





# Domain Optimization for Hierarchical Planning based on Set-Theory

Bernd Kast<sup>1</sup><sup>a</sup>, Vincent Dietrich<sup>1</sup><sup>b</sup>, Sebastian Albrecht<sup>1</sup><sup>c</sup>, Wendelin Feiten<sup>1</sup><sup>d</sup>  
and Jianwei Zhang<sup>2</sup>

<sup>1</sup>Siemens AG, Corporate Technology, Otto-Hahn-Ring 6, 81739 Munich, Germany

<sup>2</sup>University of Hamburg, Faculty of Mathematics, Informatics and Natural Sciences,  
Vogt-Kölln-Str. 30, 22527 Hamburg, Germany

Keywords: Hierarchy, Planning, Autonomy, Robotic Assembly.

Abstract: The design of planning domains for autonomous systems is a hard task, especially when different parties are involved. We present a domain optimization algorithm for hierarchical planners that uses a set-based formulation. Due to an automatic alignment we can compose models from different sources to a larger domain for efficient planning. The combination of domain optimization and hierarchical planning can handle large scale domains very efficiently. Our algorithm reduces the effects of the non-optimality that comes with the hierarchical approach. We demonstrate the scalability with a task and motion planning problem. In the scenario of a robotic assembly with up to 62 parts and plan lengths of over 1000 steps the planning times are kept within 15 minutes. We show the execution of our plans on a real-world dual-robot setup.

## 1 INTRODUCTION

Robotic systems bring together hardware and software components from different sources to solve a specific task. Only for recurring tasks it is viable for an engineer to compose the components and write or parametrize algorithms to manage the different pieces in a meaningful way.


However, even in an industrial environment, the cost of setting up the system can easily exceed the savings by the automated process. This is especially true for smaller lot sizes or even lot size one production. In order to address the automation of such a highly flexible production, we need algorithms for the composition of such systems and decision making during their runtime.


The algorithms for decision making have to handle a mixed problem, which is depicted in Figure 1 with continuous geometric and discrete properties, like attachment status or grasps. These planning problems become large for non-trivial tasks which results in unreasonable computation times due to the curse of dimensionality. This can be handled with a hierarchical approach, as presented in (Kast et al., 2019b).





Figure 1: The core operations of the process: assembly, screw, handover, and place. Each action is planned and executed with the hierarchical planner.

In an industrial environment several different parties, such as integrators or component suppliers, with varying user roles are involved to define modules, objectives, and available resources. It must be possible to provide the planner with these different pieces of information in a modular way while each module might implement its own modeling scheme. These different styles of expressing the declarative and procedural knowledge should not affect the performance of the online planning.

<sup>a</sup> <https://orcid.org/0000-0001-7838-3142>

<sup>b</sup> <https://orcid.org/0000-0003-0568-9727>

<sup>c</sup> <https://orcid.org/0000-0002-3647-4043>

<sup>d</sup> <https://orcid.org/0000-0002-7593-6298>

In this paper we propose a domain optimization algorithm that aligns the models for efficient planning. We can either apply this algorithm on-demand, prior to the planning, or as an offline step. We rely on our set-theoretic foundation to reformulate the declarative and procedural knowledge of the domain without infringing their validity. We analyze the performance of the optimization in a simulated setup with 800 test runs and tasks of varying lengths. The final experiment is conducted on a real two-arm robotic system.

## 2 RELATED WORK

There are two strands in the literature that target domain optimization problems.

A recently very active branch are data-driven algorithms, notably reinforcement learning approaches, which optimize heuristics for a specific domain. These methods show very good performances for some easy to simulate problems, like board games (Silver et al., 2016), (Silver et al., 2017) or computer games (Mnih et al., 2013), (Vinyals et al., 2019). However, application on real-world scenarios are still difficult due to the limited data available. Attempts to overcome this include large scale pick and place setups with hundreds of robots (Kalashnikov et al., 2018) and, due to the difficult nature of physics, hard but rather short tasks (Xie et al., 2019).

In (Schmitt et al., 2019) reinforcement learning is used in combination with an abstraction layer. This layer allows eased simulation, ensures viability, like collision-free movements, and provides an interface for real-world execution that handles small deviations. Still, training requires large datasets and processing power. Additionally, the trained model is a black box and thus hard to debug or transfer. In our approach we rely on models and rules rather than single data points that define the behavior of the resulting system. The explicit representation enables introspection, which is a key feature during development and for industrial environments. Additionally, less computational power is needed for our approach. However, it lacks theoretical optimality and still requires experts to program. In (Nägele et al., 2018) another approach, which relies on domain specific heuristics is proposed. In this case, however, the heuristics are computed online by analyzing the desired goal. Geometric interdependencies are broken up and the resulting plan is executed in simulation.

Another strand of related work covers optimization of modeled domains. In (Kang and Nnaji, 1993) a scheme for aligned and manually designed domains is elaborated. This, however, covers a single domain

only and brings in its benefits only when strictly followed. In a real-world scenario different parties bring in their modules, which are used for completely different problems as well, to compose the overall domain. Therefore, it is very hard to align everyone to a common scheme. In our approach each party fulfills their user-role with the representation they prefer. Just before planning, we harmonize the representations automatically.

For this type of optimization many algorithms that operate on discrete, mainly PDDL domains have been proposed. Two strands can be identified for algorithms which either transform the problem to suite the planner, or pick a planner, which can cope with the characteristics of the original problem.

Portfolio planners, such as (Seipp et al., 2012), (Katz et al., 2018) apply different planners with different heuristics on the domain and try to switch to the most appropriate combination for the current problem formulation.

In (Haslum et al., 2007), (Vallati et al., 2015) automated optimization algorithms for PDDL-domains are proposed, which allow even non-experts to apply generic planning algorithms on general PDDL-domains. In (Areces et al., 2014) a formalization for an optimization scheme was found, that was formerly applied manually on the domains. This even allows an engineer to inspect the optimized formulation to validate and further optimize it easily. However, it comes with the difficulties and limitations of PDDL to handle continuous domains.

## 3 APPROACH

Before we discuss our new domain optimization algorithm, we highlight the properties of its set-theoretic foundation, which was introduced in (Kast et al., 2019a).

Our new algorithm is effective in combination with a hierarchical planner only. We point out the relevant properties of an example hierarchical planning algorithm, which was proposed in (Kast et al., 2019b).

### 3.1 Declarative Knowledge

Following the line of (Kast et al., 2019a), we call elements of the declarative knowledge *concepts*. For us, a concept  $C$  is a subset of some concept base  $B_\Gamma$  which is a set of instances that is not necessarily finite. Concepts with the same concept base  $B_\Gamma$  belong to the same concept class  $\Gamma$ . The partial order *more-detailed than*  $\mathcal{M}_\Gamma$  between two concepts holds true if each instance described by the first concept is also an

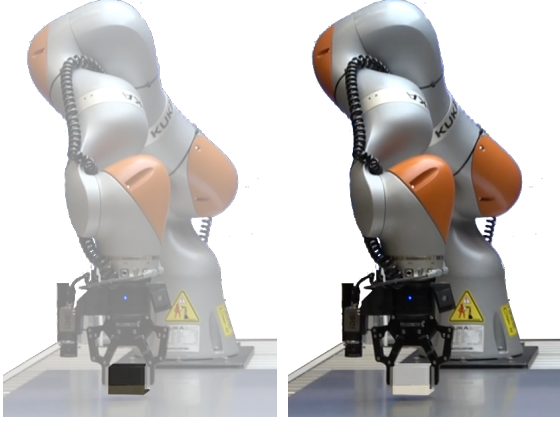


Figure 2: Either the Box is the main actor and the robot only the supporting actor, thus every property is described product-centric, or it is vice versa and the robot has properties that describe the objects in its grippers.

element of the second one. Properties that are common to a set of instances define *composite concepts*. These properties in turn can be expressed by value ranges or sets, which are concepts themselves:

$$C \cong \prod_{r \in R_C} C_r,$$

where  $C_r$  with  $r \in R_C$  are the sub-concepts and the role set  $R_C$  defines the composite structure.

Examples to this set-theoretic view, which we will reconsider in subsection 3.4, are the concept bases of *Objects and Manipulators* with exemplary concepts of a box or a robotic arm.

The box with no property but an identifier can be detailed by concepts that define dimensions or its position, like *on the table*, *grasped by the robot* with even more-detailed concepts that specify continuous positions relative to some coordinate system.

The concept of the robot is detailed by concepts that specify its current position and state. Part of the state is the content of the gripper, which can be an object, like an instance of a box, or it is empty as nothing is currently grasped.

This example shows that some phenomena can equivalently be expressed by instances of concepts with different basic types, e.g. a robot holding a box or a box in a robot's gripper as visualized in Figure 2.

### 3.2 Procedural Knowledge

Planning is all about actions, which are represented by the procedural knowledge. Only the procedures make the declarative knowledge useful. We call these building blocks of planning and execution *operators*  $\pi$ . They map between given instances of input concepts to instances of output concepts:

$$\pi : \prod_{r_i \in R_{\pi, \text{in}}} C_{r_i} \rightarrow \prod_{r_j \in R_{\pi, \text{out}}} C_{r_j},$$

where the sets  $R_{\pi, \text{in}}$  and  $R_{\pi, \text{out}}$  describe the roles of the respective input and output concepts. The mapping can either be specified explicitly by a formula or implicitly by a black box of code, some library, or even a simulated or real-world experiment.

A set of operators is more-detailed than another operator  $\tilde{\pi}$ , when a sequence or network  $\hat{\pi}$  of these operators exists with matching, more detailed outputs compared to the original operator and inputs, which are either a subset of the original inputs or orthogonal to all of them, i.e. have a different concept bases and are therefore independent to each other:

$$\forall r_i \in R_{\tilde{\pi}, \text{in}} \exists r_j \in R_{\tilde{\pi}, \text{in}} : (\hat{C}_{r_i}, \tilde{C}_{r_j}) \in \mathcal{M}_{\Gamma(\tilde{C}_{r_j})},$$

$$\forall r_i \in R_{\tilde{\pi}, \text{out}} \exists r_j \in R_{\hat{\pi}, \text{out}} : (\hat{C}_{r_i}, \tilde{C}_{r_j}) \in \mathcal{M}_{\Gamma(\tilde{C}_{r_j})},$$

and for all  $r_i \in R_{\tilde{\pi}, \text{in}}$  holds:

$$\begin{aligned} & |\{r_j \in R_{\tilde{\pi}, \text{in}} \mid \Gamma(\tilde{C}_{r_j}) \subseteq \Gamma(\hat{C}_{r_i})\}| \\ &= |\{r_j \in R_{\hat{\pi}, \text{in}} \mid \Gamma(\hat{C}_{r_j}) \subseteq \Gamma(\tilde{C}_{r_i})\}|. \end{aligned}$$

The more-detailed operators can consider additional information and are in general more costly to be applied as they implement a more comprehensive simulation or even real-world execution to tell the outcome of the action.

This partial ordering is later used by the hierarchical planner to define new sub-planning problems.

### 3.3 Hierarchical Planner

The hierarchical planner, which we proposed in (Kast et al., 2019b), is a forward state space planner that can handle domains with both, discrete and continuous properties efficiently. The basic idea of the planner is to divide the overall planning problem into small subproblems repeatedly, such that the curse of dimensionality can be alleviated. We do this by planning in an abstract domain first. In this domain the goal can be reached with a relatively small number of steps, as the abstracted operators cover huge changes of the state. Additionally, the branching factor is small due to the smaller number of possible operators that can be applied in that domain.

Once we found a solution on the coarse level, each operator that was applied in this plan by itself defines a new sub-planning problem with its inputs as starting values and outputs as goals. The operators that can be applied in this new, refined domain are the more-detailed operators as described in subsection 3.2. We apply this process recursively to each of the newly generated planning problems until there

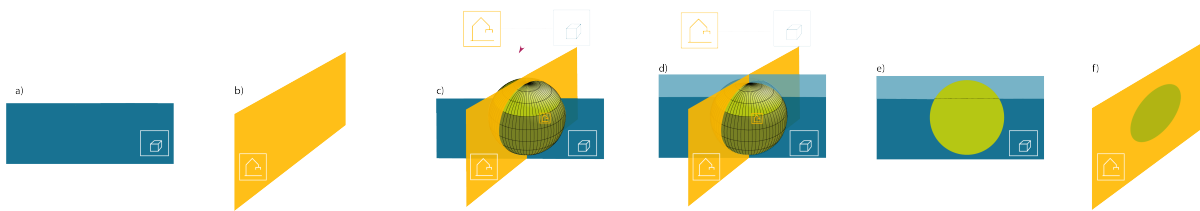


Figure 3: Geometric representation of the connection between two sets. The dark blue plane segment depicts the concept base for all objects (a) while the yellow plane represents the manipulators concept base all robots belong to (b). The intersection is a sphere (green). The projection of the sphere to the box-segment would miss the light green volume (c). This is due to a possibly empty list of grasped objects for the robot, which has no representation in the original box-set. Therefore, our algorithm extends the box-plane with the light blue segment depicting the "no object" set (d). After this extension, the volume can either be projected on the combination of the two blue plane segments (e) or the robot plane (f). This enables a reformulation of the planning domain.

is no further refinement for the operators. As the abstracted domains and their planned solutions can only be approximations of the real behavior, there can be errors and unsolvable subtasks on any level. Our solution to avoid dependency on the downward refinement property is backtracking, which means that plans on the abstract level are dismissed and new sequences to the goal are recalculated if a refinement fails. In our system, execution is the final refinement and error handling a case of backtracking.

Therefore the planning approach represents a model predictive control scheme. Both execution and error handling are first class citizens with our planning approach. Under optimal conditions, when the coarse level is a good approximation to the behavior of the real-world, our planning approach can scale linearly with the length of the task. This, however, holds only true if the downward refinement property is always guaranteed. For a bad approximation the strategy still grows exponentially as the full problem is np-hard.

Additionally, our solutions are not necessarily optimal. It depends on the modeling of the coarse domains to propose good intermediate goals for factorization to have overall solutions close to the optimum. This, however, can be a burden to the engineering, especially when different people model the levels of the domain according to their respective user roles.

### 3.4 Optimization Algorithm

The key idea of our domain optimization algorithm is to align the different levels of abstraction. This is especially important, when engineers with differing viewpoints on the problem, for example due to different user roles, model different parts of the domain. As described in subsection 3.3 the key to an efficient factorization with our hierarchical planner are the intermediate goals that arise from the abstract level plan.

Remember that we need a sequence of operators

on the refined level, which produces more detailed instances than the coarser level. According to our definition described in subsection 3.1, a concept, and therefore an instance of that concept as well, cannot be more-detailed than another, if their concept bases differ.

Consider the example of the box and the robot in subsection 3.1. Both can describe the same phenomenon, but one from a product-centric view and the other from an actor or tool-based angle. As they use different concept bases, the robot with the box cannot be a solution to the box in the gripper of the robot nor of any of the abstractions of the box with the robot. However, it is quite likely that the engineer modeling the coarse processing steps has a product-centric view on the plant, while the person that designs the cell with the robot or some other machining equipment has a tool-based angle.

During planning the coarse level would define intermediate goals, which are only reachable for the refined domain, if box and robot are separated. This causes additional steps to be planned and executed, which results in solutions that are farther off the optimal plan. To overcome this issue, we can identify concepts  $C \cong \prod_{r \in R_C} C_r$  having a sub-concept  $C_j, j \in R_C$  with a concept base matching our overall goal, i.e.  $B_\Gamma(C_j) \subseteq B_\Gamma(C_{\text{goal}})$ . Then we reformulate these concepts such that they have the same concept base as the goal concept while the set that they represent is identical: Assume that  $C_j \cong \prod_{r_j \in R_{C_j}} C_{r_j}$  then define the reformulated concept by

$$C' := \prod_{r_j \in R_{C_j}} C_{r_j} \times \prod_{r \in R_C \setminus \{j\}} C_r.$$

This concept is isomorph to a subset of  $C_j$  and thus has the concept base  $B_\Gamma(C_j)$ . This is the formal description of the isomorphism described in subsection 3.1. With this transformation, which is depicted in Figure 3, we can directly derive intermediate goals and can successfully compare newly created instances with our existing subset, as the concept bases match.

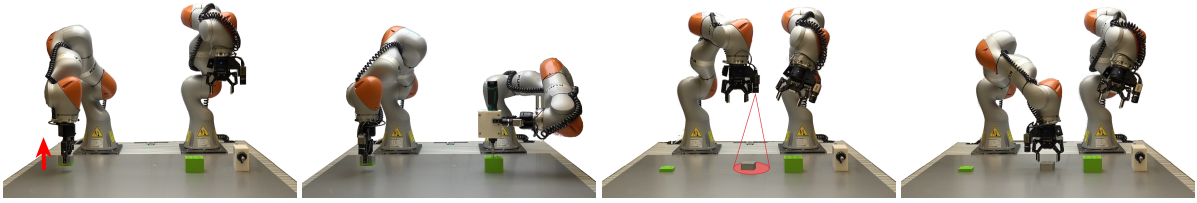


Figure 4: Extension to Figure 1 of important actions. Picking of the assembled box, localization of objects, screw out from the magazine and initial pick of the box.

However, this is only possible if all elements of the concept can be expressed in the other concept base, which is not always true. Consider the example where one concept is a composite concept having an array of another concept type as a sub-concept. For our robot-box-example, the composite concept with an array is the robot and elements in this array are of type box. Then not all composite concepts with the array can be expressed equivalently by the sub-concept, because the array might be empty, i.e. the robot's gripper is empty.

Therefore, we cannot express some instances directly in the form of the first concept base. To overcome this, we expand the first concept base with a set which contains the empty set, i.e.

$$B_{\Gamma}^* := B_{\Gamma}(C_j) \cup \{\emptyset\}.$$

Afterwards corresponding sub-concepts of all elements of that extended concept base can have additional attributes:

$$C' := \{\emptyset\} \times \prod_{r \in R_C \setminus \{j\}} C_r.$$

For example, a *no-box-object* can have the attribute *robot*. This allows to cover empty arrays as well and therefore include all relevant instances.

We then generated wrapping operators which replace the original ones that have optimizable input or output concepts. They internally call the original operators but change the type of the inputs and outputs, such that the aligned and newly generated concepts are returned. For a domain, which is optimized this way, the intermediate goals defined by the abstract layers for the refinement layers are relaxed, while the specified overall goals remain firm. Therefore, the overall result of the plan is the same for the original and the optimized domain, while possibly less steps are required for the optimized domain.

## 4 EXPERIMENT

In the real-world we conduct our experiment on a dual arm robot with a total of 14 degrees of freedom. Each of the arms has a two-finger gripper and

a RGB-camera that is used to refine the object poses. The workspace is observed with a stationary 3D camera, which is used to estimate the objects starting locations. We use a cordless screwdriver which is remotely controlled.

The assembly scenario we use to analyze the performance of our algorithm includes the mounting of a lid to a box and the insertion of multiple screws to join them. The relevant actions that need to take place are depicted in Figure 1 and Figure 4. It is necessary to refine the object positions with one of the wrist-mounted cameras, before any interaction takes place, which means prior to grasping or picking up screws. Fastening the screws can only happen with the box fixated in one gripper and the screwdriver actuated with the other arm. The objects (screwdriver and box with lid) can be placed on the table which can create loops in the state space. The box can additionally be handed over between the two arms to change the relative orientation in the gripper.

We generate collision-free robot trajectories with the constraint-based solver described in (Schmitt et al., 2019). This allows for good reachability with relatively few intermediate positions for motion planning.

The configuration of the perception system is grounded on (Dietrich et al., 2018).

We can vary the number of screws that are inserted to modify the difficulty of the task. The real box has only four screw holes, which results in maximum of six parts that can be assembled. In simulation we added virtual screw holes such that we can examine scalability of our approach for up to 62 pieces.

We analyze the performance of the optimization algorithm on a domain that was first designed on an abstract, discrete layer only and later extended with the continuous layers. There are a total of 4 modeled abstraction layers for the task at hand. They include two completely abstract levels, with and without localization of parts, simulated and real-world execution. Only the first layer is workpiece centric, while the others are robotic centric. The robot can localize each object in its workspace, pick the box, place it on the lid, localize the merged pieces again to pick both up together. The other arm can then pick the

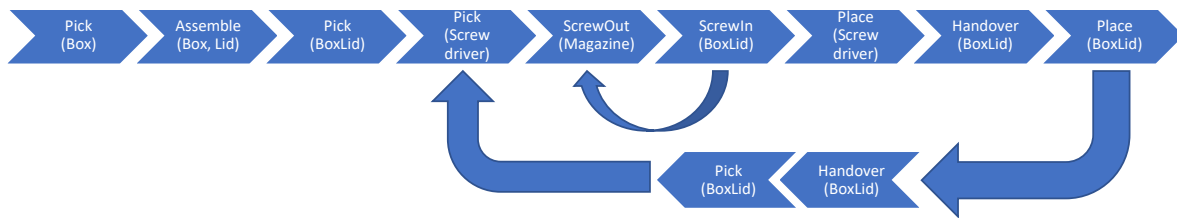


Figure 5: Nominal sequence of actions for the assembly process. Localizations, which must be executed prior to each pick, screw, or assemble are omitted in this graph, but must be added to the plan as those actions fail otherwise. Additional screws require the repetition of the last actions actions of this sequence. For the optimized domain only the small loop with two actions (and additional localizations) must be repeated. The original domain requires the execution of actions in the larger loop, as the intermediate goal is more restrictive. This includes the placement of the screwdriver such that the handovers of the box can be performed.

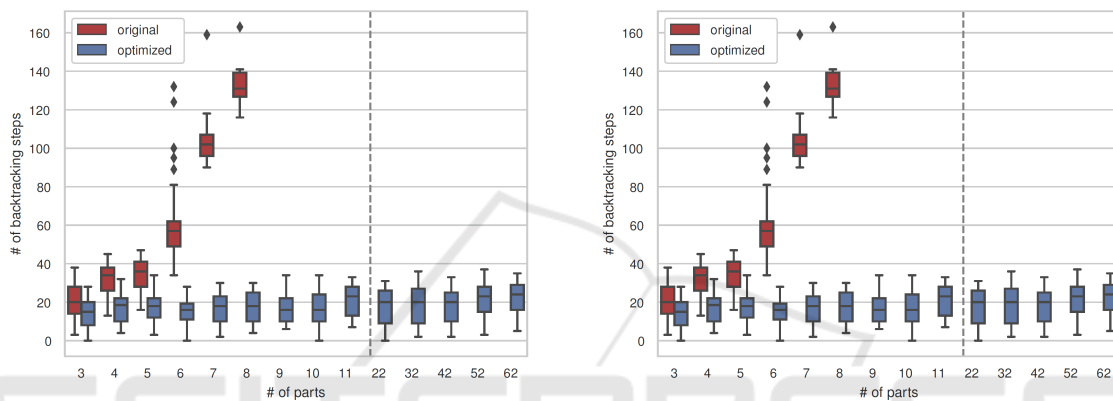


Figure 6: The number of steps in the successful solution for the task increases linearly. For the original domain more steps are necessary per additional part. For the optimized domain the intermediate product is not placed and nevertheless recognized as the interim step due to the reformulation. This is also reflected in the number of necessary backtracking steps shown on the right figure. Both numbers go hand in hand, as shorter plans have less operations, that may fail during refinement. Please notice the change in step size right of the dashed line.

screwdriver, pick up a screw from the magazine, and insert it in the air to the box with lid as depicted in Figure 4. In order to reach the intermediate goals in the optimized domain, the arm with screwdriver can localize the magazine right away and continue to pick up screws from the magazine and fasten the lid with them. For the original domain on the other hand, the box must be separated from the robot after each intermediate step to fulfill the intermediate goals posed by the coarse level. This means that the box must be placed on the table. However, the box is grasped upside down to allow access for the screwdriver. Therefore, no collision-free way to place the box upright with a single arm exists. That means an additional handover is needed, and the planner must find out that the screwdriver must be placed to free the second arm for that as highlighted in Figure 5. Afterwards the box and the screwdriver must be localized again before the process can continue similar to the optimized domain. In Figure 6 the resulting lengths of the successful plans is depicted on the left hand side. We can see that the plan lengths grow significantly faster

for the original domain due to the described process of additional placements. We tested each domain 100 times with increasing number of screws and a noise of 5 cm added to the initial object positions. The variance in plan lengths is a result of additional free-space movements to reach the goal positions. On the right hand side the number of necessary backtracking steps is depicted. This number grows for the original domain faster as well, as the additional steps that need to take place compared to the optimized domain add further points of possible failure during refinement. Therefore, not only the final plan length is shorter for the optimized domain, but also the planning process examines less dead ends during the search. This is also reflected in the planning times. Figure 7 shows the them for each of the domains for different number of parts. We can observe that the optimized domain scales almost linearly even for a huge number of parts. The original domain performs good as well, however it has a significantly faster increase in planning times compared to the optimized domain. Note that even the original domain scales near linear, de-

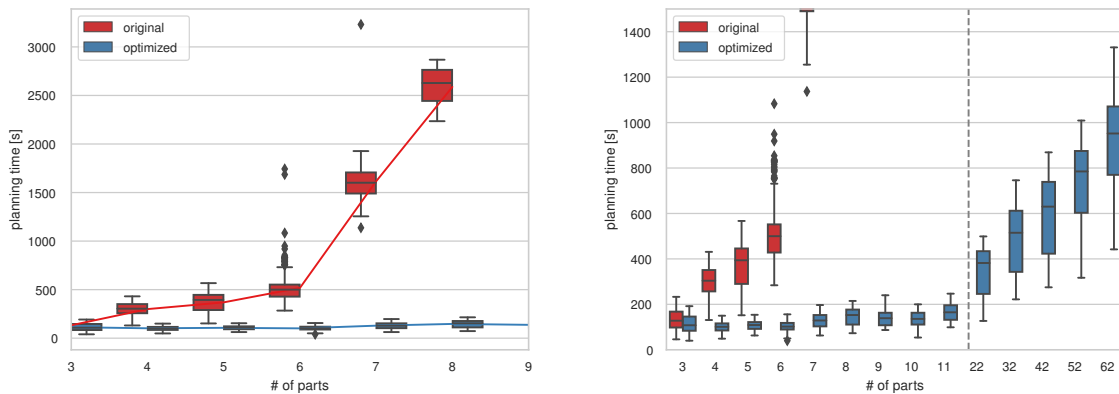


Figure 7: Planning times for the original and optimized domain for a task with three to 8 parts that need to be assembled. We can observe a near linear relationship between planning times and length of the task. However, the optimized domain scales significantly better than the original domain. Please notice the change in step size right of the dashed line.

spite the very long plans of up to 300 steps. Non-hierarchical planners would typically scale exponentially with this plan length. The reason for this is that the complexity is handled on the abstract levels of our domain which results in very few calls to the very expensive trajectory generation. The linear scaling probably ends when the abstract planning problems gains more weight on the overall planning times than the problems on the refined levels which individually stay constant in size.

## 5 CONCLUSIONS

In this paper we presented a novel optimization algorithm for planning domains, that are represented in a set-based formulation. We use a hierarchical planner that makes use of this formulation and provides near linear scalability for our problem at the cost of non-optimal solutions. Poorly modeled domains naturally result in costly and computationally expensive solutions.

Our optimization approach reduces these effects and therefore allows for an easy composition of different models to an overall planning domain. This is an indispensable prerequisite for scalable industrial autonomy and flexible manufacturing.

An alternative to this would be perfectly aligned descriptions of each part of the overall domain. This, however, can only be guaranteed in small demo scenarios, which are designed by a single person or a small team. As soon as several groups or compa-

The presented research is financed by the TransFit project which is funded by the German Federal Ministry of Economics and Technology (BMWi), grant no. 50RA1701, 50RA1702, and 50RA1703.

nies bring in modules which enable easy integration and alignment of the models must be enabled by algorithms. An important advantage of the explicit model of the domain compared to learned heuristics is the ease to debug and ability to explain the behavior of the algorithm.

## REFERENCES

- Arecas, C. E., Bustos, F., Dominguez, M., and Hoffmann, J. (2014). Optimizing planning domains by automatic action schema splitting. In *Twenty-Fourth International Conference on Automated Planning and Scheduling*.
- Dietrich, V., Kast, B., Schmitt, P., Albrecht, S., Fiegert, M., Feiten, W., and Beetz, M. (2018). Configuration of perception systems via planning over factor graphs. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–7. IEEE.
- Haslum, P., Botea, A., Helmert, M., Bonet, B., Koenig, S., et al. (2007). Domain-independent construction of pattern database heuristics for cost-optimal planning. In *AAAI*, volume 7, pages 1007–1012.
- Kalashnikov, D., Irpan, A., Pastor, P., Ibarz, J., Herzog, A., Jang, E., Quillen, D., Holly, E., Kalakrishnan, M., Vanhoucke, V., et al. (2018). Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation. *arXiv preprint arXiv:1806.10293*.
- Kang, T.-S. and Nnaji, B. O. (1993). Feature representation and classification for automatic process planning systems. *Journal of manufacturing systems*, 12(2):133–145.
- Kast, B., Albrecht, S., Feiten, W., and Zhang, J. (2019a). Bridging the gap between semantics and control for industry 4.0 and autonomous production. In *Int. Conf. on Automation Science and Engineering*. IEEE.
- Kast, B., Dietrich, V., Albrecht, S., Feiten, W., and Zhang, J. (2019b). A hierarchical planner based on set-theoretic models: Towards automating the automation

- for autonomous systems. In *Int. Conf. on Informatics in Control, Automation and Robotics*. SCITEPRESS Digital Library.
- Katz, M., Sohrabi, S., Samulowitz, H., and Sievers, S. (2018). Delfi: Online planner selection for cost-optimal planning. *IPC-9 planner abstracts*, pages 57–64.
- Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.
- Nägele, L., Schierl, A., Hoffmann, A., and Reif, W. (2018). Automatic planning of manufacturing processes using spatial construction plan analysis and extensible heuristic search. In *ICINCO (2)*, pages 586–593.
- Schmitt, P. S., Wirnshofer, F., Wurm, K. M., Wichert, G. V., and Burgard, W. (2019). Planning reactive manipulation in dynamic environments. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.
- Seipp, J., Braun, M., Garimort, J., and Helmert, M. (2012). Learning portfolios of automatically tuned planners. In *Twenty-Second International Conference on Automated Planning and Scheduling*.
- Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. (2016). Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484.
- Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al. (2017). Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359.
- Vallati, M., Hutter, F., Chrapa, L., and McCluskey, T. L. (2015). On the effective configuration of planning domain models. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*.
- Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D. H., Powell, R., Ewalds, T., Georgiev, P., et al. (2019). Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354.
- Xie, A., Ebert, F., Levine, S., and Finn, C. (2019). Improvisation through physical understanding: Using novel objects as tools with visual foresight. *arXiv preprint arXiv:1904.05538*.