MultiPush: Learning to Rearrange with Teams of Car-Like Pushers

Jeeho Ahn and Christoforos Mavrogiannis

***Abstract*— We focus on the problem of rearranging multiple objects within a constrained workspace via pushing using a team of car-like robots. While the use of multiple robots offers the potential for more efficient execution, the need for conflict resolution and the kinematic constraints arising from physics, robot design, and the workspace boundary make this problem especially challenging. Our key insight is that by exploiting the structure introduced by the car-like kinematics of the domain, we could relax the problem into an ordered assignment of Dubins curves to robots. To this end, we introduce *MultiPush*, a reinforcement-learning based framework that jointly determines an efficient schedule of pushing tasks and their allocation to available robots. In simulation with up to 14 objects and four robots, MultiPush achieves a lower makespan and a higher success rate than the baselines while planning the fastest. We demonstrate MultiPush on hardware with two and three robots rearranging 12 objects.**

I. Introduction

We study the model problem of rearranging multiple objects in a confined workspace using a team of car-like robots that manipulate objects via pushing rather than grasping. Under the quasistatic regime, a pushing car-like robot must steer within a very limited range to keep stable contact with the manipulated object [6], and a team sharing a constrained workspace further requires a schedule that avoids conflicts. Prior work has either considered a single robot [1] or restricted multirobot pushing to monotone instances in discretized spaces, as in PuSHR [9].

Our key insight is that the structure imposed by car-like constraints during transit and transfer relaxes the planning problem from a search over the continuum of multirobot paths into an ordered assignment of Dubins curves [4] to robots. To this end, we introduce *MultiPush*, which determines the schedule of the pushing tasks and their allocation jointly via reinforcement learning. The learned policy schedules without invoking the motion planner and generalizes to robot counts and a workspace unseen in training.

II. Problem Statement

We consider n identical car-like robots with flat bumpers that must rearrange m rigid, polygonal objects from their initial to their goal poses in a confined workspace along collision-free trajectories. The workspace with the robots' initial poses and the objects' initial and goal poses defines an *instance*. Rearranging object j is a *task* τ_j, executed by a single robot, and a rearranged object becomes static. An *assignment* pairs a task with the robot that executes it, and the m assignments in execution order form the *assignment sequence* σ. We seek the assignment sequence and trajectories of minimum *makespan*.

The authors are with the Department of Robotics, University of Michigan, Ann Arbor, USA. Email: {jeeho, cmavro}@umich.edu

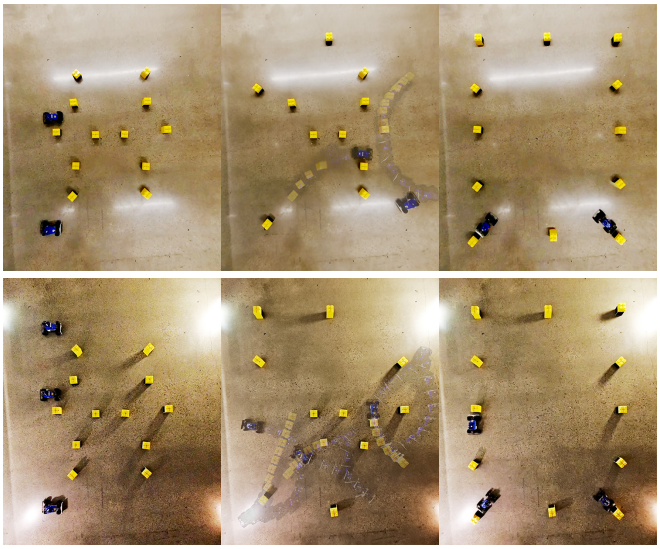


Fig. 1: Teams of two robots (top) and three robots (bottom) rearrange the same 12 objects by pushing, following the schedule and trajectories planned by *MultiPush*.

III. MultiPush

Task representation. Executing a task involves *transit* paths, along which the robot moves without pushing, and *transfer* paths, along which it pushes the object; the optimal paths are Dubins curves for both. ReloPush-BOSS [1] searches a push-traversability graph whose vertices are *pushing poses*, robot poses in contact with objects at their initial and goal poses, and whose edges are the Dubins transfers between pushing poses. Its search yields a nominal sequence σ_0 rearranging all objects with one robot, which we take as a *seed* fixing every transfer path.

Assignment sequence as a Markov decision process. We cast the construction of σ as an episodic Markov decision process with horizon m. The state s_h holds the assignments made before step h, and the action $a_h = (\tau_j, i)$ assigns task τ_j to robot i. An action is admissible only when its task is unassigned and every task ordered earlier in σ_0 whose object blocks its transfer path is already assigned. If σ is feasible, the reward is the negative makespan of the planned trajectories, normalized per instance by that of GREEDY (Sec. IV); otherwise it is a penalty below every feasible reward, ranked by how far σ progresses before failing.

Learning an assignment policy. Since an instance admits up to $m!\,n^m$ assignment sequences and evaluating one requires planning all robots' trajectories, we train a stochastic policy π_θ by reinforcement learning. While σ is being filled, a *pseudo-schedule* estimates each candidate task's completion time and its robot's next availability from the seed's transit times and task durations, without planning

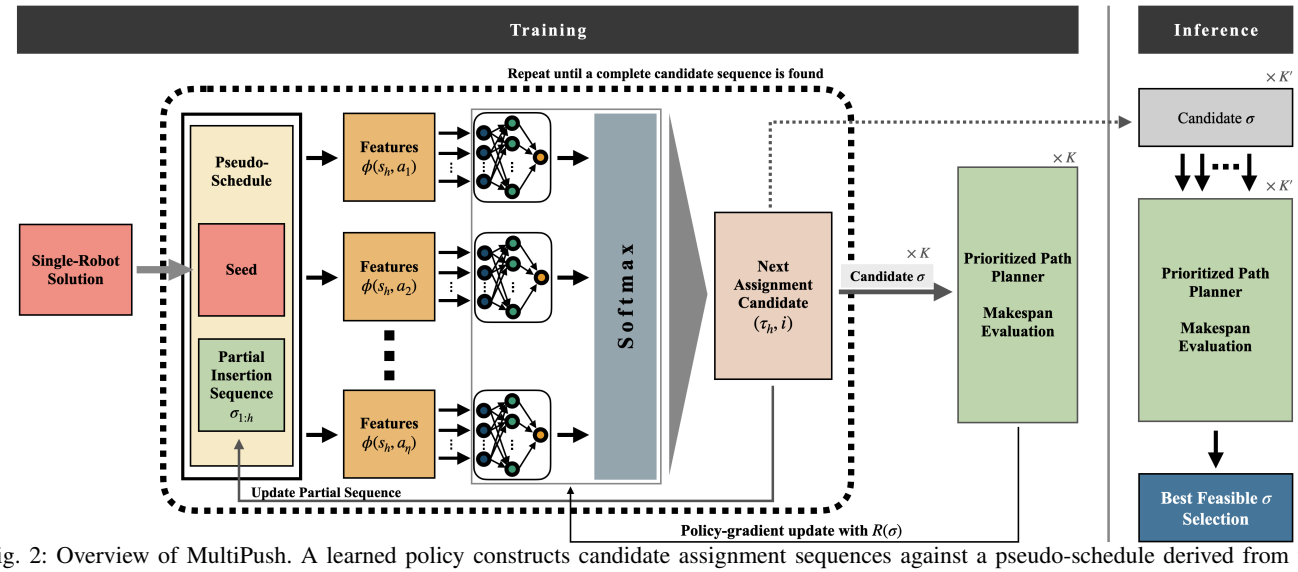


Fig. 2: Overview of MultiPush. A learned policy constructs candidate assignment sequences against a pseudo-schedule derived from the single-robot seed, and the prioritized planner verifies them.

		$4.5 \times 5.5\,\mathrm{m}^2$ (trained)			$5.0 \times 5.0\,\mathrm{m}^2$ (unseen)		
n	Method	Rel. MS	Time (s)	Succ. (%)	Rel. MS	Time (s)	Succ. (%)
2	GREEDY	1.000	–	96.0	1.000	–	93.0
	LNS	0.895	21.9	96.0	0.900	19.5	93.0
	MultiPush	**0.872**	**8.7**	**97.5**	**0.875**	**8.6**	**96.2**
3	GREEDY	1.000	–	95.5	1.000	–	92.2
	LNS	0.879	32.2	95.5	0.881	30.6	92.2
	MultiPush	**0.843**	**12.5**	**98.0**	**0.847**	**13.0**	**97.0**
4	GREEDY	1.000	–	95.0	1.000	–	92.2
	LNS	0.866	43.9	95.0	0.869	41.2	92.2
	MultiPush	**0.840**	**17.1**	**97.5**	**0.834**	**18.1**	**97.5**

TABLE I: Relative makespan (Rel. MS, against GREEDY, averaged over the instances solved by every method), planning time, and success rate on 400 test instances per workspace.

any path. A feature vector $\phi(s_h, a_h)$ computed from the seed and the pseudo-schedule measures how much a candidate extends the schedule, the balance of availabilities it leaves, and the risk of leaving the remaining tasks infeasible. A multilayer perceptron scores $\phi(s_h, a_h)$, and a softmax over the admissible candidates gives $\pi_\theta(a_h \mid s_h)$. The features carry no entry per robot, so the policy applies to any number of robots. Training uses REINFORCE [10] on the terminal reward, with a candidate's advantage taken relative to the mean reward of K candidates for the same instance [5]. At inference, the policy samples K' candidates and the verified plan of minimum makespan is executed (Fig. 2).

Prioritized multirobot path planning. Given σ, trajectories are planned one at a time in its order via Hybrid A^* [3], each plan treating already planned trajectories, recorded in a centralized space-time timetable, as dynamic obstacles [2]. Conflicts are resolved by waiting for a blocking robot or object to clear the path in bounded time, or by moving an idle blocking robot to a safe parking location before replanning. If neither succeeds, σ is infeasible.

IV. Evaluation

We evaluate MultiPush on $m \in \{8, 10, 12, 14\}$ cubic objects with two to four MuSHR [8] robots in a $4.5 \times 5.5\,\mathrm{m}^2$ workspace and a $5.0 \times 5.0\,\mathrm{m}^2$ workspace unseen in training, the policy being trained in the former with three robots only. Each method evaluates $K' = 20$ candidate sequences per instance with the same planner. *GREEDY* processes the tasks in the order of σ_0, assigning each to the earliest available robot. *LNS* refines GREEDY by Large Neighborhood Search [7], removing one or two assignments per candidate and reinserting them. *MultiPush* samples its candidates from the policy and evaluates GREEDY in addition, as LNS does.

Table I reports the results. MultiPush attains a lower relative makespan than LNS in every workspace and robot count, and LNS requires 2.3 to 2.6 times its planning time. A baseline sampling candidates uniformly at random, not shown, does not improve on GREEDY, so the gains follow from the policy rather than the budget. The advantage holds in the unseen workspace and with two and four robots, so the policy generalizes beyond its training conditions.

References

[1] J. Ahn and C. Mavrogiannis. ReloPush-BOSS: Optimization-guided nonmonotone rearrangement planning for a car-like robot pusher. *IEEE Robotics and Automation Letters*, 11(4), 2026.

[2] M. Čáp, P. Novák, A. Kleiner, and M. Selecký. Prioritized planning algorithms for trajectory coordination of multiple mobile robots. *IEEE transactions on automation science and engineering*, 12(3):835–849, 2015.

[3] D. Dolgov, S. Thrun, M. Montemerlo, and J. Diebel. Practical search techniques in path planning for autonomous driving. *ann arbor*, 1001(48105):18–80, 2008.

[4] L. E. Dubins. On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents. *American Journal of mathematics*, 79(3):497–516, 1957.

[5] Y.-D. Kwon, J. Choo, B. Kim, I. Yoon, Y. Gwon, and S. Min. Pomo: Policy optimization with multiple optima for reinforcement learning. *Advances in neural information processing systems*, 33:21188–21198, 2020.

[6] K. M. Lynch and M. T. Mason. Stable pushing: Mechanics, controllability, and planning. *The international journal of robotics research*, 15(6):533–556, 1996.

[7] P. Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In *International conference on principles and practice of constraint programming*, pages 417–431. Springer, 1998.

[8] S. S. Srinivasa, P. Lancaster, J. Michalove, M. Schmittle, C. Summers, M. Rockett, R. Scalise, J. R. Smith, S. Choudhury, C. Mavrogiannis, et al. Mushr: A low-cost, open-source robotic racecar for education and research. *arXiv preprint arXiv:1908.08031*, 2019.

[9] S. Talia, A. Thareja, C. Mavrogiannis, M. Schmittle, and S. S. Srinivasa. Pushr: A multirobot system for nonprehensile rearrangement. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5380–5387. IEEE, 2023.

[10] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256, 1992.