GPU-Accelerated A* Search with Deep Neural Network Heuristics

Misagh Soltani Forest Agostinelli

msoltani@email.sc.edu foresta@cse.sc.edu

Dept. of Computer Science & Engineering
University of South Carolina

The 19th International Symposium on Combinatorial Search

Presenter: Rojina Panta

August 15, 2026

Molinaroli College of Engineering and Computing
UNIVERSITY OF SOUTH CAROLINA

Code on GitHub

Outline

Learned heuristics move the cost of search

- Deep reinforcement learning can represent a heuristic function as a deep neural network with no domain-specific heuristic knowledge, as DeepCubeA [1] showed for the Rubik's Cube.
- Once the heuristic is a neural network, the computational bottleneck often shifts from node expansion to heuristic computation.
- Batched search answers this by exposing more parallel work per iteration, so many successors are evaluated in a single forward pass.

Batched expansion strategies

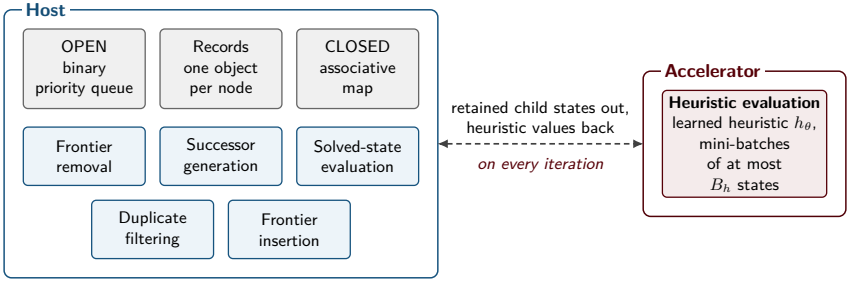
- K-Best-First Search [2] expands the top K nodes of the open list per iteration.
- Batch A* [3] and Batch Weighted A* remove K nodes from OPEN each cycle and issue all successor evaluations in one batched inference step.

[1] Forest Agostinelli et al. "Solving the Rubik's cube with deep reinforcement learning and search". In: *Nature Machine Intelligence* 1.8 (2019), pp. 356–363

[2] Ariel Felner, Sarit Kraus, and Richard E Korf. "KBFS: K-best-first search". In: *Annals of Mathematics and Artificial Intelligence* 39.1 (2003), pp. 19–39

[3] Tianhua Li et al. "Optimal search with neural networks: Challenges and approaches". In: *Proceedings of the International Symposium on Combinatorial Search*. Vol. 15. 1. 2022, pp. 109–117

Batching the heuristic leaves the rest of the search on the host



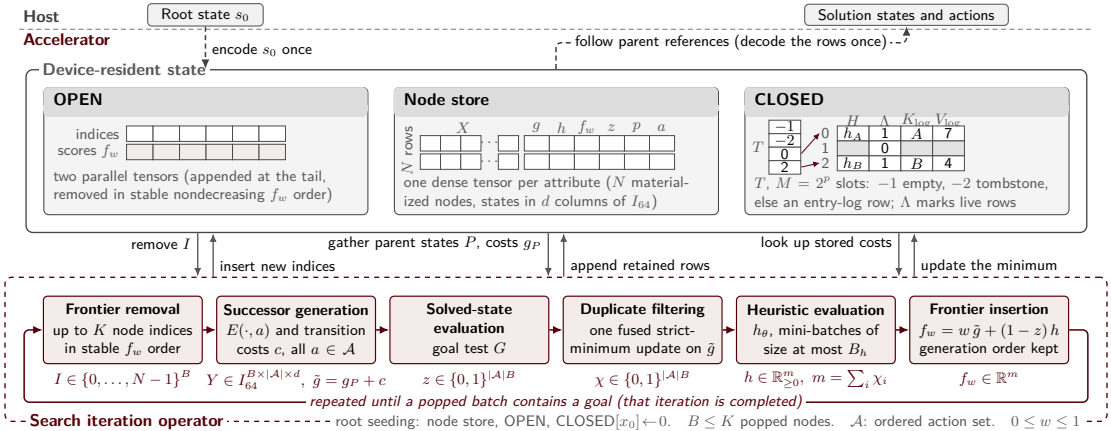
- The accelerator evaluates the heuristic and nothing else, so retained child states leave the host and heuristic values return on every iteration.
- In our host-resident implementation successor generation takes 53.52% of measured iteration time against 12.70% for heuristic evaluation.

Question

What if we perform the whole of batch weighted A* search on the accelerator (e.g., GPU)?

Our Method: GPU-Resident Batch Weighted A* Search ($\mathrm{BWAS_{GPU}}$)

- $\mathrm{BWAS_{GPU}}$ keeps the ordered batch semantics of BWAS, executed on the accelerator.



X, g, h, f_w, z, p, a: state, path cost, heuristic, score, goal flag, parent, action. E: encoded transition. χ: acceptance mask.

Ordered batch semantics

- A node stores a state, a path cost g, a heuristic value h, a goal indicator z, a parent reference, and the generating action.
- We reserve classical Weighted A* [4] for f_{WA} and rank nodes by the weighted score f_w instead, where $1 - z(n)$ sets the heuristic contribution of a goal node to zero.
- K bounds the nodes removed per iteration and B_h the states in one pass.

Ranking

$$f_{\mathrm{WA}}(n) = g(n) + \omega\, h(n), \quad \omega \geq 1$$

$$f_w(n) = w\, g(n) + \big(1 - z(n)\big)\, h(n)$$

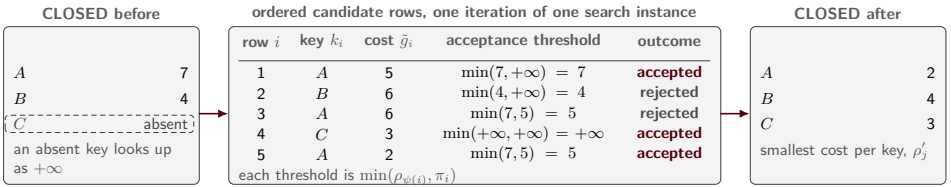
$$\text{with } 0 \leq w \leq 1$$

Termination

Search stops only after an iteration whose popped batch contains a goal node. Among the goals popped in that batch, the returned solution has minimum path cost.

[4] Ira Pohl. "Heuristic search viewed as path finding in a graph". In: *Artificial Intelligence* 1.3-4 (1970), pp. 193–204

One fused pass filters duplicates and updates CLOSED



prior min prefix min

$\underbrace{\chi_i = 1}_{\text{accept}} \iff \underbrace{\tilde{g}_i}_{\text{cand. cost}} < \underbrace{\min(\overbrace{\rho_{\psi(i)}}^{\text{prior min}}, \overbrace{\pi_i}^{\text{prefix min}})}_{\text{threshold}}$

$\underbrace{\rho'_j}_{\text{updated}} = \min(\underbrace{\rho_j}_{\text{prior stored}}, \underbrace{\min_{i:\psi(i)=j} \tilde{g}_i}_{\text{batch min}})$

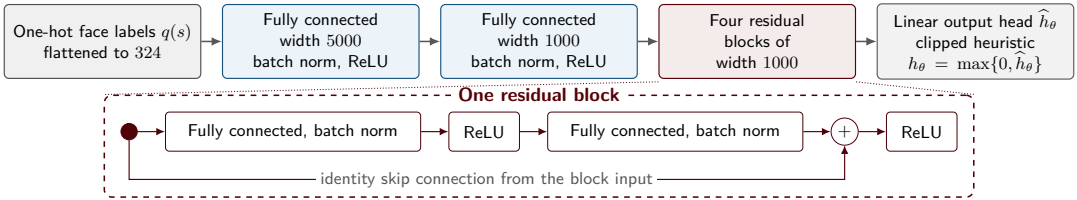
$\tilde{g}_i$: candidate path cost $g_P + c$. $k_i \in I^d_{64}$: encoded state. χ: acceptance mask.
$\psi(i)$: distinct-key index of row i. ρ_j: stored value for key, $+\infty$ if absent.

$\pi_i = \min \left\{ \tilde{g}_j : \underbrace{j < i}_{\text{earlier}}, \underbrace{k_j = k_i}_{\text{same key}} \right\}$, with $\min \emptyset = +\infty$.

Why one pass matters

CLOSED is an ordered partial map from encoded states to best known path costs, held in an address table whose live entries point into an append-only entry log. One device-resident pass combines non-improvement filtering, within-batch strict-improvement filtering, and CLOSED maintenance, so no state leaves the accelerator during duplicate filtering.

Domain, learned heuristic, and evaluation protocol

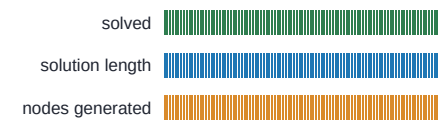


- The study is instantiated on the $3 \times 3 \times 3$ Rubik's Cube, where each of the twelve quarter turns permutes the 54 coordinates at unit cost.
- The heuristic operates on face labels rather than on raw sticker identifiers: $q_i(s) = \lfloor x_i(s)/9 \rfloor$, where $x_i(s) \in \{0, \ldots, 53\}$ is the sticker identifier at coordinate i.
- The component sweeps reuse one captured search snapshot, so both variants see the same input sequence.
- Both variants share the transition function, action ordering, and network weights, and use $w = 0.6$, $K = 10^4$, and $B_h = 10^4$ on the first 100 instances of the test set released with DeepCubeA and DeepCubeAI [5].
- All accelerator measurements use one NVIDIA H200 GPU, and the baseline evaluates the heuristic on that same GPU.

The same search outcomes in 12.03× less time

Variant	Mean (s)	Std. (s)	Min (s)	Max (s)	Solved	Soln. len.	Nodes/sec
Host-resident	71.37	51.02	8.43	267.0	100/100	21.28	97,990
Accelerator-resident	**5.931**	4.261	1.53	24.86	100/100	21.28	**1,179,067**

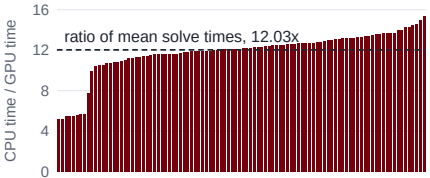
Outcome equivalence across the 100 test states



100 of 100 solved by both variants

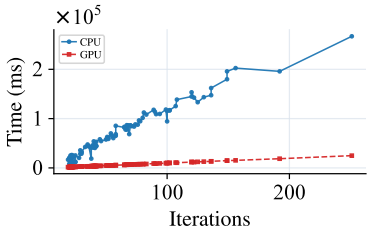
identical solution lengths and identical generated-node counts

Per-state speedup over the 100 test states

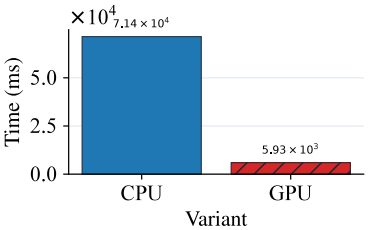


Both variants expand nodes in the same order and generate approximately 6.99×10^8 nodes and use the same 6,216 total iterations, while throughput rises from 9.80×10^4 to 1.18×10^6 nodes per second.

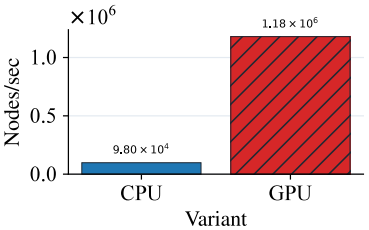
Faster in aggregate and across the whole difficulty range



Per-instance runtime by iteration count.



Average solve time over the test states.

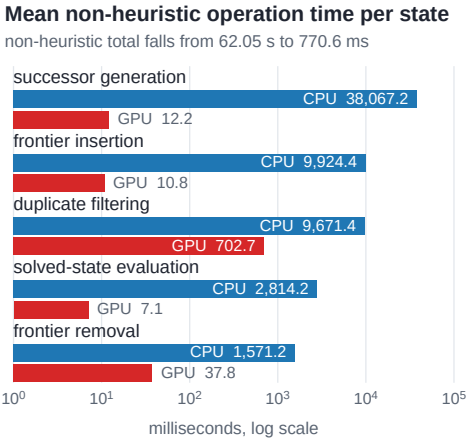


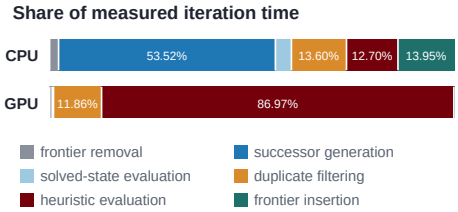
Average node generation throughput.

Reading

Mean solve time falls by 12.03× and node generation throughput rises by about the same factor, and the per-instance advantage holds across the whole problem difficulty range with no crossover.

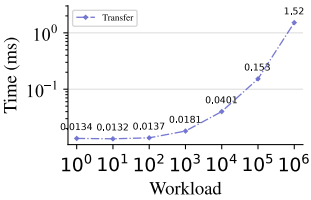
Where the iteration time goes



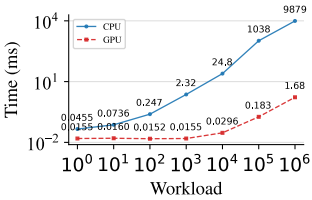


Total non-heuristic time drops by 80.5×, from 62.05 s to 770.6 ms per test state. Heuristic evaluation now accounts for 86.97% of iteration time and duplicate filtering for 11.86%, with every other stage below 1%. Duplicate filtering is the largest non-heuristic residual at 702.7 ms.

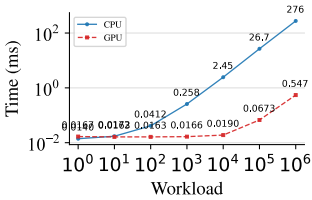
Batched stages scale strongly



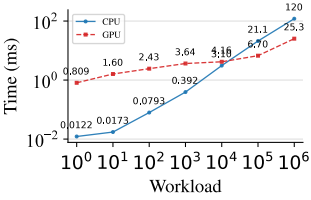
Host-to-device transfer.



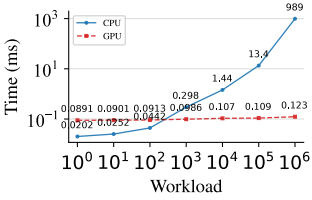
Successor generation.



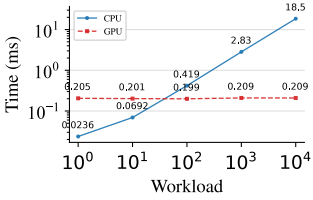
Solved-state evaluation.



Duplicate filtering, CLOSED size 10^6.

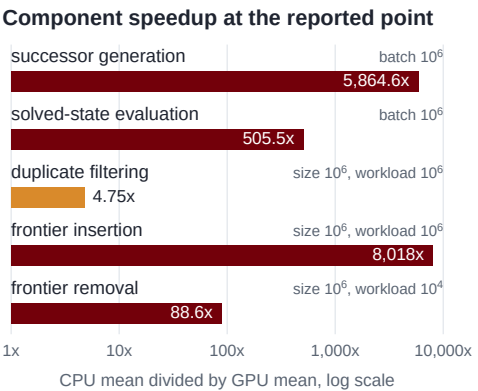


Frontier insertion, OPEN size 10^6.



Frontier removal, OPEN size 10^6.

Component summary



Stage	Point	CPU (ms)	GPU (ms)	Speedup
Succ. gen.	batch 10^6	9879.3	1.685	5864.6×
Solved-state ev.	batch 10^6	276.3	0.547	505.5×
Dup. filter	sz 10^6, wl 10^6	120.1	25.30	4.75×
Frontier ins.	sz 10^6, wl 10^6	989.2	0.123	8018×
Frontier rem.	sz 10^6, wl 10^4	18.51	0.209	88.6×

The largest gains occur in the regular batched stages and in the large-workload OPEN operations, while CLOSED duplicate filtering improves only in the largest combined structure-and-workload regime.

Takeaways

- Moving the full batched-search pipeline onto the accelerator yields a 12.03× end-to-end gain without changing the observed search behavior, so the gain comes from where the same computation is performed.
- Regular batched operators and large-workload OPEN operations become highly accelerator-favorable, whereas CLOSED duplicate filtering remains the largest bottleneck besides neural network heuristic evaluation.
- Next steps are Just-in-Time compilation of the pipeline, multi-accelerator scaling (distributed across accelerators), and hybrid architectures for dynamic load balancing.

Code on GitHub

`github.com/misaghsoltani/GPUAStar`

References

[1] Forest Agostinelli et al. "Solving the Rubik's cube with deep reinforcement learning and search". In: *Nature Machine Intelligence* 1.8 (2019), pp. 356–363.

[2] Ariel Felner, Sarit Kraus, and Richard E Korf. "KBFS: K-best-first search". In: *Annals of Mathematics and Artificial Intelligence* 39.1 (2003), pp. 19–39.

[3] Tianhua Li et al. "Optimal search with neural networks: Challenges and approaches". In: *Proceedings of the International Symposium on Combinatorial Search*. Vol. 15. 1. 2022, pp. 109–117.

[4] Ira Pohl. "Heuristic search viewed as path finding in a graph". In: *Artificial Intelligence* 1.3-4 (1970), pp. 193–204.

[5] Forest Agostinelli and Misagh Soltani. "Learning discrete world models for heuristic search". In: *Reinforcement Learning Conference*. 2024.

Thank you for your attention!

Any questions?

This material is based upon work supported by the National Science Foundation under Award No. 2426622. The authors gratefully acknowledge the computational resources provided by the Theia high performance computing cluster at the University of South Carolina, which is supported by National Science Foundation Grant No. 2320292. We also acknowledge the technical assistance and resources provided by Research Computing at the University of South Carolina (RRID:SCR_027488).

Code on GitHub