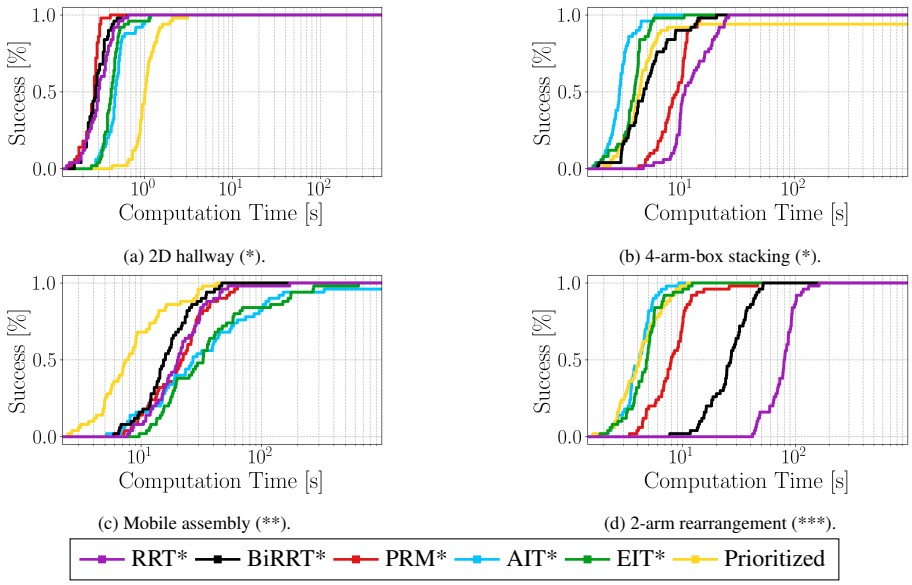



Fig. 12: Evolution of the success rate time over 50 runs. (*) indicates a fully given task sequence, (**) indicates a partial task ordering, (***) indicates unassigned and unordered tasks. Note that the prioritized planner only acts as anytime planner and improves its cost in the (**) and (***) settings, where multiple different sequences can be generated and planned for.

# Appendix

## A Success plots

In addition to the cost convergence plots in Fig. 6, we show the success rate of the planners over their runtime in Fig. 12. The planners achieve a close to perfect success rate. The exception in this scenario here is the AIT* planner, which struggles with the mobile manipulation task (Fig. 12c) which has a large configuration space and many tasks that are not fully ordered. Thus, it can happen that the exploration of the large number of modes does not reach the goal mode, and does not find a solution. In comparison, the other planners are more greedy, and thus do better in this scenario.

## B Scaling studies

We present an empirical analysis of how the proposed algorithms scale with the number of robots and the number of tasks on the example of the bidirectional RRT. We show the scaling behavior of the planner for three scenarios:

– Box stacking with multiple robots in the same workspace, i.e., the scenario shown in Fig. 1.

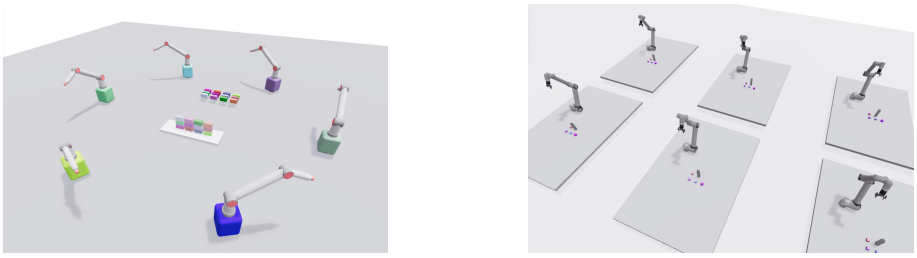
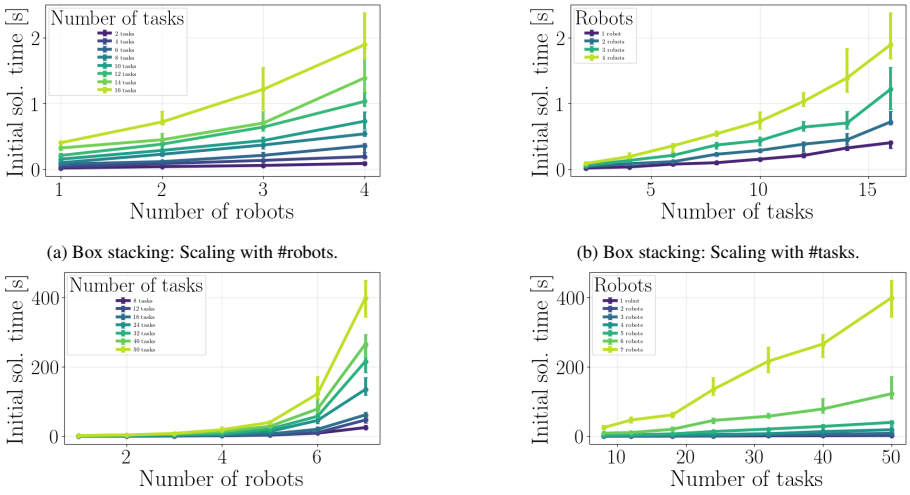


Fig. 13: Scenarios for the empirical scaling analysis. Left: Mobile manipulation with 6 robots, right: independent box stacking with 6 robots.




(a) Box stacking: Scaling with #robots.

(b) Box stacking: Scaling with #tasks.

(c) Mobile wall: Scaling with #robots.

(d) Mobile wall: Scaling with #tasks.

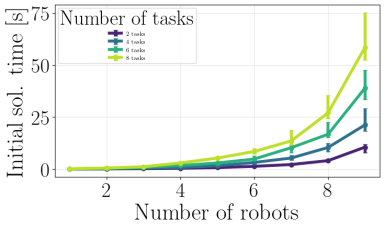
Fig. 14: Median time taken by the bidirectional RRT to find the initial solution in the box stacking and the mobile manipulation setting in shared workspaces along with the confidence intervals.

- Multiple robots building a wall, with up to 7 robots, i.e., with up to 42 Dof, and up to 25 bricks (Fig. 13, left).
- Box stacking of multiple completely independent robots in independent workspaces, i.e., the box stacking scenario in Fig. 1 with a single robot, duplicated $n$ times (Fig. 13, right).

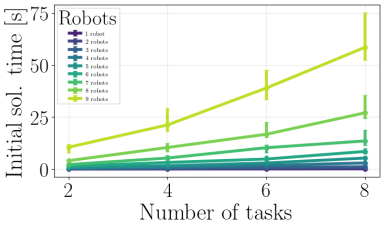
In these scenarios, we focus on the time the planner takes to reach the initial solution dependent on the number of robots in the scene, and dependent on the number of tasks. We do not analyze the convergence behavior of the planner here.

### B.1 Scaling in shared workspaces

In order to analyze the scaling behavior, we plot the number of robots against the time taken to find the initial solution (for varying numbers of tasks) in Figs. 14a and 14c, and




(a) Independent box stacking: Scaling with #robots.




(b) Independent box stacking: Scaling with #tasks.

Fig. 15: Median time taken by the bidirectional RRT to find the initial solution in the box stacking setting in independent workspaces along with the confidence intervals. Note that in this scenario, we are considering *tasks per robot* instead of the absolute number of tasks as before.

we plot the number of tasks against the time taken to find an initial solution (for varying numbers of robots) in Figs. 14b and 14d.

We can see that similar trends hold for both the environment with the tabletop box stacking and for the environment with the mobile robots: in both scenarios, we see slightly above linear complexity with the number of tasks, and we see roughly exponential complexity with the number of robots. This is more clearly visible in the scenario with the mobile robots where we have both more robots, and more tasks to note the differences.

### B.2 Scaling in independent workspaces

For the last scenario where each robot is completely independent of the others, we group by number of tasks *per robot*, as it is not easily possible to hold the number of tasks constant in the isolated environments when varying the number of robots. In this problem, an ideal planner would not be influenced by the presence of the other environments (except through the ordering constraint of the tasks), and the time taken for planning would be expected to increase linearly with the number of environments (or stay constant if we could fully parallelize everything). In Fig. 15, we can see that this is not the case for the composite space planning approach. Rather, we see similar behavior as in the previous environments, i.e., the roughly exponential increase of time taken to find the initial solution.

## C Compute breakdown

We provide a few examples of where compute time is spent in the bidirectional RRT. Particularly, we show the fractions of where the compute time is spent until an initial solution is found for all the environments also used in Table 2. We show the times here up to where the initial solution is found only, as afterwards, the profile of where the time is spent changes considerably, and is more dependent on the hyperparameters, e.g., deciding on how much shortcutting we do.

Table 3: Ratios of where time is spent in the bidirectional RRT planner up until the initial solution is found. *Rnd. Sampling* contains collision checking time as well. Major parts in *Other* are nearest neighbor computation, oracle evaluation and partially overhead from, e.g., bookkeeping for modes.

| | Fraction of total time [%] | | | | |
|---|---|---|---|---|---|
| | | Collision checking | | | |
| Environment | Rnd. Sampling | Single config | Edge (free) | Edge (blocked) | Other |
| simple | 15.3 | 2.0 | **54.1** | 11.2 | 17.4 |
| one_agent_many_goals | 16.6 | 0.5 | **55.3** | 0.8 | 26.9 |
| single_agent_mover | 19.1 | 2.0 | **44.4** | 7.4 | 27.2 |
| 2d_handover | 23.7 | 5.2 | **49.4** | 10.4 | 11.2 |
| random_2d | **51.6** | 4.3 | 17.1 | 12.1 | 14.9 |
| other_hallway | 22.8 | 6.3 | **39.5** | 7.6 | 23.8 |
| three_agents | **36.2** | 3.2 | 26.1 | 14.3 | 20.2 |
| triple_waypoints | 7.6 | 0.6 | **83.8** | 2.4 | 5.6 |
| welding | 11.2 | 0.6 | **82.1** | 1.6 | 4.7 |
| handover | **37.1** | 7.5 | 25.1 | 13.0 | 17.2 |
| eggcartons | 10.3 | 1.1 | **70.7** | 3.2 | 14.8 |
| bottles | 1.3 | 0.3 | **95.3** | 2.6 | 0.5 |
| box_rearrangement | 17.6 | 1.5 | **64.3** | 4.1 | 12.4 |
| box_reorientation | 20.4 | 2.1 | **60.9** | 5.2 | 11.5 |
| pyramid | 20.7 | 0.7 | **70.6** | 2.5 | 5.5 |
| box_stacking | 31.7 | 1.1 | **52.6** | 2.7 | 12.0 |
| mobile_wall_four | 18.1 | 8.0 | **48.2** | 17.2 | 8.5 |
| mobile_strut | 3.7 | 0.9 | **85.3** | 2.8 | 7.3 |
| spiral_tower | 28.3 | 1.5 | **61.2** | 3.5 | 5.6 |
| spiral_tower_two | 6.8 | 0.9 | **80.5** | 3.4 | 8.4 |
| cube_four | 35.8 | 1.4 | **56.9** | 3.1 | 2.8 |
| dep_piano | 29.6 | 2.7 | **39.2** | 9.8 | 18.7 |
| dep_box_stacking | 31.7 | 0.9 | **55.5** | 2.1 | 9.8 |
| dep_box_reorientation | 14.6 | 1.3 | **74.8** | 3.1 | 6.2 |
| dep_mobile_wall_four | 22.1 | 7.9 | **38.1** | 17.5 | 14.4 |
| unordered_box_reorientation | 11.9 | 0.8 | **64.9** | 1.6 | 20.8 |
| unordered_bottles | 8.1 | 1.9 | **76.3** | 4.5 | 9.2 |
| unassigned_tsp | 17.5 | 0.3 | **50.9** | 0.9 | 30.4 |
| unassigned_cleanup | 11.3 | 0.6 | **59.3** | 0.9 | 27.9 |
| unassigned_stacking | 18.9 | 0.5 | **60.5** | 1.1 | 18.9 |

We can see in Table 3 that up to the initial solution, the dominating factor is edge collision checking in most problems. We want to note that it is in particular the valid edges that take the majority of the time, and not the invalid edges.

The ratio of the main time-consuming functions is also dependent on the environment, and the dimensionality of the problem: The probability of sampling a collision free configuration decreases with increasing dimensionality of the problem space. Thus, the ratio of edge-collision checking to configuration sampling also changes depending on this.

After the planner found the initial solution, i.e., when improving the path using informed sampling and shortcutting, more time is spent doing edge-collision checking (from rewiring and from shortcutting) and configuration-collision checking (from informed sampling). The ratio of what dominates here is very dependent on the hyperparameters, since we decide how many iterations the planner does for shortcutting, and how long it tries to generate informed samples.

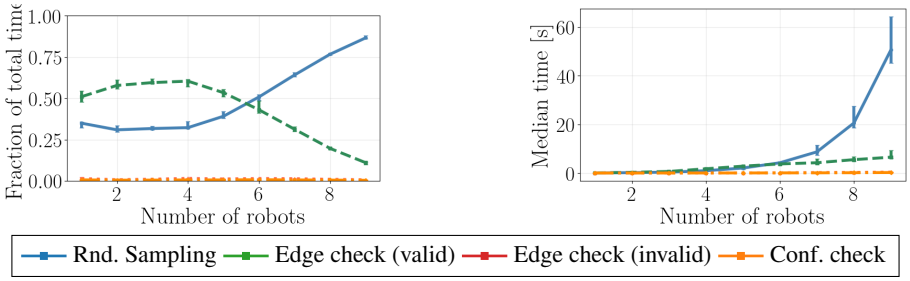



Fig. 16: Independent box stacking: Relative (left) and absolute (right) compute time spent by the bidirectional RRT in different functions depending on the number of robots in the environment.

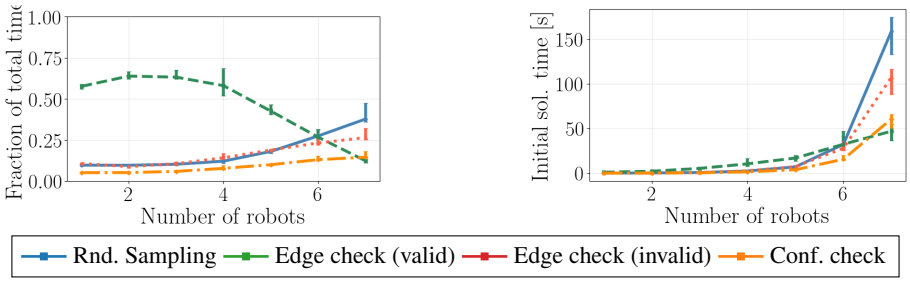



Fig. 17: Mobile manipulation: Relative (left) and absolute (right) compute time spent by the bidirectional RRT in different functions depending on the number of robots in the environment.

### C.1 Scaling of compute ratios

We further break down where time is spent if we increase the number of robots in the independent box stacking environment, and in the mobile manipulation scenario from the previous section. We focus on the time taken to find the initial solution, and not on the convergence. In order to simplify the analysis, we focus on only one specific set of tasks, i.e. we keep the number of tasks per robot constant over all the experiments.

In the following, we distinguish (i) time spent in sampling random valid configurations (which includes collision checking them, (ii) collision checking single configurations (as done, e.g., after steering), and (iii) edge collision checking, which we further break down into valid and invalid ones based on the checked results.

In the independent box stacking scenario, the environment per robot does not change with the addition of more robots. Thus, we would expect the sampling, and the collision checking effort to increase roughly linearly with the number of robots. However, as we can see in Figs. 16 and 17, the time spent in sampling of valid configurations, and edge-collision checking is increasing considerably with the number of robots in the BiRRT* planner.

Focusing on the independent box stacking environment, we can observe two main effects:

– The time spent finding valid random configurations in this particular environment scales exponentially.

– Edge collision checking effort increases significantly. We attribute this to the fact that the expected path length for an edge increases with the number of robots.

In the mobile manipulation scenario, we can see that the time spent in checking edges that are invalid increases with the number of robots, which is likely mostly due to the more congested environment due to more robots being in the same workspace.

Summarizing everything, we can see that the time to find an initial solution increases roughly exponentially with the number of robots that we plan for, which can be expected, as the volume of the search space grows exponentially with the number of degrees of freedom.

**Sampling collision free poses:** We notice that one of the large drivers of the scaling is sampling collision free poses. In hindsight, this can be expected, given that the probability of sampling a collision-free pose in the full composite space is roughly $p_{\text{free}} \propto \prod_i p_{\text{free},i}$, where we use $p_{\text{free},i}$ as probability of sampling a collision free pose for robot $i$.

**Edge collision checking:** While we would expect the robots to roughly maintain the same path length between their subsequent goals independent of how many other robots are added (especially in the independent environments), we can observe that the paths that the robots take between their goals become longer the more other robots we add to the problem. This leads to more collision checking to validate the motions of the robots, and therefore a longer time until we find the initial solution to the problem.

### C.2 Tackling the scaling effects

The described issues can be alleviated by (partially) decoupling the planning of the different robots from each other, as done in many previous works. However, completely decoupling the robots from each other sacrifices completeness and optimality. Luckily, one does not have to decouple everything completely: It is for example possible to sample full configurations more efficiently via Gibbs sampling, or simply by sampling them iteratively instead of all at once. This brings us from the exponential, to a roughly linear complexity, which we would also have if we plan in a completely decoupled space, where we treat previously planned robots as fixed.

The second issue (the growing length of the paths between goals) can also be alleviated, e.g., by biasing the paths that we plan towards good single-robot plans, which again goes in the direction of decoupling the robots from each other.