Rearrangement on lattices with pick-n-swaps: Optimality structures and efficient algorithms

Jingjin Yu

Abstract
We study a class of rearrangement problems under a novel pick-n-swap prehensile manipulation model, in which a robotic manipulator, capable of carrying an item and making item swaps, is tasked to sort items stored in lattices of variable dimensions in a time-optimal manner. We systematically analyze the intrinsic optimality structure, which is fairly rich and intriguing, under different levels of item distinguishability (fully-labeled, where each item has a unique label, or partially-labeled, where multiple items may be of the same type) and different lattice dimensions. Focusing on the most practical setting of one and two dimensions, we develop low polynomial time cycle-following-based algorithms that optimally perform rearrangements on 1D lattices under both fully- and partially-labeled settings. On the other hand, we show that rearrangement on 2D and higher-dimensional lattices become computationally intractable to optimally solve. Despite their NP-hardness, we prove that efficient cycle-following-based algorithms remain optimal in the asymptotic sense for 2D fully- and partially-labeled settings, in expectation, using the interesting fact that random permutations induce only a small number of cycles. We further improve these algorithms to provide $1.0\times$-optimality when the number of items is small. Simulation studies corroborate the effectiveness of our algorithms. The implementation of the algorithms from the paper can be found at github.com/arc-l/lattice-rearrangement.

Keywords
Rearrangement, manipulation task planning

1. Introduction
Effective object manipulation Mason (2018), a difficult task and motion planning challenge for machines to master, is essential in fulfilling the true potential of autonomous robots in factories and warehouses, and at home. In tackling the challenge, in the past few decades, while some research has emphasized integrated solutions with promising results Kaelbling and Lozano-Pérez (2011); Levine et al. (2016); Mahler et al. (2017); Zeng et al. (2022), significant efforts have also been devoted to examining key components including rearrangement planning Ben-Shahar and Rivlin (1998); Han et al. (2018); Havur et al. (2014); Haustein et al. (2015); Huang et al. (2019); Kroniris and Bekris (2015); King et al. (2016); Lee et al. (2019); Pan and Hauser (2020); Shome et al. (2021); Stilman and Kuffner (2005); Treleaven et al. (2013) and manipulation Boularias et al. (2015); Chavan-Dafle and Rodriguez (2015); Dogar and Srinivasa (2011); Goldberg (1993); Lynch and Mason (1999), among others, as a thorough understanding of these components is indispensable toward the end goal of enabling truly intelligent object manipulation.

For the same reason, in this work, we perform a systematic structural and algorithmic study on a class of prehensile rearrangement problems where items are stored in individual cells of a lattice of dimension $d = 1, 2, \ldots$ (see Figure 1, bottom row, for example, start and goal configurations), under a novel pick-n-swap model. The stored items may be fully-labeled or partially-labeled. In a fully-labeled setting, each item has a unique label and must go to a specific lattice cell. In a partially-labeled setting, multiple items may have the same label or type and are thus interchangeable. A robotic manipulator, capable of picking up items, carrying them around (a single item at a time), and executing item swaps, is tasked to rearrange items to reach a desired goal configuration in a time-optimal manner. Efficient solutions for rearrangement problems on lattices to restore the items to a desired order find many practical applications, including the rearrangement of products at stores and showrooms (see Figure 1, top row), the sorting of books on bookshelves, and inventory management in autonomous vertical warehouses, to list a few. To accomplish the task, the robot must carefully plan a sequence with which the items are picked and subsequently placed, to

Department of Computer Science, Rutgers University, USA

Corresponding author:
Jingjin Yu, Department of Computer Science, Rutgers University, 110 Frelinghuysen Road, Piscataway, NJ 08854-8019, USA.
Email: jingjin.yu@cs.rutgers.edu
minimize the number of pick-n-swaps and the associated end-effector travel.

Our study of the rearrangement problems under the pick-n-swap prehensile manipulation model is motivated by the recent emergence of dual-gripper end-effectors. Such setups have clear potential in rendering rearrangement tasks more efficiently solvable, given the capacity of carrying more than one item at a time. A dual-gripper end-effector provides functionality that falls between a single-arm-single-gripper setup and a dual-arm setup, leaning closer to a dual-arm setup. While not as flexible as a dual-arm setup, a dual-gripper setup has at least two distinct advantages: it is much more affordable and it does not require a sophisticated motion planning routine to coordinate arm movements. Whereas this research limits the workspace to be lattice-like, the structures analyzed and algorithms developed here can be extended to enable dual-gripper end-effectors to be more efficient in a fully continuous workspace as well.

Due to the intricate interaction between minimizing the number of pick-n-swaps and minimizing the end-effector travel, time-optimal rearrangement on lattices demonstrates a rich and complex structure. In exploring the structure, our study of the lattice rearrangement problem under the pick-n-swap model brings forth the following contributions:

- For the fully-labeled setting, we show that a random rearrangement instance induces the forming of $O(\log m)$ cycles over $m$ items. Each cycle decides an optimal pick-n-swap sequence within the cycle in a deterministic manner. The combinatorial cycle structure applies generally to rearrangement problems, which go beyond lattice-based settings.
- Building on the intriguing and easily computable cycle structure, for 1D lattices, for the fully-labeled setting, efficient algorithms are developed that decide the optimal pick-n-swap sequence which simultaneously minimizes the number of pick-n-swap operations and the end-effector travel. For the partially-labeled setting, our algorithms compute optimal pick-n-swap sequences that minimize the end-effector travel after minimizing the number of pick-n-swaps and allow trade-offs between the two component objectives.
- For 2D and high-dimensional lattices, we show that minimizing the end-effector travel becomes NP-hard for both fully-labeled and partially-labeled settings. This demonstrates a dichotomy in computational complexity between 1D and higher dimensions for optimally solving the lattice rearrangement problem.
- For 2D and high-dimensional lattices, despite the hardness, we show that, because the number of cycles is small, efficient algorithms can be developed that compute optimal solutions in the asymptotic sense. That is, as the number of items goes to infinity, the solution converges to the optimal one, in expectation. For the 2D setting, we further develop additional principled heuristics such that even when the number of items is small, the solution is near-optimal, that is, $1.\times$-optimal.

This paper is based on the conference publication Yu (2021). In comparison to the conference version, this manuscript provides: (1) a more thorough and clearer presentation, including complete, revised proofs for all theorems, (2) significantly stronger guarantees on our algorithms for the labeled settings as well as many unlabeled settings, showing that they often ensure global optimality (Remark 3, Corollary 3.1, Remark 6, Corollary 5.1, and Remark 8), and (3) expanded discussion of alternative
problem formulations and their implications, and interesting open problems.

Related work: Multi-object rearrangement is computationally challenging. As a variation of multi-robot motion planning problems, rearrangement inherits the PSPACE-hard complexity Hopcroft et al. (1984). When geometric constraints must be considered, the relatively simple navigation among movable obstacles (NAMO) problem is shown to be NP-hard Wilfong (1991). If consideration of plan quality is further required, as is often the case in practice, optimally resolving dependency van Den Berg et al. (2009) or planning an optimal object pick-n-place sequence is both NP-hard Han et al. (2018). Rearrangement problems addressed in this paper have a somewhat similar complexity structure.

Despite the high computational complexity, due to its high utility, multi-object rearrangement has been extensively studied, with much research working with nonprehensile (e.g., pushing) manipulations, sometimes assisted with prehensile (grasping) actions. A complete sensing-planning-control framework is proposed in Chang et al. (2012) for the singulation of objects in clutter, which uses both perturbations pushes and grasping actions. In Laskey et al. (2016), hierarchical supervised learning from demonstration is applied to the singulation task. Based on over-segmented RGB-D images, in Eitel et al. (2020), a push proposal network is constructed for push-only singulation. Results that bridge singulation and clutter removal include Huang et al. (2021); Zeng et al. (2018), where learning-based methods are trained to dictate when to push or grasp. To deal with the combinatorial explosions inherent in rearrangement, a randomized kinodynamic planner is employed in Haustein et al. (2015) for rearranging objects on a tabletop, allowing the effective exploration of configurations. King et al. (2015) further integrates a physics-based model to enable the use of the entire robotic arm for complex rearrangement manipulations. A physics-based approach is also used in Moll et al. (2017) for handling grasping in clutter. For a similar task, Bejjani et al. (2018) uses a receding horizon planner with a learned value function that interleaves planning and plan execution.

Huang et al. (2019) has developed an iterated local search (ILS) method for accomplishing multiple tabletop rearrangement tasks including singulation, separation, formation, and sorting of many identically shaped cubes. In Song et al. (2019), Monte Carlo tree search is combined with deep learning for separating many objects into coherent clusters within a bounded workspace. In contrast to Huang et al. (2019), non-convex objects are supported. More recently, Pan and Hauser (2020) proposed a bi-level planner that employs both pushing and overhead grasping that is capable of sorting up to 200 objects.

On work that uses mainly prehensile actions, the earliest is perhaps the study of NAMO problems Stilman and Kuffner (2005); Stilman et al. (2007), which applies backtracking search to effectively deal with monotone and linear NAMO instances, among others. Exploring the dependency graph structure, difficult non-monotone tabletop rearrangement instances are solved using monotone solvers as subroutines Krontiris and Bekris (2015, 2016). Han et al. (2018) shows that tabletop rearrangement embeds a feedback vertex set problem Karp (1972) and the traveling salesperson problem (TSP) Pappadimitriou (1977), both of which are NP-hard, rendering optimally solving these problems intractable. Nevertheless, integer programming models are provided that can quickly compute high-quality rearrangement solutions for practical-sized problem instances. In exploring object dependency structures, studies like Han et al. (2018); Krontiris and Bekris (2015) put more emphasis on the combinatorial aspects of object rearrangement. In Lee et al. (2019), a polynomial-time, complete planner for reasoning rearrangement for object retrieval in a constrained, shelf-like setting is proposed. In a subsequent study Nam et al. (2019), the number of objects to be relocated for retrieval is minimized while considering sensor occlusion.

Organization: The rest of the manuscript is organized as follows. In Section 2, we formally define the lattice rearrangement problems studied in this paper. In Section 3 and Section 4, we provide structural analysis and describe algorithms for the fully- and partially-labeled settings in 1D, respectively. 2D and higher dimensions are examined in Section 5. We then empirically characterize the behavior of our algorithms through simulation studies in Section 6, and discuss and conclude in Section 7.

2. Rearrangement on lattices:

Problem formulation

We examine an object rearrangement problem where items are stored in a d-dimensional lattice or grid, \( d = 1, 2, \ldots \), with dimension \( i \) having a length or capacity of \( m_i \). Items are assumed to be stored at full capacity, that is, an \( m_1 \times \ldots \times m_d \) lattice stores \( \Pi_{i=1}^d m_i \) items (see, e.g., Figure 1). The items may be fully-labeled or partially-labeled. In a fully-labeled setting, each item has a unique label from the set \( \{1, \ldots, \Pi_{i=1}^d m_i\} \) and must be relocated to a specific location or coordinate on the lattice. In a partially-labeled setting, there are \( k > 1 \) types of items where items within each type are considered interchangeable; items of the same type are to be grouped, via rearrangement, into contiguous clusters.

It is assumed that a robotic manipulator is capable of picking up an item, temporarily holding it, moving it to a different location, and swapping the held item with the item at the new location. That is, in a single pick-n-swap operation where a robot end-effector is located above a fixed lattice coordinate, the robot can execute one of the following:

- pick up an item and hold it with the robot’s end-effector (only if no item is already held by the robot),
- swap the item held by the robot’s end-effector with an item inside the lattice at the given coordinate,
place the item held by the robot’s end-effector at the lattice coordinate if no item is already at the coordinate.

In this paper, we use lattice coordinate and cell interchangeably. Such a pick-n-swap model can be readily realized, for example, using two adjacent suction cups, two parallel grippers mounted side-by-side, and so on. The pick-n-swap primitive also models, to a lesser extent, a dual-arm robot or for that matter, how humans perform such rearrangement tasks. We have also examined alternative pick-n-place models, which are discussed in Section 7.

The pick-n-swap model leads to a natural partition of the robot’s operations into pick-n-swap operations and end-effector travel operations. A rearrangement plan can then be represented as a sequence of lattice coordinates, \( P = \{ p_0, p_1, \ldots, p_N \} \), where the robot end-effector starts from the rest position \( p_0 \) and sequentially executes pick-n-swap operations at \( p_1, p_2, \) and so on. For quantifying the quality of a rearrangement plan \( P \), it is assumed that each pick-n-swap incurs a (time) cost of \( c_p \) and the (time) cost of traveling a unit distance (the distance between two adjacent cells) by the end-effector is \( c_t \). The total cost of completing a rearrangement plan is then

\[
J_T(P) = Nc_p + \sum_{i=0}^{N} \text{dist}(p_i, p_{i+1})c_t, \tag{1}
\]

where \( \text{dist}(p_i, p_{i+1}) \) is the effective distance traveled by the end-effector between \( p_i \) and \( p_{i+1} \); \( p_{N+1} = p_0 \). The distance metric may be \( L_1 \), Euclidean (\( L_2 \)), and so on, depending on the end-effector’s motion mechanism. For example, if a human is to arrange shoes for the setup shown in Figure 1 [top right], then horizontal travel is likely the main source of travel distance cost. This study works with Euclidean distances, that is, \( \text{dist}(p_i, p_{i+1}) = \|p_i - p_{i+1}\|_2 \).

Remark 1. It is straightforward to observe that plans computed for the pick-n-swap model are reversible; the associated optimality guarantees, if any, will also carry over. This implies that the algorithms developed in this work can also be applied to (near-)optimally randomize the locations of items stored in lattices.

Because a pick-n-swap operation requires precise robot arm placement and grasp planning/execution involving contact between the end-effector and objects, similar to Han et al. (2018), it is assumed that the total pick-n-swap cost dominates the total end-effector travel cost. That is, on the right side of equation (1), the first term will be considered first, yielding a sequential optimization problem in which minimizing the number of pick-n-swaps, \( N \), takes priority.

Remark 2. In this paper, we use “optimal in the asymptotic sense” to mean that the optimality ratio will approach 1 as the number of items goes to infinity, as is commonly used in asymptotic analysis in mathematics. This is not to be taken as the same as asymptotically optimal algorithms used in computer science, which generally means that an algorithm computes \( O(1) \)-optimal solutions in the worst case.

Remark 3. In general, the sequential optimization of two objectives that are not orthogonal to each other does not yield globally optimal solutions for the sum of the two objectives. In our case, however, the sequential optimization procedure leads to global optimality for the fully-labeled settings and some partially-labeled settings. For all partially-labeled settings, our algorithms can also be adjusted to globally balance between more pick-n-swaps and more end-effector travel based on the ratio \( c_p/ c_t \).

As practical robotic rearrangement operations are generally limited to one and two dimensions, our study also centers on the cases of \( d = 1, 2 \), with some discussions of higher dimensions. In the \( d = 1 \) case, let \( m_1 = m_2 \) be the capacity of the lattice, viewed as a single row. It is assumed that \( p_0 \) is at the leftmost cell. In the labeled setting, the lattice is equivalent to a row with its cells labeled \( 1, \ldots, m \) from left to right; the rearrangement problem is then to relocate the item with label \( i \) to the \( i \)-th cell. In the partially-labeled setting, there are \( k \) types of items, \( k < m \), possibly to be arranged into contiguous clusters (see, e.g., Figure 1 [lower left]), though we do not make assumptions about the goal configurations.

In the \( d = 2 \) case, we have an \( m_1( \text{row}) \times m_2( \text{column}) \) lattice; \( p_0 \) is at the top left cell of the lattice. In the labeled setting, it is assumed without loss of generality that lattice cells are labeled following a column-major order: cells in column \( i \), \( 1 \leq i \leq m_2 \), are labeled \( (i-1)m_1 + 1, \ldots, im_1 \), from top to bottom, respectively. In the goal configuration, the item labeled \( j \) must be located at cell \( j \). In the partially-labeled setting, besides considering arbitrary goal configurations, two natural goal configuration patterns are analyzed in more detail, with one having the goals of the same type aggregated (e.g., Figure 1 [bottom right]) and the other having each type occupying a single lattice column.

For convenience, we denote the one-dimensional labeled and partially-labeled problems, as stated above, as the labeled one-dimensional rearrangement (LOR) problem and the partially-labeled one-dimensional rearrangement (POR) problem, respectively. The two-dimensional problems corresponding to LOR and POR are named as LTR and PTR, respectively. In this study, we analyze the structural properties of LOR/POR/LTR/PTR as induced by minimizing the objective specified in equation (1). Based on the findings, we design efficient algorithms for computing (near-)optimal plans for these rearrangement problems.

3. Fully-labeled rearrangement in 1D

For combinatorial optimization problems involving the reconfiguration of many bodies, the labeled settings are often more challenging (e.g., Solovey and Halperin (2014); Han et al. (2018)). In arranging items stored in lattices, in
contrast, the labeled case is structurally simpler. This is due to the “linear” dependencies among the items to be relocated, as will become clear shortly. This allows the computation of (exact) optimal solutions for \( \text{LOR} \).

Recall that \( \text{LOR} \) requires rearranging a row of \( m \) items. Therefore, \( \text{LOR} \) can be viewed as going from one random permutation of \( m \) labeled items to the canonical order \( [m] = 1, \ldots, m \). An \( \text{LOR} \) instance is therefore fully specified by a permutation \( \pi \) of \( [m] \), where \( \pi_i, 1 \leq i \leq m \) is the label of the item that occupies cell \( i \) in the initial configuration. We start with a simple greedy cycle-following algorithm, \( \text{SweepCyclesLOR} \), that solves \( \text{LOR} \) near-optimally, in expectation, assuming that the initial \( \text{LOR} \) instance is randomly generated. \( \text{SweepCyclesLOR} \) minimizes the end-effector travel distance to near-optimality after minimizing the number of pick-n-swap operations to the smallest possible. Then, we describe a more involved algorithm, \( \text{OptPlanLOR} \), that computes a rearrangement plan that also minimizes the end-effector travel. Unlike \( \text{SweepCyclesLOR} \), \( \text{OptPlanLOR} \) guarantees solution optimality for each individual instance, that is, it does not depend on the \( \text{LOR} \) instance being random. After presenting \( \text{SweepCyclesLOR} \) and \( \text{OptPlanLOR} \), we further show that although they perform sequential optimization of equation (1), their optimality guarantees are in fact global.

3.1. Cycle-following with left to right sweeping

Consider an \( \text{LOR} \) instance with nine items and the initial configuration \( \pi = (3, 2, 4, 1, 7, 6, 9, 5, 8) \). The instance can be solved by starting with the leftmost item that needs rearrangement, in this case 3, and moving it from cell 1 to cell 3, which replaces item 4 that in turn replaces item 1. This yields a cycle \( 3 \rightarrow 4 \rightarrow 1 \) (see Figure 2), often written as \( (341) \). After following this cycle, the end-effector returns to cell 1. The end-effector then works with the next leftmost item that is not at goal, 7, inducing another cycle 7985. Altogether, there are two cycles, \( (341) \) and \( (7985) \), containing 7 items in total (here, we deviate slightly from how cycles are normally counted in permutations, where a single item in the correct cell would be counted as a cycle containing a single element as well, which we ignore by default).

These cycles are uniquely determined by the initial configuration \( \pi \). Noticing that each cycle requires one more pick-n-swap than the number of items in the cycle, the instance is solved using a minimum \((3 + 1) + (4 + 1) = 9\) pick-n-swaps. After processing all cycles, the \( \text{LOR} \) instance is solved and the end-effector returns to its rest position (cell 1). The straightforward algorithm \( \text{SweepCyclesLOR} \), a natural greedy algorithm, is outlined in Alg. 1, in which the routine \( \text{Swap}(i, j) \) will pick up item \( j \) at cell \( \ell \) (if it is not \( \epsilon \), denoting a phantom item) and swap it with item \( i \) being held (if it is not \( \epsilon \)). It is clear that \( \text{SweepCyclesLOR} \) runs in linear or \( O(m) \) time.

The optimality properties of \( \text{SweepCyclesLOR} \) are established in Proposition 3.1 and Proposition 3.2.

**Proposition 3.1.** \( \text{SweepCyclesLOR} \) minimizes the number of required pick-n-swap operations for \( \text{LOR} \).

**Proof.** To solve a given cycle, it is clear that an item on the cycle must be first picked up without any other items of the same cycle already held by the end-effector (note that the end-effector may hold items from other cycles). Therefore, for each cycle, one additional pick is unavoidable. Induction over the cycles of \( \pi \) then proves the proposition.

**Proposition 3.2.** After minimizing the number of pick-n-swaps, \( \text{SweepCyclesLOR} \) computes optimal solutions for \( \text{LOR} \) in the asymptotic sense, minimizing end-effector travel in expectation, assuming that \( \pi \) is a random permutation.

**Proof.** Given the initial configuration \( \pi \), rearranging each cycle will cause the end-effector to end at where it starts following \( \text{SweepCyclesLOR} \), which is the leftmost location where a cycle starts. The total distance traveled by the end-effector can be factored into (i) the distance traveled to solve each cycle, and (ii) the overhead of traveling after solving a cycle to the next, including the overhead before starting the first cycle and after completing the last cycle. For (i), each cycle must be rearranged to minimize the number of pick-n-swaps, the distance for solving each cycle is already at the minimum possible. For (ii), the end-effector travels from left to right in between solving cycles. This adds no more than \( 2m \) distance in total. We show that \( 2m \) is inconsequential in comparison to the distance incurred by (i). To compute distance incurred by (i), given a random \( \pi \), for a fixed \( i \), the expected distance between item \( i \) (located in cell \( \pi_i^{-1} \), which is uniformly randomly distributed between 1 and \( m \)) and cell \( i \) is

\[
\mathbb{E}_i = \frac{i - 1 + \cdots + 1 + 0 + 1 + \cdots + m - i}{m}.
\]

Tallying over \( i \) from 1 to \( m \), the expected total distance due to resolving all cycles is then

\[
\mathbb{E} = \frac{1 + \cdots + 1 + 0 + 1 + \cdots + m - 1}{m} = \frac{m + 1}{2} - 1.
\]
The number of items that are involved is generally not very large.

### 3.2. Cycle sweeping with cycle switching

In SWEEPCYCLESLOR, each cycle is followed through one by one, without switching to another cycle before one is complete. If we interleave the completion of cycles, however, end-effector travel can be shortened without adding the number of pick-n-swaps. In the example illustrated in Figure 3, the plan by SWEEPCYCLESLOR uses 6 pick-n-swaps and a total end-effector distance of 14. The alternative plan, which breaks cycles during the rearrangement process, uses also 6 pick-n-swaps but only a total distance of 10.

From the example, we observe a switch from one cycle to another cycle before completing rearranging the first can reduce end-effector travel. The saved distance corresponds to reducing the end-effector travel without holding an item.

In the example, the bottom plan avoids traveling from the leftmost location to item 5’s initial location (and back), saving a distance of 2 + 2 = 4. The observation leads to the OPTPLANLOR algorithm that groups cycles for more effective rearrangement. To describe the algorithm, some definitions are in order. Given a permutation \( \pi \), let \( C_\pi \) be the set of all cycles induced by \( \pi \). For a cycle \( C \subset C_\pi \), let \( \min(C) \) and \( \max(C) \) be the smallest and largest item labels of items in \( C \), respectively.

We group elements of \( C_\pi \) into equivalence classes as follows. Initially, let \( C_\pi := \{ \{ c_1 \} \mid c_1 \in C_\pi \} \). Elements of \( C_\pi \) are grouped (via union) if their ranges overlap. That is, for two cycle groups \( C_1, C_2 \subset C_\pi \), if their ranges \([\min(C_1), \max(C_1)]\) and \([\min(C_2), \max(C_2)]\) intersect, we update \( C_\pi \) to \( \{C_\pi \setminus \{C_1, C_2\}\} \cup (C_1 \cup C_2) \).

Since there are only a finite number of cycles, the grouping process will stop and yield a set of cycle groups that are pairwise disjoint. Then, a rearrangement plan can be computed as follows. Let the leftmost group of cycles be \( C_1 = \{c_1, c_2, \ldots\} \) where \( \min(c_1) < \min(c_2) < \ldots \). We start performing cycle-following on \( c_1 \) until the end-effector passes over location \( \min(c_2) \) (where \( c_2 \) starts) for the first time, at which point we pause following \( c_1 \) and switch to following \( c_2 \). Similarly, as \( c_2 \) is being followed, we will switch to following \( c_3 \) as the end-effector passes over...
min \(|c_1|\) for the first time, and so on. At some point, the end-effector will reach \(\max(|c_1|)\), the rightmost reach of the cycle group \(C_1\). If there are additional cycle groups on the right of \(C_1\), we pause working with \(C_1\) and start working with the next cycle group on the right of \(C_1\), and return to \(C_1\) after all items to the right of \(C_1\) are rearranged (iteratively).

An outline of the \textsc{optplanlor} algorithm is given in Alg. 2, which in turn calls Alg. 3 and Alg. 4. It is not difficult to observe that \textsc{optplanlor} runs in \(O(m)\) time. We further prove its distance optimality.

\begin{algorithm}[h]
\caption{\textsc{optplanlor} \((\pi)\)}
\begin{algorithmic}[1]
\State retrieve cycles as singleton sets
\State \(C_e \leftarrow \text{getcycles}(\pi)\)
\State group cycles with overlapping ranges
\While{\(\exists C_1, C_2 \in C_e\) with overlapping ranges}
\State \(C_e \leftarrow (C_e \setminus \{C_1, C_2\}) \cup \{C_1 \cup C_2\}\)
\EndWhile
\State process cycle groups
\State \(C \leftarrow \) leftmost cycle group in \(C_e\)
\State \textsc{processcyclegroup}(\(p, C, \pi, C_0\))
\end{algorithmic}
\end{algorithm}

\begin{algorithm}[h]
\caption{\textsc{processcyclegroup} \((p, C, \pi, C_0)\)}
\begin{algorithmic}[1]
\State \(c \leftarrow \) leftmost cycle in cycle group \(C\)
\State \textsc{processcycle}(\(p, c, C, \pi, C_0\))
\end{algorithmic}
\end{algorithm}

\begin{algorithm}[h]
\caption{\textsc{processcycle} \((p, c, C, \pi, C_0)\)}
\begin{algorithmic}[1]
\State \(c' \leftarrow \) leftmost cycle in group \(C\) after \(c\)
\State \(C' \leftarrow \) leftmost cycle group to the right of \(C\) in \(C_e\)
\State process cycle \(c\) from left
\State \(i \leftarrow \) \(\min(\pi)\)\; \(\text{swap}(i, p, \pi)\); \(g \leftarrow \pi_i\)
\While{\(g \neq i\)}
\State switch cycles within group?
\EndWhile
\State \(c' \neq \) null and \(g > \min(\pi')\)
\State \textsc{processcycle}(\(p, c', C, \pi, C_0\))
\EndIf
\State \(c' \leftarrow \) leftmost cycle in cycle group \(C\) after \(c'\)
\State switch to the next cycle group?
\If{\(g = \max(\pi)\) and \(c' = \) null}
\State \textsc{processcyclegroup}(\(p, C', \pi, C_0\))
\EndIf
\State following the current cycle
\State \text{\textsc{swap}}(\(g, g, \pi_\beta\)); \(g \leftarrow \pi_\beta\)
\State \text{\textsc{swap}}(\(i, i, p\))
\end{algorithmic}
\end{algorithm}

**Theorem 3.1.** For \textsc{lor}, \textsc{optplanlor} computes a re-arrangement plan with minimal end-effector travel, among all plans that minimize the number of pick-n-swaps.

**Proof.** We prove the theorem by showing that: (a) end-effector travel is minimal within each cycle group, and (b) end-effector travel between two adjacent cycle groups is minimized (this is trivially true).

To prove (a), for an item with label \(i\) let \(\pi^{-1}_i\) be its initial location (this is natural since we then have \(\pi^{-1}_i = i\)). Notice that the minimum distance the end-effector must travel while carrying item \(i\) is \(|\pi^{-1}_i - i|\). In \textsc{processcycle}, within a cycle group, an item \(i\) is moved exactly a total distance of \(|\pi^{-1}_i - i|\), even though it may be done in multiple steps. For example, in the top figure of Figure 3, item 4 is moved a distance of 3 in a single move. In the bottom figure, item 4 is first moved a distance of 2 and later followed by a move of distance 1. We note that an item \(i\) may be moved more than \(|\pi^{-1}_i - i|\) by \textsc{optplanlor} if it is carried from one cycle group to another, but such travel is attributed to travel between adjacent cycle groups, which is unavoidable.

It is clear that Proposition 3.3 and Theorem 3.1 optimize equation (1) globally because each of the two terms of equation (1) is optimized globally.

**Corollary 3.1.** For \textsc{lor}, in minimizing Equation (1), \textsc{sweepcycleslor} computes globally optimal solutions in the asymptotic sense in expectation and \textsc{optplanlor} computes globally optimal solutions.

**Remark 5.** Alg. 2 has a pre-processing stage where the cycles are ordered. We note that this stage can be interleaved with the actual processing stage (Alg. 3). We have opted for the standalone pre-processing stage to make the algorithm hopefully more clear and the running time analysis more straightforward.

### 4. Partially-labeled rearrangement in 1D

In the (fully) labeled setting, each item requiring rearranging has a single possible destination, limiting the combinatorial explosion of feasible rearrangement plans. This is no longer the case in the partially-labeled setting where each item of a given type can have multiple goal arrangements (e.g., in Figure 1 [left], item 6 may go to locations 4, 5, or 6). In other words, a partially-labeled problem can be viewed as many labeled problems mixed together, demanding additional computational efforts for selecting a best labeling to solve the problem. Nevertheless, we show that the one-dimensional partially-labeled problem \textsc{por} can still be optimally solved despite the significant added complexity.

We describe an optimal algorithm for \textsc{por}, \textsc{optplanpor}, that applies to arbitrary goal configurations. Because the algorithm is somewhat involved, for readability, we describe the algorithm over a natural but restricted class of \textsc{por} instances where each type of item form a contiguous section in the goal configuration (see, e.g., Figure 4). In such instances, for a given type \(1 \leq t \leq k\), let \(n_t\) be the number of items of type \(t\), \(\sum_{t=1}^{k} n_t = m\). In the goal configuration, items of type \(t\) fill locations between \(\left\lceil \frac{1}{t} \sum_{i=1}^{t-1} n_i + 1 \right\rceil\) and \(r_t = \sum_{i=1}^{t} n_i\), inclusive. Define \(\text{range}(t) = [l_t, r_t]\) An instance of this restricted \textsc{por} problem is then fully specified by the initial configuration of the items as a sequence of types, that is, \((t_1, \ldots, t_m)\), where \(1 \leq t_i \leq k\).

#### 4.1. Algorithm description

In the first phase, simple matchings are made between initial and goal locations. Starting from the left side, for the item at location \(i\) with type \(t_i\), \(i \notin \text{range}(t_i)\), we select its goal to be the leftmost available one. Figure 4 [top left] shows an example. The matchings induce a set of cycles (Figure 4, [top right]), with the addition of the vertical edges corresponding to pick-n-swap operations, which are distance optimal for end-effector travel but do not minimize the number of pick-n-swaps. As is the case with \textsc{lor}, each cycle requires one more pick-n-swap plus the number of items in the cycle. We
call the subroutine FORMCYCLES and note that additional end-effector travel between these cycles is needed for obtaining a full rearrangement plan.

In the second phase, the initial set of cycles are merged in a pairwise manner when two cycles have edges going to the same item type in the same direction (either both left or both right). Formally, two cycles can be merged in this phase if they contain two items $t_i$ and $t_j$, respectively, and $t_i$, $t_j$ satisfy $t_i = t_j$ and $i, j$ are either both on the left of range($t_i$) or both on the right of range($t_i$). For example, the left two cycles in Figure 4 [top right] can be merged by swapping the goals of the left two type 2 edges (first and third from the top row), yielding Figure 4 [bottom left]. The process reduces the number of pick-n-swap operations but does not incur additional end-effector travel, because each merge keeps the total distance unchanged. We call this subroutine MERGECYCLES.

In the third phase, cycles are further merged in a pairwise manner if two cycles have edges going to the same type but in different directions. Formally, two cycles can be merged in this phase if they contain two items $t_i$ and $t_j$, respectively, $t_i$, $t_j$ are on two different sides of range($t_i$). For example, in Figure 4 [bottom left], the two cycles both have edges going into the third type, but in different directions. Merging these two cycles, as shown in Figure 4 [bottom right], will reduce the number of pick-n-swaps by one but will incur additional end-effector travel. We note that to ensure total distance optimality, the merge here needs to be done by computing a minimum spanning tree (MST) based on the added distances when two cycles are merged. We call the associated subroutine MERGECYCLESMST.

After the third phase completes, we obtain a set of cycles that minimizes the number of pick-n-swaps. These cycles are now much like the cycles in the labeled case and are swept through and switched similarly. We call this last subroutine GROUPSWEEPCYCLESPOR, which connects all cycles and composes the full rearrangement plan.

4.2. Algorithm outline and optimality properties

The OPTPLANPOR algorithm and the MERGECYCLESMST subroutine are outlined in Alg. 5 and Alg. 6. The other subroutines, FORMCYCLES, MERGECYCLES, and GROUPSWEEPCYCLESPOR are relatively straightforward to implement based on the description; we omit the pseudo-code.

**Algorithm 5: OPTPLANPOR ($t_1, \ldots, t_m$)**

1. $C \leftarrow$ FORMCYCLES ($t_1, \ldots, t_m$)
2. $C' \leftarrow$ MERGECYCLES ($C$)
3. $C'' \leftarrow$ MERGECYCLESMST ($C'$)
4. GROUPSWEEPCYCLESPOR ($C''$)

**Algorithm 6: MERGECYCLESMST ($C$)**

1. $V_C \leftarrow C; E_C \leftarrow \{\}$; $W \leftarrow \{\}$; $G_C \leftarrow (V_C, E_C, W)$;
2. compute ‘merge distance’ between cycles
3. for all $c_i, c_j \in V_C$ do
4. $E_C \leftarrow E_C \cup \{c_{ij} = (c_i, c_j)\}$
5. $w_{ij} \leftarrow CYCLEDISTANCE(c_i, c_j)$; $W \leftarrow W \cup \{w_{ij}\}$
6. compute a minimum spanning forest over $G_C$
7. $F \leftarrow$ MST ($G_C$)
8. for each tree $T$ in $F$ do
9. while $T$ has an edge $e_{ij} = (c_i, c_j)$ do
10. $c_i \leftarrow$ merge $c_i$ and $c_j$
11. collapse edge $e_{ij}$ in $T$
12. Add the single cycle $c$ in $T$ to $C'$
13. return $C'$

In MERGECYCLESMST, a graph $G_C$ is constructed that captures the distances between cycles that can be merged to reduce the number of pick-n-swaps. The function CYCLEDISTANCE computes the closest distance between two cycles for merging in the obvious way. This distance in Figure 4 [bottom right] is 1 (we note that the actual swap will incur a cost doubling this distance). After $G_C$ is constructed, which can have multiple connected components, a minimum spanning tree algorithm is executed, for example, Prim’s algorithm (Prim (1957)), yielding a spanning forest $F$ of $G_C$. Each tree $T$ in $F$ will result in a single merged cycle.
different merged cycles from different trees cannot be merged further to reduce the number of pick-n-swaps.

We proceed to establish key properties of OptPlanPOR and the subroutines. From here on, arbitrary goal configurations are assumed unless stated otherwise.

**Proposition 4.1.** For POR with $k$ types of items, MergeCycles creates cycles that are distance optimal. When goals are aggregated by item types, there are at most $k$ cycles after completing the MergeCycles subroutine.

**Proof.** FORMCycles creates cycles that are distance optimal by construction. This is clear by looking at the minimum number of times the end-effector must pass over or stop at a given cell of the lattice in order to move items of a given type to the goals. For each initial and goal pair, and type, this number is fully determined and realized by FORMCycles. For example, for sorting the first type in Figure 4, the end-effector must pass cell 3 at least once and must also stop at the cell at least once, because all type 1 items should be to the left of cell 4 in the goal configuration and there are two type 1 items in the initial configuration to the right of cell 3. The cycles created by FORMCycles realize this minimum end-effector travel. Then, because MERGECYCLES does not add additional distance, the total distance remains at the minimum.

For the rest of the proposition, in the case where the goal configuration has the types aggregated, each type can participate in at most two cycles after MERGECYCLES is performed. For example, in Figure 4 [bottom left], two cycles exist that contain items of type 3. There is a single cycle for types 1, 2, and 4. Moreover, the leftmost and rightmost types in the goal configuration can each only participate in a single cycle. This is because all items of type $t$ in the initial configuration to the left (or right) of range($t$) will be in a single cycle after MERGECYCLES is performed, by construction. Since each cycle requires at least two types of items, for $k$ types, there can be at most $k - 1$ cycles after MERGECYCLES.

**Lemma 4.1.** For POR, MergeCyclesMST reduces the number of pick-n-swaps to the minimum while incurring the minimum amount of additional end-effector travel.

**Proof.** After MERGECYCLES, for each pair of the resulting cycles, they cannot be merged to reduce the number of pick-n-swaps without incurring additional end-effector travel. To see that this is the case, after MERGECYCLES, for each pair of cycles, say $c_1$ and $c_2$, they can be merged to reduce the number of pick-n-swaps if and only if they contain items of the same type. Suppose that $c_1$ and $c_2$ both involve items of the same type, say $t$ (there could be multiple such types for a pair of cycles), and $c_1$ is to the left of $c_2$. Then it must be the case that edges of $c_1$ for restoring type $t$ items and edges of $c_2$ for restoring type $t$ items do not intersect (by the construction of MERGECYCLES). The third type in Figure 4 [bottom left] gives an example. For a type $t$, denote the set of edges of $c_1$ (resp., $c_2$) for restoring type $t$ items as $E'_1$ (resp., $E'_2$). To merge $c_1$ and $c_2$, it requires for exactly one $t$, one edge of $E'_1$ and one edge of $E'_2$ to swap their ends so that $E'_1$ and $E'_2$ will cross over range($t$). This operation will incur additional end-effector travel that cannot be avoided.

The optimal way to merge two cycles $c_1$ and $c_2$ sharing the same item types is by doing the merge on the type $t$ where $E'_1$ and $E'_2$ are closest to each other. Without loss of generality, assume $E'_1$ is to the left of $E'_2$. The distance between $E'_1$ and $E'_2$ is then simply the distance between the rightmost position reached by $E'_1$ and the leftmost position reached by $E'_2$. Computing this over all applicable types then yields the distance between $c_1$ and $c_2$ (this is done in CYCLEDISTANCE in MERGECYCLESMST).

For all cycles that can be merged into a single cycle, the merging process naturally induces a spanning tree of the involved cycles. The optimal merging sequence is then given by a minimum spanning tree as computed in MERGECYCLESMST.

**Theorem 4.1.** For POR, OptPlanPOR computes a re-arrangement plan with the minimum end-effector travel after minimizing the total number of pick-n-swaps.

**Proof.** By Lemma 4.1, after MERGECYCLESMST, we obtain a set of cycles corresponding to the least number of pick-n-swaps and the minimum end-effector travel to realize this. What is left is to “connect” these cycles together to yield a complete rearrangement plan. This connection process is performed using GROUPSWEPCYCLES, which maintains the number of pick-n-swaps and adds only the minimum travel distance for cycles that are spatially disjoint. As a result, the overall OPTPLANPOR algorithm ensures distance optimality after minimizing the number of pick-n-swaps.

**Remark 6.** From the discussions, theorems, and proofs, before the step of running MergeCyclesMST, OptPlanPOR optimizes both terms of equation (1) simultaneously. In executing MergeCyclesMST, in merging two cycles $c_1$ and $c_2$, let the distance cost of merging them be $\text{dist}(c_1, c_2)$. If $\text{dist}(c_1, c_2) \geq c_p$, and $c_1$ and $c_2$ overlap (meaning that $c_1$ and $c_2$ can be swept without additional distance cost but with one extra pick-n-swap), then merging the two cycles will increase the total cost as defined by equation (1). In this case, we can skip the merging of $c_1$ and $c_2$. More generally, OptPlanPOR can be augmented to globally optimize equation (1) based on different $c_p$: $c_t$ ratios. Because the required change is fairly involved and the benefit of doing global optimization is small in comparison to optimizing equation (1) sequentially (since MergeCyclesMST will only apply to a very few cycles for random instances), we omit further details.

In terms of running time, FORMCycles can be performed in linear time using multiple passes over the initial and goal configurations. During the execution of FORMCycles, data structures can be built so that cycles are associated with types. With the proper data structures, MERGECYCLES can be run in linear time by going through the types one by one, resulting in an $O(m)$ running time. For MERGECYCLESMST, computing $G_C$ and the distances between cycles can be done in linear time through amortization analysis. Computing a minimum spanning tree can be done in $O(|E_C| + |V_C| \log |V_C|)$ time Johnson (1975). Merging cycles can be done at the
same time as the minimum spanning tree is built, which does not take additional time. \textsc{groupsweepcyclespor} takes $O(m)$ time. The total running time of \textsc{optplanpor} is then $O(m(n + |E| + |F| \log |V|)$. If the goals are aggregated based on types, there are at most $k - 1$ cycles (Proposition 4.1) entering \textsc{mergecyclestmst}, resulting in a total running time of $O(m + k \log k)$. In the general case, the running time is $O(m \log m)$.

5. Rearrangement in 2D and higher dimensions

For higher dimensions, it is straightforward to observe that the cycle-following structure for \textsc{lor} and \textsc{por} carry over. However, minimizing end-effector travel becomes more challenging, as the problem now contains a TSP, as will be shown. Nevertheless, strategies can be derived that yield optimality in the asymptotic sense.

5.1. Fully-labeled 2D and higher dimensions

5.1.1. Labeled rearrangement in 2D. An \textsc{ltr} instance is specified by its lattice dimension $m_1$, $m_2$, and a permutation $\pi$ of $[m_1 m_2]$. Similar to \textsc{lor}, minimizing the number of pick-n-swaps for \textsc{ltr} can be achieved via cycle-following. This allows proving results for \textsc{ltr} similar to Propositions 3.1 and 3.2. A natural extension \textsc{tosweepcycleslor} can be made to support the 2D setting: for an \textsc{ltr} instance with an initial configuration $\pi$, we compute all its cycles as $c_1, \ldots, c_k$. The new algorithm, which we call \textsc{sweepcyclesltr}, again performs cycle-following of the cycles and moves to cycle $c_{i+1}$ after completing cycle $c_i$.

Then, we note that the cycle switching procedure for \textsc{lor} (e.g., the process illustrated in Figure 3) can be generalized to \textsc{ltr}. That is, for consecutive items $a_1$ and $b_1$ belonging to cycle $c_1$ and an item $a_2$ belonging to cycle $c_2$, instead of going from $a_1$ to $b_1$, it can potentially save travel distance by going from $a_1$ to $a_2$, finishing $c_2$ (and possibly additional cycles), and then going to $b_1$ to finish $c_1$. The optimal switching schedule can be computed using a minimum spanning tree procedure somewhat similar to \textsc{mergecyclestmst}. There is, however, a key difference: in \textsc{mergecyclestmst}, the cycles to be merged have symmetric distances but the distance from cycle $c_1$ to $c_2$ and the distance from cycle $c_2$ to $c_1$ are different in merging cycles for \textsc{ltr}. That is, the graph where the cycles are vertices, over which a minimum spanning tree is to be constructed, is now directed. This means that we need to apply a directed minimum spanning tree algorithm (Chu (1965); Edmonds (1967)). The end-effector rest position should also be considered in computing the directed minimum spanning tree.

We call the overall cycle switching procedure for \textsc{ltr} as \textsc{switchcyclesltr}, which clearly runs in polynomial time. We omit the pseudo-code for these two algorithms given their similarity to \textsc{lor} and \textsc{por}. These algorithms are optimal in the asymptotic sense, in expectation.

**Proposition 5.1.** The number of pick-n-swaps is minimized by \textsc{sweepcyclesltr} and \textsc{switchcyclesltr}. Furthermore, they compute distance-optimal solutions in the asymptotic sense for \textsc{ltr} in minimizing end-effector travel in expectation, assuming that $\pi$ is a random permutation.

**Proof.** It is clear that \textsc{sweepcyclesltr} and \textsc{switchcyclesltr} minimize the number of pick-n-swaps. Without loss of generality, assume $m_1 \geq m_2$. Following similar reasoning as used in the proof of Proposition 3.2, a random permutation $\pi$ will require an average distance $E_\pi = \Omega((m_1 + m_2) m_1 m_2) = \Omega(m_1^2 m_2)$. In expectation, there are $H_{m_1 m_2} \approx \log m_1 m_2 = O(\log m_1)$ cycles. Traveling through all these cycles once then incurs a distance cost of no more than $O(m_1 \log m_1)$, which is inconsequential as compared to $\Omega(m_1^2 m_2)$.

For $m_1 = m_2 = m$, the expected distance from cycles over a random \textsc{ltr} instance can be readily computed as $((2 + \sqrt{2} + 5\ln(\sqrt{2} + 1))/15)m^2 \approx 0.52 m^2$. We omit the details but note that this is equivalent to computing the average distance between two points in a unit square (see, e.g., Santaló (2004)), via a double integration, and multiply that distance by $m^2$. \textsc{sweepcyclesltr} can be implemented by making a constant number of linear passes over the $m_1 \times m_2$ lattice, yielding an $O(m_1 m_2)$ running time. \textsc{switchcyclesltr} requires more work; a naive implementation requires $O(m_1^2 m_2^2)$ running time, mainly for checking switching distances between cycles. In a sense, the optimality (in the asymptotic sense) provided by \textsc{sweepcyclesltr} and \textsc{switchcyclesltr} is the best one can do, because optimizing distance for \textsc{ltr}, unlike for \textsc{lor}, is NP-hard. We note that the hardness holds regardless of whether the number of pick-n-swaps is minimized.

**Theorem 5.1.** Minimizing the total end-effector travel distance for \textsc{ltr} is NP-hard.

**Proof.** We prove the claim via a reduction from the Euclidean TSP Papadimitriou (1977). Given a Euclidean TSP instance specified by a set of points embedded in a rectangular region (e.g., Figure 5, left), we superimpose a lattice over the region at some resolution $m_1 \times m_2$. To construct the \textsc{ltr} instance, we set the initial condition $\pi$ to be the identity permutation, that is, $\pi_i = i$ for all $1 \leq i \leq m_1 m_2$. Then, we update $\pi$ for each of the internal points (i.e., excluding the top left “starting” point) in the TSP instance. For a given internal point in the TSP instance, let its coordinates be $(x_1, x_2)$. Without loss of generality, we may assume that $(x_1, x_2)$ satisfy $1 < x_1 < m_1$ and $1 < x_2 < m_2$. We update $\pi$ such that $\pi_{\pi_1(x_2 + 1)} = m_2(x_2 - 1) + 1$ and $\pi_{\pi_2(x_1 - 1) + x_2} = m_1 x_2 + x_1$. That is, each internal point in the TSP instance is converted to two adjacent items (pairs of adjacent items in Figure 5, right) that must be exchanged. We refer to each pair of the adjacent items to be exchanged as a cluster.

By selecting sufficiently large $m_1$ and $m_2$, the end-effector travel cost for solving each cluster becomes negligible. Therefore, for the \textsc{ltr} instance, the optimal end-effector travel cost is determined by the cost of end-effector travel between clusters. An optimal solution to the
TSP problem then maps to an optimal solution to the LTR instance that minimizes end-effector travel (minor details are omitted).

On the other hand, in any valid solution to the LTR instance, the end-effector must start from the rest position (the top left point), go to each cluster to make the exchange, and eventually return to the rest position. Because the travel cost for solving each cluster locally is negligible (again, some minor but formal arguments are omitted here), a distance optimal solution then translates back to an optimal solution to the initial Euclidean TSP instance.

It is clear that the decision version of the LTR instance is NP-complete because the distance of a given rearrangement plan can be checked in linear time.

**Remark 7.** While LOR (and POR) can be optimally solved in polynomial time, LTR becomes NP-hard, implying that we cannot optimally solve it exactly in polynomial time unless P = NP. What is changed is that LOR is a 1D problem whereas LTR is a 2D problem. Such transitions of computational complexity due to dimension changes are also observed elsewhere, for example, computing shortest paths or the visibility graph becomes hard as dimension rises from two to three Canny and Reif (1987); Mitchell and Sharir (2004).

Similar to the case for LOR, we make the observation that the algorithms for LTR simultaneously optimize the two terms in equation (1): both SWEEPCYCLESLTR and SWITCHCYCLESLTR ensure that the number of pick-n-swaps is minimized and the total end-effector travel is optimal in the asymptotic sense. As a result, these algorithms provide global optimality guarantees for LTR.

**Corollary 5.1.** For LTR, in minimizing Equation (1), SWEEPCYCLESLTR and SwitchCyclesLTR compute globally optimal solutions in the asymptotic sense.

5.1.2. Labeled rearrangement in higher dimensions. We briefly discuss extensions to dimensions higher than two. It is straightforward to observe that Proposition 5.1 and Theorem 5.1 continue to hold in higher dimensions through a direct embedding, that is, a two-dimensional problem can be readily reduced to a $d$-dimensional problem via adding additional orthogonal dimensions.

**Corollary 5.2.** For labeled rearrangement on $d$-dimensional lattices, with a fixed $d \geq 2$, a cycle-following procedure, after minimizing the number of pick-n-swaps, also yields a total end-effector travel distance that is optimal in the asymptotic sense, in expectation.

**Corollary 5.3.** Minimizing the total end-effector travel distance for labeled rearrangement on $d$-dimensional lattices, with a fixed $d \geq 2$, is NP-hard.

5.2. Partially-labeled 2D and higher dimensions

Our main focus on PTR is finding near-optimal solutions. We present such an algorithm for minimizing the number of pick-n-swaps and show that it is distance-optimal in the asymptotic sense, in expectation, for two common goal configuration patterns shown in Figure 6. In goal configuration pattern $A$, items of the same type are aggregated in both dimensions. In pattern $B$, each type occupies a single column of the lattice. For notational convenience, it is assumed that the number of types $k = m_1 = m_2$ is a perfect square, for example, $k = 4 = 2^2$. Our analysis does not depend on this last assumption.

For PTR, minimizing the number of pick-n-swaps can be realized in polynomial time, as it is for POR: for each type, matchings can be made to yield cycles with minimum end-effector travel. These cycles can be merged using a procedure similar to MERGECYCLES and MERGECYCLESMSST, which minimizes the number of pick-n-swaps.

On the other hand, similar to LTR, minimizing the end-effector travel is computationally intractable, regardless of whether the number of pick-n-swaps is minimized. This is true because LTR is a special case of PTR.

**Corollary 5.4.** Computing a rearrangement plan for minimizing the travel distance for PTR is NP-hard.

We note that we can also show that the hardness results continue to hold for goal configuration patterns $A$ and $B$. For pattern $A$, we may apply the same proof used for
proving Theorem 5.1 by treating each $\sqrt{k} \times \sqrt{k}$ cell as a single cell, which reduces the PTR problem to an LTR problem. For pattern $B$, the proof for Theorem 5.1 directly applies.

Next, we describe a general algorithm for PTR, which does not depend on the goal configuration pattern. The algorithm is similar to OPTPLANPOR for POR but with only three phases; the two cycle-merging phases in POR are merged into a single phase. In the first phase, for each type of item that needs to be rearranged, a bipartite graph is constructed that connects all applicable initial configurations to all goal configurations, with the edge weight being the distance between a pair of start and goal configurations. Matching is then performed on this bipartite graph to get a 1-1 mapping between initial and goal configurations. This can be done efficiently using the Hungarian algorithm Kuhn (1955). After the first phase, denoted as FORMCYCLESPTR, a set of initial cycles, $C$, is formed.

In the second phase, cycles in $C$ are merged through goal swaps. Two cycles can be merged through goal swaps (similar to what is done in Figure 4 [bottom left] \rightarrow Figure 4 [bottom right]) if they contain items of the same type. Unlike POR, where some merges do not change end-effector travel distance, a merge here will cause the total distance to increase in general. Therefore, only a single cycle-merging phase is done for PTR. The procedure for doing so is the same as MERGECYCLESMST for POR: a graph $G$ is constructed where each cycle is a node and there is an edge between each pair of mergeable cycles with the edge weight being the additional distance that is incurred for the merge, due to goal swaps. A minimum spanning forest is then constructed for merging all mergeable cycles. After the second phase, which we call MERGECYCLESPTR, no more than $k/2$ cycles are left and the number of pick-n-swaps is minimized. Let this set of cycles be $C'$.

In the third and last phase, cycles in $C'$ are again connected to form a complete rearrangement plan. This is realized through another round of minimum spanning tree computation, for which another graph $G'$ is needed for capturing the distance between cycles. Here, for two cycles $c_1$ and $c_2$, the distance between them is simply $\min_{v_1 \in c_1, v_2 \in c_2} \text{dist}(v_1, v_2)$. Similar to the LTR case, here, the distance between two cycles is directed. Therefore, $G'$ is also directed. We also include $p_0$, the end-effector’s initial location, as a vertex in $G'$ and compute its distance to cycles in $C'$ in the same way. After the minimum spanning tree $T$ is computed for $G'$, a double covering of this tree starting at $p_0$, going through the cycles and back, yields a complete rearrangement plan with the minimum number of pick-n-swaps. Denoting the last phase as SWEEPCYCLESPTR and the overall algorithm PLANPTR (we omit the algorithm outline since it is fairly similar to OPTPLANPOR), we proceed to analyze the distance optimality.

**Theorem 5.2.** PlanPTR computes distance-optimal solutions in the asymptotic sense for PTR with goal configuration patterns $A$ and $B$, in expectation.

**Proof.** We first examine pattern $A$. Intuitively, the required amount of distance for moving items to a proper goal dominates other distances. To establish this, we estimate the different costs. For a single item of a given type, the initial location can be anywhere in the lattice. Therefore, the expected distance for restoring it, regardless of where the goal is, is $\Omega(k)$. The total cost, in expectation, is then $E = \Omega(k^3)$. It is clear that the cycles computed by FORMCYCLESPTR, in expectation, have a total length no more than $E$. Because the MERGECYCLESPTR subroutine only swaps goals within a $\sqrt{k} \times \sqrt{k}$ square region, each swap will add at most $O(\sqrt{k})$ additional distance (we omit the straightforward computation based on the triangle inequality). Therefore, MERGECYCLESPTR will add at most $O(k^{3/2})$ distance. In the last phase, because at most $k/2$ cycles are connected, SWEEPCYCLESPTR inverts an additional connection distance cost of $O(\sqrt{k})$. Because MERGECYCLESPTR and SWEEPCYCLESPTR only add costs that are asymptotically inconsequential as compared to $E$, PLANPTR is distance-optimal in the asymptotic sense for PTR in expectation with goal configuration pattern $A$.

For pattern $B$, it is clear that the expected cost remains at $E = \Omega(k^3)$. For MERGECYCLESPTR, although goal swaps may happen over a distance of up to $k$, we note that no two different swaps will ever cross each other in the vertical direction. We readily see that (again, via an application of the triangle inequality) the cumulative distance increase per type is bounded by $2k$. The total additional cost over all types due to MERGECYCLESPTR is then bounded by $O(k^2)$. For pattern $B$, SWEEPCYCLESPTR essentially goes from the leftmost column to the rightmost and back, which incurs $O(k)$ distance. Therefore, PLANPTR is distance-optimal in the asymptotic sense, in expectation, for PTR with goal configuration pattern $B$.

**Remark 8.** Similar to the labeled setting, the structural results obtained for POR and PTR readily extend to higher
dimensions. For patterns A and B, it is clear that PlanPTR optimizes equation (1) globally.

6. Simulation studies

In this section, based on simulation studies, we highlight some properties of the rearrangement problems and corroborate the guarantees provided by our algorithms. We implemented all algorithms described in the paper in Python. Each data point presented in a figure is an average of over 100 randomly-generated instances according to some distribution to be stated. We mention that our basic Python implementation is fairly efficient; each instance, with some containing 10,000 items, is solved within 1 s. For practical-sized problems with a few hundred items, each takes less than $10^{-3}$ second to solve. We do not present the computation time here as it will not be representative of an optimized implementation in C/C++. The source code, with implementations of both greedy and optimized/optimal algorithms for LOR/POR/LTR/PTR problems, is available at https://github.com/arc-l/lattice-rearrangement/.

For LOR and LTR, since cycle-following is a natural strategy that minimizes the number of pick-n-swaps, only end-effector travel is examined here. In Figure 7 [left], LOR instances are generated following the uniform random distribution. We then take the end-effector travel distance computed by OPTPLANLOR and divide it by $m^2$. The figure shows that the ratio converges to 1/3 (the gray dotted horizontal line) as expected. We further note that the ratio’s range, between 0.33 and 0.345, is very small. A contributing reason that the total travel cost is close to $m^2/3$, even when $m$ is small, is that there are not many cycles so the distance due to traveling between cycles is very minimal.

In Figure 7 [right], the ratio of the travel distance between SWEEPCYCLESLOR (non-optimal) and OPTPLANLOR (optimal) is evaluated over three item distribution patterns: uniform random, 10-random, and $\sqrt{m}$-random, where $x$-random means that every block of $x$ items, counting from the left, are uniformly randomly distributed in the generated LOR instances. OPTPLANLOR does significantly better than the greedy (best-first) SWEEPCYCLESLOR, especially when the number of items $m$ is small, which actually corresponds to more practical settings. For the 10-random and $\sqrt{m}$-random settings, there are more cycles due to the partitioning, therefore allowing more opportunities for cycle switching to engage in OPTPLANLOR, providing more distance savings as a result.

For LTR, since cycle-following is again natural, we focus on end-effector travel (all plans have optimal numbers of pick-n-swaps). On an $m \times m$ square lattice where $m$ is also a perfect square, we evaluate the performance of cycle-following algorithms over three distributions: uniform random, column random, where items are uniformly randomly distributed within columns, and block random, where items are randomized within $\sqrt{m} \times \sqrt{m}$ blocks. Figure 8 [left] presents the ratio of the distance from all cycles divided by $m^2$, which is the same for SWEEPCYCLESLTR and SWITCHCYCLESLTR. That is, traveling between cycles is not included. We observe that both uniform random and block random settings have travel distances that converge to about 0.52 $m^2$ as predicted, whereas the column random setting, essentially a one-dimensional problem, shows convergence to $m^2/3$, also as expected.

In Figure 8 [right], for each randomness setting, the solid (resp., dashed) line shows the ratio between the distance cost from SWITCHCYCLESLTR (resp., SWEEPCYCLESLTR) and the distance from cycles only. SWITCHCYCLESLTR clearly outperforms the greedy SWEEPCYCLESLTR algorithm in all cases. For both uniform random and column random, SWITCHCYCLESLTR incurs little extra distance beyond the necessary distance needed for following cycles. For the uniform random setting, different cycles are all entangled in the $m \times m$ square, allowing plenty of opportunities for cycle switching. Similarly, for the column random setting, edges of a cycle in a column are likely to pass closely by a vertex of another cycle in a nearby column, presenting opportunities for cycle switching. Cycles in different blocks in the block random setting tend to be more apart; it less likely for the edge of a cycle in an $\sqrt{m} \times \sqrt{m}$ block to pass closely by the vertex of another cycle in another block.

For POR and PTR, we look at both the end-effector travel distance and the number of pick-n-swaps. The ratios of
distance and the number of pick-n-swaps between the greedy algorithm and OPTPLANPOR are given in Figure 9 for different numbers of item types. The optimal OPTPLANPOR algorithm is 1–2% better than the greedy algorithm on distance, and up to over 5% better on the number of pick-n-swaps (which carries more importance).

Figure 9 shows that OPTPLANPOR is consistently more effective in reducing both the number of pick-n-swaps and the end-effector travel distance as compared with a greedy algorithm. This is not surprising. For POR, due to many items having the same types, greedy strategies tend to work more locally, leading to sub-optimality on both the number of pick-n-swaps and the end-effector travel.

For PTR, while we no longer have algorithms for computing the optimal distance (recall that the problem is NP-hard), our minimum spanning tree-based algorithm still demonstrates a much better performance when compared to greedy best-first approaches, as shown in Figure 10, for both patterns (pattern A and pattern B). The cost differences, in this case, can be similarly explained as for the POR case, where greedy approaches tend to work more locally.

7. Conclusions and discussions
In this paper, we have performed a systematic study of lattice-based robotic rearrangement using the pick-n-swap
primitive. For both the fully-labeled and the partially-labeled settings under all lattice dimensions, we either provide efficient algorithms for optimally solving the problem (i.e., LOR, POR), or provide algorithms that are optimal in the asymptotic sense when the problem is NP-hard (LTR, PTR). We have demonstrated, via simulation, that our algorithms perform fairly well with respect to absolute optimality measures and in comparison with the already decent greedy best-first approaches. For the majority of the settings explored, our algorithms also provide global optimality guarantees.

In addition to providing characterization and solutions for the specific problems, our analysis points to a general solution structure for such rearrangement problems: forming cycles naturally and then optimally connecting them, for example, using a minimum spanning tree. Combined with proper analysis, guarantees can often be obtained. We believe this general cycle-following + connecting structural insight applies to enhancing the efficiency in solving rearrangement problems beyond lattice-based settings.

We conclude the paper with some open-ended discussions.

Non-random item distribution: The current study assumes that items are randomly distributed in the lattice. If item distribution is not random, in all cases, the number of pick-n-swaps can still be readily minimized. For overall optimality, for LOR, the guarantee by sweepcycleslor no longer holds, but optplandlor continues to compute globally optimal solutions. For POR, optplaplor also continues to ensure solution optimality guarantee as before. For LTR and PTR, the distance optimality guarantee (in the asymptotic sense) no longer holds. We project, however, that the associated algorithms should continue to compute high-quality solutions.

Domain topology: The lattices examined in this work are embedded in Euclidean spaces. This assumption may be relaxed. For example, an application may call for rearranging items that form a circle. The algorithms developed in this study can be adapted to work for such scenarios with relatively minor changes. The main update surrounds the distance computation for two lattice points, which changes as the domain’s topology changes. Depending on whether the end-effector travels along the circle or in straight lines between two consecutive pick-n-swaps, the optimal rearrangement plan will change. If the end-effector travels along the circle, then the optimality guarantee for LOR and POR continues to hold (the algorithms will require some minor modifications). If the end-effector travels along straight lines between two points on a circle, the situation is closer to the 2D setting with similar optimality guarantees.

Bi-criteria optimization: In our treatment of lattice-based rearrangement problems, there exists a fairly good level of flexibility that allows balancing between the two (sometimes competing) objectives in equation (1). For example, in POR, a minimum total-distance feasible solution is first computed, allowing a subsequent trade-off between reducing the number of pick-n-swaps and adding additional end-effector travel. If we enforce that the rearrangement task must be completed, then OPTPLANPOR (Alg. 5) computes the full relevant Pareto frontier. On the other hand, our algorithms do not produce the entire Pareto optimal frontier for the two objectives if partial solutions are also considered.

Bounded optimality: While not a focus of this work, if it is desirable, the algorithms in this work can be shown to provide bounded optimality guarantees, in addition to ensuring optimality in the asymptotic sense. This can be achieved by comparing the extra travel distance with the minimum required distance for realizing the rearrangement task.

Alternative pick-n-place primitives: We have examined a few other natural pick-n-place primitives. It would appear that the pick-n-swap model provides a very nice balance between the complexity of the system (e.g., end-effector, workspace) design and achievable efficiency. For example, if the end-effector cannot swap items, for example, it moves a picked item to a temporary location if it cannot be directly placed, it will make the system twice as inefficient; it will double the number of pick-n-place operations as picks and places are always executed with end-effector travel in between. The travel distance also doubles as a result. It could be interesting to look at a dual-arm extension of this work. Because the two arms can be at two places, additional efficiency gain should be possible. However, because collision avoidance must be considered in a dual-arm setting, the coordination of arms becomes non-trivial and can interfere with the rearrangement planning process. Some related studies on the dual-arm setting but not for lattice rearrangement can be found in Shome et al. (2021). It is also interesting to examine when a single end-effector can hold $k$ items, which should also allow additional efficiency gain. In our preliminary experiments, we noticed additional travel distance savings, as a function of $k$, diminish quickly as $k$ increases. Both dual-arm and hold-$k$ settings will likely incur more computation costs.

Acknowledgments

We sincerely thank the anonymous reviewers for bringing up many insightful suggestions and intriguing questions, which have helped improve the quality and depth of the study.

Declaration of conflicting interests

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Funding

The author(s) disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: This work was supported by the This work is supported in part by NSF awards IIS-1734419, IIS-1845888, and CCF-1934924.

ORCID iD

Jingjin Yu https://orcid.org/0000-0002-7594-7798
Supplemental Material

Supplemental material for this article is available online.

Note

1. \( H_m = \sum_{i=1}^{m} \frac{1}{i} \approx \log m \), where log refers to natural logarithm.

References

Bejiani W, Papallas R, Leonetti M, et al. (2018) Planning with a receding horizon for manipulation in clutter using a learned value function. In: 2018 IEEE-RAS 18th international conference on humanoid robots (humanoids), Beijing, China, Nov 2018. IEEE, pp. 1–9.

Ben-Shahar O and Rivlin E (1998) Practical pushing planning for rearrangement tasks. IEEE Transactions on Robotics and Automation 14(4): 549–565.

Boullarias A, Bagnell J and Stentz A (2015) Learning to manipulate unknown objects in clutter by reinforcement. In: Proceedings of the AAAI Conference on Artificial Intelligence. Austin, Texas, USA, Jan 2015, Vol. 29.

Canny J and Reif J (1987) New lower bound techniques for robot motion planning problems. In: 28th annual symposium on foundations of computer science (sfcs 1987), Los Angeles, CA, USA, 12–14 October 1987. IEEE, pp. 49–60.

Chang L, Smith JR and Fox D (2012) Interactive singulation of objects from a pile. In: 2012 IEEE international conference on robotics and automation, Saint Paul, MN, USA, Jun 2012: IEEE, pp. 3875–3882.

Chavan-Daflé N and Rodriguez A (2015) Prehensile pushing: In-hand manipulation with push-primitives. In: 2015 IEEE/RSJ international conference on intelligent robots and systems (IROS), Hamburg, Germany, Oct 2015: IEEE, pp. 6215–6222.

Chu YJ (1965) On the shortest arborescence of a directed graph. Scientia Sinica 14: 1396–1400.

Dogan M and Srinivasa S (2011) A framework for push-grasping in clutter. Robotics: Science and Systems VII 1. Los Angeles, CA, USA.

Edmonds J (1967) Optimum branchings. Journal of Research of the National Bureau of Standards B 71(4): 233–240.

Eitel A, Hauff N and Burgard W (2020) Learning to singulate objects using a push proposal network. In: Robotics: Research. Springer, pp. 405–419.

Flajolet P and Sedgewick R (2009) Analytic Combinatorics. Cambridge University Press.

Goldberg KY (1993) Orienting polygonal parts without sensors. Algorithmica 10(2): 201–225.

Han SD, Stiffler NM, Krontiris A, et al. (2018) Complexity results and fast methods for optimal tabletop rearrangement with overhand grasps. The International Journal of Robotics Research 37(13–14): 1775–1795.

Haustein JA, King J, Srinivasa SS, et al. (2015) Kinodynamic randomized rearrangement planning via dynamic transitions between statically stable states. In: 2015 IEEE international conference on robotics and automation (ICRA), Seattle, WA, USA, Jul 2015. IEEE, pp. 3075–3082.

Havur G, Ozbilgin G, Erdem E, et al. (2014) Geometric rearrangement of multiple movable objects on cluttered surfaces: a hybrid reasoning approach. In: 2014 IEEE international conference on robotics and automation (ICRA), Hong Kong, China, 31 May–07 June 2014. IEEE, pp. 445–452.

Hopcroft JE, Schwartz JT and Sharir M (1984) On the complexity of motion planning for multiple independent objects; PSPACE-hardness of the “warehouseman’s problem”. The International Journal of Robotics Research 3(4): 76–88.

Huang B, Han SD, Boullarias A, et al. (2021) Dipn: deep interaction prediction network with application to clutter removal. In: Proceedings IEEE international conference on robotics and automation. Note: to appear.

Huang E, Jia Z and Mason MT (2019) Large-scale multi-object rearrangement. In: 2019 international conference on robotics and automation (ICRA), Montreal, QC, Canada, 20–24 May 2019. IEEE, pp. 211–218.

Johnson DB (1975) Priority queues with update and finding minimum spanning trees. Information Processing Letters 4(3): 53–57.

Kaelbling LP and Lozano-Pérez T (2011) Hierarchical task and motion planning in the now. In: 2011 IEEE international conference on robotics and automation, Shanghai, China, Aug 2011. IEEE, pp. 1470–1477.

Karp RM (1972) Reducibility among combinatorial problems. In: Complexity of Computer Computations. Springer US: Springer, pp. 85–103.

King JE, Cognetti M and Srinivasa SS (2016) Rearrangement planning using object-centric and robot-centric action spaces. In: 2016 IEEE international conference on robotics and automation (ICRA), Stockholm, Sweden, Jun 2016. IEEE, pp. 3940–3947.

King JE, Haustein JA, Srinivasa SS, et al. (2015) Nonprehensile whole arm rearrangement planning on physics manifolds. In: 2015 IEEE international conference on robotics and automation (ICRA), Seattle, WA, USA, Jul 2015. IEEE, pp. 2508–2515.

Krontiris A and Bekris KE (2015) Dealing with difficult instances of object rearrangement. In: Robotics: Science and Systems. Rome, Italy: Jul 2015. Vol. 1123.

Krontiris A and Bekris KE (2016) Efficiently solving general rearrangement tasks: a fast extension primitive for an incremental sampling-based planner. In: 2016 IEEE international conference on robotics and automation (ICRA), Stockholm, Sweden, Jun 2016. IEEE, pp. 3924–3931.

Kuhn HW (1955) The Hungarian method for the assignment problem. Naval Research Logistics Quarterly 2(1–2): 83–97.

Laskey M, Lee J, Chuck C, et al. (2016) Robot grasping in clutter: using a hierarchy of supervisors for learning from demonstrations. In: 2016 IEEE international conference on automation science and engineering (CASE), Fort Worth, TX, USA, Nov 2016. IEEE, pp. 827–834.

Lee J, Cho Y, Nam C, et al. (2019) Efficient obstacle rearrangement for object manipulation tasks in cluttered environments. In: 2019 international conference on robotics and automation
Levine S, Finn C, Darrell T, et al. (2016) End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research* 17(1): 1334–1373.

Lynch KM and Mason MT (1999) Dynamic nonprehensile manipulation: controllability, planning, and experiments. *The International Journal of Robotics Research* 18(1): 64–92.

Mahler J, Liang J, Niyaz S, et al. (2017) Dex-net 2.0: deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics. arXiv preprint arXiv:1703.09312.

Mason MT (2018) Toward robotic manipulation. *Annual Review of Control, Robotics, and Autonomous Systems* 1: 1–28.

Mitchell JS and Sharir M (2004) New results on shortest paths in three dimensions. In: Proceedings of the twentieth annual symposium on computational geometry, Brooklyn, New York, USA, Jun 2004, pp. 124–133, Association for Computing Machinery.

Moll M, Kavraki L, Rosell J et al. (2017) Randomized physics-based motion planning for grasping in cluttered and uncertain environments. *IEEE Robotics and Automation Letters* 3(2): 712–719.

Nam C, Lee J, Cho Y, et al. (2019) Planning for target retrieval using a robotic manipulator in cluttered and occluded environments. arXiv preprint arXiv:1907.03956.

Pan Z and Hauser K (2020) Decision making in joint push-grasp action space for large-scale object sorting. arXiv preprint arXiv:2010.10064.

Papadimitriou CH (1977) The euclidean travelling salesman problem is np-complete. *Theoretical Computer Science* 4(3): 237–244.

Prim RC (1957) Shortest connection networks and some generalizations. *The Bell System Technical Journal* 36(6): 1389–1401.

Santaló LA (2004) *Integral Geometry and Geometric Probability*. Cambridge University Press.

Shome R, Solovey K, Yu J, et al. (2021) Fast, high-quality two-arm rearrangement in synchronous, monotone tabletop setups. *IEEE Transactions on Automation Science and Engineering* 18(3): 888–901.

Solovey K and Halperin D (2014) k-color multi-robot motion planning. *The International Journal of Robotics Research* 33(1): 82–97.

Song H, Haustein JA, Yuan W, et al. (2019) Multi-object rearrangement with monte carlo tree search: a case study on planar nonprehensile sorting. arXiv preprint arXiv:1912.07024.

Stilman M and Kuffner JJ (2005) Navigation among movable obstacles: Real-time reasoning in complex environments. *International Journal of Humanoid Robotics* 2(04): 479–503.

Stilman M, Ju Schamburek, Kuffner J, et al. (2007) Manipulation planning among movable obstacles. In: Proceedings 2007 IEEE international conference on robotics and automation, Rome, Italy, Apr 2007. IEEE, pp. 3327–3332.

Treleaven K, Pavone M and Frazzoli E (2013) Asymptotically optimal algorithms for one-to-one pickup and delivery problems with applications to transportation systems. *IEEE Transactions on Automatic Control* 58(9): 2261–2276.

van Den Berg J, Snoeyink J, Lin MC, et al. (2009) Centralized path planning for multiple robots: optimal decoupling into sequential plans. In: *Robotics: Science and Systems*. Seattle, USA: MIT Press, Cambridge, Massachusetts, Vol. 2, pp. 2–3.

Wilfong G (1991) Motion planning in the presence of movable obstacles. *Annals of Mathematics and Artificial Intelligence* 3(1): 131–150.

Yu J (2021) Rearrangement on lattices with pick-n-swaps: optimality structures and efficient algorithms. In: *Robotics: Science and Systems*. Virtual: July 2021.

Zeng A, Song S, Welker S, et al. (2018) Learning synergies between pushing and grasping with self-supervised deep reinforcement learning. In: 2018 IEEE/RSJ international conference on intelligent robots and systems (IROS), Madrid, Spain, Jan 2018. IEEE, pp. 4238–4245.

Zeng A, Song S, Yu KT, et al. (2022) Robotic pick-and-place of novel objects in clutter with multi-affordance grasping and cross-domain image matching. *The International Journal of Robotics Research* 41(7): 690–705.