Rearrangement on Lattices with Pick-n-Swaps: Optimality Structures and Efficient Algorithms

Jingjin Yu

***Abstract*—We propose and 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 becomes computationally intractable to optimally solve. Despite their NP-hardness, we prove that efficient cycle-following based algorithms remain asymptotically optimal 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 $1.x$-optimality when the number of items is small. Simulation studies corroborate the effectiveness of our algorithms.**

code: github.com/rutgers-arc-lab/lattice-rearrangement

I. Introduction

Effective object manipulation [1], a task and motion planning challenge that remains difficult for machines to master, is essential in fulfilling the true potential of autonomous robots. In the past few decades, in tackling the challenge, while some research has emphasized integrated solutions with promising results [2]–[6], significant efforts have been devoted to examining key components including perception [7]–[10], rearrangement planning [11]–[22], and manipulation [23]–[30], 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 $d = 1, 2, \ldots$, under a novel *pick-n-swap* model (see Fig. 1, bottom row, for example start and goal configurations). 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 find many practical applications, including the rearrangement of products at stores and show rooms (see Fig. 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 minimize the number of pick-n-swaps and end-effector travel, which is usually easier to execute than pick-n-swaps and less time-consuming.

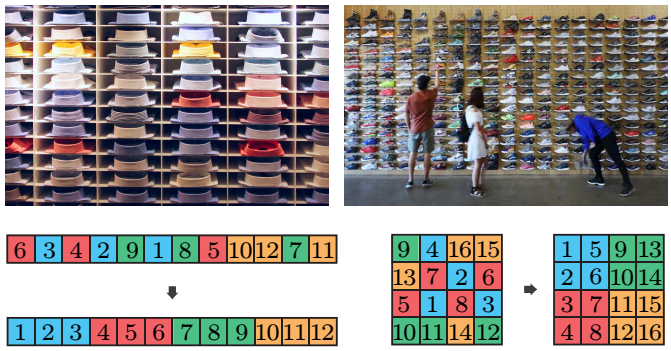

Fig. 1. [top row] Real world examples of items stored in lattice/grids that need rearrangement from time to time. [top left] Shirts. [top right] Shoe display. [bottom row] Examples of the rearrangement problem formulations studied in this work. [bottom left] A set of 12 items in a row that must be rearranged either according to the labels or according to the types (colors). [bottom right] A set of 16 items in a two-dimensional lattice that must be rearranged either according to the labels or according to the types (colors).

As a summary of our findings and contributions, we observe that, 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 particular, there is an interesting dichotomy, in terms of computational complexity, between one-dimensional lattices and lattices in higher dimensions. For 1D lattices, both the fully-labeled setting and the partially-labeled setting can be optimally solved in low-polynomial time through a carefully designed *cycle-following* procedure based on the fact that optimal pick-n-swap sequences naturally induce one or more cycles [31], [32]. Here, the partially-labeled setting possesses a more complex structure as compared to the fully-labeled setting. Once we move from 1D lattices to

J. Yu are with the Department of Computer Science, Rutgers, the State University of New Jersey, Piscataway, NJ, USA. E-Mail: jingjin.yu@cs.rutgers.edu.

2D lattices, minimizing the end-effector travel becomes computationally intractable for both fully- and partially-labeled settings. Nevertheless, through careful analysis, we are able to show that our algorithmic solutions still achieve asymptotic optimality in expectation. That is, as the number of items goes to infinity, the solution converges to the optimal solution, in expectation. When the number of items is small, the solution is near-optimal, i.e., $1.x$-optimal.

From an application perspective, focusing on the scheduling of pick-n-swaps and end-effector travel, our study reveals intrinsic combinatorial structures, as highlighted above, that apply generally to robotic rearrangement problems, which go beyond settings that are lattice-based.

Related work. Multi-object rearrangement is computationally challenging. As a variation of multi-robot motion planning problems, rearrangement inherits the PSPACE-hard complexity [33]. When geometric constraints must be considered, the relatively simple Navigation Among Movable Obstacle (NAMO) problem is shown to be NP-hard [34]. If consideration of plan quality is further required, as is often the case in practice, optimally resolving dependency [35] or planning an optimal object pick-n-place sequence are both NP-hard [19]. 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 many research working with non-prehensile (e.g., pushing) manipulations, sometimes assisted with prehensile (grasping) actions. A complete sensing-planning-control framework is proposed in Chang et al. [36] for the singulation of objects in clutter, which uses both perturbation pushes and grasping actions. In [37], hierarchical supervised learning from demonstration is applied to the singulation task. Based on over-segmented RGB-D images, in [38], a push proposal network is constructed for push-only singulation. Results that bridge singulation and clutter removal include [39], [40], 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 [15] for rearranging objects on a tabletop, allowing the effective exploration of configurations. King et al. [41] 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 [42] for handling grasping in clutter. For a similar task, Bejjani et al. [43] uses a receding horizon planner with a learned value function that interleaves planning and plan execution.

Huang et al. [20] has developed an Iterated Local Search (ILS) method for accomplishing multiple tabletop rearrangement tasks including singulating, separation, formation, and sorting of many identically shaped cubes. In [44], Monte Carlo Tree Search is combined with deep-learning for separating many objects into coherent clusters within a bounded workspace. In contrast to [20], non-convex objects are supported. More recently, Pan and Hauser proposed a bi-level planner [22] that employs both pushing and overhand 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 [12], [45], 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 [16], [46]. Han et al. [19] shows that tabletop rearrangement embeds a Feedback Vertex Set (FVS) problem [47] and the Traveling Salesperson Problem (TSP) [48], 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 [16], [19] put more emphasis on the combinatorial aspects of object rearrangement. To that end, classical Vehicle Routing Problems (VRP) [49] become relevant. In [21], a polynomial-time, complete planner for reasoning rearrangement for object retrieval in a constrained, shelf-like setting is proposed. In a subsequent study [50], the number of objects to be relocated for retrieval is minimized while considering sensor occlusion.

Organization. The rest of the paper is organized as follows. In Sec. II, we formally define the rearrangement problems studied in this paper. In Sec. III and Sec. IV, we provide structural analysis and describe algorithms for the fully- and partially-labeled settings in 1D, respectively. 2D and higher dimensions are examined in Sec. V. We characterize the behavior of our algorithms through simulation studies in Sec. VI, and discuss and conclude in Sec. VII.

II. Rearrangement on Lattices

We examine a multi-object rearrangement problem where items are stored in a d-dimensional lattice or grid, $d = 1, 2, \ldots$, with dimension i having a capacity or length of m_i. Items are assumed to be stored at full capacity, i.e., an $m_1 \times \ldots \times m_d$ lattice stores $\Pi_{i=1}^{d} m_i$ items (see, e.g., Fig. 1). The items, to be rearranged, 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 (coordinate) on the $m_1 \times \ldots \times m_d$ lattice. In a partially-labeled setting, there are $k > 1$ types of items where items within a given type are considered interchangeable; items of the same type are to be grouped, via rearrangement, possibly 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 may execute one of the following:

- pick up an item and hold it on 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, or

- 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 is briefly discussed in Sec. VII.

The pick-n-swap model leads to a natural partition of the robot operation 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

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

where $dist(p_i, p_{i+1})$ 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 the end-effector's motion mechanism. For example, if a human is to arrange shoes for the setup shown in Fig. 1 [top right], then horizontal travel is the main source of travel distance cost. This study works with Euclidean distances, i.e., $dist(p_i, p_{i+1}) = \|p_i - p_{i+1}\|_2$.

Because a pick-n-swap involves precise robot arm placement and challenging grasp planning/execution, similar to [19], it is assumed that the total pick-n-swap cost dominates the total end-effector travel cost. That is, on the right side of (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. We mention that our analysis provides individual treatments for minimizing the number of pick-n-swaps and the travel cost, allowing practitioners to adapt the algorithms based on 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 $m_1 = m$ 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 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., Fig. 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, beside considering arbitrary goal configurations, two natural goal configuration patterns are analyzed in more detail, with one having the goals of the same type aggregated (see, e.g., Fig. 1 [bottom right]) and the other having each type occupying a single column of the lattice.

For convenience, we denote the one dimensional labeled and partially-labeled problems, as stated above, as the **l**abeled **o**ne-dimensional **r**earrangement (LOR) problem and the **p**artially-labeled **o**ne-dimensional **r**earrangement (POR) problem, respectively. The **t**wo-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 (1). Based on the findings, we design efficient algorithms for computing (near)-optimal plans.

III. Fully-Labeled Rearrangement in 1D

For combinatorial optimization problems involving the reconfiguration of many bodies, the labeled settings are often more challenging (e.g., [19], [51]). In our case, however, the labeled case is structurally simpler, due to the absence of dependencies among the items to be relocated. This allows the computation of (exact) optimal solutions for LOR.

Recall that LOR requires rearranging a row of m items. Therefore, LOR can be viewed as going from one random permutation of m labeled items to the canonical order $[m] := 1, \ldots, m$. An LOR instance is therefore fully specified by a permutation π of $[m]$, where π_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 *cycle-following* algorithm, SweepCyclesLOR, that solves LOR near-optimally, i.e., minimizing 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, OptPlanLOR, that computes a rearrangement plan that also minimizes the end-effector travel.

A. Cycle Following with Left to Right Sweeping

Consider an LOR instance with 9 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* 341 (see Fig. 2). 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. All together, there are two cycles, (341) and (7985), containing 7 items in total (here, we deviate slightly with how cycles are normally counted in permutations, where a single

item in the correct cell would be counted as a cycle as well, which we do not by default).

Fig. 2. [left] The initial configuration of an LOR instance with two cycles marked in different colors. A plan is shown that rearranges items following the cycle (341). [right] The resulting intermediate configuration.

These cycles are uniquely determined by the initial configuration π. 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 LOR instance is solved and the end-effector returns to its rest position (cell 1). The straightforward algorithm SWEEPCYCLESLOR is outlined in Alg. 1, in which the routine SWAP(ℓ, i, j) will pick up item j at cell ℓ (if it is not ε, denoting a null item) and swap it with item i being held (if it is not ε). It is clear that SWEEPCYCLESLOR runs in linear or $O(m)$ time.

Algorithm 1: SWEEPCYCLESLOR (π)

```
▷ are there more cycles?
while there are more items to be rearranged do
    ▷ follow & resolve the leftmost cycle
    i ← leftmost i where π_i ≠ i
    SWAP (i, ε, π_i); g ← π_i
    while g ≠ i do
        SWAP (g, g, π_g); g ← π_g
    SWAP (i, i, ε)
```

The optimality properties of SWEEPCYCLESLOR are established in Proposition III.1 and Proposition III.2.

Proposition III.1. SWEEPCYCLESLOR *minimizes the number of required pick-n-swap operations for* LOR*.*

Proof. To solve each cycle, it is clear an item on the cycle must be first picked without any other items on the cycle already held by the end-effector to swap. This means that for each cycle, one additional pick is unavoidable. Induction over the cycles of π then proves the proposition. □

In what follows, by *asymptotic optimality*, we mean that the solution converges to the optimal solution as the number of items goes to infinity.

Proposition III.2. *After minimizing the number of pick-n-swaps,* SWEEPCYCLESLOR *computes asymptotically-optimal solutions for* LOR*, minimizing end-effector travel in expectation, assuming that π is a random permutation.*

Proof. Given the initial configuration π, rearranging each cycle will cause the end-effector to end where it starts following 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), because 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 (i), the end effector travel 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 the distance incurred by (i). To compute distance incurred by (i), given a random π, for a fixed i, the expected distance between item i and cell i is

$$\mathbb{E}_i = \frac{i-1+\ldots+1+0+1+\ldots+m-i}{m}.$$

Tallying over i from 1 to m, the expected total distance due to resolving all cycles is then $\mathbb{E}_\pi = \mathbb{E}_1 + \ldots + \mathbb{E}_m \approx \frac{m^2}{3}$, which dominates $2m$. Therefore, the total end-effector travel distance produced by SWEEPCYCLESLOR is asymptotically optimal in expectation. □

Let H_m denote the m-th harmonic number. We can further estimate the expected total cost according to Eq. (1).

Proposition III.3. *For* LOR *with random initial configurations, the expected total rearrangement cost is*

$$T_{LOR}(m) = (m + H_m - 2)c_p + \frac{m^2 c_t}{3}. \tag{2}$$

Proof. To compute the expected total cost including pick-n-swaps, we know that the number of cycles (here cycles of size 1 are included) in a random permutation π of $[m]$ is H_m [52], the m-th harmonic number. Given any π, the probability of any item i is already at cell i is $\frac{1}{m}$. The expected number of cycles of length 1 is then 1, making the expected number of cycles of at least 2 in a random permutation $H_m - 1$. The total number of pick-n-swaps is then $m-1$, the number of items that must be rearranged, plus $H_m - 1$, the extra number of pick-and-swap operations for completing cycles. The total (time) cost of rearrangement, in expectation, is then given by (2). □

B. Cycle Sweeping with Cycle Switching

In SWEEPCYCLESLOR, each cycle is followed through one bye 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 Fig. 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

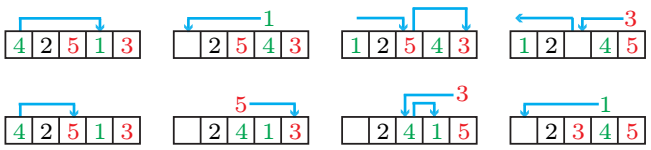

Fig. 3. [top] A rearrangement plan with a total distance of $3+3+4+4 = 14$ as computed by SWEEPCYCLESLOR [bottom] A rearrangement plan with a total distance of $2 + 2 + 3 + 3 = 10$. Both plans require six pick-n-swap operations; the later one does not wait for one cycle to finish.

algorithm that groups cycles for more effective rearrangement. To describe the algorithm, some definitions are in order. Given a permutation π, let C_π be the set of all cycles induced by π. For a $c \in C_\pi$, let $\min(c)$ and $\max(c)$ be the smallest and largest item labels of items in c, respectively. With a slight abuse of notation, the definitions $\min$ and $\max$ extend to a set of cycles, i.e., for $C \subset C_\pi$, $\min(C) = \min_{c \in C} \min(c)$ and $\max(C) = \max_{c \in C} \max(c)$.

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

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 $\min(c_3)$ 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 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 OPTPLANLOR algorithm is given in Alg. 2, which in turn calls Alg. 3 and Alg. 4. It is not difficult to observe that OPTPLANLOR runs in $O(m)$ time. We further prove its distance optimality.

Algorithm 2: OPTPLANLOR (π)

```
  ▷ retrieve cycles as singleton sets
1 C_π ← GETCYCLES (π)
  ▷ group cycles with overlapping ranges
2 while ∃C1, C2 ∈ C_π with overlapping ranges do
3 |   C_π ← (C_π \ {C1, C2}) ∪ {C1 ∪ C2}
  ▷ process cycle groups
4 C ← left most cycle group in C_π
5 PROCESSCYCLEGROUP (ε, C, π, C_π)
```

Algorithm 3: PROCESSCYCLEGROUP (p, C, π, $\mathcal{C}_\pi$)

```
1 c ← leftmost cycle in cycle group C
2 PROCESSCYCLE (p, c, C, π, C_π)
```

Algorithm 4: PROCESSCYCLE (p, c, C, π, $\mathcal{C}_\pi$)

```
1  c' ← leftmost cycle in cycle group C after c
2  C' ← leftmost cycle group to the right of C in C_π
   ▷ process cycle c from left
3  i ← min(c); SWAP (i, p, π_i); g ← π_i
4  while g ≠ i do
   |   ▷ switch cycles within group?
5  |   while c' ≠ null and g > min(c') do
6  |   |   PROCESSCYCLE (g, c', C, π, C_π)
7  |   |   c' ← leftmost cycle in cycle group C after c'
   |   ▷ switch to the next cycle group?
8  |   if g == max(C) and C' ≠ null then
9  |   |   PROCESSCYCLEGROUP (g, C', π, C_π)
   |   ▷ following the current cycle
10 |   SWAP (g, g, π_g); g ← π_g
11 SWAP (i, i, p);
```

Theorem III.1. *For* LOR, OPTPLANLOR *computes a rearrangement plan with minimum 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 π_i^{-1} be its initial location (this is natural, since we then have $\pi_{\pi_i^{-1}} = i$). Notice that the minimum distance the end-effector must travel while carrying item i is $|\pi_i^{-1} - i|$. In PROCESSCYCLE, within a cycle group, an item i is moved exactly a total distance of $|\pi_i^{-1} - i|$, even though it may be done in multiple steps. For example, in the top figure of Fig. 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 $|\pi_i^{-1} - i|$ by OPTPLANLOR if it is carried from one cycle group to another, but such travel is attributed to travel between adjacent cycle groups. □

Remark. 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 opted for the standalone pre-processing stage to make the algorithm hopefully more clear and the running time analysis more straightforward.

IV. 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

Fig. 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 POR can still be optimally solved despite the significant added complexity.

We describe an optimal algorithm for POR, 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 POR instances where each type of items form a contiguous section in the goal configuration (see, e.g., Fig. 4). In such instances, for a given type $1 \leq t \leq k$, let n_i 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 $\ell_t = \sum_{i=1}^{t-1} n_i + 1$ and $r_t = \sum_{i=1}^{t} n_i$, inclusive. Define $range(t) := [\ell_t, r_t]$. An instance of this restricted POR problem is then fully specified by the initial configuration of the items as a sequence of types, i.e., $(t_1, \ldots, t_m)$, where $1 \leq t_i \leq k$.

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, $i \neq t_i$, we select its goal to be the leftmost available one. Fig. 4 [top left] shows an example. The matchings induce a set of cycles (Fig. 4, [top right]), which are distance optimal but do not minimize the number of pick-n-swaps. As is the case with 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.

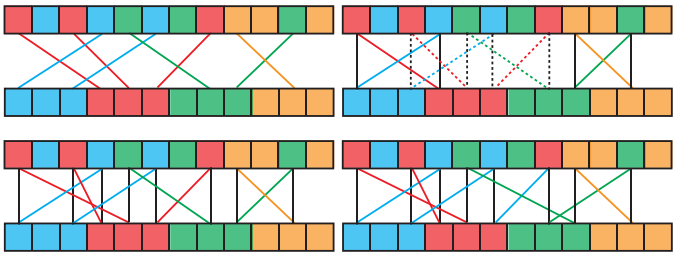

Fig. 4. Illustration of the first two phases of the OPTPLANPOR algorithm. [top left] The colored edges show a simple matching of items with proper goals. [top right] The matchings induce three cycles, two solid ones and one dashed one. [bottom left] Swapping the two red edges on the left merge two cycles without adding end-effector travel. [bottom right] Swapping the two green edges merge two cycles but incur additional end-effector travel costs.

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 Fig. 4 [top right] can be merged by swapping the goals of the left two red edges, yielding Fig. 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, $t_i = t_j$, and i, j are on different side of $range(t_i)$. For example, in Fig. 4 [bottom left], the two cycles both have edges going into the third (green) type, but in different directions. Merging these two cycles, as shown in Fig. 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.

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 these pseudo-code.

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

```
  ▷ phase 1: form cycles, C is a list of cycles
1 C ← FORMCYCLES (t_1, ..., t_m)
  ▷ phase 2: merge cycles, w/o added distance
2 C' ← MERGECYCLES (C)
  ▷ phase 3: merge cycles, w/ added distance
3 C'' ← MERGECYCLESMST (C')
  ▷ phase 4: group, sweep, and switch cycles
4 GROUPSWEEPCYCLESPOR (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 Fig. 4 [bottom right] is 1 (we note that the actual swap will incur a cost doubling this distance). After the G_C is constructed, which can have multiple connected components, a minimum spanning tree algorithm is executed, e.g., Prim's algorithm [53], yielding a spanning forest F of G_C. Each tree T in F will result in a single merged cycle.

We proceed to establishing important properties of OPTPLANPOR and the subroutines. From here on, arbitrary goal configurations are assumed unless stated otherwise.

Algorithm 6: MERGECYCLESMST (C)

```
▷ initialize a cycle merge distance graph
V_C ← C; E_C ← {}; W ← {}; G_C ← (V_C, E_C, W);
▷ compute ''merge distance'' between cycles
for all c_i, c_j ∈ V_C do
    E_C ← E_C ∪ {e_ij = (c_i, c_j)}
    w_ij ← CYCLEDISTANCE (c_i, c_j); W ← W ∪ {w_ij}
▷ compute a minimum spanning forest over G_C
F ← MST (G_C)
▷ merge cycles for each tree T in F
C' ← {}
for each tree T in F do
    while T has an edge e_ij = (c_i, c_j) do
        c_i ← merge c_i and c_j
        collapse edge e_ij in T
    Add the single cycle c in T to C'
return C'
```

Proposition IV.1. *For* POR *with k types of items,* MERGECYCLES *creates cycles that are distance optimal. When goals are aggregated by items types, there are at most $k-1$ cycles after completing the* MERGECYCLES *subroutine.*

Proof. FORMCYCLES creates cycles that are distance optimal by construction, which 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 configurations pair, and a type, this number is fully determined and realized by FORMCYCLES. For example, for sorting the first type (cyan) in Fig. 4, the end-effector must pass cell 3 at least once and must also stop at the cell at least once, because all cyan items should be to the left of cell 4 in the goal configuration and there are two cyan items in the initial configuration to the right of cell 3. The cycles created by FORMCYCLES realizes 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. 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. □

Lemma IV.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 cross (by construction of MERGECYCLES). The third (green) type in Fig. 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^t (resp., E_2^t). To merge c_1 and c_2, it requires for exactly one t, one edge of E_1^t and one edge of E_2^t to swap their ends so that E_1^t and E_2^t 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 same item types is by doing the merge on the type t where E_1^t and E_2^t are closest to each other. Without loss of generality, assume E_1^t is to the left of E_2^t. The distance between E_1^t and E_2^t is then simply the distance between the right most position reached by E_1^t and the leftmost position reached by E_2^t. 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 IV.1. *For* POR, OPTPLANPOR *computes a rearrangement plan with the minimum end-effector travel after minimizing the total number of pick-n-swaps.*

Proof. By Lemma IV.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 GROUPSWEEPCYCLESPOR, 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. □

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 [54]. Merging cycles can be done at the same time as the minimum spanning tree is built, which does not take additional time. GROUPSWEEPCYCLESPOR takes $O(m)$ time. The total running time of OPTPLANPOR is then $O(m + |E_C| + |V_C| \log |V_C|)$. If the goals are aggregated based on types,

there are at most $k-1$ cycles (Proposition IV.1) entering MERGECYCLESMST, resulting in a total running time of $O(m + k \log k)$. In the general case, the running time is $O(m \log m)$.

V. REARRANGEMENT IN 2D AND HIGHER DIMENSIONS

For higher dimensions, the cycle-following structure for LOR and POR carry over. However, minimizing end-effector travel becomes more challenging, as the problem now contains a Traveling Salesperson Problem (TSP), as will be shown. Nevertheless, strategies can be derived that yield asymptotic optimality.

A. Fully-Labeled 2D (LTR) and Higher Dimensions

1) Labeled Rearrangement in 2D (LTR): An LTR instance is specified by its lattice dimension m_1, m_2, and a permutation π of $[m_1 m_2]$. Similar to LOR, minimizing the number of pick-n-swaps for LTR can be achieved via cycle following. This allows proving results for LTR similar to Propositions III.1 and III.2. A natural extension to SWEEPCYCLESLOR can be made to support the 2D setting: for an LTR instance with an initial configuration π, we compute all its cycles as $c_1, \ldots, c_k$. The new algorithm, which we call SWEEPCYCLESLTR, again performs cycle following of the cycles and moving to cycle c_{i+1} after completing cycle c_i.

Then, we note that the cycle switching procedure for LOR (e.g, the process illustrated in Fig. 3) can be generalized to 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 MERGECYCLESMST. There is however a key difference: in MERGECYCLESMST, 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 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 [55], [56]. The end effector rest position should also be considered in computing the directed minimum spanning tree.

We call the overall cycle switching procedure for LTR as SWITCHCYCLESLTR, which clearly runs in polynomial time. We omit the pseudo code for these two algorithms given their similarity to LOR and POR. These algorithms are asymptotically optimal in expectation.

Proposition V.1. *The number of pick-n-swaps is minimized by* SWEEPCYCLESLTR *and* SWITCHCYCLESLTR. *Furthermore, they compute asymptotically distance-optimal solutions for* LTR *in minimizing end-effector travel in expectation, assuming that π is a random permutation.*

Proof. It is clear that SWEEPCYCLESLTR and 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 III.2, a random permutation π will require an average distance $\mathbb{E}_\pi = \Omega((m_1 + m_2)m_1 m_2) = \Omega(m_1^2 m_2)$. 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 LTR instance can be readily computed as $((2 + \sqrt{2} + 5 \ln(\sqrt{2} + 1))/15)m^2 \approx 0.52m^2$. We omit the details but note that this is equivalent to computing the average distance between two points in a unit square and multiply that distance by m^2. SWEEPCYCLESLTR can be implemented by making a constant number of linear passes over the $m_1 \times m_2$ lattice, yielding an $O(m_1 m_2)$ time. SWITCHCYCLESLTR requires more work; a naive implementation requires $O(m_1^3 m_2^3)$ time, mainly for checking switching distances between cycles. In a sense, the asymptotic optimality provided by SWEEPCYCLESLTR and SWITCHCYCLESLTR is the best one can do, because optimizing distance for LTR, unlike for LOR, is NP-hard. We note that the hardness holds regardless of whether the number of pick-n-swaps is minimized.

Theorem V.1. *Minimizing the total end-effector travel distance for* LTR *is NP-hard.*

Proof. We prove the claim via a reduction from the Euclidean TSP [48]. Given an Euclidean TSP instance specified by a set of points embedded in a rectangular region (e.g., Fig. 5, left), we superimpose a lattice over the region at some resolution $m_1 \times m_2$. To construct the LTR instance, we set the initial condition π to be the identity permutation, i.e., $\pi_i = i$ for all $1 \leq i \leq m_1 m_2$. Then, we update π for each of the black points in the TSP instance. For a given black point in the TSP instance, let its coordinates be (x_1, x_2). Without loss of generality, we may assume that the (x_1, x_2) satisfy $1 < x_1 < m_1$ and $1 < x_2 < m_2$. We update π such that $\pi_{m_1 x_2 + x_1} = m_1(x_2 - 1) + x_1$ and $\pi_{m_1(x_2-1)+x_1} = m_1 x_2 + x_1$. That is, each black point in the TSP instance is converted to two adjacent items (red points in Fig. 5, right) that must be exchanged. We refer to each pair of the adjacent items to be exchanged as a *cluster*.

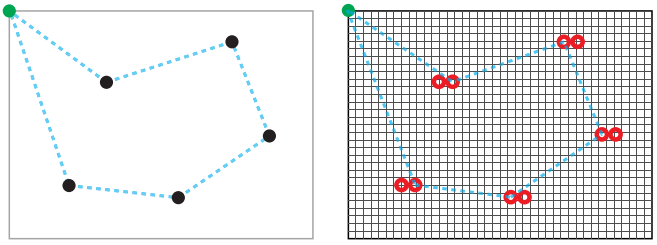

Fig. 5. [left] An Euclidean TSP instance, fully specified by a the set of (black and green) points. [right] A corresponding LTR instance where each black point in the TSP instance is replaced by two items that must be exchanged. The green point corresponds to the end-effector rest position.

By selecting sufficiently large m_1 and m_2, end-effector travel cost for solving each cluster becomes negligible. Therefore, for the 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. □

Remark. 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. Such transition of computational complexity is also observed elsewhere as dimension changes, e.g., shortest paths or visibility become hard as dimension rises from two to three.

2) Labeled Rearrangement in Higher Dimensions: We briefly discuss extensions to dimensions higher than two. It is straightforward to observe that Proposition V.1 and Theorem V.1 continue to hold in higher dimensions through a direct embedding, i.e., a two-dimensional problem can be readily reduced to a d-dimensional problem via adding additional dimensions.

Corollary V.1. *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 asymptotically optimal.*

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

B. Partially-Labeled 2D (PTR) 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 asymptotically distance-optimal in expectation for two common goal configuration patterns shown in Fig. 6. In goal configuration pattern A, items of the same type is 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, e.g., $k = 4 = 2^2$. 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 MERGECYCLESMST, 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. We note that we can also show that the hardness results continue to hold for goal configuration patterns A and B.

Fig. 6. [left] An example of a PTR start configuration. [middle] The corresponding goal configuration pattern A. [right] The corresponding goal configuration pattern B.

Corollary V.3. *Computing a rearrangement plan for minimizing the travel distance for PTR is NP-hard.*

Next, we describe our algorithm for PTR, which applies generally and does not depend on the goal configuration pattern. The algorithm is fairly 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 items that need 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. A 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 [57]. After the first phase, which we call FORMCYCLESPTR, some initial cycles C are 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 Fig. 4 [bottom left] $\rightarrow$ Fig. 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 $\min_{v_1 \in c_1, v_2 \in c_2} dist(v_1, v_2)$. Similar to the LTR case, here, the distance between two cycles are *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 V.2. PLANPTR *computes asymptotically distance-optimal solutions 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 $\mathbb{E} = \Omega(k^3)$. It is clear that the cycles computed by FORMCYCLESPTR, in expectation, has a total length no more than $\mathbb{E}$. 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 $O(k^{5/2})$ distance. In the last phase, because at most $k/2$ cycles are connected, SWEEPCYCLESPTR incurs an additional connection distance cost of $O(k^2)$. Because MERGECYCLESPTR and SWEEPCYCLESPTR only add costs that are asymptotically inconsequential as compared to $\mathbb{E}$, PLANPTR is asymptotically distance-optimal in expectation for PTR with goal configuration pattern A.

For pattern B, it is clear that the expected cost remains at $\mathbb{E} = \Omega(k^3)$. For MERGECYCLESPTR, although goal swaps may happen over a distance of up to k, we note that no two different swaps will never 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 right most and back, which incurs $O(k)$ distance. Therefore, PLANPTR is asymptotically distance-optimal in expectation for PTR with goal configuration pattern B. □

We mention without elaboration that, similar to the labeled setting, the structural results obtained for POR and PTR readily extend to higher dimensions.

VI. 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 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 10000 items, is solved within 1 second. 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/rutgers-arc-lab/lattice-rearrangement/.

For LOR and LTR, since cycle following is a natural strategy which minimizes the number of pick-n-swaps, only end-effector travel is examined here. In Fig. 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. 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.

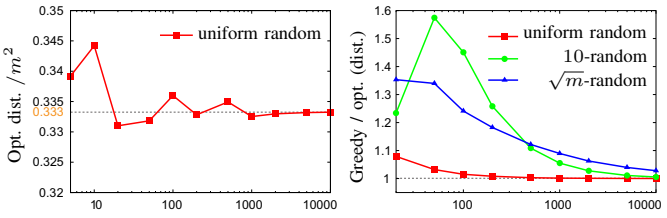

Fig. 7. Two plots illustrating properties of LOR and the associated algorithms. The y-axes are unit-less. The x-axes are the number of items in an instance, which is the case for all figures in this section. [left] The optimal end-effector travel distance for LOR (computed by OPTPLANLOR) divided by m^2 where m is the number of items. [right] End-effector travel distance ratio between SWEEPCYCLESLOR and OPTPLANLOR (optimal) for three different initial arrangement patterns.

In Fig. 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 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. Fig. 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 $0.52m^2$ as predicted, whereas the column random setting, essentially a one-dimensional problem, shows convergence to $m^2/3$, also as expected.

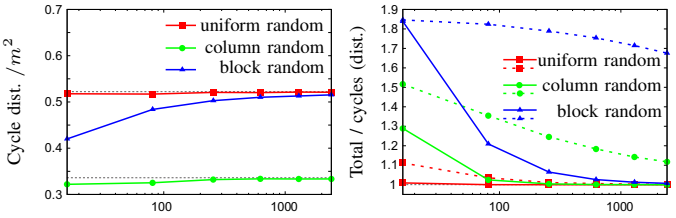

Fig. 8. Properties of LTR and the associated algorithms. [left] Ratio of distance incurred by following cycles versus m^2 for three item distributions. Two gray dotted horizontal line at around 0.52 and $1/3$ are added for reference. [right] Total distance from algorithms versus the total cycle distances for three distributions and two algorithms.

In Fig. 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 block random, SWITCHCYCLESLTR incurs little extra distance beyond the necessary distance needed for following cycles.

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 number of pick-n-swaps between the greedy algorithm and OPTPLANPOR are given in Fig. 9 for different number 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).

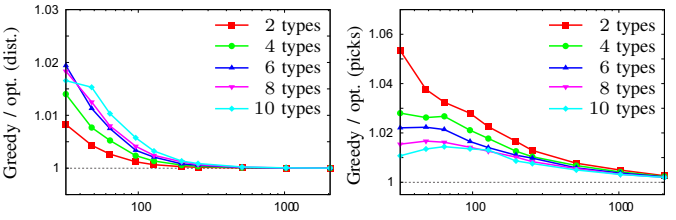

Fig. 9. Performance of the greedy algorithm versus the optimal algorithm (OPTPLANPOR) for POR. [left] End-effector travel distance ratios for different number of types. [right] Ratios between the number of pick-n-swaps for the two algorithms for different number of types.

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 Fig. 10, for both patterns (pattern A and pattern B).

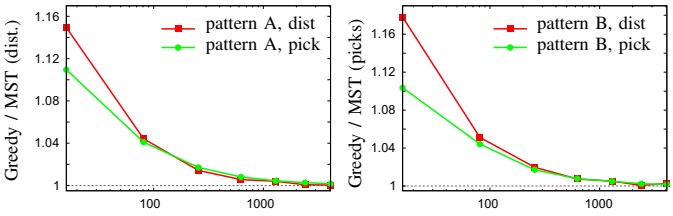

Fig. 10. Performance of the greedy algorithm versus the minimum spanning tree based cycle merging algorithm for PTR. [left] Distance and pick-n-swap ratios for "block" item distribution. [right] Distance and pick-n-swap ratios for "column" item distribution.

VII. CONCLUSION AND DISCUSSION

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 asymptotically optimal 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.

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, e.g., using a minimum spanning tree. Combined with proper analysis, guarantees can often be obtained. We believe this general structural insight applies to enhancing the efficiency in solving rearrangement problems beyond lattices.

We conclude the paper with some open-ended discussion.

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 may change.

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 Eq. (1). For example, in POR, a minimum total-distance feasible solution is first computed, allowing 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 asymptotic optimality. 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 system (e.g., end-effector, workspace) design and achievable efficiency. For example, if the end-effector cannot swap items, e.g., 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. We are also looking at further optimization of a dual-arm solution over this work; because the two arms can be at two places, additional efficiency appears possible.

ACKNOWLEDGMENTS

This work is supported in part by NSF awards IIS-1734419, IIS-1845888, and CCF-1934924. We sincerely thank the anonymous reviewers for bringing up many insightful suggestions and intriguing questions.

REFERENCES

[1] M. T. Mason, "Toward robotic manipulation," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 1, pp. 1–28, 2018.

[2] L. P. Kaelbling and T. Lozano-Pérez, "Hierarchical task and motion planning in the now," in *2011 IEEE International Conference on Robotics and Automation*. IEEE, 2011, pp. 1470–1477.

[3] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 1334–1373, 2016.

[4] J. Mahler, J. Liang, S. Niyaz, M. Laskey, R. Doan, X. Liu, J. A. Ojea, and K. Goldberg, "Dex-net 2.0: Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics," *arXiv preprint arXiv:1703.09312*, 2017.

[5] A. Zeng, S. Song, K.-T. Yu, E. Donlon, F. R. Hogan, M. Bauza, D. Ma, O. Taylor, M. Liu, E. Romo *et al.*, "Robotic pick-and-place of novel objects in clutter with multi-affordance grasping and cross-domain image matching," in *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2018, pp. 3750–3757.

[6] A. M. Wells, N. T. Dantam, A. Shrivastava, and L. E. Kavraki, "Learning feasibility for task and motion planning in tabletop environments," *IEEE robotics and automation letters*, vol. 4, no. 2, pp. 1255–1262, 2019.

[7] A. Saxena, J. Driemeyer, and A. Y. Ng, "Robotic grasping of novel objects using vision," *The International Journal of Robotics Research*, vol. 27, no. 2, pp. 157–173, 2008.

[8] M. Gualtieri, A. Ten Pas, K. Saenko, and R. Platt, "High precision grasp pose detection in dense clutter," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2016, pp. 598–605.

[9] C. Mitash, K. E. Bekris, and A. Boularias, "A self-supervised learning system for object detection using physics simulation and multi-view pose estimation," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 545–551.

[10] Y. Xiang, T. Schmidt, V. Narayanan, and D. Fox, "PoseCNN: A convolutional neural network for 6d object pose estimation in cluttered scenes," in *Robotics: Science and Systems*, 2018.

[11] O. Ben-Shahar and E. Rivlin, "Practical pushing planning for rearrangement tasks," *IEEE Transactions on Robotics and Automation*, vol. 14, no. 4, pp. 549–565, 1998.

[12] M. Stilman and J. J. Kuffner, "Navigation among movable obstacles: Real-time reasoning in complex environments," *International Journal of Humanoid Robotics*, vol. 2, no. 04, pp. 479–503, 2005.

[13] K. Treleaven, M. Pavone, and E. Frazzoli, "Asymptotically optimal algorithms for one-to-one pickup and delivery problems with applications to transportation systems," *IEEE Transactions on Automatic Control*, vol. 58, no. 9, pp. 2261–2276, 2013.

[14] G. Havur, G. Ozbilgin, E. Erdem, and V. Patoglu, "Geometric rearrangement of multiple movable objects on cluttered surfaces: A hybrid reasoning approach," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2014, pp. 445–452.

[15] J. A. Haustein, J. King, S. S. Srinivasa, and T. Asfour, "Kinodynamic randomized rearrangement planning via dynamic transitions between statically stable states," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2015, pp. 3075–3082.

[16] A. Krontiris and K. E. Bekris, "Dealing with difficult instances of object rearrangement." in *Robotics: Science and Systems*, vol. 1123, 2015.

[17] J. E. King, M. Cognetti, and S. S. Srinivasa, "Rearrangement planning using object-centric and robot-centric action spaces," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2016, pp. 3940–3947.

[18] R. Shome, K. Solovey, J. Yu, K. Bekris, and D. Halperin, "Fast, high-quality dual-arm rearrangement in synchronous, monotone tabletop setups," in *International Workshop on the Algorithmic Foundations of Robotics*. Springer, 2018, pp. 778–795.

[19] S. D. Han, N. M. Stiffler, A. Krontiris, K. E. Bekris, and J. Yu, "Complexity results and fast methods for optimal tabletop rearrangement with overhand grasps," *The International Journal of Robotics Research*, vol. 37, no. 13-14, pp. 1775–1795, 2018.

[20] E. Huang, Z. Jia, and M. T. Mason, "Large-scale multi-object rearrangement," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 211–218.

[21] J. Lee, Y. Cho, C. Nam, J. Park, and C. Kim, "Efficient obstacle rearrangement for object manipulation tasks in cluttered environments," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 183–189.

[22] Z. Pan and K. Hauser, "Decision making in joint push-grasp action space for large-scale object sorting," *arXiv preprint arXiv:2010.10064*, 2020.

[23] R. H. Taylor, M. T. Mason, and K. Y. Goldberg, "Sensor-based manipulation planning as a game with nature," in *Fourth International Symposium on Robotics Research*, 1987, pp. 421–429.

[24] K. Y. Goldberg, "Orienting polygonal parts without sensors," *Algorithmica*, vol. 10, no. 2, pp. 201–225, 1993.

[25] K. M. Lynch and M. T. Mason, "Dynamic nonprehensile manipulation: Controllability, planning, and experiments," *The International Journal of Robotics Research*, vol. 18, no. 1, pp. 64–92, 1999.

[26] M. Dogar and S. Srinivasa, "A framework for push-grasping in clutter," *Robotics: Science and systems VII*, vol. 1, 2011.

[27] J. Bohg, A. Morales, T. Asfour, and D. Kragic, "Data-driven grasp synthesis—a survey," *IEEE Transactions on Robotics*, vol. 30, no. 2, pp. 289–309, 2013.

[28] N. C. Dafle, A. Rodriguez, R. Paolini, B. Tang, S. S. Srinivasa, M. Erdmann, M. T. Mason, I. Lundberg, H. Staab, and T. Fuhlbrigge, "Extrinsic dexterity: In-hand manipulation with external forces," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2014, pp. 1578–1585.

[29] A. Boularias, J. Bagnell, and A. Stentz, "Learning to manipulate unknown objects in clutter by reinforcement," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 29, no. 1, 2015.

[30] N. Chavan-Dafle and A. Rodriguez, "Prehensile pushing: In-hand manipulation with push-primitives," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2015, pp. 6215–6222.

[31] A. Gál and P. B. Miltersen, "The cell probe complexity of succinct data structures," *Theoretical computer science*, vol. 379, no. 3, pp. 405–417, 2007.

[32] E. Curtin and M. Warshauer, "The locker puzzle," *The Mathematical Intelligencer*, vol. 28, no. 1, pp. 28–31, 2006.

[33] J. E. Hopcroft, J. T. Schwartz, and M. Sharir, "On the complexity of motion planning for multiple independent objects; PSPACE-hardness of the" warehouseman's problem"," *The International Journal of Robotics Research*, vol. 3, no. 4, pp. 76–88, 1984.

[34] G. Wilfong, "Motion planning in the presence of movable obstacles," *Annals of Mathematics and Artificial Intelligence*, vol. 3, no. 1, pp. 131–150, 1991.

[35] J. van Den Berg, J. Snoeyink, M. C. Lin, and D. Manocha, "Centralized path planning for multiple robots: Optimal decoupling into sequential plans." in *Robotics: Science and systems*, vol. 2, no. 2.5, 2009, pp. 2–3.

[36] L. Chang, J. R. Smith, and D. Fox, "Interactive singulation of objects from a pile," in *2012 IEEE International Conference on Robotics and Automation*. IEEE, 2012, pp. 3875–3882.

[37] M. Laskey, J. Lee, C. Chuck, D. Gealy, W. Hsieh, F. T. Pokorny, A. D. Dragan, and K. Goldberg, "Robot grasping in clutter: Using a hierarchy of supervisors for learning from demonstrations," in *2016 IEEE International Conference on Automation Science and Engineering (CASE)*. IEEE, 2016, pp. 827–834.

[38] A. Eitel, N. Hauff, and W. Burgard, "Learning to singulate objects using a push proposal network," in *Robotics research*. Springer, 2020, pp. 405–419.

[39] A. Zeng, S. Song, S. Welker, J. Lee, A. Rodriguez, and T. Funkhouser, "Learning synergies between pushing and grasping with self-supervised deep reinforcement learning," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 4238–4245.

[40] B. Huang, S. D. Han, A. Boularias, and J. Yu, "Dipn: Deep interaction prediction network with application to clutter removal," in *Proceedings IEEE International Conference on Robotics & Automation*, 2021, note: to appear.

[41] J. E. King, J. A. Haustein, S. S. Srinivasa, and T. Asfour, "Nonprehensile whole arm rearrangement planning on physics manifolds," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2015, pp. 2508–2515.

[42] M. Moll, L. Kavraki, J. Rosell *et al.*, "Randomized physics-based motion planning for grasping in cluttered and uncertain environments," *IEEE Robotics and Automation Letters*, vol. 3, no. 2, pp. 712–719, 2017.

[43] W. Bejjani, R. Papallas, M. Leonetti, and M. R. Dogar, "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)*. IEEE, 2018, pp. 1–9.

[44] H. Song, J. A. Haustein, W. Yuan, K. Hang, M. Y. Wang, D. Kragic, and J. A. Stork, "Multi-object rearrangement with monte carlo tree search: A case study on planar nonprehensile sorting," *arXiv preprint arXiv:1912.07024*, 2019.

[45] M. Stilman, J.-U. Schamburek, J. Kuffner, and T. Asfour, "Manipulation planning among movable obstacles," in *Proceedings 2007 IEEE international conference on robotics and automation*. IEEE, 2007, pp. 3327–3332.

[46] A. Krontiris and K. E. Bekris, "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)*. IEEE, 2016, pp. 3924–3931.

[47] R. M. Karp, "Reducibility among combinatorial problems," in *Complexity of computer computations*. Springer, 1972, pp. 85–103.

[48] C. H. Papadimitriou, "The euclidean travelling salesman problem is np-complete," *Theoretical computer science*, vol. 4, no. 3, pp. 237–244, 1977.

[49] G. B. Dantzig and J. H. Ramser, "The truck dispatching problem," *Management science*, vol. 6, no. 1, pp. 80–91, 1959.

[50] C. Nam, J. Lee, Y. Cho, J. Lee, D. H. Kim, and C. Kim, "Planning for target retrieval using a robotic manipulator in cluttered and occluded environments," *arXiv preprint arXiv:1907.03956*, 2019.

[51] K. Solovey and D. Halperin, "k-color multi-robot motion planning," *The International Journal of Robotics Research*, vol. 33, no. 1, pp. 82–97, 2014.

[52] P. Flajolet and R. Sedgewick, *Analytic combinatorics*. cambridge University press, 2009.

[53] R. C. Prim, "Shortest connection networks and some generalizations," *The Bell System Technical Journal*, vol. 36, no. 6, pp. 1389–1401, 1957.

[54] D. B. Johnson, "Priority queues with update and finding minimum spanning trees," *Information Processing Letters*, vol. 4, no. 3, pp. 53–57, 1975.

[55] Y.-J. Chu, "On the shortest arborescence of a directed graph," *Scientia Sinica*, vol. 14, pp. 1396–1400, 1965.

[56] J. Edmonds, "Optimum branchings," *Journal of Research of the national Bureau of Standards B*, vol. 71, no. 4, pp. 233–240, 1967.

[57] H. W. Kuhn, "The Hungarian method for the assignment problem," *Naval research logistics quarterly*, vol. 2, no. 1-2, pp. 83–97, 1955.