Interleaving Monte Carlo Tree Search and Self-Supervised Learning for Object Retrieval in Clutter

Baichuan Huang   Teng Guo   Abdeslam Boularias   Jingjin Yu

***Abstract*— In this study, working with the task of object retrieval in clutter, we have developed a robot learning framework in which Monte Carlo Tree Search (MCTS) is first applied to enable a Deep Neural Network (DNN) to learn the intricate interactions between a robot arm and a complex scene containing many objects, allowing the DNN to partially clone the behavior of MCTS. In turn, the trained DNN is integrated into MCTS to help guide its search effort. We call this approach learning-guided Monte Carlo tree search for Object REtrieval (MORE), which delivers significant computational efficiency gains and added solution optimality. MORE is a self-supervised robotics framework/pipeline capable of working in the real world that successfully embodies the System 2 → System 1 learning philosophy proposed by Kahneman, where learned knowledge, used properly, can help greatly speed up a time-consuming decision process over time. Videos and supplementary material can be found at https://github.com/arc-l/more.**

I. Introduction

Kahneman [1] proposed a thought-provoking hypothesis of human intelligence: in solving real-world problems, humans engage fast or "System 1" (S1) type of thinking for making split-second decisions, e.g., speech, driving, and so on. For other decision-making processes, e.g., playing chess, a slow or "System 2" (S2) approach is taken, where the brain would perform a search over some structured domain for the best actions to take. After repeatedly using S2 thinking to solve a given problem, patterns can be distilled over time and burned into S1 to speed up the overall process. In playing chess, for example, good chess players can instinctively identify good candidate moves. First-time or beginner drivers rely heavily on S2 and gradually converge to S1 as they gain more experience. This S2→S1 thinking has gained significant attention and has been explored in many directions in machine learning, including attempts at building machines with consciousness [2]. But, perhaps the most prominent line of work in reinforcement learning [3] that closely aligns with this paradigm is the application of Monte Carlo Tree Search (MCTS) for carrying out self-supervised learning in games [4], [5], where an "understanding" of a game emerges from a lifelong self-play and is gradually distilled so that it significantly reduces the search effort. Gradually, the overall system learns enough useful information that allows it to play perfect games with much less time and computing resources.

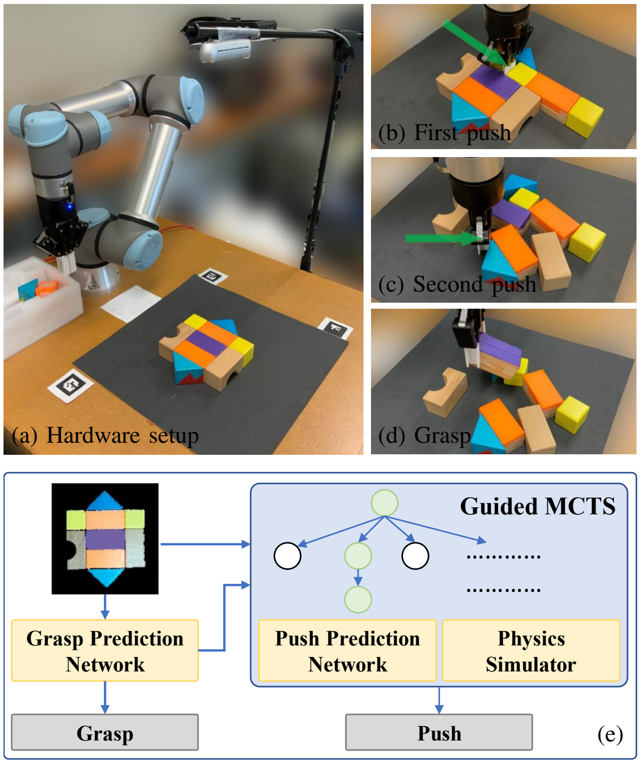

Fig. 1: (a) The hardware setup for object-retrieval-from-clutter includes a Universal Robots UR5e manipulator with a Robotiq 2F-85 two-finger gripper, and an Intel RealSense D455 RGB-D camera. The objects are placed in a square workspace and the target object is masked in purple. (b)(c) Two push actions (shown with green arrows) are used to enable the grasping of the target (purple) object. (d) The target object is successfully grasped and retrieved. (e) The overview of our overall system.

Inspired by [4], [5] that show a search-and-learn approach for realizing S2→S1 applies well to game-like settings with relatively well-defined rules, we set out to find out whether we could build a similar framework that enables real robots to interact with real-world physics and uncertainties to perform physical tasks, somewhat akin to [6]. Specifically, we focus on the task of retrieving an object enclosed in clutter using non-prehensile actions, such as pushing and poking, followed by prehensile two-finger grasping. The goal is to obtain a computationally efficient system and produce high-quality solutions (i.e., using the minimum number of actions).

As pointed out by Valpola [7], due to the difficulty in exploring the landscape of the state space of real-world problems, in addition to uncertainty, naive applications of the S2→S1 paradigm often lead to undesirable behavior. Non-trivial design as well as engineering efforts are needed to build such S2→S1 systems. In the object-retrieval-from-clutter setting, the challenge lies in the difficulty of predicting

B. Huang, G. Teng, A. Boularias, and J. Yu are with the Department of Computer Science, Rutgers, the State University of New Jersey, Piscataway, NJ, USA. Emails: {baichuan.huang, teng.guo, abdeslam.boularias, jingjin.yu}@rutgers.edu.
This work is supported by NSF awards IIS-1734492 and IIS-1846043, IIS-1845888, CCF-1934924 and IIS-2132972.

the outcome of push actions, with the tip of the gripper, when many objects are involved. This is due to discontinuities inherent in object interactions; while a certain pushing action will move a given object, a slightly different direction can miss that same object entirely.

The main contribution of this work is proving the feasibility of applying the S2→S1 philosophy to build a self-supervised robotic object retrieval system capable of continuously improving its computational efficiency, through *cloning* the behavior of the time-consuming initial MCTS phase. Through the careful design and integration of two Deep Neural Networks (DNNs) with MCTS, our proposed self-supervised method, named **M**onte Carlo tree search and learning for **O**bject **RE**trieval (MORE), enables a DNN to learn from the manipulation strategies discovered by MCTS. Then, learned DNNs are fed back to the MCTS process to guide the search. MORE significantly reduces MCTS computation load and achieves identical or better outcomes, i.e., retrieving the object using very few strategic push actions. In other words, our method "closes the loop". This contrasts with [6], which only learns to replace the rollout function of MCTS.

II. Related Work

Grasping. Grasping approaches can be classified as being *analytical* or *data-driven* [8]. Analytical methods examine precise object models to predict the stability of a grasp based on *force-closure* or *form-closure* [9]–[11]. However, high precision 3D models of objects, e.g., YCB objects [12], are hard to come by. In addition, other material properties, such as friction and inertia, are challenging to measure. These challenges have given rise to data-driven methods that learn from data, where many works focus on isolated objects [13]–[16]. Recently, grasping in clutter has received more attention [17]–[21]. Convolutional Neural Networks are widely used to construct grasp proposal networks such as Dex-net 4.0 [22], which are trained to detect 6D grasp poses in point clouds [23]. In this paper, we use a self-supervised Deep Q-Network similar to [24] for grasping in clutter.

Singulation. Singulation, i.e., isolating specific object(s) from the rest [25], is necessary for object retrieval. Usually, a sequence of pushing and grasping actions is used to clear the clutter that surrounds the target object. In [26], a *model-free* method was used to learn a reactive pushing policy without long-horizon reasoning. Later, other model-free reinforcement learning algorithms [24], [27] used learned push policies to improve grasping. In contrast to existing work on singulation, we explicitly seek to minimize the number of actions needed to isolate a target object for grasping sufficiently.

Object Retrieval. Object retrieval from clutter, the focus of this study, can be viewed as a form of rearrangement planning [6], [28], [29]. Online planning for object search with partial observations has been discussed in [30]. Retrieving objects under occlusion was also recently considered in [31] where parallel-jaw and suction grasping were used along with pushing to de-clutter surroundings of target objects. A model-free reinforcement learning technique has also been used for searching for objects in [32]. In [33], an agent was trained to find a continuous trajectory of a gripper that pushes away clutter or pushes the target object to free space, mimicking human-like behavior. A human in-the-loop solution was proposed in [34] help with searching for objects in clutter. A deep Q-Learning method [35] considers a similar task and setup but uses additional primitives such as sliding objects from the top. Our work partially builds on [36], which explores the use of MCTS for the same object retrieval problem. In contrast to existing object retrieval works, we focus on developing the machinery that enables the S2→S1 philosophy to reduce the computational burden of the related search problem while using real robots and objects.

III. Preliminaries

The object-retrieval-from-clutter task consists in using a robot manipulator to retrieve a hard-to-reach target object (Fig. 1). Objects are rigid bodies with various shapes, sizes, and colors; the target object is assigned a unique color. Similar to [36], a top-down fixed camera is installed to observe the workspace. The camera takes an RGB-D image of the workspace (e.g., the top-left image of Fig. 1), which serves as the only input to our system.

Pushing and *grasping* actions are allowed, the execution of each is considered as one *atomic action*. A grasp action is defined as a top-down overhead grasp motion $a^g = (x, y, \theta)$, corresponding to the gripper's target location and orientation, based on a coordinate system defined over the input image. A push action is defined as a quasi-static planar motion $a^p = (x_0, y_0, x_1, y_1)$ where (x_0, y_0) and (x_1, y_1) are the start and the end locations of the gripper tip. The horizontal push distance is fixed and it is 10cm in our experiments. Each primitive action is transformed to the real-world coordinates for execution, but all the planning and reasoning are in image coordinates. The robotic arm keeps pushing objects until the target object can be grasped or until the target object is pushed outside of the workspace, in which case the task is considered a failure. The problem is to find a policy that maximizes the frequency of successfully grasping the target object, while also minimizing the number of pre-grasp pushing actions.

IV. Methodology

The MORE framework consists of three components: a Grasp Prediction Network (GPN), a Monte Carlo Tree Search (MCTS) routine, and a Push Prediction Network (PPN). GPN is a neural network that predicts the success probabilities of grasp actions. It is trained online similarly to [36]. The success probabilities can be interpreted as immediate rewards. MCTS uses a physics engine as a transition function to simulate long sequences of consecutive push actions that end with a terminal grasp action. Each branch in MCTS is composed of push actions as internal nodes, and a grasp action as a leaf. Grasp actions are evaluated with GPN, and the returned rewards are back-propagated to evaluate their corresponding branches. The branch with the highest discounted reward, or *Q-value*, is selected for execution by the robot.

While highly effective in finding near-optimal paths, MCTS suffers from a high computation time that makes it impractical.

To solve this, MORE employs a second neural network, PPN, to prioritize the action selection in the rollout policy. The robot starts by relying entirely on MCTS (S2 type of thinking) to solve various instances of the object-retrieval problem. Instead of throwing away the computation performed by MCTS for solving the various instances, we use the computed Q-values as ground-truth to train PPN. Note that this computation data is free, since it is generated by the simulations performed by MCTS as a byproduct of solving the actual problem. PPN is a neural network that learns to imitate MCTS and clone its behavior, while avoiding heavy computation and physics simulations by MCTS. As PPN becomes more accurate in predicting the outcome of MCTS, the robot starts relying on both MCTS and PPN for action selection. In a nutshell, PPN is used for orienting the search in MCTS toward more promising push actions that rearranges the scene and renders the target object graspable. After a long experience, PPN's accuracy in predicting the Q-values of push actions matches that of MCTS, and the robot switches entirely to PPN to make decisions in a few milliseconds (S1 type of thinking).

A. *Grasp Prediction Network (*GPN*)*

GPN is a deep neural network based on the model proposed in [24] and further customized to estimate the expected grasp reward [36]. We used the pre-trained model from [36]. with a ResNet-18 FPN [37], [38] backbone [39]. For training, only successful grasps are given fixed non-zero rewards. The Grasp Network takes a single RGB-D image s_t as input and outputs a pixel-wise reward map $R_g(s_t) \in [0,1]^{H\times W}$ with the same size (H and W are height and width of s_t). To enable GPN to account for gripper orientation, s_t is rotated 16 times in the range of $[0, 2\pi]$, adding another dimension to the reward map and making it $R_gm(s_t) \in [0,1]^{H\times W\times k}$ with $k = 16$. Because the goal is to retrieve a specific object, a mask is imposed on the target object using Mask R-CNN [40], effectively truncating the reward map. If the largest reward from the map $\max_{i,j,\theta} R_{gm}(s_t)[i,j,\theta]$ is larger than some preset threshold, R_{g*}, GPN suggests grasping as the next action to execute. The location $[i,j]$ and rotation θ of the best grasp is retrieved from the reward map R_{gm}.

B. *Monte-Carlo Tree Search*

Monte-Carlo Tree Search (MCTS) [41] is used in MORE for both decision-making and training PPN. A typical MCTS routine has four steps: selection, expansion, simulation, and back-propagation. In our case, the goal of the search is to find the shortest action sequence; we can stop the search as soon as the best solution is found without exploring the rest. The search stops in two cases: 1) the number of iterations n exceeds a pre-set budget N_{max}, or 2) the expanded node with state that the target object can be grasped, and all nodes in parent level are expanded. A node is considered as a leaf if $\max_{i,j,\theta} R_{gm}(s_t)[i,j,\theta] > R_{g*}$ where $R_{gm}(s_t)$ is obtained from GPN and R_{g*} is a pre-defined high probability. The maximum depth of the tree is limited to d, where d is set to 4 in our experiments.

In the selection phase, we find an expandable node starting from the root according to the search policy

$$\pi_n(s) = \arg\max_{a^p}(Q(s,a^p) + C\sqrt{\frac{\ln N(s)}{N(s,a^p)}}), \tag{1}$$

where $N(s)$ is the number of visits to node (state) s and $N(s,a^p)$ is the number of times push action a^p has been selected at node (state) s. The Q-value is calculated as

$$Q(s,a^p) = \frac{\sum_{i=1}^{m} r_i(s,a^p)}{\min\{N(n_i), m\}}, \tag{2}$$

where $r(s,a^p)$ is the returned long-term reward and m is a pre-set maximum. Only the best m terms $r_i(s,a^p)$ are used to compute the Q-value in the equation above. m is set to 10 when expanding nodes and 1 when selecting the best solution. C is the coefficient of the exploration term, and it is set to 2 when expanding nodes and 0 when selecting the best solution. In the expansion phase, we use a physics simulator to execute the chosen push action a^p at state s_i and predict new state s_{i+1}. Then, a random policy is used to sample actions to simulate until a grasp is possible or a failure is encountered. The reward r is predicted by GPN at a terminal state s_t. Reward r is set to 1 if $\max_{i,j,\theta} R_{gm}(s_t)[i,j,\theta] > R_{g*}$, and 0 otherwise. One additional term $\delta \max(R_{gm}(s_t))$ is added to r, to distinguish between good and bad push actions. We set δ to be 0.2. In the last step, reward r is propagated back to its parent nodes to update their Q-values with a discount factor $\gamma = 0.5$.

As the push action space is enormous even after discretization, we further sample a subset of actions such that all push actions start around the contour of an object and point to the center of the object (Fig. 2). This action sampling method has been discussed in [36] and was empirically proven efficient for a similar setup of object retrieval.

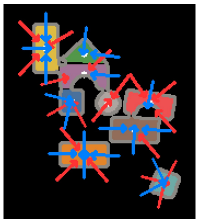

Fig. 2: Sampled push actions.

In our implementation, N_{max} is set to 300 when MCTS is used to collect data to train PPN. The second and the third conditions for stopping the search are only activated after at least 50 roll-outs, so that the number of visits to a state is statistically significant and to reduce the variance of PPN.

C. *Push Prediction Network (*PPN*)*

As previously mentioned, PPN learns to imitate MCTS. PPN is a deep neural network with ResNet-34 FPN [37], [38] as the backbone, where the P2 level of the FPN connects to the head. It takes a two-channel input and outputs a single channel pixel-wise push Q-value map, similar to the reward map produced by GPN. An example input is shown in Fig. 3, where the first channel is a segmented image of all objects and the second channel is a binary image of the target object. The output is the image on the right of Fig. 3, where the arrow shows a push action with the highest Q-value. PPN estimates the Q-value (discounted rewards) $Q_p(s_t)$ of executing push actions at the corresponding pixel, where the action is assumed to push 10cm to the right. $\max(Q_p(s))$ is limited to the range $[0, \eta]$, where η is the maximum reward of a terminal state.

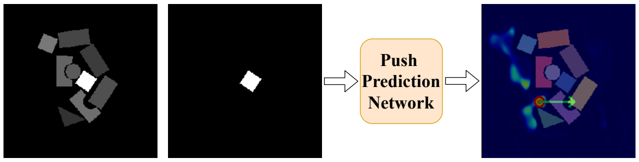

Fig. 3: The left two figures are the input to PPN. The first is a segmentation of objects; the second is the mask of the target object. The image on the right is the output from the PPN. We use Jet colormap to represent the reward value, where the value ranges from red (high) to blue (low). The pixel with the highest Q-value is plotted with a circle and attached with an arrow on the right image, representing pushing action starting at the circle and moving to the right with a distance of 10cm.

When MCTS is used to generate training cases, it builds a tree and saves the transitions for each case: the state (image) s, the push action a^p, the Q-value $Q(s, a^p)$, and the visited number $N(s, a^p)$. As such, PPN is trained in a self-supervised manner. The input image is rotated based on a push action so that the corresponding push action points to the right. Because a single action is generated by MCTS (i.e., a δ signal over the entire input), which is not conducive to training PPN, we "expand" the Q-value over a 3×3 patch centered around MCTS action but set invalid pushes (e.g., if part of the patch is inside an object) to be zero. Now, the label is relatively dense compared to a one-hot pixel, so we can use Smooth L1 loss from Pytorch [42] with β equals to 0.8 to regress. Only gradients on the labeled pixels are used. Loss weighting is also applied: label values from the MCTS are weighted based on $N(s, a^p)$, label values (zero Q-value) from void push actions are weighted with a small number, 0.001 for collision and 0.0001 for pushing thin air. We observe that PPN has difficulty learning to create clearance around the target object. Data augmentation is applied here so that for each training case, we also randomly choose the target object for the MCTS to solve; so each arrangement becomes many training cases. It mitigates over-fitting; given similar visual information, it could learn different strategies, as the target object could be anywhere.

The head model is an FCN with four layers, where the first two layers have a kernel size of 3, the last two 1, and the strides of four layers are all 1. Batch normalization is used at each layer of the head model except the last. Bilinear interpolation ($\times 2$) is applied interleaved between the last three layers of the head model to scale up the hidden state to the same size as the input image. The training process has two stages, one to train the network with a batch size of 8, learning rate starts at $1\mathrm{e}{-4}$, epoch of 50. The learning rate decays with cosine annealing [43], where the maximum number of iterations is set to be the same as the epoch number 50 and the minimum learning rate is $1\mathrm{e}{-8}$. The second is a fine-tuning stage; we increase the batch size to 28 and the learning rate to $1\mathrm{e}{-5}$ with an epoch of 20.

D. Guided Monte-Carlo Tree Search

With the trained GPN and PPN, a guided MCTS is implemented to accelerate the search process, cutting cost from time-consuming expansion and simulation phases. GPN is again used to determine the terminal state and if so, calculate its estimated reward, as discussed in IV-B. PPN, trained with data from MCTS, can estimate how much reward can be gained from taking a push action at a certain state.

For this combination of MCTS with PPN, some additional updates are made (compared to IV-B) to incorporate the guidance from PPN. The exploration term is removed from the search policy, so C in equation (1) is set to 0. Similar to [44], we use the estimated reward from PPN as a prior, so the Q-value is calculated as follows

$$Q_{guide}(s, a^p) = \frac{\max(Q_p(s)) + \sum_{i=1}^{m} r_i(s, a^p)}{N(s, a^p)}, \quad (3)$$

where m is set to 3 when expanding nodes and $N(s, a^p)$ is initialized to 1 for all state-action pairs. Instead of computing an average as standard MCTS, only best m of Q_p are considered, this is due to the number of rollout is small, a good action could be averaged out. To select the best action as the next step solution, the Q-value is calculated without the denominator

$$Q_{best}(s, a^p) = \max(G_p(s)) + \max(r_i(s, a^p)), \quad (4)$$

where only the best explored solution is considered.

The push action space of the guided MCTS is limited to a subset (like Fig. 2) so that the estimated reward from PPN is more accurate and the branching factor of the tree is of a reasonable size. To make the selection mimic the training data, we rotate the image for each sampled push action such that the push action in the rotated image is always pointing to the right. Then, we only use the estimated Q-value at the corresponding pixel (push action) of the output Q-value map. An example of guided MCTS is given in Fig. 4. The expansion of the tree is prioritized by PPN, where the push action with higher Q-value is sampled earlier, and the rollout policy is also prioritized. The maximum depth of the tree is limited to 3 instead of 4 as used in the earlier version of MCTS for collecting data to train PPN.

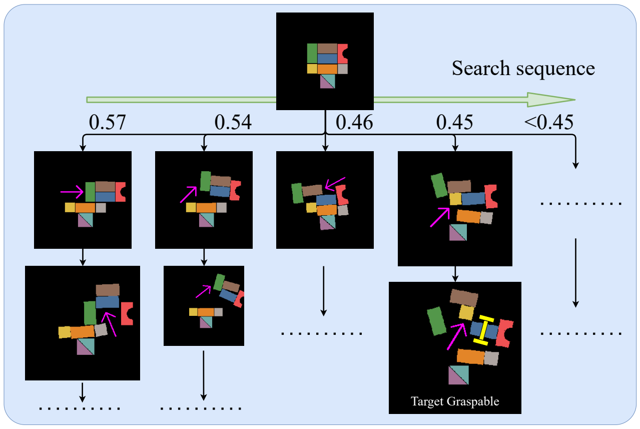

Fig. 4: An example of the guided MCTS with a budget of 10 iterations. State with larger image have higher estimated Q-values. All expanded nodes are plotted. The numbers in the first levels represent the estimated Q-value returned by PPN for corresponding push action. These values, together with the reward returned from simulation, guide the tree search.

V. Experimental Evaluation

We evaluated the proposed technique both in simulation (PyBullet [45]) and on adversarial test cases on a UR5e robot with a Robotiq 2F-85 gripper using real objects. The robot, workspace, objects, and camera are the same in simulation and real-world experiments, so that we can seamlessly transfer from simulation to the real setup. The workspace is limited to a square with a side length of 0.448m; it is discretized as a grid of 224 × 224 cells during the image processing step. The friction of objects and table cannot be accurately measured; nevertheless, high-fidelity physical properties do not seem to be needed for this particular application. The results demonstrate that the proposed method significantly outperforms MCTS [36] in terms of time efficiency while returning plans of equal quality. The plans returned by the proposed technique contain fewer actions and yield higher success rates than those returned by the purely learning-based solution presented in [35]. Training and evaluation are completed on a machine with an Intel i7-9700K CPU and an Nvidia GeForce RTX 2080 Ti.

A. Simulation experiments

Tasks. Given an arrangement of heterogeneous and tightly packed objects, a target object is to be retrieved using push and grasp actions from a two-finger gripper. In simulation, we benchmark on 22 adversarial test cases from [36] (Fig. 5) and 10 from [24], [35]. Here "adversarial" means that at least one push action has to be executed for a grasp action to be feasible (insert gripper without collision). Random cases, which are too easy [36], [39], are not discussed here.

Metrics. We use four metrics: 1) the number of actions used to retrieve the target object, 2) the total time used for retrieving the target object, which includes both planning time and execution time for simulation results, 3) the completion rate, failures occur when the target object is pushed out of the workspace, and 4), the grasp success rate, which is the number of successful grasps divided by the total number of grasping attempts. The number of re-arrangement actions that are needed to make the target object graspable and time are the two main metrics. The completion and grasp success rates are also reported but are not the main focus as they are often close to 100%.

Baseline Methods. We compare with three methods: 1) A self-supervised reinforcement learning method denoted as go-PGN [35], which trains a grasp DQN and a push DQN then selects an action with the highest Q-value out of the two networks to execute. 2) MCTS as described in Section IV-B. This is adapted from [36], but we use here a simulator to predict the next state instead of the originally used learned model, for fair comparisons. 3) PPN as described in Section IV-C. PPN proposes push actions based on their predicted Q-values and the robot executes those actions until the target object can be grasped according to GPN.

Simulation Studies. We ran our method and the three alternative methods on 22 cases [36] and 10 cases [24], [35], in simulation first. Tables I and II show the overall performance of the four methods, where MCTS based methods are limited to a budget of 50 iterations per test case. In this paper, we denote the tree search methods with different budgets of search iterations as MCTS-10/20/50 and MORE-10/20/50, where the suffix denotes the iterations limit. The 22 cases are generally harder to solve than the 10 cases, where the target object can be retrieved after one push action. The time metric records the average time (out of 5 trials) for retrieving the object, including planning and execution times.

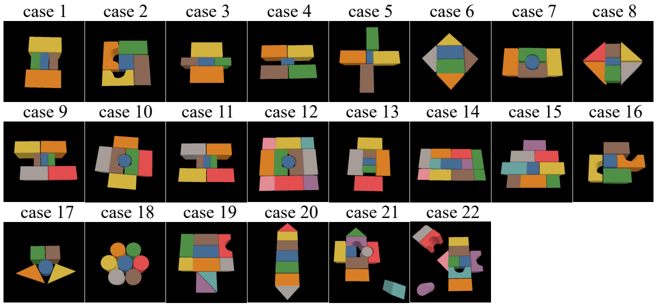

Fig. 5: 22 cases [36] used in simulation experiments, where the target object is masked in blue at the center.

For the baseline go-PGN, results on 10 cases are directly quoted from the paper (at the time of our submission, we could not obtain the trained model or the information necessary for fully reproducing go-PGN). MORE uses the fewest number of actions to solve the task. Performance details on 22 cases can be found in Fig. 6 for the number of actions and 7 for the running time. PPN is fast as it is a one-stage DNNs solution. It learned a policy that creates free spaces around the target object, but it is less consistent and less stable than the tree search solutions. From our observation, PPN can propose non-prevailing pushing actions. MCTS provides a consistent and good quality solution, but requires a much longer planning time. MORE, combining the benefits of both, reduces the planning time and delivers high-quality solutions.

	Num. of Actions	Time	Completion	Grasp Success
MORE-50	**2.61**	82s	100%	99.2%
MCTS-50 [36]	2.69	208s	100%	99.1%
PPN	3.68	8s	100%	97.7%

TABLE I: Simulate experiment results for 22 cases [36]. Budgets of MCTS and MORE are limited up to 50 iterations.

	Num. of Actions	Time	Completion	Grasp Success
MORE-50	**2.10**	16s	100%	100%
MCTS-50 [36]	2.20	32s	100%	93.4%
PPN	2.70	4s	100%	95.0%
go-PGN [35]	2.77		99.0%	90.0%

TABLE II: Simulate experiment results for 10 cases [35]. Budgets of MCTS and MORE are limited up to 50 iterations.

Ablation Studies Although the data generated by MCTS for training PPN is free because it is collected fully automatically in simulation, we set to explore data efficiency in training, which can be important for building larger models in practice. For this purpose, we collected 243 training cases (65384 transitions in 30 hours with PyBullet) with MCTS as

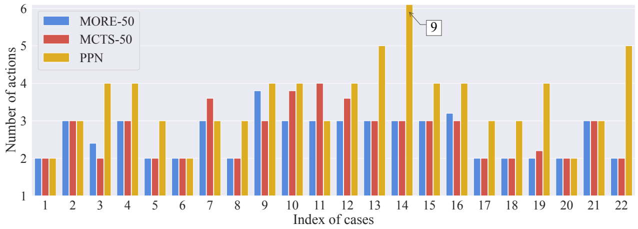

Fig. 6: The average number (out of 5 trials) of action used to solve one case for 22 cases.

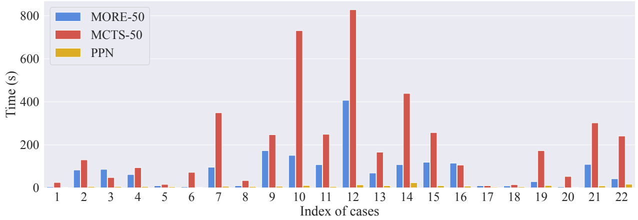

Fig. 7: The average time (of 5 trials) used to solve one case for 22 cases.

described in Section IV-B. Training on PPN on all data used around 22 hours. As shown in Fig. 8, we tested MCTS and MORE with different budgets. Also, MORE is trained on different numbers of training data. Clearly, the problem can be solved by all tested methods with fewer actions when the search iteration limits are increased. But the time for solving the problem also increases as a consequence. The proposed MORE technique can retrieve target objects with only 2.8 executed actions and using only 10 iterations of MCTS that last 36 seconds on average. This is close to the best that MCTS without PPN can achieve, 2.69 actions, after 50 iterations that last 208 seconds. When we limit the number of iterations of MCTS (without MORE) to 10, the number of executed actions increases to 3.19, and the search time remains relatively high (127 seconds). This clearly shows the out-performance of the proposed approach in terms of both time and action efficiency.

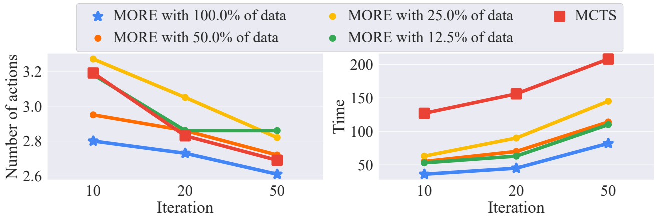

Fig. 8: Different amounts of training data are used to train PPN, which are evaluated on MORE with different budgets (iteration). This is the evaluation of the 22 cases.

B. Robot Experiments

We evaluated the four methods on six real test cases (four from [35] and two from [36]). These six test cases are representative in that they contain more objects and often require at least two push actions to solve. For these real experiments, the results are shown in Table. III and Fig. 10. The budget of MCTS and MORE is limited to 10 iterations. We note that the results for go-PGN are taken from [35]. The execution time of PPN is not listed in Table. III as it is a near-constant small value as we had in the simulation experiments. From the result, we observe only negligible performance degradation in comparison to simulation, which may be due to differences in friction, slight differences in the dimensions of the objects between simulation and real world, statistical error, or a combination of these. Overall, the sim-to-real transfer was very successful and showed that MORE can learn in simulation and directly apply the learned skill to real-world tasks. We assume models of objects are known, such that simple pose estimation can be used to locate objects in the real world and placed in simulation for planning. We could also use sophisticated tracking systems [46]–[48] for general purpose.

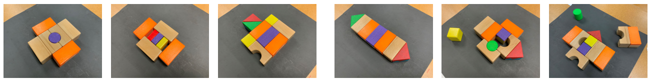

Fig. 9: Manually generated cases similar to [35], [36]. The target object is masked in purple. These cases are used also in simulation experiments as shown in Fig. 5.

	Num. of Actions	Time	Completion	Grasp Success
MORE-10	**2.83**	36s	100%	100%
MCTS-10 [36]	3.67	190s	100%	95.8%
PPN	3.72	3s	94.5%	95.8%
go-PGN [35]	4.62		95.0%	86.6%

TABLE III: Real experiment results for six cases as shown in Fig. 9. The budget of MCTS and MORE is limited to 10 iterations. For go-PGN, only the first four cases apply, and results are from [35]. Only planning time is recorded (robot execution was intentionally slowed down for safety). The computation time for PPN to solve a task is 3 seconds on average (estimated).

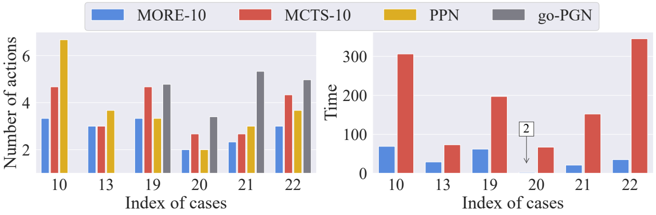

Fig. 10: The number of action and time used on solving six cases. The budget is up to 10 iterations for MCTS and MORE.

VI. Discussion and Conclusion

The main limitation of this work is that we need to know the models of the objects to do the planning. One possible solution is instead of using an explicit simulator, we can use a learned model [39] to simulate the push results. Generalization to novel objects could then be possible. We can further utilize the Push Prediction Network to estimate the simulation (rollout) result instead of using a physics engine. However, this can introduce additional uncertainties that typically result from using DNNs, which can cause unexpected behaviors such as pushing objects out of the workspace. Building on the know-hows gains from developing MORE, we are exploring other real-world robotic manipulation tasks that would benefit from the S2→S1 search-and-learn philosophy. We point out that MORE can be further sped up by implementing a parallel version of MCTS, as we only utilized a single CPU thread in our implementation and PPN (on GPU) is not being used most of the time.

References

[1] D. Kahneman, *Thinking, fast and slow*. Macmillan, 2011.

[2] Y. Bengio, "The consciousness prior," *arXiv preprint arXiv:1709.08568*, 2017.

[3] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[4] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, *et al.*, "A general reinforcement learning algorithm that masters chess, shogi, and go through self-play," *Science*, vol. 362, no. 6419, pp. 1140–1144, 2018.

[5] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, *et al.*, "Mastering atari, go, chess and shogi by planning with a learned model," *Nature*, vol. 588, no. 7839, pp. 604–609, 2020.

[6] 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," *CoRR*, vol. abs/1912.07024, 2019. [Online]. Available: http://arxiv.org/abs/1912.07024

[7] R. Boney, N. Di Palo, M. Berglund, A. Ilin, J. Kannala, A. Rasmus, and H. Valpola, "Regularizing trajectory optimization with denoising autoencoders," *Advances in Neural Information Processing Systems*, vol. 32, pp. 2859–2869, 2019.

[8] J. Bohg, A. Morales, T. Asfour, and D. Kragic, "Data-driven grasp synthesis—a survey," *Trans. Rob.*, vol. 30, no. 2, p. 289–309, Apr. 2014.

[9] A. Bicchi and V. Kumar, "Robotic grasping and contact: A review." Proceedings - IEEE International Conference on Robotics and Automation 1, 2000.

[10] H. Liang, X. Ma, S. Li, M. Görner, S. Tang, B. Fang, F. Sun, and J. Zhang, "PointNetGPD: Detecting grasp configurations from point sets," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2019.

[11] A. Rodriguez, M. T. Mason, and S. Ferry, "From caging to grasping," *The International Journal of Robotics Research*, vol. 31, no. 7, pp. 886–900, 2012.

[12] B. Calli, A. Singh, J. Bruce, A. Walsman, K. Konolige, S. Srinivasa, P. Abbeel, and A. M. Dollar, "Yale-cmu-berkeley dataset for robotic manipulation research," *The International Journal of Robotics Research*, vol. 36, no. 3, pp. 261–268, 2017.

[13] A. Boularias, O. Kroemer, and J. Peters, "Learning robot grasping from 3-d images with markov random fields," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2011, San Francisco, CA, USA, September 25-30, 2011*, 2011, pp. 1548–1553. [Online]. Available: http://dx.doi.org/10.1109/IROS.2011.6094888

[14] A. Mousavian, C. Eppner, and D. Fox, "6-dof graspnet: Variational grasp generation for object manipulation," in *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*. IEEE, 2019, pp. 2901–2910. [Online]. Available: https://doi.org/10.1109/ICCV.2019.00299

[15] C. Gabellieri, F. Angelini, V. Arapi, A. Palleschi, M. G. Catalano, G. Grioli, L. Pallottino, A. Bicchi, M. Bianchi, and M. Garabini, "Grasp it like a pro: Grasp of unknown objects with robotic hands based on skilled human expertise," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 2808–2815, 2020.

[16] Q. Lu, M. Van der Merwe, B. Sundaralingam, and T. Hermans, "Multifingered grasp planning via inference in deep neural networks: Outperforming sampling by learning differentiable models," *IEEE Robotics & Automation Magazine*, vol. 27, no. 2, pp. 55–65, 2020.

[17] A. Boularias, J. A. Bagnell, and A. Stentz, "Efficient optimization for autonomous robotic manipulation of natural objects," in *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada.*, 2014, pp. 2520–2526. [Online]. Available: http://www.aaai.org/ocs/index.php/AAAI/AAAI14/paper/view/8414

[18] 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," 2017.

[19] D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke, and S. Levine, "Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation," 2018.

[20] H.-S. Fang, C. Wang, M. Gou, and C. Lu, "Graspnet-1billion: A large-scale benchmark for general object grasping," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 11 444–11 453.

[21] B. Wen, W. Lian, K. Bekris, and S. Schaal, "Catgrasp: Learning category-level task-relevant grasping in clutter from simulation," *arXiv preprint arXiv:2109.09163*, 2021.

[22] J. Mahler, M. Matl, V. Satish, M. Danielczuk, B. DeRose, S. McKinley, and K. Goldberg, "Learning ambidextrous robot grasping policies," *Science Robotics*, vol. 4, no. 26, p. eaau4984, 2019.

[23] A. ten Pas, M. Gualtieri, K. Saenko, and R. P. Jr., "Grasp pose detection in point clouds," *CoRR*, vol. abs/1706.09911, 2017. [Online]. Available: http://arxiv.org/abs/1706.09911

[24] 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.

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

[26] A. Eitel, N. Hauff, and W. Burgard, "Learning to singulate objects using a push proposal network," in *Robotics Research*, N. M. Amato, G. Hager, S. Thomas, and M. Torres-Torriti, Eds. Cham: Springer International Publishing, 2020, pp. 405–419.

[27] M. Danielczuk, J. Mahler, C. Correa, and K. Goldberg, "Linear push policies to increase grasp access for robot bin picking," in *2018 IEEE 14th International Conference on Automation Science and Engineering (CASE)*, 2018, pp. 1249–1256.

[28] K. Gao, S. W. Feng, and J. Yu, "On minimizing the number of running buffers for tabletop rearrangement," in *Robotics: Sciences and Systems*, 2021.

[29] J. Yu, "Rearrangement on lattices with swaps: Optimality structures and efficient algorithms," in *Robotics: Sciences and Systems*, 2021.

[30] Y. Xiao, S. Katt, A. t. Pas, S. Chen, and C. Amato, "Online planning for target object search in clutter under partial observability," in *International Conference on Robotics and Automation*, 2019.

[31] M. Danielczuk, A. Kurenkov, A. Balakrishna, M. Matl, D. Wang, R. Martín-Martín, A. Garg, S. Savarese, and K. Goldberg, "Mechanical search: Multi-step retrieval of a target object occluded by clutter," *CoRR*, vol. abs/1903.01588, 2019. [Online]. Available: http://arxiv.org/abs/1903.01588

[32] T. Novkovic, R. Pautrat, F. Furrer, M. Breyer, R. Siegwart, and J. I. Nieto, "Object finding in cluttered scenes using interactive perception," *CoRR*, vol. abs/1911.07482, 2019. [Online]. Available: http://arxiv.org/abs/1911.07482

[33] A. Kurenkov, J. Taglic, R. Kulkarni, M. Dominguez-Kuhne, R. Martín-Martín, A. Garg, and S. Savarese, "Visuomotor mechanical search: Learning to retrieve target objects in clutter," in *IEEE/RSJ Int. Conference. on Intelligent Robots and Systems (IROS)*, 2020.

[34] R. Papallas and M. R. Dogar, "Non-prehensile manipulation in clutter with human-in-the-loop," *CoRR*, vol. abs/1904.03748, 2019. [Online]. Available: http://arxiv.org/abs/1904.03748

[35] K. Xu, H. Yu, Q. Lai, Y. Wang, and R. Xiong, "Efficient learning of goal-oriented push-grasping synergy in clutter," *IEEE Robotics and Automation Letters*, vol. 6, no. 4, pp. 6337–6344, 2021.

[36] B. Huang, S. D. Han, J. Yu, and A. Boularias, "Visual foresight trees for object retrieval from clutter with nonprehensile rearrangement," *IEEE Robotics and Automation Letters*, vol. 7, no. 1, pp. 231–238, 2021.

[37] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[38] T. Lin, P. Dollár, R. B. Girshick, K. He, B. Hariharan, and S. J. Belongie, "Feature pyramid networks for object detection," *CoRR*, vol. abs/1612.03144, 2016. [Online]. Available: http://arxiv.org/abs/1612.03144

[39] B. Huang, S. D. Han, A. Boularias, and J. Yu, "Dipn: Deep interaction prediction network with application to clutter removal," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 2021.

[40] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask r-cnn," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2961–2969.

[41] R. Coulom, "Efficient selectivity and backup operators in monte-carlo tree search," in *International conference on computers and games*. Springer, 2006, pp. 72–83.

[42] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[43] I. Loshchilov and F. Hutter, "Sgdr: Stochastic gradient descent with warm restarts," in *International Conference on Learning Representations ICLR*, 2017.

[44] J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, T. Pfaff, T. Weber, L. Buesing, and P. W. Battaglia, "Combining q-learning and search with amortized value estimates," in *International Conference on Learning Representations ICLR*, 2019.

[45] E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation for games, robotics and machine learning," http://pybullet.org, 2016–2021.

[46] B. Wen, C. Mitash, B. Ren, and K. E. Bekris, "se (3)-tracknet: Data-driven 6d pose tracking by calibrating image residuals in synthetic domains," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 10 367–10 373.

[47] B. Wen and K. Bekris, "Bundletrack: 6d pose tracking for novel objects without instance or category-level 3d models," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2021, pp. 8067–8074.

[48] C. Mitash, B. Wen, K. Bekris, and A. Boularias, "Scene-level pose estimation for multiple instances of densely packed objects," in *Conference on Robot Learning*. PMLR, 2020, pp. 1133–1145.