Planning for Manipulation Among Movable Objects: Deciding Which Objects Go Where, In What Order, And How

Anonymous submission

Abstract

We are interested in pick-and-place style robot manipulation tasks in cluttered and confined 3D workspaces among movable objects that may be rearranged by the robot and may slide, tilt, lean or topple. A recently proposed algorithm, M4M, determines *which* objects need to be moved and *where* by solving a Multi-Agent Pathfinding (MAPF) abstraction of this problem. It then utilises a nonprehensile push planner to compute actions for *how* the robot might realise these rearrangements and a rigid body physics simulator to check whether the actions satisfy physics constraints encoded in the problem. However, M4M greedily commits to valid pushes found during planning, and does not reason about orderings over pushes if multiple objects need to be rearranged. Furthermore, M4M does not reason about other possible MAPF solutions that lead to different rearrangements and pushes. In this paper, we extend M4M and present Enhanced-M4M (E−M4M) – a systematic graph search-based solver that searches over orderings of pushes for movable objects that need to be rearranged and different possible rearrangements of the scene. We introduce several algorithmic optimisations to circumvent the increased computational complexity, discuss the space of problems solvable by E−M4M and show that experimentally, both on the real robot and in simulation, it significantly outperforms the original M4M algorithm, as well as other state-of-the-art alternatives when dealing with complex scenes.

Introduction

Simple pick-and-place robot manipulation tasks can be difficult to solve for motion planning algorithms that do not reason about how 'movable' objects in the confined workspace might need to be rearranged in order to find a feasible solution path. Such situations are commonly encountered when robot arms have to grasp and extract desired objects from cluttered shelves or pack several objects in a box. Solving these "Manipulation Among Movable Objects" (MAMO) problems (Alami, Laumond, and Simeon 1994; Stilman et al. 2007) requires a planning algorithm to decide *which* objects should be moved (Hauser 2014), *where* to move them, and *how* they may be moved. For the scene shown in Figure 1 (a), the tomato soup can is the "object-of-interest" (OoI) to be retrieved. In order to do so, the PR2 robot must first move the coffee can and potted meat can out of the way so that the grasp pose for the OoI becomes reachable.

(a) (b)

Figure 1: **(a)** The tomato soup can (yellow outline) is the object-of-interest (OoI) to be retrieved. The potted meat can and coffee can in front of it must be rearranged out of the way in order to retrieve the OoI and solve the MAMO problem. **(b)** Trying to retrieve the beer can (OoI, yellow outline) leads to a complex interaction with the movable potted meat can being tilted by the robot arm.

Existing state-of-the-art approaches in literature commonly assume prehensile (pick-and-place) rearrangement actions, e.g. Stilman et al. (2007); Wang et al. (2022) and/or planar robot-object and object-object interactions, e.g. van den Berg et al. (2008); Vieira et al. (2022). Prehensile rearrangements not only preclude manipulation of big, bulky and otherwise ungraspable objects, but also assume access to known grasp poses for all movable objects and availability of stable placement locations for them in a cluttered and confined workspace. The planar world assumption does not account for realistic physics interactions between objects in a real-world scene. In contrast to these, our emphasis is on solving MAMO problems (i) in a 3D workspace where robot actions can lead to complex multi-body interactions where objects tilt, lean on each other, slide, and topple (Figure 1 (b)); and (ii) with nonprehensile push actions for rearranging the clutter in the scene. With this we allow for more seamless and natural manipulation that rearranges objects aside without picking them up while considering complicated toppling, sliding, and leaning effects.

The MAMO problem definition includes information about which objects are *movable* and which are static or *immovable* obstacles. All objects have a set of *interaction constraints* associated with them that define valid robot-object

and object-object interactions in the workspace. Interaction constraints encode that neither the robot nor any other object can make contact with immovable obstacles (an object that cannot be interacted with, such as a wall), and movable objects cannot fall off the shelf, tilt too far (beyond $25°$), or move with a high instantaneous velocity (above $1\,\mathrm{m\,s^{-1}}$). These constraints help model realistic and desirable robot-object interactions since we want to prevent robots from carelessly hitting, pushing or throwing objects around. In order to forward simulate the effect of nonprehensile robot pushes on the objects, we use a rigid body physics simulator to evaluate the interaction constraints and determine the resultant state of the workspace.

In recent work (_____ 2023), we proposed the M4M algorithm for MAMO to answer the questions of *which* objects to move *where*, and *how*. M4M ("Multi-Agent Pathfinding for Manipulation Among Movable Objects") relies on an MAPF abstraction of MAMO problems where the movable objects are artificially actuated agents with the goal of avoiding collisions with (i) the robot arm as it retrieves the OoI, (ii) each other, and (iii) immovable obstacles. A solution to this MAPF abstraction informs M4M of *which* objects must be moved and *where* so that the OoI can be retrieved to solve the MAMO problem. M4M then samples nonprehensile pushes to try and realise the rearrangements suggested by the MAPF solution in the real-world, thereby addressing the third question of *how* objects may be moved.

However, M4M is greedy and can fail to find solutions in many cases where one may exist. It is greedy in three different ways. First, M4M does not search over all possible orderings of object rearrangements if multiple movable objects need to be moved. It greedily commits to the first valid push it finds and continues searching for a solution from the resultant state of that push. Second, M4M never reconsiders solving the MAPF problem again for a different solution that might require objects to be rearranged differently. As such, it does not search over all possible rearrangements of the scene. This is important because in cases where an object cannot be rearranged successfuly as per the MAPF solution (perhaps due to robot kinematic limits, the presence of immovable obstacles, interaction constraint violations etc.), we must replan the MAPF solution and consider a different way to rearrange the scene that may indeed be feasible. Finally, even in cases where we successfully rearrange an object to a particular location, M4M never reconsiders moving that object differently, which may be required if no solution can be found from the resultant state of the valid push.

This paper extends M4M and presents Enhanced-M4M (E−M4M), an algorithm that addresses all three shortcomings of M4M and does so by searching a much larger space for solutions. It considers different orderings for rearranging objects, replans MAPF solutions as and when required, and considers different ways to rearrange any particular object. Although the search space for E−M4M grows tremendously as a result, E−M4M exploits the information gained during its execution to reduce redundant exploration of the solution space. There is redundancy in considering the same or similar pushing actions for an object in different nodes of E−M4M search tree such that if one action succeeds or fails (i.e. its validity is determined by forward simulating it for interaction constraint verification), it is likely that the other actions will succeed or fail as well. We exploit this by introducing caching of positive and negative simulation results and learning a probabilistic estimate of solving a particular subtree of the search, and use these within E−M4M to bias its exploration. We make the following contributions as part of our E−M4M algorithm:

1. A best-first graph search for MAMO problems that searches over orderings of object rearrangements, different rearrangements of the scene, and different ways to rearrange each object.
2. Caching results of successful (valid) pushes to avoid simulations of similar pushes repeatedly.
3. Caching results of unsuccessful (invalid) pushes to feedback information to the MAPF solver to efficiently search the space of rearrangements of the scene.
4. A learned probabilistic model for solving a particular subtree to bias exploration of the best-first search.
5. Significant quantitative improvements over M4M and several other state-of-the-art MAMO baselines.

Related Work

In recent years, MAMO planning algorithms have continued to rely on at least one of two simplifying assumptions. The first limits the action space of the robot to prehensile or pick-and-place rearrangements of movable objects (Stilman et al. 2007; Krontiris et al. 2014; Krontiris and Bekris 2015; Lee et al. 2019; Nam et al. 2020; Shome and Bekris 2021; Wang et al. 2022). This simplifies the planning problem as grasped objects behave as rigid bodies attached to the robot end effector, and rearrangement paths can be computed by avoiding collisions with other objects in the scene. It is important to note that these paths can only be found if we assume (i) all objects that may need to be rearranged are graspable by the robot, (ii) we have known grasp poses for all graspable objects, (iii) the existence of stable placement locations for these objects in the cluttered workspace, and (iv) a relatively large volume of object-free space so that collision- and contact-free rearrangement paths with a grasped object exist. In our work, we make none of these assumptions, instead relying on nonprehensile pushing actions to rearrange the scene. This allows us to manipulate a much larger set of objects, and also lets us rearrange multiple objects simultaneously. However, using these actions within a planning algorithm necessitates the ability to accurately predict their effect on the configurations of movable objects in order to compute the resultant state of the world after the push.

The second simplification assumes planar robot-object and object-object interactions, while allowing nonprehensile push actions for rearrangement. This planar assumption halves the size of the configuration space of movable objects from $SE(3)$ to $SE(2)$. To predict the effect of push actions on the scene, some existing algorithms make use of simple analytical or learned physics models (van den Berg et al. 2008; Dogar and Srinivasa 2012; Huang et al. 2022), while others use computationally cheap 2D physics simulators (King 2016; Huang, Jia, and Mason 2019). Assuming

planar interactions does not capture the complex multi-body physics of the 3D real-world where objects may tilt, lean on each other, topple etc., something we account for in the E−M4M algorithm. As part of our experiments, we compare against an implementation of the algorithm from (Dogar and Srinivasa 2012) that uses the same nonprehensile push planner as E−M4M in conjunction with a 3D rigid body physics simulator. The original algorithm is not viable for our MAMO problems as it uses a 2D analytical model to predict the result of planar robot-object interactions, and only allows the robot to rearrange one object at a time.

Existing work which uses a full 3D rigid body physics simulator to forward simulate the effect of robot actions on the scene does not account for the difficult interaction constraints we include in our MAMO problems which makes it harder to find a feasible solution. Instead, they either only deal with simple constraints (Suhail Saleem and Likhachev 2020; Saxena, Saleem, and Likhachev 2021) where objects are not allowed to fall off the workspace shelf, or do not include any such constraints (Park et al. 2022; Vieira et al. 2022) and ignore cases where objects topple. We include a comparison against (Suhail Saleem and Likhachev 2020) in our experiments, albeit with the full interaction constraint set that E−M4M considers.

We also include comparisons against two general purpose sampling-based planning algorithms, KPIECE (Sucan and Kavraki 2012) and RRT (LaValle and Kuffner 2001), and our own recent M4M algorithm developed for this MAMO domain. KPIECE is a randomised algorithm developed for planning problems where it is expensive to determine the resultant state of an action (like querying a physics-based simulator for the effect of robot pushes). M4M, as discussed earlier, decouples the search for a solution to the MAMO problem between solving an abstract MAPF problem that reasons about the configuration of movable objects but does not require forward simulating a simulation-based model, and solving a simulation-based arm motion planning problem that does not need to search over the possible configurations of movable objects.

Problem Formulation

We are interested in solving MAMO problems with a q degrees-of-freedom robot manipulator $\mathcal{R}$ whose configuration space $\mathcal{X}_\mathcal{R} \subset \mathbb{R}^q$. The workspace is populated with objects $\mathcal{O} = \{O_1, \ldots, O_n\}$ whose configuration spaces $\mathcal{X}_{O_k} \equiv SE(3)$. We assume we know which objects $\mathcal{M} \subset \mathcal{O}$ are *movable* and which objects $\mathcal{I} \subset \mathcal{O}$ are *immovable*. Each object O_k is associated with *interaction constraints* described earlier that help determine whether any state x in the search space $\mathcal{X} := \mathcal{X}_\mathcal{R} \times \mathcal{X}_{O_1} \times \cdots \times \mathcal{X}_{O_n}$ is valid or not. The planning algorithm is provided the initial configurations of all movable objects (denoted as $\mathcal{M}^{\text{init}}$) and immovable obstacles ($\mathcal{I}$), information about which object is the "object-of-interest" (OoI), desired grasp pose for the OoI, and a "home" configuration outside the workspace shelf where the OoI must be moved. Our goal is to find a path of valid states in $\mathcal{X}$ that successfully retrieves the OoI from the cluttered and confined workspace shelf.

We make no assumptions about the MAMO problem being *monotone* where each movable object may only be moved once, and we allow the robot to rearrange several movable objects at the same time. We do assume access to a rigid body physics simulator to evaluate the effect of robot actions on the states of the objects in the workspace.

E−M4M

This paper presents the E−M4M algorithm, an enhanced version of our previous M4M algorithm. In this section we provide details about E−M4M, the MAPF abstraction and nonprehensile push planner used within it, and discuss when and why E−M4M will solve a MAMO problem (or not).

In order to solve the MAMO problems of interest to us, E−M4M must answer questions about *which* objects should be moved, *where* they may be moved, and *how* the robot can move them. Like M4M, it relies on two modules to answer these questions – an MAPF solver (Sharon et al. 2015) is used to answer the first two questions, while our nonprehensile push planner uses the MAPF solution to try and answer the third. Unlike M4M however, E−M4M runs a best-first search over a graph $G = (V, E)$ with the help of these two modules. The vertices $v \in V$ represent a set of configurations (alternatively a *rearrangement*) of the movable objects $\mathcal{M}$ in the scene. Edges $e = (u, v) \in E$ represent a successful rearrangement action changing the configuration of *at least* one $O_m \in \mathcal{M}$ between u and v. The overall E−M4M search expands vertices in an order dictated by some priority function $f : V \to \mathbb{R}_{\geq 0}$. In contrast M4M (i) greedily commits to the first valid push found (it does not search over orderings of rearrangements of multiple movable objects like E−M4M), (ii) only obtains a single MAPF solution for each rearrangement it sees (it never replans the MAPF solution based on the result of push actions like E−M4M), and consequently (iii) only tries to rearrange a movable object along a single MAPF solution path for each rearrangement (it does not consider alternate ways to push an object for the same rearrangement like E−M4M).

Main Algorithm

Algorithm 1 contains the pseudocode for E−M4M. Initially, E−M4M computes a trajectory $\gamma_{\text{OoI}} \subset \mathcal{X}_\mathcal{R}$ for the robot to grasp and extract the OoI while pretending no movable objects $\mathcal{M}$ exist in the scene (Line 28). The argument for the PLANRETRIEVAL function is the set of objects to be considered as obstacles during planning. The volume occupied by the robot arm and OoI during execution of γ_{OoI}, written as $\mathcal{V}(\gamma_{\text{OoI}})$, creates a *negative goal region* (NGR) (Dogar and Srinivasa 2012). We define an NGR parameterised with a robot trajectory as some volume in the workspace that, if free of all objects, will lead to successful retrieval of the OoI upon execution of that robot trajectory. Note that if the trajectory γ_{OoI} cannot be found, the overall MAMO problem as specified is unsolvable. It may be solvable given a different grasp pose for the OoI, however grasp planning is beyond the scope of this work. Once the initial NGR $\mathcal{V}(\gamma_{\text{OoI}})$ has been computed (Line 29), E−M4M executes a best-first search using a priority queue ordered by f.

Algorithm 1: E−M4M

```
1:  procedure CreateVertex(M, v, γ)
2:      v'.M ← M, v'.parent ← v, v'.γ ← γ
3:      return v'
4:  procedure Done(v)
5:      if v.M ∩ V(γ_OoI) = ∅ then
6:          return true
7:      γ̂_OoI ← PlanRetrieval(v.M ∪ I)
8:      if γ̂_OoI exists then
9:          γ_OoI ← γ̂_OoI
10:         return true
11:     return false
12: procedure ExpandState(v)
13:     κ ← InvalidGoals(v)
14:     v.{π_m}_{m∈M} ← RunMAPF(v, κ, V(γ_OoI))
15:     if MAPF failed then
16:         Remove v from OPEN
17:         return
18:     for m ∈ v.M do
19:         if v.π_m ≠ ∅ then
20:             γ_m ← PlanPush(v.π_m, v.M)
21:             M', valid ← IsValid(γ_m)
22:             if valid then
23:                 v' ← CreateVertex(M', v, γ_m)
24:                 Insert v' into OPEN with priority f(v')
25:             else
26:                 Add final state in v.π_m to InvalidGoals(v)
27: procedure Main(M^init, I)
28:     γ_OoI ← PlanRetrieval(I)
29:     Compute V(γ_OoI)
30:     OPEN ← ∅, v_start ← CreateVertex(M^init, ∅, ∅)
31:     Insert v_start into OPEN with priority f(v_start)
32:     while OPEN is not empty and time remains do
33:         v ← OPEN.top()
34:         if Done(v) then
35:             return ExtractRearrangements(v)
36:         ExpandState(v)
37:     return ∅
```

Every time a vertex v is expanded from this queue during the search (Line 36), E−M4M calls an MAPF solver (Line 14). The set of solution paths returned by the MAPF solver is then used by our nonprehensile push planner to generate and evaluate successor rearrangement states (the loop from Line 18). For each object O_m that "moves" in the MAPF solution, our push planner tries to compute a trajectory $\gamma_m \subset \mathcal{X}_{\mathcal{R}}$ to push O_m along its MAPF solution path π_m (Line 20). If γ_m is found, it is forward simulated in a rigid body physics simulator for interaction constraint verification (Line 21). If γ_m successfully rearranges the scene, i.e. at least one object is moved and no constraints are violated, E−M4M generates a successor state v' with the resultant rearrangement and adds it to the queue (Lines 23 and 24).

A vertex v is *closed* in Line 16 and never re-expanded again if the MAPF solver fails to return a solution in Line 14. Otherwise when a vertex v is re-expanded, we ensure that the MAPF solver returns a different solution than one obtained during any previous expansion of v by including a set κ of invalid goals (Line 13). κ contains configurations for each movable object $O_m \in v.\mathcal{M}$ that *cannot* be the final state in the path π_m found by the MAPF solver. This helps E−M4M search over different MAPF solutions for the same rearrangement $v.\mathcal{M}$, thereby helping it search over different ways to rearrange $v.\mathcal{M}$.

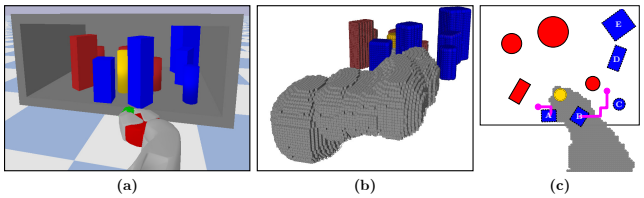


Figure 2: **(a)** A MAMO problem with five movable objects and four immovable obstacles, **(b)** the initial NGR $\mathcal{V}(\gamma_{\text{OoI}})$ found for this scene (in gray), and **(c)** a 2D projection of the scene with the MAPF solution paths in pink. This MAPF solution suggests that objects A and B should be rearranged as per the pink paths to be outside $\mathcal{V}(\gamma_{\text{OoI}})$.

E−M4M terminates with v as the goal state when the OoI can be successfully retrieved given the rearrangement $v.\mathcal{M}$. This may be achieved in one of two ways. If the movable objects $\mathcal{M}$ in v have been successfully rearranged to be outside the initial NGR $\mathcal{V}(\gamma_{\text{OoI}})$, we know the robot can execute γ_{OoI} to retrieve the OoI (Line 6). Alternatively, if a different trajectory $\hat{\gamma}_{\text{OoI}}$ (and therefore its NGR $\mathcal{V}(\hat{\gamma}_{\text{OoI}})$) can be found in the presence of all objects (movable and immovable) as obstacles, the robot can execute $\hat{\gamma}_{\text{OoI}}$ to retrieve the OoI without making contact with any other object (Line 10).

MAPF Abstraction

We make the observation that by solving a carefully constructed MAPF problem with movable objects as agents, we can obtain information about which objects our MAMO planner should consider rearranging and where. We use Conflict-Based Search (CBS) (Sharon et al. 2015) as the solver for our abstract MAPF problem. For a vertex v in E−M4M, we include all movable objects as agents in CBS starting at their current poses in $v.\mathcal{M}$. Each agent is assigned a goal of being outside the initial NGR $\mathcal{V}(\gamma_{\text{OoI}})$, while avoiding collisions with each other and all immovable obstacles $\mathcal{I}$. Although agent states in CBS specify their configuration in $\mathcal{X}_{O_m} \equiv SE(3)$, agents use a 2D action space on a four-connected grid in the MAPF abstraction that only changes the $x-$ or $y-$coordinates of their state. The CBS solution for this MAPF problem is a set of paths $\{\pi_m\}_{m\in\mathcal{M}}$ such that O_m ends up outside $\mathcal{V}(\gamma_{\text{OoI}})$ after following π_m. Figure 2 shows a simulated MAMO problem, the initial NGR $\mathcal{V}(\gamma_{\text{OoI}})$ for the scene, and a 2D projection of the scene which shows the MAPF solution found.

In order to search over all possible ways to rearrange $v.\mathcal{M}$, E−M4M includes a set of invalid goals κ when it calls CBS in Line 14. For each vertex v, κ includes information about where each object $O_m \in v.\mathcal{M}$ *cannot* end up in any future CBS solution. During a previous expansion of v, if path π_m led to an invalid push γ_m, the final state of π_m is added as an invalid goal for $O_m \in v.\mathcal{M}$ since we failed to rearrange O_m along π_m (Line 26). As a result, the next solution from CBS would return a new path $\pi'_m \neq \pi_m$ which

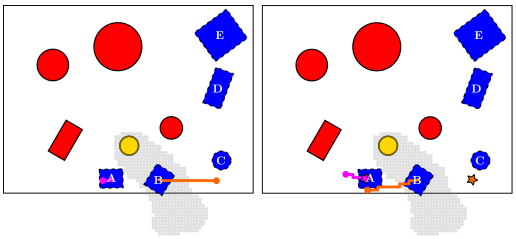


Figure 3: **(a)** First MAPF solution that led to invalid pushes γ_A and γ_B, **(b)** Adding the final states of π_A and π_B (colour coded stars) to κ leads to a new MAPF solution.

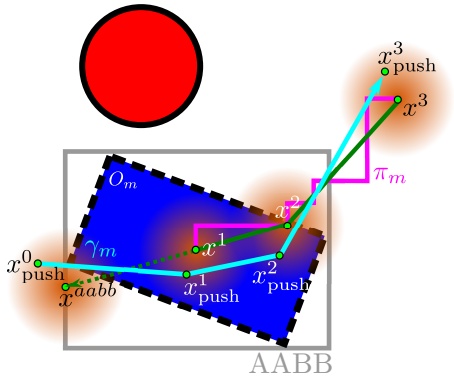


Figure 4: 2D illustration of our push planner. Movable object O_m is blue, and an immovable obstacle is drawn in red. The green path is obtained after shortcutting the pink MAPF solution path π_m. x^{aabb} is the point-of-intersection between the first segment (x^1, x^2) and the axis-aligned bounding box for O_m. The cyan segments depict the path along which inverse kinematics is used to obtain γ^i_m.

in turn would cause our push planner to consider a different rearrangement action. When all invalid pushes from v have been included in the previous call to CBS, we also include the final states of valid pushes found previously as invalid goals in κ so as to ensure we consider all possible ways to rearrange $v.\mathcal{M}$. If CBS fails to find a solution, or if no new invalid goals can be added to κ from the last call to CBS, we *close* vertex v and stop it from being re-expanded (Line 16) since no new way to rearrange $v.\mathcal{M}$ can be found. Figure 3 shows the effect of invalid goals κ on the MAPF solution – when both pushes γ_A and γ_B computed as per the MAPF solution in (a) failed, adding the final states of π_A and π_B to κ results in a different MAPF solution in (b).

Nonprehensile Push Planner

The goal for our push planner is to find robot trajectories $\gamma_m \subset \mathcal{X}_\mathcal{R}$ that rearrange a movable object along the path π_m returned by our MAPF solver. If we are able to precisely rearrange each O_m to the final state of π_m, we know the robot can execute γ_{OoI} to solve the MAMO problem. In order to do so, we assume the push planner is provided a shortcut path π_m (accounting for immovable obstacles $\mathcal{I}$) in Line 20.

Each call to PLANPUSH stochastically generates a robot trajectory γ_m, as shown in Figure 4. The starting location of the push, x^0_{push}, is sampled around the point of intersection x^{aabb} between the first segment of π_m and the axis-aligned bounding box of O_m. If the push planner finds an approach path $\gamma^0_m \subset \mathcal{X}_\mathcal{R}$ to x^0_{push} in the presence of all objects (movable and immovable) as obstacles, it samples waypoints x^i_{push} around the corresponding states x^i in π_m. It uses inverse kinematics to find segments of the push γ^i_m between the waypoints x^{i-1}_{push} and x^i_{push}. If all γ^i_m are found, the planner returns the final push trajectory as a concatenation of the individual pieces. In this case, the push γ_m is forward simulated in a rigid body physics simulator to detect if any interaction constraints are violated during its execution and get the resultant rearrangement of the scene.

What E−M4M Can and Cannot Solve

Solutions to MAMO problems lie in a space $\mathcal{X} = \mathcal{X}_\mathcal{R} \times \mathcal{X}_{O_1} \times \cdots \times \mathcal{X}_{O_n}$ that grows exponentially with the number of movable objects. There are several subtle reasons due to which E−M4M might fail to find solutions to complicated MAMO problems in this space. When trying to rearrange an object to a specific location, E−M4M only considers moving it along the particular path π_m found by CBS and not along all such paths. Its reliance on CBS and our push planner also means that E−M4M might fail to find interesting non-monotone solutions where it must rearrange an object O_a partially along its solution path π_a before moving O_b along π_b and finally going back to move O_a the remainder of the way along π_a. The MAPF abstraction itself uses a simple 2D action space which fails to capture all possible rearrangements of the scene in $\mathcal{X}_{O_1} \times \cdots \times \mathcal{X}_{O_n}$. Finally, E−M4M does not actively search over all OoI retrieval trajectories $\gamma_{\text{OoI}} \subset \mathcal{X}_\mathcal{R}$, and consequently the same NGR is used to specify goals for movable objects in all MAPF calls.

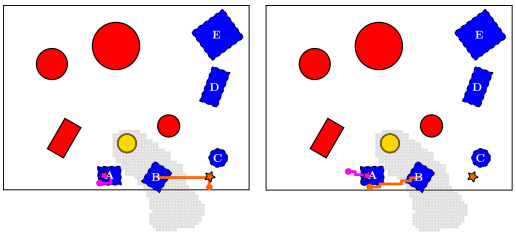
Despite these limitations, by relying on our MAPF abstraction and nonprehensile push planner, E−M4M does search over 'allowed' (i) orderings of movable object rearrangements, (ii) potential rearrangements for the set of movable objects, and (iii) ways to rearrange the same movable object. In doing so it makes progress towards a complete MAMO planning algorithm that uses nonprehensile actions for rearrangement in a 3D workspace with complex multi-body interactions where movable objects may tilt, lean, topple etc. Our quantitative analysis shows that E−M4M performs better than many state-of-the-art planning algorithms for these MAMO problems.

Speeding up the Algorithm

This section discusses three algorithmic optimisations we propose as part of E−M4M that significantly improve its quantitative performace.

Caching Unsuccessful Push Actions

The goal set for movable objects in the MAPF abstraction includes any configuration outside the initial NGR $\mathcal{V}(\gamma_{\text{OoI}})$ that is free of collision from all other objects. Given vertex v, consider an object O_m in $v.\mathcal{M}$ and its corresponding



(a) (b)

Figure 5: **(a)** First MAPF solution that led to invalid pushes γ_A and γ_B, **(b)** Adding the final states of π_A and π_B (colour coded stars) to κ leads to a new MAPF solution.

path π_m which achieves this goal. If the resulting push γ_m is invalid, the next call to CBS from v will lead to a path $\pi'_m \neq \pi_m$. However, naively including the last state in π_m as an invalid goal state for CBS will likely lead to the new path π'_m ending in a neighbouring state of the invalid goal (since CBS is an optimal MAPF solver). This in turn will lead to a push $\gamma'_m \approx \gamma_m$ that is also likely to be invalid.

To mitigate this, for every CBS call from a vertex v, for each object O_m, E−M4M caches the goals that were previously determined to be invalid in a nearest neighbour data structure. We use this cached information to bias the solutions produced by CBS to avoid moving objects to states close to known invalid goals for the respective objects. This biasing is done by assigning penalties to each potential goal location for each object O_m, and then during each low-level search within CBS finding the solution that minimises the summation of getting to a goal plus the penalty associated with the goal. This can be done by introducing one single 'pseudogoal' that the search searches towards and connecting all the potential goal locations to this 'pseudogoal' with edges whose cost is proportional to the respective penalty. This helps penalise paths to states close to known invalid goals, and lets E−M4M search the space of allowed rearrangements of $v.\mathcal{M}$ more efficiently. Figure 5 (a) shows the new MAPF solution when we include invalid goals naively – the new paths π'_A and π'_B are very similar to π_A and π_B (from Figure 3 (a)) and end in final states very close to the invalidated goals. However, using the above explained approach that modifies the single-agent search to use our nearest neighbour data structure for invalid goals, we get a very different MAPF solution in Figure 5 (b).

Caching Successful Push Actions

During the search, CBS solutions in different vertices of the graph E−M4M may contain the same path π_m for object O_m. In such cases, if we have computed the push γ_m in one of these vertices and found it to be valid in simulation while generating the successor rearrangement state, we would like to reuse this result whenever possible since simulating a push action is computationally expensive (around $2-6$s per action simulation). This reuse of successful push actions is enabled by storing the successful pushes in a database.

If a push γ_m from path π_m for object O_m rearranged the scene from $v.\mathcal{M}$ to $v'.\mathcal{M}$, we index into this database with the key (O_m, π_m). While any push is simulated, we keep track of objects that are $relevant$ for that push – these are all objects whose configurations are changed between $v.\mathcal{M}$ and $v'.\mathcal{M}$. For each (O_m, π_m) tuple, the database stores the value $(v.\mathcal{M}, v'.\mathcal{M}, \gamma_m, relevant$ objects$)$. During the expansion of some other vertex u, if CBS returns the same path π_m for O_m, we try to reuse the result of the stored push γ_m to generate the successor state u' corresponding to rearranging O_m. However, this reuse is only possible if all $relevant$ objects are in the same configurations in $v.\mathcal{M}$ and $u.\mathcal{M}$, and all other 'irrelevant' objects in $u.\mathcal{M}$ are in configurations that will not be affected by γ_m. If both these conditions are true, we can simply reuse the result of γ_m stored in the database to say that the $relevant$ objects in $u.\mathcal{M}$ will be rearranged to their repective configurations $v'.\mathcal{M}$, and the 'irrelevant' objects will remain unaffected.

Learned Priority Function

E−M4M searches an extremely large space for MAMO solutions since it searches over orderings of object rearrangements, different rearrangements of the scene, and different ways to rearrange each object. In an attempt to focus its search effort on more promising vertices of the search tree, we learn to predict the probability of a particular vertex leading to a solution for the MAMO problem. The learned function is used as the priority function f in the best-first E−M4M search. We predict the probability of a vertex v leading to a solution based on features of the rearrangement $v.\mathcal{M}$ – the percentage volume of the initial NGR $\mathcal{V}(\gamma_{\text{OoI}})$ occupied by movable objects (ϕ_1), the number of movable objects inside the NGR (ϕ_2), and for each such object the product of its mass, coefficient of friction and percentage volume inside the NGR (ϕ_3). These features indicate how difficult it is to clear the NGR and therefore solve the MAMO problem. Our predictive model f makes the Naive Bayes assumption (John and Langley 1995) that these features are conditionally independent of the others. Thus if E is the event that vertex v leads to a MAMO solution,

$$f(v) = P(E) \times P(\phi_1 \mid E) \times P(\phi_2 \mid E) \times \Pi_{O_m \in \mathcal{V}(\gamma_{\text{OoI}})} P(\phi_3 \mid E)$$

We model $P(\phi_1 \mid E)$ as a beta distribution, and $P(\phi_2 \mid E)$ and $P(\phi_3 \mid E)$ are both exponential distributions. Their parameters are estimated via maximum likelihood estimation from a dataset of self-supervised MAMO problems. We generate this set of problems by running E−M4M breadth-first on 20 MAMO scenes with a $10\,\text{min}$ planning timeout. We store each vertex generated during these E−M4M runs as a separate MAMO problem, and then run E−M4M depth-first with a $60\,\text{s}$ timeout on these problems to get our training dataset of 2400 datapoints.

Experimental Analysis

This section compares the performance of E−M4M against several MAMO baselines, studies the effect of the algorithmic improvements we propose in an ablation study, and provides results from real-world experiments on a PR2 robot.

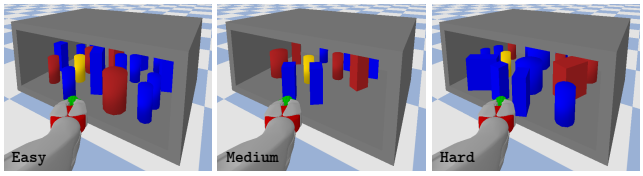


Figure 6: Example `Easy`, `Medium`, and `Hard` scenes.

Table 1: Number of problems solved by various MAMO planning algorithms in simulation experiments

Difficulty	Planning Algorithm					
	E−M4M	M4M	DOGAR	SELSIM	RRT	KPIECE
`Easy` (98)	97	78	7	16	33	16
`Medium` (63)	45	25	0	8	7	1
`Hard` (39)	15	7	0	1	1	0

Simulation Experiments Against MAMO Baselines

Our simulation experiments randomly generate MAMO workspaces with one OoI, four immovable obstacles, and five, ten or fifteen movable objects. All object properties (shapes, sizes, mass, coefficient of frictions, initial poses) are randomised and known to each planner prior to planning. We categorise these workspaces into three difficulty levels based on the number of movable objects inside the initial NGR $\mathcal{V}(\gamma_{\text{OoI}})$. Problems are `Easy`, `Medium`, or `Hard` depending on whether there are one, two, or more than two movable objects overlapping with the initial NGR. Figure 6 shows sample MAMO workspaces of each difficulty level. We test the performance of all algorithms on 98 `Easy`, 63 `Medium`, and 39 `Hard` problems with a 5 min planning timeout.

In addition to (1) M4M, we compare the performace of E−M4M against four other baselines. (2) We reimplement DOGAR (Dogar and Srinivasa 2012) to use our push planner with a physics-based simulator. It recursively searches backwards in time for objects that need to be rearranged outside the most recent NGR. If it rearranges an object successfully, the volume spanned by the rearrangement trajectory is added to the previous NGR and the recursion continues. However, DOGAR only has information about which objects need to be rearranged but not where they should be moved. Our implementation randomly samples points outside the latest NGR as goal locations for our push planner. (3) SELSIM (Suhail Saleem and Likhachev 2020) interleaves planning a trajectory while simulating interactions with 'relevant' objects with tracking the found trajectory in the presence of all objects. If tracking violates interaction constraints, the 'culprit' object is identified and added to the set of relevant objects for the next iteration. SELSIM uses simple motion primitives that change only one of the q degrees-of-freedom of the robot, which does not lead to meaningful robot-object interactions in this domain. Finally, we compare against the standard OMPL (Şucan, Moll, and Kavraki 2012) implementations of general-purpose sampling-based planning algorithms (4) RRT (LaValle and Kuffner 2001) and (5) KPIECE (Sucan and Kavraki 2012).

Table 1 contains the number of problems solved by each planning algorithm for the different difficulty levels. It is apparent that E−M4M far outperforms all other algorithms, and that all baselines struggle to solve `Medium` and `Hard` problems. The quantitative performance of all algorithms in terms of total planning time and time spent simulating robot actions is shown in Figure 7. E−M4M is able to achieve a good balance of time spent computing robot actions (with the MAPF solver and push planner) and forward simulating them for interaction constraint verification. Since M4M never replans the MAPF solution, it spends most of its time trying to sample push actions to be simulated. DOGAR repeatedly fails to find solutions, even for simple problems, because it (i) has no information about where objects should move, choosing to randomly sample uninformed pushes instead, (ii) only considers pushes to be successful if they rearrange an object to be completely outside the NGR, and (iii) never considers rearranging an object more than once. The performance of SELSIM is particularly interesting. It is only able to solve problems where the first planned path succeeds when tracked without any interaction constraints being violated, resulting in negligible planning times for its successes (and no time spent in simulation). Otherwise, owing to its primitive action space, it spends most of its planning budget in simulation trying to rearrange the scene with small robot actions that are incapable of significantly changing object configurations. RRT performs better than these special-purpose MAMO baselines since it samples long robot motions that can rearrange objects with favourable physical properties (low masses and coefficients of friction) with high likelihood. The poor performance of KPIECE in comparison to RRT can be explained by the 3D projective space being inadequate for capturing information about progress to the goal (of retrieving the OoI) since it contains no notion of which objects still remain to be rearranged, and how difficult that might be.

E−M4M Ablation Study

To study the effect of the algorithmic improvements we propose as part of E−M4M, we present results from an ablation study where we compare different versions of E−M4M. NEG-DB only caches information from unsuccessful push actions; POS-DB only caches information from successful push actions; NO-DB does not cache any information from push actions; and TIEBREAK assigns priorities by lexicographically tiebreaking E−M4M vertex feature vectors $(\phi_1, \phi_2, \sum_{O_m \in \mathcal{V}(\gamma_{\text{OoI}})} \phi_3)$. NEG-DB, POS-DB, and NO-DB all use the learned priority function like E−M4M, while TIEBREAK caches information from unsuccessful and successful push actions like E−M4M.

Table 1 shows that each of these E−M4M ablations solve fewer MAMO problems in comparison to E−M4M which combines all of them. Quantitatively, their performance can be compared from the plots in Figure 8. While there is no significant difference between the different ablations for `Easy` problems, for `Medium` and `Hard` problems we can see that performance degrades as we remove cached information. POS-DB performs worse than NEG-DB since it is not as likely for E−M4M to find the same push multiple

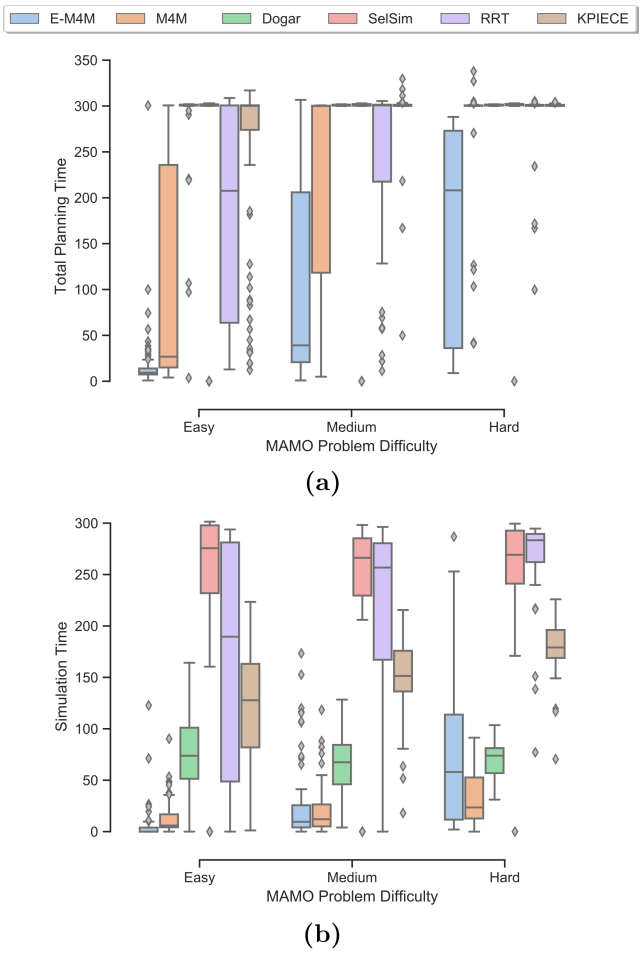


Figure 7: **(a)** Total planning time and **(b)** time spent querying a physics-based simulator for MAMO planning algorithms across planning problems with varying difficulty levels.

times during a search as it is for it to require several different MAPF solutions. Finally, even with a naive tiebreaking based priority function, TIEBREAK performs only slightly worse than E−M4M for `Hard` problems. This suggests that the learned priority function (using the Naive Bayes assumption) is not as useful for these problems, perhaps due to the 60 s timeout imposed during data collection being insufficient to result in a rich set of datapoints for `Hard` problems.

Real-World Experiments

We ran experiments with the PR2 robot with a refrigerator compartment as our MAMO workspace (Figure 1). Problems were initialised with five objects from the YCB Object Dataset (Çalli et al. 2017). The tomato soup can was always our OoI, while all other objects were initialised as movable.

We ran E−M4M on 20 perturbations of the scenes in Figure 1 with a 5 min planning timeout. 15 runs resulted in successful OoI retrieval, with the others failing due to unforeseen discrepancies between simulated and real-world robot-object interactions. Four failures were due to inaccurate computation of coefficients of friction of movable objects. One failure was the result of an object getting stuck in ridges in the real-world refrigerator shelf that were not modeled in simulation. These discrepancies highlight the sim-to-real gap that E−M4M can suffer from, since it blindly relies on the result of the physics based simulator used in the algorithm. On average, for the 15 successful retrievals, E−M4M took a total time of 39.3 ± 28.2 s of which 0.9 ± 1.2 s was spent calling the MAPF solver, 33.5 ± 25.8 s was spent planning pushes, and 7.1 ± 5.3 s was spent simulating them.

Table 2: Number of problems solved E−M4M ablations

Difficulty	Ablation				
	E−M4M	NEG-DB	POS-DB	NO-DB	TIEBREAK
`Easy` (98)	97	87	86	82	85
`Medium` (63)	45	24	29	25	36
`Hard` (39)	15	7	8	7	13

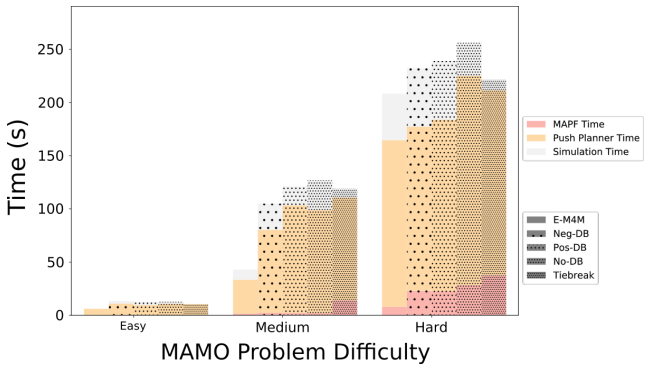


Figure 8: Median statistics for time spent calling the MAPF solver, push planner, and simulator for different E−M4M ablations.

Conclusion and Future Work

The Enhanced-M4M algorithm presented in this paper builds upon our prior work on Multi-Agent Pathfinding for Manipulation Among Movable Objects (_____ 2023). E−M4M utilises an MAPF abstraction of MAMO, a non-prehensile push planner, and a rigid body physics simulator within a best-first graph search for solving MAMO problems that require determing *which* movable objects should be moved, *where* to move them, and *how* they can be moved. E−M4M searches over different orderings of object rearrangements, different rearrangements of the workspace, and different ways to rearrange the same object.

Currently, the MAPF solver does not take into account any information about robot kinematics, movable object properties, or immovable obstacle poses (other than for collision checking against agents) when computing solution paths. Since the E−M4M algorithm uses these paths downstream for nonprehensile push planning, in future work we wish to explore an experience-based learning formulation to take these factors into account as part of the MAPF cost function so as to find paths that are more likely to result in valid pushes.

References

Alami, R.; Laumond, J.-P.; and Simeon, T. 1994. Two manipulation planning algorithms. In Goldberg, K.; Halperin, D.; Latombe, J.-C.; and Wilson, R., eds., *WAFR Proceedings of the workshop on Algorithmic foundations of robotics* , 109–125. A. K. Peters, Ltd. Natick, MA, USA.

Çalli, B.; Singh, A.; Bruce, J.; Walsman, A.; Konolige, K.; Srinivasa, S. S.; Abbeel, P.; and Dollar, A. M. 2017. Yale-CMU-Berkeley dataset for robotic manipulation research. *Int. J. Robotics Res.*, 36(3): 261–268.

Dogar, M. R.; and Srinivasa, S. S. 2012. A Planning Framework for Non-Prehensile Manipulation under Clutter and Uncertainty. *Auton. Robots*, 33(3): 217–236.

Hauser, K. K. 2014. The minimum constraint removal problem with three robotics applications. *Int. J. Robotics Res.*, 33(1): 5–17.

Huang, B.; Han, S. D.; Yu, J.; and Boularias, A. 2022. Visual Foresight Trees for Object Retrieval From Clutter With Nonprehensile Rearrangement. *IEEE Robotics Autom. Lett.*, 7(1): 231–238.

Huang, E.; Jia, Z.; and Mason, M. T. 2019. Large-Scale Multi-Object Rearrangement. In *2019 International Conference on Robotics and Automation (ICRA)*, 211–218.

John, G. H.; and Langley, P. 1995. Estimating Continuous Distributions in Bayesian Classifiers. In Besnard, P.; and Hanks, S., eds., *UAI '95: Proceedings of the Eleventh Annual Conference on Uncertainty in Artificial Intelligence, Montreal, Quebec, Canada, August 18-20, 1995*, 338–345. Morgan Kaufmann.

King, J. E. 2016. *Robust Rearrangement Planning Using Nonprehensile Interaction*. Ph.D. thesis, Carnegie Mellon University, Pittsburgh, PA.

Krontiris, A.; and Bekris, K. E. 2015. Dealing with Difficult Instances of Object Rearrangement. In Kavraki, L. E.; Hsu, D.; and Buchli, J., eds., *Robotics: Science and Systems XI, Sapienza University of Rome, Rome, Italy, July 13-17, 2015*.

Krontiris, A.; Shome, R.; Dobson, A.; Kimmel, A.; and Bekris, K. E. 2014. Rearranging similar objects with a manipulator using pebble graphs. In *14th IEEE-RAS International Conference on Humanoid Robots, Humanoids 2014, Madrid, Spain, November 18-20, 2014*, 1081–1087. IEEE.

LaValle, S. M.; and Kuffner, J. J. 2001. Randomized Kinodynamic Planning. *The International Journal of Robotics Research*, 20(5): 378–400.

Lee, J.; Cho, Y.; Nam, C.; Park, J.; and Kim, C. 2019. Efficient Obstacle Rearrangement for Object Manipulation Tasks in Cluttered Environments. In *International Conference on Robotics and Automation, ICRA 2019, Montreal, QC, Canada, May 20-24, 2019*, 183–189. IEEE.

Nam, C.; Lee, J.; Cheong, S.; Cho, B. Y.; and Kim, C. 2020. Fast and resilient manipulation planning for target retrieval in clutter. In *2020 IEEE International Conference on Robotics and Automation, ICRA 2020, Paris, France, May 31 - August 31, 2020*, 3777–3783. IEEE.

Park, S.; Chai, Y.; Park, S.; Park, J.; Lee, K.; and Choi, S. 2022. Semi-Autonomous Teleoperation Via Learning Non-Prehensile Manipulation Skills. In *IEEE International Conference on Robotics and Automation (ICRA)*.

Saxena, D. M.; Saleem, M. S.; and Likhachev, M. 2021. Manipulation Planning Among Movable Obstacles Using Physics-Based Adaptive Motion Primitives. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 6570–6576.

Sharon, G.; Stern, R.; Felner, A.; and Sturtevant, N. R. 2015. Conflict-based search for optimal multi-agent pathfinding. *Artif. Intell.*, 219: 40–66.

Shome, R.; and Bekris, K. E. 2021. Synchronized Multi-arm Rearrangement Guided by Mode Graphs with Capacity Constraints. In LaValle, S. M.; Lin, M.; Ojala, T.; Shell, D. A.; and Yu, J., eds., *Proceedings of the Fourteenth Workshop on the Algorithmic Foundations of Robotics, WAFR 2021*.

Stilman, M.; Schamburek, J.; Kuffner, J.; and Asfour, T. 2007. Manipulation Planning Among Movable Obstacles. In *2007 IEEE International Conference on Robotics and Automation, ICRA*. IEEE.

Sucan, I. A.; and Kavraki, L. E. 2012. A Sampling-Based Tree Planner for Systems With Complex Dynamics. *IEEE Trans. Robotics*.

Şucan, I. A.; Moll, M.; and Kavraki, L. E. 2012. The Open Motion Planning Library. *IEEE Robotics & Automation Magazine*, 19(4): 72–82. https://ompl.kavrakilab.org.

Suhail Saleem, M.; and Likhachev, M. 2020. Planning with Selective Physics-based Simulation for Manipulation Among Movable Objects. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 6752–6758.

_____. 2023. Planning for Complex Non-prehensile Manipulation Among Movable Objects by Interleaving Multi-Agent Pathfinding and Physics-Based Simulation. *Submitted to 2023 IEEE International Conference on Robotics and Automation (ICRA)*.

van den Berg, J. P.; Stilman, M.; Kuffner, J.; Lin, M. C.; and Manocha, D. 2008. Path Planning among Movable Obstacles: A Probabilistically Complete Approach. In Choset, H.; Morales, M.; and Murphey, T. D., eds., *Selected Contributions of the Eighth International Workshop on the Algorithmic Foundations of Robotics, WAFR 2008*.

Vieira, E.; Nakhimovich, D.; Gao, K.; Wang, R.; Yu, J.; and Bekris, K. E. 2022. Persistent Homology for Effective Non-Prehensile Manipulation. In *IEEE International Conference on Robotics and Automation (ICRA)*.

Wang, R.; Gao, K.; Yu, J.; and Bekris, K. 2022. Lazy Rearrangement Planning in Confined Spaces. In *International Conference on Automated Planning and Scheduling*.