen in monolithic manipulators [17]. Alternatively, meta-heuristics, such as simulated annealing, can be applied to deal with the large search space [18]. Mixed integer programming [19] or an augmented Lagrangian technique [20] were introduced to optimize for the static reachability of MR without considering possible robot motion. The work in [6] deploys a heuristic search leveraging reinforcement learning to solve tasks with multiple goals and obstacles. However, due to the large number of evaluations necessary for training, the authors have to simplify the tasks by discretizing the space occupied by obstacles, training the agent on tasks with one goal only, and solving for reachability only instead of performing motion planning.

By applying hierarchical elimination, i.e., evaluating robots on a sequence of increasingly computationally expensive criteria, the authors of [5], [21], [22] significantly speed up the assessment of the feasibility of a particular MR for a given task. However, these works fail to navigate the vast search space effectively, as they do not harness the information acquired during hierarchical elimination, nor do they employ information about the robots themselves

The authors are with the Department of Computer Engineering, Technical University of Munich, 85748 Garching, Germany. `jonathan.kuelz@tum.de, althoff@tum.de`.

[1]Visit `https://cobra.cps.cit.tum.de/tasks` and search for tasks with the name `kuelz_lexicographic_ga`.



(a) For narrow orientation tolerances at the goal positions, our optimization yields a robot with six degrees of freedom.

(b) An MR with just four degrees of freedom solves the task if the orientation tolerances at the desired goal poses are broad.

Fig. 1. Adapting task-tailored MR compositions leads to optimized setup and production costs: In the example shown above, a small change in the task requirements results in two robots of significantly different complexity. While a six-degree-of-freedom robot is necessary to reach all desired goal positions exactly (a), a relaxation of orientation tolerances allows us to design an MR with four degrees of freedom only to solve the task (b).

strategically to guide the search process, resulting in the necessity of imposing structural constraints on the module composition.

Genetic algorithms (GAs) [23] are another frequently used approach to optimizing MR composition. By mimicking natural evolutionary processes, genetic algorithms can be utilized to maximize their fitness for a given environment [24], [25]. In a GA, a gene sequence composes a chromosome representing a solution candidate. Starting from an initial population, a new generation of chromosomes is recursively created by applying crossover, mutation, and selection operations to the previous population. Previous work, such as in [26], has explored GA-based optimization of robot compositions. However, this research (a) focuses on a limited subset of robot modules, (b) does not consider different or mobile bases, (c) imposes strict constraints on their structure, i.e., the number of joints and the ordering of joint and link modules, and (d) simplifies performance metrics and omits complex factors like the computation of a workspace trajectory, which is crucial in real-world applications. These challenges are common in related studies [6], [27]–[29]. In contrast, the authors of [7] propose a two-step approach involving the elimination of infeasible MRs through hierarchical elimination. This approach realizes motion planning on a reduced set of promising MRs. Still, it imposes constraints on the number of robot modules and their degrees of freedom to manage the search space.

II. Problem Statement

Throughout this work, we aim to find a composition of heterogeneous modules that can be assembled to a robot to solve a given task. We assume that a module m has b bodies connected by $b-1$ joints and proximal and distal connectors c_p and c_d, respectively, defining interfaces to connect it to other modules. Our method explicitly permits the inclusion of empty bodies, enabling the creation of more complex module structures. For instance, this flexibility allows one to combine two linear joints with an empty body to construct a planar joint, which can be used to model a mobile base module. We consider two special types of modules, bases and end effectors. For base modules, c_p represents the reference frame of the base; for end effector modules, c_d defines the tool center point (TCP). Any connector is attached to a body and has a fixed type. The corresponding modules can be connected if a distal and a proximal connector have the same type. In this work, we consider any MR that adheres to the following structure:

$$\textit{Base - } M \textit{ - ... - } M \textit{ - End Effector}$$

Here, M are arbitrary regular modules, so the structure fits any serial manipulator with exactly one base and end effector module. An assembled composition of n_M modules is referred to as robot $R = (m_1, \ldots, m_{n_M})$.

A task T, as displayed in Fig. 2, is defined by the tuple $\langle G, \mathcal{T}, \mathcal{O} \rangle$, where $G = (g_1, \ldots, g_{n_G})$ is a sequence of goals, $\mathcal{T}$ is a set of tolerances for all goals, and $\mathcal{O}$ is the workspace occupied by obstacles. Joint limits for a robot with n_J joints are given by lower limits $\underline{\boldsymbol{q}} \in \mathbb{R}^{n_J}$ and upper limits $\overline{\boldsymbol{q}} \in \mathbb{R}^{n_J}$. We define the set of all valid joint configurations as $\mathcal{Q} = \left[\underline{\boldsymbol{q}}, \overline{\boldsymbol{q}}\right]$. A goal $g \in G \subset SE3$ is defined as a position $\boldsymbol{p}_g \in \mathbb{R}^3$ and a desired orientation $\boldsymbol{n}_g \in SO3$. We introduce the operator

$$rot(\boldsymbol{n_1}, \boldsymbol{n_2}) = \langle \boldsymbol{e}, \theta \rangle, \|\boldsymbol{e}\|_2 = 1, \theta \in [0, \pi] \tag{1}$$

that returns the rotation from $\boldsymbol{n_1}$ to $\boldsymbol{n_2}$ in axis-angle representation, defined as a unit vector $\boldsymbol{e}$ and an angle θ. Further, we define the forward kinematics (FK) for robot R with joint angles $\boldsymbol{q}$ as

$$[\boldsymbol{p}_{\text{TCP}}, \boldsymbol{n}_{\text{TCP}}] = FK(R, \boldsymbol{q}), \tag{2}$$

with TCP position $\boldsymbol{p}_{\text{TCP}} \in \mathbb{R}^3$ and TCP orientation $\boldsymbol{n}_{\text{TCP}} \in SO3$. The workspace occupied by the robot is given by $\mathcal{A}(R, \boldsymbol{q})$. The tolerances $\mathcal{T}$ are composed of a position tolerance $t_p \in \mathbb{R}_+$ and an orientation tolerance, given as the tuple $t_o = \langle \boldsymbol{t}, \varphi \rangle, \boldsymbol{t} \in \{\boldsymbol{x} \in \mathbb{R}^3_+ \mid \|\boldsymbol{x}\|_\infty \leq 1\}, \varphi \in (0, \pi]$, where $\boldsymbol{t}$ defines the upper bound on the absolute value of the Euler axis of rotation between $\boldsymbol{n}_{\text{TCP}}$ and $\boldsymbol{n}_g$ (3).

A goal g can be be reached by robot R, if there exists a joint configuration $\boldsymbol{q} \in \mathcal{Q}$ such that

$$\|\boldsymbol{p}_{\text{TCP}}(R, \boldsymbol{q}) - \boldsymbol{p}_g\|_2 \leq t_p \wedge \theta \ |\boldsymbol{e}| \leq \varphi \ \boldsymbol{t}\,, \tag{3}$$

where $\langle \boldsymbol{e}, \theta \rangle = rot(\boldsymbol{n}_g, \boldsymbol{n}_{\text{TCP}})$. In (3), $|\boldsymbol{e}|$ denotes the element-wise absolute value and the inequality holds if it is true for all dimensions. We denote this predicate as $r(R, \boldsymbol{q}, g, \mathcal{T})$, which evaluates to `true` if robot R reaches goal g with the joint configuration $\boldsymbol{q}$ and `false` otherwise. We also introduce the predicate $\hat{r}$ that determines whether a trajectory $\boldsymbol{q}(t)$ reaches all goals in the order specified by G as

$$\begin{aligned} \hat{r}(R, \boldsymbol{q}(t), G, \mathcal{T}) = \exists\,(t_1, \ldots, t_n) : \\ t_1 \leq t_2 \leq \cdots \leq t_n = t_{max} \\ \wedge\, \forall i \in \{1, \ldots, n\} : r(R, \boldsymbol{q}(t_i), g_i, \mathcal{T})\,. \end{aligned} \tag{4}$$

A task T is considered to be achieved by robot R, if there exists a trajectory $\boldsymbol{q}(t)$ that is collision-free (5a), for which joint velocity (5b), acceleration (5c), and torque limits (5d) are met, and for which all goals are reached in the intended order (5e). We denote the set of all trajectories fulfilling these properties by:

$$\chi_T(R) = \{\boldsymbol{q}(t) : [0, t_{max}] \to \mathcal{Q} \mid \tag{5}$$

$$\mathcal{O} \cap \mathcal{A}(R, \boldsymbol{q}(t)) = \emptyset \tag{5a}$$

$$\wedge\, \dot{\boldsymbol{q}}(t) \in \left[\, \underline{\dot{\boldsymbol{q}}}, \dot{\overline{\boldsymbol{q}}} \,\right] \tag{5b}$$

$$\wedge\, \ddot{\boldsymbol{q}}(t) \in \left[\, \underline{\ddot{\boldsymbol{q}}}, \ddot{\overline{\boldsymbol{q}}} \,\right] \tag{5c}$$

$$\wedge\, |\boldsymbol{\tau}| \leq \boldsymbol{\tau}_{max} \tag{5d}$$

$$\wedge\, \hat{r}(R, \boldsymbol{q}(t), G, \mathcal{T}\}\,. \tag{5e}$$

If $\chi_T(R) \neq \emptyset$, we assume there is a path planning algorithm that computes a trajectory $\boldsymbol{q}(t)$ for task T with the robot R. Under this assumption, we can define our objective function

$$C_T(R) = w_s C_s(R) + w_p C_{p,T}(R, \boldsymbol{q}(t))\,, \tag{6}$$

which is composed of a weighted sum of robot setup costs C_s (such as acquisition cost or module availability) and process cost $C_{p,T}$ (such as cycle time or energy consumption) arising during operation. Consequently, the robot defined by the optimal composition is given by

$$R_T^* = \arg\min_R (C_T(R)),\ \text{s.t. } \boldsymbol{q}(t) \in \chi_T\,. \tag{7}$$

III. Method

In a GA, multiple solution candidates (*chromosomes*) form the population, which is altered via the genetic operators in every generation. To encode a manipulator as chromosome c of fixed length n_c, we index the set of available modules $\mathcal{M}$ and write $c = (m_1, \ldots, m_{n_c}), m_i \in \{0, 1, \ldots, |\mathcal{M}|\}$ to encode the sequence of assembled module identifiers. A gene encodes an empty slot when set to zero ($m_i = 0$) and a regular module from $\mathcal{M}$ otherwise. In a slight abuse of notation, we will use a gene m_i and the encoded module M_i interchangeably in the remainder of this paper. A series of genes $(u, 0, v)$ encodes a partial configuration of modules (M_u, M_v). This flexibility allows us to encode any manipulator with n_M modules in a chromosome of consistent length $n_c \geq n_M$. As a result, we neither predetermine the number of modules in a solution candidate nor constrain the search space to alternate joint and link modules, allowing us to explore unconventional compositions.

A. Genetic Operators

Within every generation, we compute the fitness value for every chromosome. Based on the fitness, we perform steady state *selection*: The p solution candidates (population) with the highest fitness among all individuals are chosen to be the parents for the next generation. We then replace any chromosome not selected by a new one created by a single-point *crossover* between two randomly selected parents. We denote the set of all modules with the same connectors by $\mathcal{V}_m$ and, finally, select a replacement candidate uniformly from $\mathcal{V}_m$ for every gene in the current population with a *mutation* probability of p_m. If the mutated chromosome m_i is neither a base nor an end effector ($i \notin \{1, n_c\}$), the empty module is added to $\mathcal{V}_m$ if a connection between the adjacent modules (m_{i-1}, m_{i+1}) is possible. Due to the inherent validity check of connections during population generation, we can incorporate constraints on module composition, such as those given by connector properties inherently, and before computing the fitness. Therefore, even complex module libraries, e.g., modules of different sizes, can be optimized using our GA.

B. Fitness Function

As the fitness function f determines the quality of a solution candidate, we expect the relation $C_T(R_i) < C_T(R_j) \Leftrightarrow f(R_i) > f(R_j)$ between fitness and task cost to hold. However, due to the computationally expensive evaluation of the cost function in (6) that requires attempting to compute a solution trajectory, it is infeasible to use it directly as a fitness function. Given that our selection process relies on the order of fitness scores rather than their exact numerical values, we implement a lexicographic fitness function [30]: We select a sequence of fitness objectives $f(R) = \langle f_1(R), \ldots, f_n(R) \rangle$, ranked descending by importance and computational simplicity, and define the ordering

$$\begin{aligned} f(R_a) > f(R_b) \Leftrightarrow & \exists k \in \{1, \ldots, n\} : f_k(R_a) > f_k(R_b) \\ & \wedge\, \forall i < k : f_i(R_a) = f_i(R_b)\,, \end{aligned} \tag{8}$$

$$f(R_a) = f(R_b) \Leftrightarrow \forall i \in \{1, \ldots, n\} : f_i(R_a) = f_i(R_b)\,. \tag{9}$$

As an illustrative example, let us consider two robots, R_1 and R_2, and evaluate them on two objectives. The first objective, $f_1(R)$, is a binary classifier determining whether a trajectory

$\boldsymbol{q}(t) \in \chi_T(R)$ exists for each robot. The second objective, $f_2(R) = -n_J(R)$, is defined as the negative of the number of joints for each robot. Applying a lexicographic ordering, R_1 is preferred over R_2 if a trajectory exists for R_1 but not for R_2 or if a trajectory exists for both robots but R_1 has fewer joints than R_2. Specifically, we introduce the following objectives for a robot R:

1) Trivial reachability: A goal can only be reached if a robot arm is sufficiently long. We introduce the maximum Euclidean distance between the distal and the proximal connector of module M_i as $d_{max}(M_i)$ and define

$$f_1(R) = \begin{cases} 1, \text{ if } \sum_R d_{max}(M_i) \geq \max_{g \in G} \|\boldsymbol{p}_g\|_2 \\ 0, \text{ otherwise.} \end{cases} \tag{10}$$

2) Reachable number of goals: Objective

$$f_2(R) = \sum_{g \in G} \left(\begin{cases} 1, \text{ if } \exists \boldsymbol{q} \in \mathcal{Q} : r(R, \boldsymbol{q}, g, \mathcal{T}) \\ 0, \text{ otherwise} \end{cases} \right) \tag{11}$$

returns the number of goal positions that can be reached by the robot R. The existence of a valid joint configuration $\boldsymbol{q}$ is determined using a numeric inverse kinematics algorithm.

3) Reachable number of goals, considering collisions: Objective

$$f_3(R) = \sum_{g \in G} \left(\begin{cases} 1, \text{ if } \exists \boldsymbol{q} \in \mathcal{Q} : & r(R, \boldsymbol{q}, g, \mathcal{T}) \\ & \wedge \mathcal{O} \cap \mathcal{A}(R, \boldsymbol{q}) = \emptyset \\ 0, \text{ otherwise} \end{cases} \right) \tag{12}$$

returns the number of goal positions that can be reached while there are no collisions between robot and environment.

4) Cost objective: The final fitness objective measures robot setup complexity and task performance and can be chosen freely. Following (6), (7), we define

$$f_4(R) = \begin{cases} -\infty, & \text{if } \chi_T(R) = \emptyset \\ -C_T(R), & \text{otherwise.} \end{cases} \tag{13}$$

We incrementally compute the lexicographic fitness value by subsequently evaluation these criteria and stopping if a composition does not achieve the maximum fitness value $f_i(R) \neq f_{i,max}$ for an intermediate objective $i \leq 3$, where $f_{1,max} = 1$, $f_{2,max} = f_{3,max} = |G|$. This can be interpreted as a hierarchical pruning procedure, backed by computational evidence about a robot's performance. However, unlike prior work using hierarchical elimination to optimize MR compositions [7], we retain information about partial solutions, such as the ability to reach the final goal in the task and use it for the selection process. Moreover, the pruning relies exclusively on task performance, avoiding a human bias introduced by manually crafted rules and thereby preserving the explorative capabilities of GAs. Finally, the introduction of a lexicographic fitness function enhances interpretability. Rather than managing and fine-tuning weighted sums for numerous task criteria for a scalar fitness value, a human evaluator can easily conclude which constraints different solution candidates satisfy.

IV. Experiments

A. Robot Modules

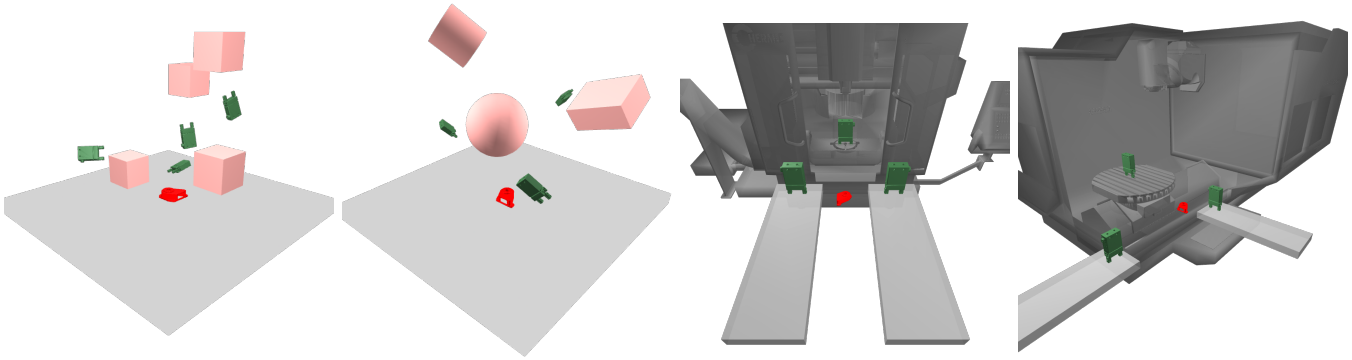
For our experiments, we used data from 29 different modules manufactured by RobCo[2]. The set consisted of four bases, 20 static link modules, four modules with a joint, and one end effector. The static link modules differed in size and shape (L-shaped and I-shaped), and the bases differed in size and mounting orientation. We divided the database into six large modules, including two bases, and 22 smaller modules, including the end effector that can only be attached to one of the large modules using a special connecting link module. Without the constraints imposed by the connectors and the necessity of a base and end effector module, there would have been $\sum_{i=n}^{12} 29^n \approx 10^{17}$ distinct compositions with at most twelve modules that could be built from this module set. Considering the different module sizes and types, the resulting size of the workspace was still beyond 10^{12} (one trillion) possible compositions for a chromosome length of twelve and, to the best of our knowledge, exceeds those in prior work, such as 15552 in [7], 32768 in [5], and around 10^6 in [6] significantly.

B. Task Definition

We evaluated our approach on two types of tasks: Randomly generated synthetic tasks and manually curated industrial manufacturing tasks. We assessed the algorithm's performance for each type in two different settings and three difficulty levels each. Fig. 2 shows an example for every type of task.

For the first setting (Synthetic I), we implemented a scenario similar to the one proposed in [6] by discretizing a space of 1.25m^3, centered in the point $[0, 0, 0.625]^T\text{m}$ in $5 \times 5 \times 5$ voxels of edge width 0.25m. For the different difficulty levels, we randomly sampled $d \in \{3, 4, 5\}$ box-shaped obstacles, filling one voxel each and d goal positions centered in a voxel each. In addition, we created non-discretized synthetic tasks (Synthetic II), where goal and obstacle position and orientation were sampled randomly in a half-ball of radius $1.2m$ and with a positive z coordinate. In this setting, obstacles could be spheres, boxes, or cylinders, and their position and volume were determined randomly. For all goals in the synthetic tasks, we set the orientation tolerance to half a degree around an arbitrary axis ($\boldsymbol{e} = [1, 1, 1]^T, \varphi = \frac{\pi}{360}$). Finally, we applied a simple, non-exhaustive heuristic to discard tasks that are not solvable, e.g., because obstacles covered one of the goal positions or the base position. In total, we defined 120 synthetic tasks, 20 for each setting with $d \in \{3, 4, 5\}$.

[2] https://www.robco.de/en



(a) Synthetic I, $d = 4$ (b) Synthetic II, $d = 3$ (c) Manufacturing I (d) Manufacturing II

Fig. 2. We evaluated our algorithm on two synthetic and two manufacturing settings: Synthetic obstacles are shown in red, milling machines and conveyor belts are shown in grey. Every goal consists of a desired position and orientation, indicated by an end effector. The robot base placement is indicated by a red base module.

Furthermore, we evaluated our approach in two settings inspired by real-world machine tending problems (Manufacturing I & II), where the goals represent a pick position for raw material, a position inside a milling machine, and a position to place the final workpiece on. For each one, a difficulty level was determined by using one of three orientation tolerances, mimicking different workpiece geometries:

- Sphere-like geometry ($t_o = t_{o,1}$): The TCP orientation can be chosen freely within an arbitrary rotation of 90°, so we set $\boldsymbol{e}_1 = [1, 1, 1]^T, \varphi_1 = \frac{\pi}{2}$.
- Partially symmetric geometry ($t_o = t_{o,2}$): The TCP orientation is free around its z-axis but otherwise constrained, so we set $\boldsymbol{e}_2 = [\frac{1}{360}, \frac{1}{360}, 1]^T, \varphi_2 = \pi$.
- Arbitrary geometry ($t_o = t_{o,3}$): The TCP orientation is pre-determined and we allow a minor deviation only, so we set $\boldsymbol{e}_3 = [1, 1, 1]^T, \varphi_3 = \frac{\pi}{360}$.

For all tasks, we set a position tolerance of $t_p = 10^{-3}$m.

C. Baseline

We compared our approach to a two-level GA based on hierarchical elimination, as proposed in [7]. We eliminated unfit solution candidates that could not reach the first and last goal as indicated by (12) before evaluation. Instead of utilizing the previously introduced lexicographic fitness function, we adopted [7, (3)-(10)]. Consequently, the baseline fitness function depended on weighting factors k_i, linear distance L, angular distance A, dexterity D, number of modules n_M, number of joints n_J, joint value differences V, and the percentage of reachable intermediate points P:

$$f_B = e^{-(k_1 R + k_2 L + k_3 A + k_4 D + k_5 n_M + k_6 n_J + k_7 V)} + k_8 P\,. \tag{14}$$

We waived the obstacle proximity criterion initially introduced in the referenced work, which is defined solely for spherical obstacles, and replaced the involved module criterion I by the number of links n_M and the number of joints n_J. All other criteria remained as defined in the reference. In the second optimization step, we selected the best 25 individuals for each scenario and computed trajectories using the same method as employed for our solutions.

D. Setup

All experiments were conducted using Timor-Python [29] on a desktop PC with an Intel i7-11700KF processor. For trajectory generation, we followed the method from [31] in combination with the RRT-Connect planner provided by the Open Motion Planning Library (OMPL) [32] with a timeout of 3 seconds. To enhance the efficiency of evaluating recurring module compositions, we optimized the process by caching the kinematic and dynamic models derived from module data for the most recent 1000 robots assessed. However, due to the stochastic nature of the RRT-Connect planner, we calculated the fitness value for each composition, regardless of whether it had been evaluated before.

We defined the optimization criterion as a function of the number of robot joints n_J, robot modules n_M, and trajectory cycle time t_{max} as:

$$C_T = n_J + 0.2 n_M + t_{max}\,. \tag{15}$$

The algorithms ran for 200 generations on a population size of 25, which was initialized using weighted random sampling, where each initial module had a probability of $p_j = 0.9$ to contain a joint. For evaluating the performance on the manufacturing tasks, we performed optimization over ten random seeds, whereas due to run time constraints, we ran optimization once on all synthetic tasks. If not stated otherwise, all results refer to the mean computed over the difficulty levels $d \in \{3, 4, 5\}$ for synthetic tasks and for tolerances $t_{o,1}$, $t_{o,2}$, and $t_{o,3}$ for the manufacturing tasks with differing seeds.

Additionally, to analyze the the ability of both algorithms to adapt to different preferences, we evaluated them in the "Manufacturing II" setting with tolerance $t_{o,3}$ and cost functions $C_T(w_J) = w_J n_J + (5 - w_J) t_{max}$, where we

TABLE I

SUMMARY OF ACHIEVED TASKS, AVERAGE NUMBER OF JOINTS, AND AVERAGE CYCLE TIME.

Task Setting		Achieved	C_T	n_J	t_{max}
Synthetic I	$d=3$	16	11.86	6.2	3.1s
	$d=4$	10	15.16	6.8	5.7s
	$d=5$	10	16.42	6.7	7.1s
Synthetic II	$d=3$	20	12.27	6.4	3.3s
	$d=4$	20	13.80	6.7	4.6s
	$d=5$	20	15.33	6.8	5.9s
Synthetic (average over all)		**16.0**	**14.14**	**6.6**	**5.0s**
Baseline		15.7	19.61	8.0	14.9s
Manufacturing I	$t_{o,1}$	10	12.81	5.3	5.0s
	$t_{o,2}$	10	14.33	6.7	5.0s
	$t_{o,3}$	10	15.05	6.6	5.8s
Manufacturing II	$t_{o,1}$	10	10.15	4.5	3.4s
	$t_{o,2}$	10	12.48	6.4	3.6s
	$t_{o,3}$	10	12.27	6.0	3.8s
Manufacturing (average over all)		**10.0**	**12.85**	**5.9**	**4.5s**
Baseline		10.0	18.17	7.8	12.9s

C_T: Cost as defined in eq. (6), n_J: Number of joints, t_{max}: Cycle time

evaluated both algorithms' performance over ten random seeds for weights $w_J \in \{0, 1, \ldots, 5\}$.

E. Results

The evaluation took $960ms$ per individual on average, resulting in 1.2 hours for one optimization over 200 generations. This is a significant speedup compared to determining the cost function for every individual and running into the path planner's 3s time limit for every unfit solution candidate. Table I reports the number of tasks achieved in different settings and the average lowest cost for all tasks with a valid solution. In addition, we report the mean number of joints n_J and cycle time t_{max} for all solutions with minimal cost, as well as the aggregated mean over all synthetic and manufacturing tasks for our approach and the baseline according to section IV-C.

Our algorithm found a valid solution for 80% of the synthetic tasks. Upon manual inspection, we discovered that some remaining tasks are inherently unsolvable due to the proximity of goals and obstacles. For the hand-curated industrial tasks, our optimization resulted in valid solutions for every trial. On average, our algorithm yielded solutions with costs about 30% lower than those generated by our baseline method. For the synthetic tasks, our approach reduced the average number of joints by 1.4 and decreased cycle times by 66%. In manufacturing tasks, we observed an average reduction of 1.9 joints and a decrease in cycle times by 65%. Notably, the robots' complexity changed with the task difficulty. Two different robots for the "Manufacturing II" setting with $t_o = t_{o,1}$ and $t_o = t_{o,3}$ are shown in Fig. 1.

The effect of different cost functions on the optimization outcome is shown in Fig. 3. For all weighting factors w_J, the proposed approach outperformed the baseline significantly. The results of our approach show a tradeoff between the

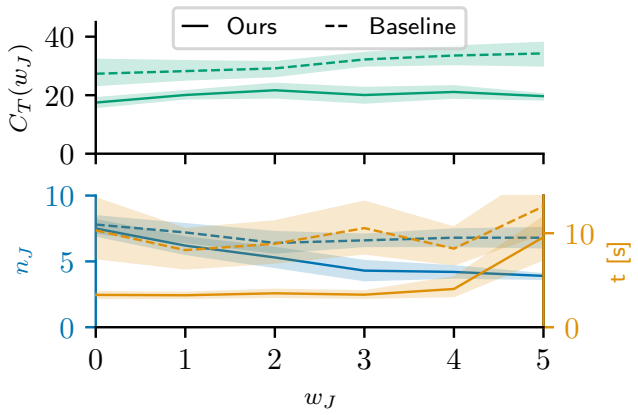


Fig. 3. Our approach outperformed the baseline regardless of the weighting factor w_J. Especially for our approach, the tradeoff between n_J and t becomes visible. Shaded areas indicate 95% confidence intervals computed using bootstrapping.

number of joints and the trajectory cycle time. The baseline procedure found solutions with fewer joints if incentivized but failed to reduce the cycle time for lower w_J. This result strongly indicates that including trajectory cycle time in the optimization process, as described in (13), provides a significant advantage over the two-level approach deployed for the baseline.

V. CONCLUSION

This paper presents a genetic algorithm with a lexicographic fitness function for the task-based optimization of modular manipulators. Compared to existing approaches, we impose no prior assumptions on the structure or the number of modules in an optimal solution. Our experimental validation shows that our GA can find solutions adapted to the complexity of a task. Moreover, the unconventional but high-performant module compositions resulting from optimizing MR for manufacturing tasks with broad orientation tolerances showcase that use-case-tailored MRs do not necessarily follow human intuition. Our approach applies to all kinds of serially connected MRs, regardless of module complexity. Additional task constraints can easily be integrated by extending the intermediate objectives evaluated during fitness computation. Lastly, our method can be used for any task without the necessity of simplification, e.g., via discretization.

ACKNOWLEDGMENT

The authors gratefully acknowledge financial support by the Horizon 2020 EU Framework Project CONCERT under grant 101016007.

REFERENCES

[1] M. R. Pedersen, L. Nalpantidis, R. S. Andersen, C. Schou, S. Bøgh, V. Krüger, and O. Madsen, "Robot skills for manufacturing: From concept to industrial deployment," *Robotics and Computer-Integrated Manufacturing*, vol. 37, pp. 282–291, 2015.

[2] D. Gorecky, S. Weyer, A. Hennecke, and D. Zühlke, "Design and instantiation of a modular system architecture for smart factories," *IFAC-PapersOnLine*, vol. 49, no. 31, pp. 79–84, 2016.

[3] L. Ribeiro and M. Bjorkman, "Transitioning from standard automation solutions to cyber-physical production systems: An assessment of critical conceptual and technical challenges," *IEEE Systems Journal*, vol. 12, no. 4, pp. 3816–3827, 2018.

[4] M. Yim, W.-M. Shen, B. Salemi, D. Rus, M. Moll, H. Lipson, E. Klavins, and G. S. Chirikjian, "Modular self-reconfigurable robot systems [grand challenges of robotics]," *IEEE Robotics and Automation Magazine*, vol. 14, no. 1, pp. 43–52, 2007.

[5] M. Althoff, A. Giusti, S. B. Liu, and A. Pereira, "Effortless creation of safe robots from modules through self-programming and self-verification," *Science Robotics*, vol. 4, no. 31, 2019.

[6] J. Whitman, R. Bhirangi, M. Travers, and H. Choset, "Modular robot design synthesis with deep reinforcement learning," in *Proc. of the AAAI Conf. on Artificial Intelligence (AAAI)*, vol. 34, no. 06, 2020, pp. 10 418–10 425.

[7] E. Icer, H. Hassan, K. El-Ayat, and M. Althoff, "Evolutionary cost-optimal composition synthesis of modular robots considering a given task," in *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2017, pp. 3562–3568.

[8] M. Mayer, J. Külz, and M. Althoff, "CoBRA: A composable benchmark for robotics applications," in *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2024.

[9] G. Pritschow and K. Tuffentsammer, "A new modular robot system," *CIRP Annals*, vol. 35, no. 1, pp. 89–92, 1986.

[10] T. Fukuda, S. Nakagawa, Y. Kawauchi, and M. Buss, "Self organizing robots based on cell structures – CEBOT," in *Proc. of the Int. Workshop on Intelligent Robots (IROS)*, 1988, pp. 145–150.

[11] D. Schmitz, P. Khosla, and T. Kanade, "The CMU reconfigurable modular manipulator system," Tech. Rep. 88–7, 1988.

[12] C. Nainer, M. Feder, and A. Giusti, "Automatic generation of kinematics and dynamics model descriptions for modular reconfigurable robot manipulators," in *IEEE Int. Conf. on Automation Science and Engineering (CASE)*, vol. 17, 2021, pp. 45–52.

[13] T. Zhang, Q. Du, G. Yang, C. Wang, C.-Y. Chen, C. Zhang, S. Chen, and Z. Fang, "Assembly configuration representation and kinematic modeling for modular reconfigurable robots based on graph theory," *Symmetry*, vol. 14, no. 3, 2022.

[14] A. Yun, D. Moon, J. Ha, S. Kang, and W. Lee, "ModMan: An advanced reconfigurable manipulator system with genderless connector and automatic kinematic modeling algorithm," *IEEE Robotics and Automation Letters*, vol. 5, no. 3, pp. 4225–4232, 2020.

[15] E. Romiti, J. Malzahn, N. Kashiri, F. Iacobelli, M. Ruzzon, A. Laurenzi, E. M. Hoffman, L. Muratore, A. Margan, L. Baccelliere, S. Cordasco, and N. Tsagarakis, "Toward a plug-and-work reconfigurable cobot," *Transactions on Mechatronics*, vol. 27, no. 5, pp. 2319–3231, 2022.

[16] A. Dogra, S. Padhee, and E. Singla, "Optimal synthesis of unconventional links for modular reconfigurable manipulators," *Journal of Mechanical Design*, vol. 144, no. 8, 2022.

[17] S. B. Liu and M. Althoff, "Optimizing performance in automation through modular robots," in *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2020, pp. 4044–4050.

[18] C. J. J. Paredis and P. K. Khosla, "Synthesis methodology for task based reconfiguration of modular manipulator systems," in *Proc. of the Int. Symp. on Robotics Research (ISRR)*, 1993.

[19] A. Valente, "Reconfigurable industrial robots: A stochastic programming approach for designing and assembling robotic arms," *Robotics and Computer-Integrated Manufacturing*, vol. 41, pp. 115–126, 2016.

[20] S. Singh, A. Singla, and E. Singla, "Modular manipulators for cluttered environments: A task-based configuration design approach," *Journal of Mechanisms and Robotics*, vol. 10, no. 5, 2018.

[21] E. Icer and M. Althoff, "Cost-optimal composition synthesis for modular robots," in *Proc. of the IEEE Conference on Control Applications (CCA)*, 2016, pp. 1408–1413.

[22] E. Icer, A. Giusti, and M. Althoff, "A task-driven algorithm for configuration synthesis of modular robots," in *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2016, pp. 5203–5209.

[23] D. E. Goldberg and J. H. Holland, "Genetic algorithms and machine learning," *Machine Learning*, vol. 3, pp. 95–99, 1988.

[24] K. Sims, "Evolving virtual creatures," in *Proc. of the Ann. Conf. on Computer Graphics and Interactive Techniques (SIGGRAPH)*, vol. 21, 1994, pp. 15–22.

[25] H. Lipson and J. B. Pollack, "Automatic design and manufacture of robotic lifeforms," *Nature*, vol. 406, pp. 974–978, 2000.

[26] J. Han, W. K. Chung, Y. Youm, and S. H. Kim, "Task based design of modular robot manipulator using efficient genetic algorithm," in *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, 1997, pp. 507–512.

[27] O. Chocron and P. Bidaud, "Evolutionary algorithms in kinematic design of robotic systems," in *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, vol. 2, 1997, pp. 1111–1117.

[28] O. Chocron, "Evolutionary design of modular robotic arms," *Robotica*, vol. 26, no. 3, pp. 323–330, 2008.

[29] J. Külz, M. Mayer, and M. Althoff, "Timor Python: A toolbox for industrial modular robotics," in *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2023, pp. 424–431.

[30] C. A. Coello, "An updated survey of GA-based multiobjective optimization techniques," *ACM Computing Surveys*, vol. 32, no. 2, pp. 109–143, 2000.

[31] T. Kunz and M. Stilman, "Time-optimal trajectory generation for path following with bounded acceleration and velocity," in *Robotics: Science and Systems*, 2012.

[32] I. A. Şucan, M. Moll, and L. E. Kavraki, "The open motion planning library," *IEEE Robotics and Automation Magazine*, vol. 19, no. 4, pp. 72–82, 2012.