Large-Scale Multi-Object Rearrangement

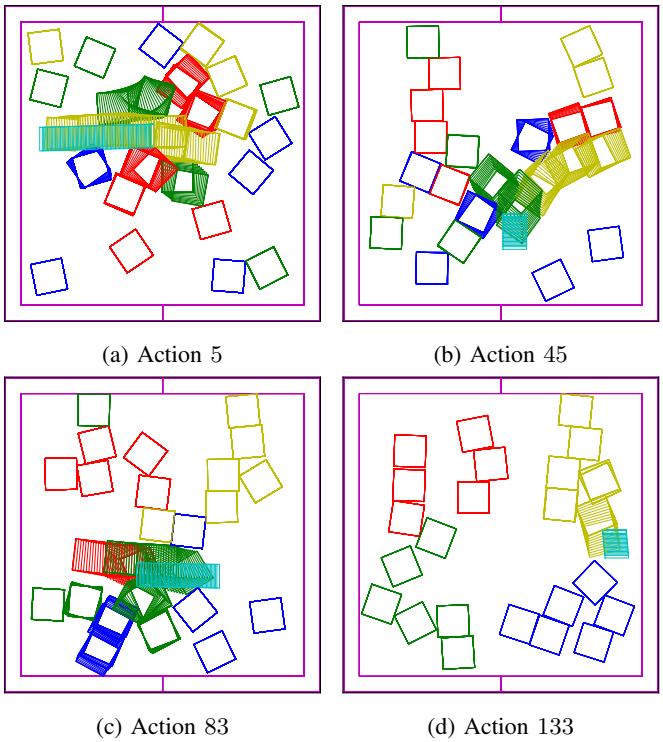
Eric Huang, Zhenzhong Jia, and Matthew T. Mason

***Abstract*—This paper describes a new robotic tabletop rearrangement system, and presents experimental results. The tasks involve rearranging as many as 30 to 100 blocks, sometimes packed with a density of up to 40%. The high packing factor forces the system to push several objects at a time, making accurate simulation difficult, if not impossible. Nonetheless, the system achieves goals specifying the pose of every object, with an average precision of $\pm 1\,\mathrm{mm}$ and $\pm 2°$. The system searches through policy rollouts of simulated pushing actions, using an Iterated Local Search technique to escape local minima. In real world execution, the system executes just one action from a policy, then uses a vision system to update the estimated task state, and replans. The system accepts a fully general description of task goals, which means it can solve the singulation and separation problems addressed in prior work, but can also solve sorting problems and spell out words, among other things. The paper includes examples of several solved problems, statistical analysis of the system's behavior on different types of problems, and some discussion of limitations, insights, and future work.**

I. Introduction

Past robotics research has identified certain scenarios in which robots need to *rearrange* multiple objects in their environment in order to accomplish a goal. One early example of such a scenario is the problem of navigation among moveable obstacles (NAMO) introduced by Wilfong [30]. Later on, Stilman and Kuffner [27] demonstrated the possibility of real-time NAMO in tight household spaces containing upwards of 90 pieces of furniture, despite NAMO problems being fundamentally NP-hard to solve [8]. Home environments are also a natural setting for scenarios involving pick-and-place rearrangement planning. The most difficult problems in this domain are *non-monotone*, i.e. the solutions require moving objects more than once. Recently, Krontiris and Bekris [19] and Han et al. [11] presented algorithms which handle non-monotonicity during pick-and-place rearrangement on shelves and tabletops, respectively; however, finding optimal solutions is often NP-hard [11].

This paper focuses on *non-prehensile* rearrangement planning, a type of rearrangement which emphasizes the need for multi-contact manipulation. For the rest of this paper, the term rearrangement planning implies the non-prehensile variant. We categorize non-prehensile rearrangement planning problems based on whether the objective is to singulate, separate, reposition, arrange, or sort objects. We define *singulate* as isolating a single object for grasping [14, 20, 7, 25, 9, 1, 32], *separate* as acheiving a minimum separating distance between all objects [5, 10], *reposition* as moving a subset of the objects into target positions (no orientation constraints) [16, 17, 18, 12, 13, 31, 3], *arrange* as moving the objects into a target configuration (with position and orientation constraints) [2], *sort* as grouping objects based on some similarity metric.



(a) Action 5 (b) Action 45

(c) Action 83 (d) Action 133

Fig. 1: Example pushes generated by our algorithm as it sorts 24 blocks by color.

Prior work has been limited to solving a *large-scale* local rearrangement problem, i.e. singulation where the behaviors of non-target objects (obstacles) are largely inconsequential, or a *small-scale* global rearrangement problem, such as separating a maximum of 12 objects [6] or repositioning 2 out of 6 objects max [12, 13]. We define a *local* rearrangement problem as one whose solution only requires the robot to interact with a small neighborhood around the object(s) of interest. Likewise, a rearrangement problem is *global* if the solution requires interaction with all objects in the scene. Historically, global problems are harder. For instance, Zeng et al. [32] could singulate 1 object out of 30 (1/30) from a random initial configuration but only 1/6 from a packed initial configuration. Table I provides an extensive list of prior results.

Our work introduces the first algorithm capable of solving large-scale global rearrangement planning problems. We validate our algorithm on rearrangement problems which contain

	max packing factor					max # objects	avg success
	singulate	separate	reposition	rearrange	sorting		
Chang et al. [5]	-	$12/12^{\dagger}$, 0.22^{*}	-	-	-	$12/12$, 0.22^{*}	98%
Hermans et al. [14]	$1/6$, 0.07^{*}	-	-	-	-	$1/6$, 0.07^{*}	72%
Laskey et al. [20]	$1/4$, 0.20^{*}	-	-	-	-	$1/4$, 0.20^{*}	65%
King et al. [16, 17, 18]	-	-	$1/6^{\ddagger}$, 0.09^{*}	-	-	$1/6^{\ddagger}$, 0.09^{*}	100%
Eitel et al. [10]	-	$8/8$, 0.10^{*}	-	-	-	$8/8$, 0.10^{*}	57%
Haustein et al. [12, 13]	-	-	$2/6^{\ddagger}$, 0.04^{*}	-	-	$2/6^{\ddagger}$, 0.04^{*}	99%
Danielczuk et al. [7]	$1/10$, 0.21^{*}	-	-	-	-	$1/10$, 0.21	?%
Yuan et al. [31]	-	-	$1/4$, 0.02^{*}	-	-	$1/4$, 0.02^{*}	70%
Muhayyuddin et al. [25]	$1/40^{\ddagger}$, 0.25^{*}	-	-	-	-	$1/40^{\ddagger}$, 0.25^{*}	90%
Dogar et al. [9, 1, 3]	$1/16^{\ddagger}$, 0.17	-	$3/3^{\ddagger}$, 0.07^{*}	-	-	$1/16^{\ddagger}$, 0.17	93%
Anders et al. [2]	-	-	-	$9/9^{\dagger}$, 0.20^{*}	-	$9/9^{\dagger}$, 0.20^{*}	96%
Zeng et al. [32]	$\mathbf{1/30}^{\dagger}$, $\mathbf{0.47}^{*}$	-	-	-	-	$1/30^{\dagger}$, 0.47^{*}	81%
Our work	$1/33$, 0.41	$\mathbf{25/25}$, $\mathbf{0.31}$	$\mathbf{32/32}$, $\mathbf{0.20}$	$\mathbf{32/32}$, $\mathbf{0.20}$	$\mathbf{24/24}$, $\mathbf{0.31}$	$\mathbf{100/100}$, $\mathbf{0.10}$	91%

TABLE I: Comparison of prior work with ours in non-prehensile rearrangement planning. Authors are ordered by latest publication date. All numbers are given in terms of the max achieved. †Grasping included as a action primitive. ‡Pusher motion restricted to be in-plane. *Packing factor estimated from images in paper.

up to 100 objects and packing factors of up to 40%. We believe this result is impressive (and surprising) for a number of reasons. First, the planner must reason about complex multi-object, multi-contact dynamics. Second, the problem is strongly non-monotonic. In the tightly packed environments we study, objects clump together and must be acted upon *en masse*. Finally, the search space is large. The continuous action space and large number of objects generates very high branching factors.

This paper introduces Iterated Local Search (ILS) with an annealing ε-greedy policy as the first known algorithm for solving large-scale global rearrangement planning problems. The ILS algorithm consists of two components, the inner the local search, i.e. the annealing ε-greedy policy, and the outer loop with iterates over local searches. The ε-greedy policy starts from the current state, and randomly selects and rolls out pushing actions. If the action is greedy, then the rollout terminates when the distance to goal is minimized. A non-greedy action would continue to some pre-determined maximum. By repeating the random selection of mostly greedy actions, the policy rollout would produce a sequence of actions and a corresponding system trajectory. This local search would tend to descend the gradient, but might get stuck in a local minimum. We solve this issue using an iterated local search. Two departures are used to define the iterated local search. First, each local search selectively uses non-greedy actions – actions which will proceed a considerable distance without regard to distance-to-goal. These non-greedy actions are employed according to an annealing schedule. They appear frequently at the beginning of the local search, and less frequently later on. Second, the local search is repeated several times until a sequence of actions which improves the overall cost-to-goal is found. The result is a plan that might be kicked out of local minima several times but always moves closer to the goal.

This paper's results have effectively shown that a simple greedy heuristic search can solve non-prehensile rearrangement planning problems which are more difficult than those previously considered. Our algorithm and results suggest that this problem space is *approximately convex*. We believe this novel insight will be important for any future work in this domain. Moreover, our algorithm transfers from simulation to real-world experiments without any parameter tuning, uncertainty modeling, or learning.

The rest of the paper is organized as follows. Section II discusses related work in more depth and quantitatively compares past results in Table I. Section III provides a background for our algorithm by formally defining non-prehensile rearrangement problems and Markov Decision Processes. Section IV describes our algorithm by first introducing ILS and then the annealing ε-greedy policy. Section VI presents our simulated and real-world experimental results. Section VII discusses the reasons behind our algorithm's performance. Section VIII reviews our algorithm's limitations and proposes future work, while Section IX gives concluding remarks.

II. Related Work

Chang et al. [5] and Hermans et al. [14] both proposed a pushing policy for singulating objects based on visual boundary information. Laskey et al. [20] learned a deep grasping policy which pushes clutter away to achieve a grasp on a target object. King et al. [16, 17, 18] published a number of different planning methods which used pushing primitives to reposition a single object in the presence of limited clutter. Eitel et al. [10] learned a push proposal network for separating objects. Haustein et al. [13] solve non-prehensile rearrangement planning problems by employing a variant of Rapidly-Exploring Random Trees (RRTs) augmented with a learned state generator and a learned policy (action generator). Similar to their previous work [12], Haustein et al. [13] restricted the pusher motion to lie in-plane, which likely makes their problems as difficult as NAMO[1]. Danielczuk et al.

[1]In our opinion, this restriction is unnecessary given that the target task is tabletop rearrangement

[7] proposed two novel deterministic pushing policies for singulating objects in a bin using only one push. Unfortunately, their results lacked any post-push grasping success rates. Yuan et al. [31] adopted Deep Q-Learning (DQN) to rearrangement planning, noting how both Atari games and tabletop pushing tasks are essentially 2D; however, their results were limited to repositioning 1 out of 2-4 objects. Muhayyuddin et al. [25] generated plans for non-prehensile reach-to-grasp tasks using Kinodynamic Motion Planning by Interior and Exterior Cell Exploration (KPIECE) with uncertainty propagation and safe motion biases. Dogar and Srinivasa [9] introduced push-grasping, i.e. singulating an object for grasping using non-prehensile actions, and described an action library approach to generating push-grasp trajectories. Agboh and Dogar [1] formulate push-grasping as an online stochastic trajectory optimization problem. Bejjani et al. [3] also applied Deep Q-Learning to rearrangement planning problems but were only able reposition 3 out of 3 objects. Anders et al. [2] placed and then pushed individual blocks one at a time into a packed configuration using forward search through in belief space with a learned transition model. Though they demonstrated the ability to rearrange objects into target positions and orientations, they were only able to handle a very specific type of goal configuration, i.e. a corner-pyramid configuration supported by the bottom and right walls. Zeng et al. [32] used Q-learning to train a fully convolutional network which labeled pixels as potential pushing or grasping actions locations. They tested on scenes with 30 randomly placed objects and scenes with 6 tightly-packed objects.

To the best of our knowledge, all related papers (see Table I) only address a single category at a time. Moreover, no prior work has demonstrated the capability to arrange objects to arbitrary configurations or sort objects. Though Anders et al. [2] demonstrated arranging, they were only able to handle a very specific corner-pyramid configuration, whereas our algorithm can arrange blocks into any character or word (Section VI). This limitation indicates the lack of a *general* approach. With the sole exception of singulation problems, Table I shows that our algorithm outperforms all prior work with respect to the number of objects and packing factors they can handle.

III. Background

A. Push Planning for Rearrangement Tasks

We take the definition of table-top rearrangement planning from King et al. [16]. Let there be n (not necessarily unique) moveable objects on a flat surface. Assuming the objects do not roll or topple, the n objects form the state space $S = \mathbb{R}^{3n}$. Given a start state $s \in S$, the goal of table-top rearrangement planning is to find a sequence of non-prehensile actions $a_1, \ldots, a_n$ which rearrange the objects into any state g in the goal set $G \subset S$.

Similar to prior work, we restrict non-prehensile actions to pushes. We define a pushing action as the tuple $a = \langle h, v, t_v, greedy \rangle$, where $h \in \mathbb{R}^3$ is the pre-push pose of the pusher, $v \in \mathbb{R}^2$ is the pusher velocity (no rotation, i.e. a straight line push), t_v is the pushing duration, and $greedy \in \{0, 1\}$ determines whether the action is evaluated greedily or not. In practice, we also fix a maximum greedy distance d_g and a maximum random distance d_r at which to stop evaluating the action. We assume the pushing actions are evaluated in a quasi-static environment, i.e. inertial forces are negligible [24].

B. Markov Decision Processes

In this paper, we model the rearrangement planning problem as a Markov Decision Process (MDP) with continuous state and action spaces [28]. Recall that an MDP $\mathcal{M}$ is described by a five-tuple $\langle S, A, T, R, \gamma \rangle$, where $S \subseteq \mathbb{R}^N$ is an N-dimensional state space, $A \subseteq \mathbb{R}^D$ is an D-dimensional action space, $T(s, a, s')$ is the probability that action a applied to state s will lead to state s' in the next time step, $R(s, a, s')$ is the immediate reward received after transitioning from state s to state s' due to action a, and $\gamma \in [0, 1]$ is the discount factor on future rewards. A non-deterministic policy $\pi(s, a)$ specifies the probability that the agent, or robot, chooses action a in state s. The expected value of a policy $V^\pi(s) = \sum_{t=0}^{\infty} \gamma^t R_t$ is the discounted sum over the expected rewards starting at state s and following policy π. The solution to an MDP is an optimal policy π^* which maximizes $V^\pi(s), \forall s \in S$.

C. Policy Rollouts

A *policy rollout* generates a solution sequence $a_1, \ldots, a_n$ to an MDP $\langle S, A, T, R, \gamma \rangle$ with start s and goals G by sampling and rolling out actions from a policy $\pi(s, a)$ until a terminal state is reached. Not unexpected, the policies themselves determine the effectiveness of this solution method. Thus, Subsection IV-A describes a policy suitable for non-prehensile rearrangement planning.

IV. Methods

A. ε-greedy Pushing Policy

This section adopts the classic ε-greedy policy formulation to continuous, non-prehensile action spaces [28]. Given a probability ε, an ε-greedy policy π_ε selects random actions with probability ε and greedy actions with probability $1 - \varepsilon$. To extend π_ε to continuous action spaces, we have π_ε execute greedy actions using a *steering function* [21]. To increase sample efficiency, we have π_ε sample pushing actions from an *object-centric* action space [17].

1) Steering Functions: We adopted the steering function from a class of sample-based planning algorithms known as Rapidly-Exploring Random Trees (RRT) [21]. Let a be a greedy action and suppose taking action a at state s generates the trajectory $\mathcal{T}(t)$, $t \in [0, \tau]$. Given a goal set G, the steering function returns the state s' along the trajectory $\mathcal{T}(t)$ which minimizes the cost-to-goal, that is

$$s' = \underset{x \in \mathcal{T}}{\arg\min} \{ \|x - g\| \mid g \in G \}. \tag{1}$$

Greedy execution using a steering function means action a is only applied up until we reach state s', or up to time $\mathcal{T}^{-1}(s')$. If a is random (not greedy) then the action is taken in full,

Algorithm 1 Iterated Local Search

```
1: function ITERATED-LOCAL-SEARCH(s, g)
2:     x ← s
3:     T ← [ ]
4:     for i ∈ RANGE(N_max) do
5:         T' ← LOCAL-SEARCH(x, g)
6:         if DIST(T') < DIST(T) then
7:             T ← T ∪ T'
8:             x ← FINAL-STATE(T)
9:         if TERMINAL(x) then
10:            return T
11:    return T
```

Algorithm 2 Annealing ε-greedy Policy Rollouts

```
1: function LOCAL-SEARCH(s, g)
2:     x ← s
3:     T ← [ ]
4:     for i ∈ RANGE(N_search) do
5:         ρ ← ρ_0 / i
6:         ε ← 1 / (1 + exp 1/ρ)
7:         a ← π_ε(x)
8:         T ← T ∪ ROLLOUT(x, a)
9:         x ← FINAL-STATE(T)
10:        if TERMINAL(x) then
11:            return T
12:    return T
```

i.e. $s' = \mathcal{T}(\tau)$. Note, examples of distance functions are given in Subsections V-A and V-B.

2) Object-Centric Sampling: An action is said to be object-centric if it interacts in a targeted way with a single object [17]. In this work, we sample an object-centric action in two phases. First, select an object not at its goal, i.e. $\|p - g\| > \delta$, where p is the object pose, g is the goal pose, and δ is the goal tolerance. Next, we sample a pusher velocity v from a velocity set $V \subseteq \mathbb{R}^2$ and a collision-free pre-push pose h such that the pusher motion from h with velocity v intersects the selected object. Furthermore, if the action is greedy, we sample velocities from the positive half-space $V^+ = \{v | v \cdot (g - x) \geq 0, v \in V\}$ instead of V (for efficiency).

Examples of V include $V_{com} = \{v \,|\, \|v\| = 1\}$, the set of unit velocities which pass through the object center, $V_{NESW} = \{(\cos\theta, \sin\theta) \,|\, \theta = n\pi/2, n \in \mathbb{Z}\}$, the set of cardinal directions, and $V_{OCT} = \{(\cos\theta, \sin\theta) \,|\, \theta = n\pi/4, n \in \mathbb{Z}\}$, the set of principal directions.

B. Iterated Local Search

The Iterated Local Search (ILS) meta-algorithm iteratively builds a sequence of solutions generated by the embedded local search heuristic [22]. At each iteration, ILS runs the heuristic once and saves the returned sequence only if it reduces the cost to goal. Algorithm 1 lists ILS pseudo-code.

In this work, we use an annealing ε-greedy policy rollout as the embedded heuristic. We schedule ε according to a temperature-controlled acceptance function (lines 5 and 6). This raises the chance of a non-greedy action significantly when the number of search iterations is very small, which helps kick the system out of local minima.

V. Implementation

Recall that the steering function applies an action a up to the point where a distance function between the rolled out state and the goal is minimized. Subsections (V-A and V-B) describe the two distance functions used in our algorithm.

A. Weighted Euclidean Distance Function

Given a set of object poses $p_1, \ldots, p_n$, corresponding goal poses $g_1, \ldots, g_n$, and non-negative weights $w_1, \ldots, w_n$, the weighted Euclidean distance function returns

$$Dist = \sum_{i=1}^{n} \|p_i - g_i\|_{w_i}. \tag{2}$$

Note, we first wrap the angular component of p_i and g_i to be within $[0, 2\pi/m_i)$, where m_i is the number of radial symmetries of object i.

B. Linear Assignment Distance Function

Suppose that some objects are not unique. Because non-unique objects can be assigned to their corresponding goals in any permutation, we must modify the distance function to additionally solve the *assignment problem* [15].

Let each set of identical objects O_k be assigned an index k. Then we can write the set of object poses and goal poses for O_k as $p_1^k, \ldots, p_{n_k}^k$ and $g_1^k, \ldots, g_{n_k}^k$, respectively, where $n_k = \|O_k\|$. Let a $n \times n$ cost matrix C be comprised of the weighted Euclidean distances between all pairs of object poses p_i^k and goals g_j^k. The linear assignment problem is to find an bijective assignment $A_k : \{1, \ldots, n_k\} \to \{1, \ldots, n_k\}$ such that the cost function

$$\sum_{i=1}^{n_k} C_{i, A_k(i)} \tag{3}$$

is minimized. This problem can be solved in $O(n^3)$ time using the Jonker-Volgenant algorithm [15]. The total distance across all sets of identical objects is given by

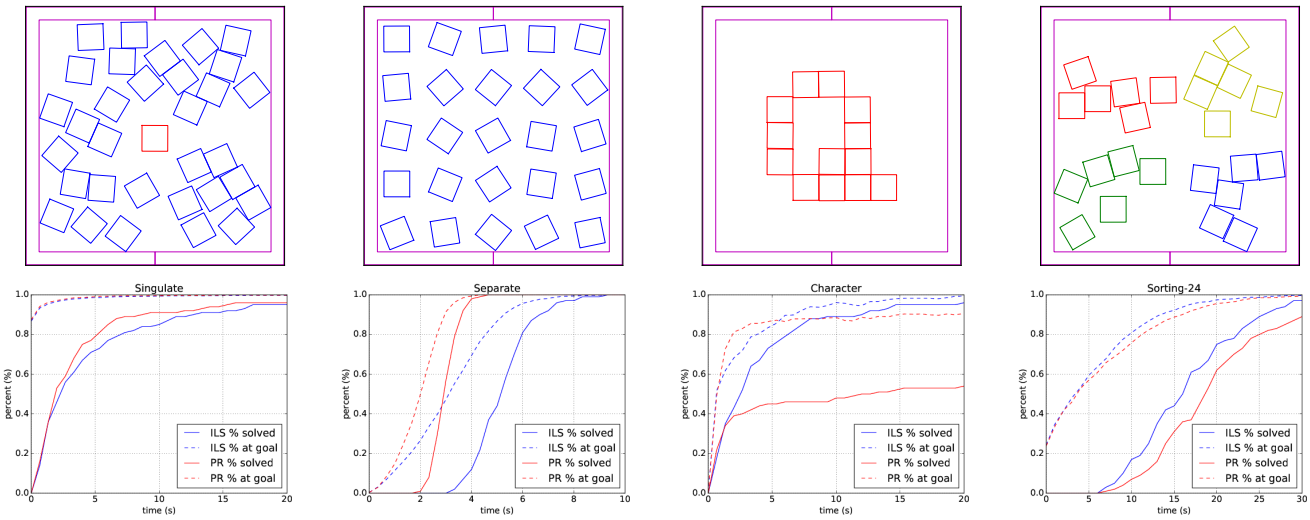
$$\sum_{k=1}^{n_o} \sum_{i=1}^{n_k} C_{i, A_k(i)}, \tag{4}$$

where n_o is the number of object sets.

VI. Experiments

A. Simulation Problem Setups

We used the following simulation environment setup in our experiments. The objects consisted of 4×4 colored blocks. The workspace was restricted to a 40×40 square with a virtual out-of-bounds region with thickness 2. We introduced the virtual out-of-bounds region to give the robot enough space to push blocks out from edges and corners. For an example workspace, see Figure 1. Unless otherwise specified, the robot



(a) Singulate (b) Separate (c) Character (d) Sorting-24

Fig. 2: Top: Example final states for each problem. Bottom: Plots of the percentage of problems solved over time for ILS (solid-blue) and ε-greedy policy rollouts (solid-red) and the percentage of objects at goal over time for ILS (dotted-blue) and ε-greedy policy rollouts (dotted-red).

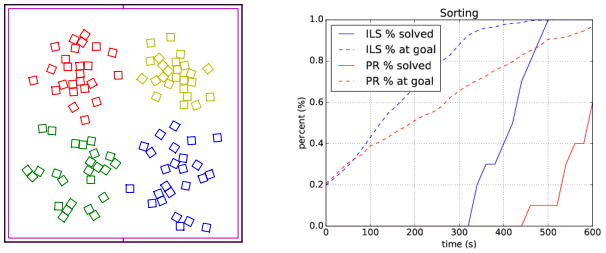


Fig. 3: Left: Example final state of *Sorting-100*. Right: Plots of the percentage of *Sorting-100* problems solved over time for ILS (solid-blue) and ε-greedy policy rollouts (solid-red) and the percentage of objects at goal over time for ILS (dotted-blue) and ε-greedy policy rollouts (dotted-red).

was equipped with a 3×0.5 fence pusher. All algorithms were run on a computer with an Intel i7-7820x CPU (3.5 MHz, 16 threads). The simulation environment was implemented in Box2D [4]. Note, we have limited our presented results to the hardest problems in each category. We have also validated our algorithm on smaller versions of these problems.

1) Singulate: This problem required the robot to singulate the block nearest to the center of the work space away from the 32 other blocks (total 33 blocks). For the target block, we used a weighted euclidean distance function with goal $g_t = [0, 0, 0]$, weights $w_t = [1, 1, 0]$, and goal tolerance $\delta_t = 0.5$. For all other blocks, we used a linear assignment distance function with $g_i \in [\pm 9, \pm 9, 0]$, $w_i = [1, 1, 0]$, and $\delta_i = 9$. Packing factor ≈ 0.41.

2) Separate: This problem required the robot to separate 25 blocks into a 5×5 grid. We used a linear assignment distance function with $g_{i,j} = [15 - 7.5i, 15 - 7.5j, 0]$, $w_{i,j} = [1, 1, 0]$, $\delta_{i,j} = 0.1$, and $i, j \in [0, 5)$. The robot was equipped with a 0.5×0.5 square pusher to help it squeeze between tightly packed blocks.

3) Character: This problem required the robot to rearrange 4×4-sized blocks into characters from the alphabet. The number of blocks in each character ranged from 3 to 13. We used a linear assignment distance function with g_i as the i-th block's location in the character, $w_i = [1, 1, 5]$, and $\delta_i = 0.1$. Max packing factor ≈ 0.16.

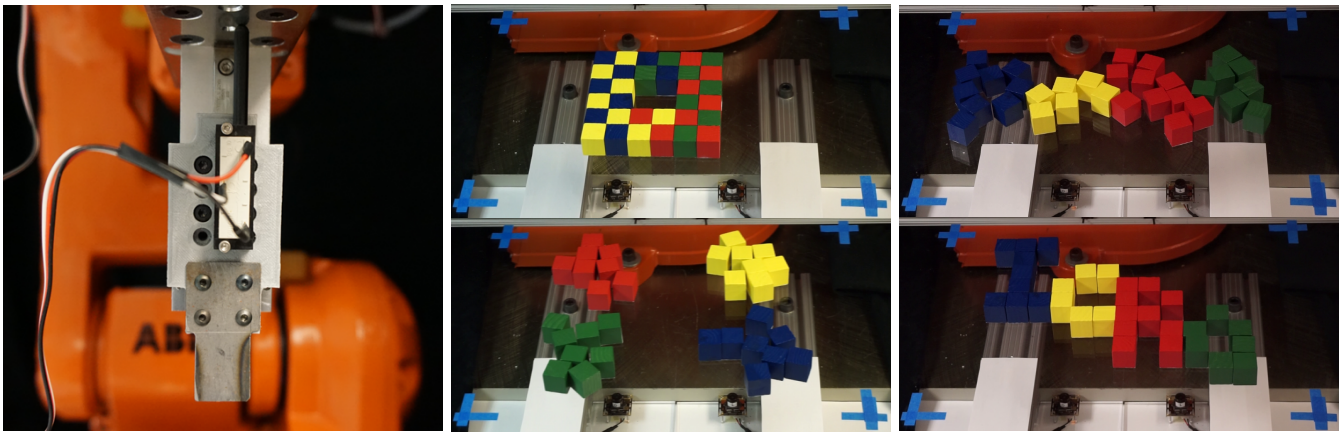
4) Sorting-24: This problem required the robot to sort 4 sets of 6 blocks by color (red, blue, yellow, and green). We used a weighted Euclidean distance function for each color c with each goal g_c being one of $[\pm 9, \pm 9, 0]$, $w_c = [1, 1, 0]$, and $\delta_c = 9$. Packing factor ≈ 0.30.

5) Sorting-100: This problem is the same as *Sorting-24* except with 25 blocks in each set (total 100), a workspace of size 125×125, $g_c \in [\pm 30.25, \pm 30.25, 0]$, and $\delta_c = 30.25$. Packing factor $\approx 10\%$.

B. Simulation Results

Both ILS and ε-greedy policy rollouts were used to solve the above problems. Both algorithms used the velocity set V_{NESW}, as we found it to have good performance across all problems. ILS used an initial temperature of $\rho_0 = 1.7$ for all problems. The following parameters are given in the order of problem numbering. For both algorithms, we set $d_g = 20, 20, 20, 20, 50$ and $d_r = 8, 4, 4, 4, 25$. For ε-greedy policy rollouts, we set ε to 0.85, 0.999, 0.999, 0.85, and 0.75. We set time limits for both algorithms to 20 s, 20 s, 20 s, 30 s, and 600 s. We collect our results in Tables II and III.

We observed that ILS statistically outperforms ε-greedy policy rollouts on *Character*, *Sorting-24*, and *Sorting-100*. *Character* problems required a large number of actions to solve, which can make the wrong non-greedy action very



(a) Fence pusher (b) Sorting-32 (c) Word-IcRa

Fig. 4: Images of our experimental setup for physical experiments. (a) Fence pusher with linear potentiometer. (b) Initial and final configurations for the *Sorting-32* problem. (c) Initial and final configurations for the *Word-IcRa* problem. The blue corner tape in (b) and (c) denote the workspace limits of the robot.

	# objects, pf	% objects at goal	successes
Singulate	1/33, 0.41	99.73%	$95\% \pm 2\%$
Separate	25/25, 0.31	100%	$100\% \pm 0\%$
Character	13/13, 0.16	99.42%	$\mathbf{96\%} \pm \mathbf{2\%}$
Sorting-24	24/24, 0.30	99.66%	$\mathbf{97\%} \pm \mathbf{2\%}$
Sorting-100	100/100, 0.10	100%	$\mathbf{100\%} \pm \mathbf{0\%}$

TABLE II: ILS results

	# objects, pf	% objects at goal	successes
Singulate	1/33, 0.41	99.81%	$96\% \pm 2\%$
Separate	25/25, 0.31	100%	$100\% \pm 0\%$
Character	13/13, 0.16	90.56%	$54\% \pm 5\%$
Sorting-24	24/24, 0.30	95.585%	$89\% \pm 3\%$
Sorting-100	100/100, 0.10	96.7%	$60\% \pm 5\%$

TABLE III: ε-greedy results

costly. The main advantage of ILS over ε-greedy is that ILS can erase mistakes by throwing away bad sequences. This feature also translates into better search efficiency, which explains the higher success rate in *Sorting-100*. ILS uses the time saved from undoing mistakes to search for better actions.

C. Real-World Problem Setups

We used the following hardware setup in our physical experiments. The objects consisted of $25.4\,\mathrm{mm}$ (1 inch) square blocks of various colors. The workspace was restricted to a $420\,\mathrm{mm}$ by $310\,\mathrm{mm}$ rectangle with a virtual out-of-bounds region of thickness $20\,\mathrm{mm}$. The workspace surface was made from transparent, scratch resistant acrylic so that the objects could be tracked using AprilTags [29] from underneath. The robot was equipped with a $19.05\,\mathrm{mm}$ flat fence pusher (Figure 4a). The pusher was mounted on a linear potentiometer with spring return. The potentiometer was used to detect contact of the fence pusher against the top of the blocks, typically resulting from errors in sensing. On top-contact, the pusher was repositioned $3\,\mathrm{mm}$ behind and the action retried. We tested our algorithm on the following problems.

1) Sorting-32: The goal of this problem is to sort 4 sets of 8 blocks by color. We used a weighted Euclidean distance function with $g_c \in [\pm 95\,\mathrm{mm}, -495\,\mathrm{mm} \pm 67.5\,\mathrm{mm}, 0]$, $w_c = [1, 1, 0]$, and $\delta_c = 60\,\mathrm{mm}$. Similar to [32], the blocks are adversarially packed to increase the problem difficulty. Initial and final configurations are shown in Figure 4b.

2) Word-IcRa: The goal of this problem is to simultaneously rearrange 9 blue, 6 yellow, 10 red, and 7 green blocks into the letters "IcRa", respectively. We used a linear assignment distance function for each letter with g_i^k as the i-th block's location in the k-th letter, $w_i^k = [1, 1, 31.75]$, and $\delta_i^k = 3$. We mixed upper/lower case letters due to constraints on the workspace and total available numbers of blocks of each color. In their initial configurations, the blocks are pre-sorted by color and randomly arranged. In their final configuration, the letters are block aligned and follow a diagonal baseline. Initial and final configurations are shown in Figure 4c.

D. Real-World Results

The ε-greedy pushing policy was used to solve the *Sorting-32* and *Word-IcRa* problems 5 times each. We did not use ILS in our real world experiments because executing an open-loop sequence of actions quickly leads to divergent states. We propose a method to overcome this difficulty in Section VIII. For both problems, execution time was capped to 30 minutes. We set ε to 0.75 and 0.9999, $d_g = 100, 100$, $d_r = 50, 25$, and the velocity set to V_{NESW}. Our results are presented in Table IV.

	# objects, pf	% objects at goal	successes
Sorting-32	32/32, 0.20	100%	5/5
Word-IcRa	32/32, 0.20	98.75%	3/5

TABLE IV: Real-world experimental results

The ε-greedy policy failed twice on the *Word-ICRA* problem. In one failure, the policy cycled between dislodging a yellow block from the counter-space (hole) in "R" and trapping the yellow block back in "R". In the other failure, the policy alternated between fixing the legs of "R" and fixing the counter-space of "a".

VII. Discussion

A. Simulation to Reality

Notably, we transferred the pushing policies from simulation to the real world without *any* tuning or system identification. We hope the following discussion of pushing dynamics can elucidate why this works. Pushing is quasi-static within a surprisingly large range of velocities. For instance, our robot end-effector moved at $0.15\,\mathrm{m\,s^{-1}}$. Under quasi-static dynamics, objects stop when they lose contact with the pusher. Moreover, pushed objects always move in the direction of the pusher. In other words, a pusher velocity v implies an object velocity $\dot{x}$ such that $\dot{x} \cdot v \geq 0$. By using a fence pusher, our system is designed to take advantage of two-point stable pushing [23]. For a single point of contact, contact point location relative to CoM and contact velocity primarily determine whether the pushed object turns left or right [24], meaning the turning direction maps easily from simulation to reality given low measurement error. Lastly, running our algorithm online allows the robot to iteratively correct errors between simulation and reality.

B. Algorithm and Problem Insights

Why were ε-greedy policy rollouts so effective at solving non-prehensile rearrangement planning problems? In our experiments, we observed the following macro-sequence of actions repeat itself over and over again. The sequence begins with a non-greedy action perturbing a group of tightly packed objects. Initially, the cost-to-goal has increased due to the perturbation; however, subsequent greedy actions take advantage of the new openings to better separate the objects toward their goals. Eventually, the sequence achieves a lower cost-to-goal and the next macro-sequence starts in a state with fewer difficult groups of objects and more free space for the remaining objects to move around in.

C. Parameter Selection

Parameter selection was greatly aided by visualizing the various scenarios where the algorithm would get stuck. In *Character*, we observed that 1) precisely building a character can take hundreds of pushes, 2) a long random push can easily destroy all progress, but 3) random pushes are required at the very end to jump out of the local minima. For these reasons, we set ε to 0.999 and d_r to 4 for policy rollouts in *Character*. On the other hand, the *Sorting* required much less precision, allowing us to sample more non-greedy actions to escape local minima quicker. We also observed that longer fence pushers, e.g. length 8, struggled to separate blocks. Using a smaller pusher with length 3 opened up the action space significantly more.

D. Computational Complexity

We conjecture that NP-complete sliding puzzles, such as the n-puzzle, can be reduced to an instance of non-prehensile rearrangement [26]. While we abstain from a proof here, this conjecture suggests the following parallels. Like the n-puzzle, non-prehensile rearrangement planning problems can be solved in polynomial time; however, finding an optimal solution is NP-hard [26].

VIII. Limitations & Future Work

We did not validate our algorithm on problems with non-convex objects due to time and space constraints. The intent of this paper was to demonstrate that a tractable algorithm even exists for large-scale global rearrangement problems. However, future efforts should also determine whether this algorithm works on non-convex objects. In addition, the presented algorithm does not provide any theoretical guarantees on probabilistic completeness or solvability. In practice, we observed that a properly tuned algorithm can solve almost any problem given enough time. Proving probabilistic completeness would be a good direction for future theoretical research.

We observed our algorithms require anywhere from 100 to 200 actions to solve sorting problems with 24 objects and a packing factor of 0.30. Moreover, improving action efficiency is likely NP-hard (see Subsection VII-D). Despite the challenge, we see this as a strong opportunity to apply an AlphaGo Zero style of deep reinforcement learning. Many rearrangement planning problems can be parameterized as a top-down 2D image. In our case, we can rasterize our scene and predict good pre-push positions using fully convolutional neural networks. As a first step, these networks can learn the solutions generated by ILS and implicitly transfer that knowledge to the real world during policy rollouts.

IX. Conclusion

This paper provides algorithmic insight into the nature of pushing multiple objects in clutter. Specifically, we demonstrate that policy rollouts with a greedy action space are sufficient for push planning on table-top rearrangement tasks. We also show that using an iterated local search technique can help escape local minima and improve results. We successfully applied this algorithm to singulating, separating, arranging, and sorting large-scale clutter in simulated and physical experiments.

References

[1] Wisdom C. Agboh and Mehmet R. Dogar. Real-time online re-planning for grasping under clutter and uncertainty. *CoRR*, 2018. URL https://arxiv.org/abs/1807.09049.

[2] Ariel S. Anders, Leslie P. Kaelbling, and Tomas Lozano-Perez. Reliably arranging objects in uncertain domains. In *IEEE Conference on Robotics and Automation (ICRA)*, 2018. URL http://lis.csail.mit.edu/pubs/anders-icra18.pdf.

[3] Wissam Bejjani, Rafael Papallas, Matteo Leonetti, and Mehmet Remzi Dogar. Planning with a receding horizon for manipulation in clutter using a learned value function. 2018. URL https://arxiv.org/abs/1803.08100.

[4] Erin Catto. Box2d, 2013. URL https://box2d.org.

[5] Lillian Chang, Joshua R Smith, and Dieter Fox. Interactive singulation of objects from a pile. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 3875–3882. IEEE, 2012.

[6] Lillian Y. Chang, Joshua R. Smith, and Dieter Fox. Interactive singulation of objects from a pile. *2012 IEEE International Conference on Robotics and Automation*, pages 3875–3882, 2012.

[7] Michael Danielczuk, Jeffrey Mahler, Chris Correa, and Ken Goldberg. Linear push policies to increase grasp access for robot bin picking. *CASE*, 2018.

[8] Erik D. Demaine, Isaac Grosof, and Jayson Lynch. Push-pull block puzzles are hard. *CoRR*, abs/1709.01241, 2017. URL http://arxiv.org/abs/1709.01241.

[9] Mehmet Dogar and Siddhartha Srinivasa. A framework for push-grasping in clutter. *Robotics: Science and systems VII*, 7, 2011.

[10] Andreas Eitel, Nico Hauff, and Wolfram Burgard. Learning to singulate objects using a push proposal network. *"International Symposium on Robotics Research"*, 2017.

[11] Shuai D Han, Nicholas M Stiffler, Athansios Krontiris, Kostas E Bekris, and Jingjin Yu. High-quality tabletop rearrangement with overhand grasps: Hardness results and fast methods. In *arXiv preprint arXiv:1705.09180*, 2017.

[12] J.A. Haustein, J. King, S.S. Srinivasa, and T. Asfour. Kinodynamic randomized rearrangement planning via dynamic transitions between statically stable configurations. In *IEEE International Conference on Robotics and Automation*, 2015.

[13] Joshua A Haustein, Isac Arnekvist, Johannes Stork, Kaiyu Hang, and Danica Kragic. Learning manipulation states and actions for efficient non-prehensile rearrangement planning.

[14] Tucker Hermans, James M Rehg, and Aaron Bobick. Guided pushing for object singulation. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 4783–4790. IEEE, 2012.

[15] Roy Jonker and Anton Volgenant. A shortest augmenting path algorithm for dense and sparse linear assignment problems. *Computing*, 38(4):325–340, 1987.

[16] J. King, J.A. Haustein, S.S. Srinivasa, and T. Asfour. Nonprehensile whole arm rearrangement planning with physics manifolds. In *IEEE International Conference on Robotics and Automation*, 2015.

[17] J. King, M. Cognetti, and S.S. Srinivasa. Rearrangement planning using object-centric and robot-centric action spaces. In *IEEE International Conference on Robotics and Automation*, 2016.

[18] J. King, V. Ranganeni, and S. S. Srinivasa. Unobservable monte carlo planning for nonprehensile rearrangement tasks. In *IEEE International Conference on Robotics and Automation*, 2017.

[19] Athanasios Krontiris and Kostas E Bekris. Dealing with difficult instances of object rearrangement. In *Robotics: Science and Systems*, 2015.

[20] Michael Laskey, Jonathan Lee, Caleb Chuck, David Gealy, Wesley Hsieh, Florian T Pokorny, Anca D Dragan, and Ken Goldberg. Robot grasping in clutter: Using a hierarchy of supervisors for learning from demonstrations. In *Automation Science and Engineering (CASE), 2016 IEEE International Conference on*, pages 827–834. IEEE, 2016.

[21] Steven M LaValle. Rapidly-exploring random trees: A new tool for path planning. 1998.

[22] Helena R Lourenço, Olivier C Martin, and Thomas Stützle. Iterated local search. In *Handbook of metaheuristics*, pages 320–353. Springer, 2003.

[23] Kevin M. Lynch and Matthew T. Mason. Stable pushing: Mechanics, controllability, and planning. 15(6):533–556, 1996. doi: 10.1177/027836499601500602.

[24] Matthew T. Mason. *Mechanics of Robotic Manipulation*. MIT Press, Cambridge, MA, USA, 2001. ISBN 0-262-13396-2.

[25] Muhayyuddin, Mark Moll, Lydia E. Kavraki, and Jan Rosell. Randomized physics-based motion planning for grasping in cluttered and uncertain environments. *IEEE Robotics and Automation Letters*, 3(2):712–719, April 2018. doi: 10.1109/LRA.2017.2783445.

[26] Daniel Ratner and Manfred K Warmuth. Finding a shortest solution for the n$\times$ n extension of the 15-puzzle is intractable. In *AAAI*, pages 168–172, 1986.

[27] Mike Stilman and James J Kuffner. Navigation among movable obstacles: Real-time reasoning in complex environments. *International Journal of Humanoid Robotics*, 2(04):479–503, 2005.

[28] Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998. ISBN 0262193981.

[29] John Wang and Edwin Olson. Apriltag 2: Efficient and robust fiducial detection. In *IROS*, pages 4193–4198, 2016.

[30] Gordon T. Wilfong. Motion planning in the presence of movable obstacles. *Annals of Mathematics and Artificial Intelligence*, 3:131–150, 1988.

[31] Weihao Yuan, Johannes A. Stork, Danica Kragic, Michael Yu Wang, and Kaiyu Hang. Rearrangement with nonprehensile manipulation using deep reinforcement learning. *IROS*, abs/1803.05752, 2018.

[32] Andy Zeng, Shuran Song, Stefan Welker, Johnny Lee, Alberto Rodriguez, and Thomas A. Funkhouser. Learning synergies between pushing and grasping with self-supervised deep reinforcement learning. *IROS*, abs/1803.09956, 2018.