Parallel Heuristic Search as Inference for Actor-Critic Reinforcement Learning Models (Extended Abstract)

Itamar Mishani[1,*], **Hanlan Yang**[1,2,*], **Luca Pivetti**[1,3], **Zachary Kingston**[2], **Maxim Likhachev**[1]
[1]Carnegie Mellon University
[2]Purdue University
[3]University of Milano-Bicocca
{imishani, lpivetti, maxim}@cmu.edu, {yang3034, zkingston}@purdue.edu

Introduction

Reinforcement learning (RL) has become a central paradigm for robot control, enabling behaviors that are difficult to manually specify. However, actor-critic RL models are typically deployed as one-step predictors during execution, discarding the critic network after training and lacking multi-step forward reasoning or backtracking capabilities. While much research has focused on improving training, comparatively little attention has been given to inference strategies.

The integration of learning and search has been demonstrated in game-playing domains such as AlphaGo (Silver et al. 2016), which combines policy and value networks with Monte Carlo Tree Search, and DeepCubeA (Agostinelli et al. 2019), which uses learned heuristics for combinatorial puzzles. However, these approaches operate in discrete state and action spaces with fast state transitions, and do not address the computational challenges of robotic planning where edge evaluations (physics simulation, collision checking) dominate planning time.

We present PACHS (Parallel Actor-Critic Heuristic Search), an efficient parallel best-first search algorithm that leverages both components during inference: the actor generates candidate actions while the critic provides learned cost-to-go estimates to guide the search. The key observation is that the critic's Q-function, when trained with negated costs as rewards, directly provides an action-conditioned heuristic: $h(s, a) = -Q_\phi(s, a)$. For clarity, we henceforth treat $Q_\phi(s, a)$ as approximating expected cumulative cost. This equivalence enables multi-level parallelization—batching neural network calls on GPU and parallelizing edge expansions across CPU threads—making search-based inference tractable for real-time robotics. Experiments on collision-free motion planning and contact-rich pushing tasks demonstrate that PACHS achieves higher success rates and better generalization to out-of-distribution scenarios.

Approach

PACHS integrates Soft Actor-Critic (SAC) models within a best-first search framework inspired by ePA*SE (Mukherjee, Aine, and Likhachev 2022). Given a trained actor π_θ and critic Q_ϕ, we formulate planning as graph search where vertices represent states and edges represent state-action pairs.

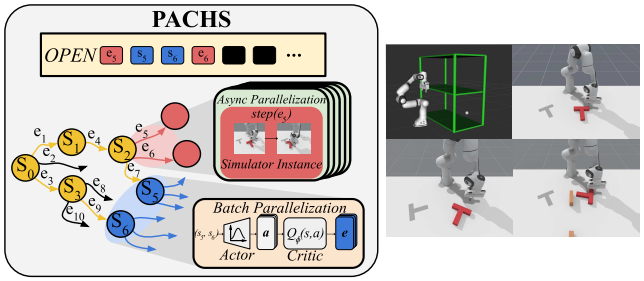

Figure 1: Left: PACHS performs best-first search using actor-critic networks, with multi-level parallelization. Yellow elements indicate completed expansions. The OPEN list stores edge candidates ordered by priority. Red edges (e_5, e_6) are expanded and evaluated in parallel across threads, while blue states (s_5, s_6) generate actions and heuristics in batches. Right: Experimental environments—collision-free motion planning in a shelf and contact-rich PushT manipulation.

Edge Prioritization with Learned Heuristics. In robotic planning, edge evaluation (physics simulation, collision checking) dominates computation time. PACHS maintains a priority queue of edges rather than states, enabling multiple edges to be evaluated in parallel before their successor states are known. Each edge $e = (s, a)$ is prioritized by:

$$f(e) = g(s) + w \cdot Q_\phi(s, a)$$

where $g(s)$ is the cost-to-come and w is a weight parameter, and the critic provides action-specific cost-to-go estimates, allowing the search to distinguish between actions from the same state.

Actor-Guided Action Generation. When expanding a state, we sample a batch of actions from the actor's distribution $\pi_\theta(a|s)$ rather than enumerating a discrete action space, replacing hand-crafted motion primitives with learned actions.

Multi-Level Parallelization. PACHS achieves efficiency through two parallelization strategies (Figure 1): (1) GPU batch-level parallelism for actor action generation and critic Q-value computation, and (2) CPU thread-level parallelism for edge expansion and evaluation (physics simulation,

*Equal contribution.

Algorithm 1 PACHS Worker Thread (runs in parallel)

while goal not reached and budget remaining **do**
  lock: pop (s, a) from OPEN; **if** $a = a^d$ (dummy) it's expansion
  if expansion edge (s, a^d) **then**
    $\{a_i\} \leftarrow \pi_\theta(\cdot|s)$; $\{Q_i\} \leftarrow Q_\phi(s, a_i)$ *// GPU batch*
    lock: insert (s, a_i) with $f = g(s) + w \cdot Q_i$; mark s closed
  else {evaluation edge (s, a) with stored Q}
    $s', c \leftarrow \text{EVALUATE}(s, a)$ *// lock-free, parallel*
    lock: **if** $s' \notin \text{CLOSED} \wedge g(s') > g(s) + c$:
      $g(s') \leftarrow g(s) + c$; insert (s', a^d) with $f = g(s') + w(Q - c)$
  end if
end while

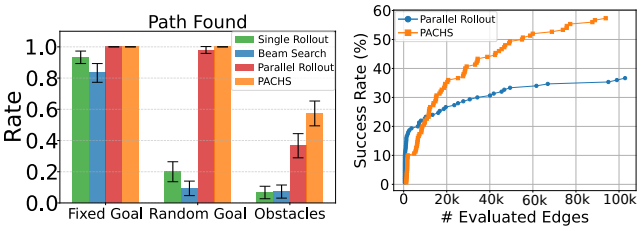

Figure 2: Left: Success rates on PushT tasks (100k budget). Right: Success rate vs. evaluation budget—PACHS continues improving while parallel rollout plateaus.

collision checking). Worker threads process edges asynchronously, with locks only for data structure updates while expensive operations run lock-free.

Algorithmic Structure. The algorithm maintains two edge types in its priority queue: (1) *expansion edges* (s, a^d) using a dummy action a^d as a marker for states ready for action generation, and (2) *evaluation edges* (s, a) representing state-action pairs awaiting evaluation (Algorithm 1). When an evaluation edge yields successor state s' with cost c, the stored Q-value is adjusted by c to estimate s''s priority. States are marked CLOSED after expansion to prevent redundant work.

Experiments

We evaluated PACHS on a 7-DoF Franka Panda arm across two domains (Figure 2): collision-free motion planning in a shelf environment and contact-rich PushT manipulation tasks. SAC models were trained using ManiSkill (Mu et al. 2021). Experiments were conducted on an Intel i7-11800H CPU with NVIDIA RTX 3070 GPU.

Baselines. We compare against: (1) *Single rollout*—sequential policy execution; (2) *Parallel rollout*—batch rollouts with restarts on failure; (3) *Beam search*—parallel beam search using Q-values for selection; and (4) *ePA*SE* with hand-crafted heuristics (for motion planning only).

Collision-Free Motion Planning. In the Panda-Shelf task, the robot must move its end-effector to target poses within a shelf. Both PACHS and ePA*SE achieve 100% success rates across 100 test instances, while parallel rollout (54%) and beam search (39%) show significantly lower performance. Despite neural network overhead, PACHS matches ePA*SE's planning time and solution quality. Importantly, for identical numbers of expanded nodes, PACHS evaluates $\sim 10\times$ fewer edges, demonstrating that critic-guided prioritization reduces computational waste.

PushT Manipulation. The robot must push a T-shaped object to a target pose (90% overlap required). We tested three variants with a 100k edge-evaluation budget: In fixed goal, single rollout achieves 93% while PACHS achieves 100%. In random goal, single rollout drops to 20% while PACHS maintains 100%. In the obstacle scenario, where we added obstacles to the environment and tested the model trained without them, single rollout achieves only 7%, parallel rollout 37%, and PACHS 57%.

Closed-Loop Execution. In replanning experiments (3-second budgets, 10-action horizons), PACHS achieves 100%, 98%, and 37% success rates on fixed goal, random goal, and obstacle tasks respectively, compared to parallel rollout's 87%, 80%, 21% and single rollout's 90%, 22%, 10%. While parallel rollout's performance drops from open-loop to closed-loop in sim-to-sim settings, PACHS maintains similar performance—suggesting it may also improve the sim-to-real transfer.

Conclusion

PACHS demonstrates that leveraging both actor and critic networks during inference, combined with parallel best-first search, enables zero-shot generalization without model retraining. Our results show substantial improvements in success rates and robustness, particularly in environments with obstacles not encountered during training. The multi-level parallelization strategy makes search-based inference practical for real-time robotic applications.

Key advantages include: (1) utilizing the full actor-critic model rather than discarding the critic, (2) enabling backtracking unavailable in standard rollouts, and (3) zero-shot adaptation to novel constraints without retraining. Future work includes physical robot deployment and integration with world models.

References

Agostinelli, F.; McAleer, S.; Shmakov, A.; and Baldi, P. 2019. Solving the Rubik's Cube with Deep Reinforcement Learning and Search. *Nature Machine Intelligence*, 1(8): 356–363.

Mu, T.; Ling, Z.; Xiang, F.; Yang, D.; Li, X.; Tao, S.; Huang, Z.; Jia, Z.; and Su, H. 2021. ManiSkill: Generalizable Manipulation Skill Benchmark with Large-Scale Demonstrations. *arXiv preprint arXiv:2107.14483*.

Mukherjee, S.; Aine, S.; and Likhachev, M. 2022. ePA*SE: Edge-Based Parallel A* for Slow Evaluations. In *Proceedings of the International Symposium on Combinatorial Search*, volume 15, 136–144.

Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; Van Den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; et al. 2016. Mastering the Game of Go with Deep Neural Networks and Tree Search. *Nature*, 529(7587): 484–489.