A* vs. IDA* on N/Rectangular Sliding Puzzles

Authors: Eran Fishbein and Danielle Eldor (submitted in pairs).

Abstract. We empirically compare A* and IDA* on 3×3, 3×4, 3×5, and 4×4 sliding-tile puzzles under admissible Manhattan and Linear-Conflict heuristics, with and without BPMX. Our instrumented Python implementations report runtime, node counts, duplicates, and memory proxies. IDA* dominates at moderate depths on all boards; on the 15-puzzle a crossover appears around depth ~14–18 where A* matches or overtakes IDA*, consistent with duplicate-pruning benefits when memory suffices. BPMX shows no speedup at our depths in Python. We conclude with guidance on when to prefer each algorithm and discuss extensions.

1) Introduction

We study informed search on sliding-tile puzzles, focusing on A* and IDA. *A* maintains OPEN/CLOSED and avoids re-expansions via a best-first frontier; IDA* performs iterative deepening on (f=g+h) and keeps only a DFS stack (memory-lean but with some re-expansions). We use admissible heuristics (Manhattan; Linear Conflict) and, for IDA*, test BPMX (bidirectional pathmax), which can increase h-values along edges to tighten f-bounds.

Scope & coverage

Boards: **3×3 (8-puzzle), 3×4, 3×5, 4×4 (15-puzzle)**.
Heuristics: **Manhattan, Linear Conflict**.
Algorithms: **A***, **IDA*** (± **BPMX**), plus **BFS/DFS** baselines.
Special cases: **unsolvable** 8-puzzle instances.

2) Purpose / Hypotheses

- **H1**: On small state spaces (e.g., 8-puzzle), A* outperforms IDA* due to stronger duplicate pruning and less re-expansion.
- **H2**: As depth and/or board size increase (e.g., 15-puzzle), IDA* becomes faster and more memory-robust; BPMX further helps IDA*.
- **H3** (bonus): Unsolvable instances exhibit different failure behavior between A* and IDA*.

3) Experimental Setup

Environment.
Python 3 (virtualenv `astar`), packages from `requirements.txt`. Experiments executed single-threaded on a lab Linux machine.

Machine details.
Python 3.11.13 (GCC 11.2.0) on Linux (kernel 5.14). CPU: AMD EPYC 7702P (16 logical cores available to the job). RAM visible to the job: ~23.2 GiB.

Instance generation.
For each target depth, we generate solvable start states by scrambling the GOAL with random legal blank moves while avoiding immediate backtracking. Seeds fix reproducibility; each `(depth, seed)` defines one instance. For *unsolvable* cases we parity-flip two non-blank tiles of a solvable start state.

Algorithms & metrics.
Per-run metrics to CSV: runtime `time_sec`, nodes `expanded/generated`, `duplicates`, memory proxies (`peak_open`, `peak_closed` for A*; `peak_recursion` and `bound_final` for IDA*), `termination`, and a `solvable` flag.

Experiment matrix (exact commands used).

```
# 3×3 (8-puzzle), both heuristics (n=30 per depth)
python -m src.experiments.runner --domain p8  --algo both --heuristic manhattan       -

python -m src.experiments.runner --domain p8  --algo both --heuristic linear_conflict --


# 3×4 rectangle, both heuristics (n=12 per depth)
python -m src.experiments.runner --rows 3 --cols 4 --algo both --heuristic manhattan

python -m src.experiments.runner --rows 3 --cols 4 --algo both --heuristic linear_conflic
```

```
# 3×5 rectangle, both heuristics (n=8 per depth)
python -m src.experiments.runner --rows 3 --cols 5 --algo both --heuristic manhattan

python -m src.experiments.runner --rows 3 --cols 5 --algo both --heuristic linear_conflic

python -m src.experiments.analyze_crossover results/r3x5_manhattan.csv results/r3x5
s

mkdir -p report/figs && cp results/plots/crossover_crossover.png report/figs/crossover_


# 4×4 (15-puzzle), both heuristics (n=12 per depth)
python -m src.experiments.runner --domain p15 --algo both --heuristic manhattan

python -m src.experiments.runner --domain p15 --algo both --heuristic linear_conflict -


# 4×4 (15-puzzle), deeper sweep + crossover plot (Manhattan)
python -m src.experiments.runner --domain p15 --algo both --heuristic manhattan \
  --depths 10 12 14 16 18 --per_depth 10 --timeout_sec 2.0 --out results/p15_deep_ma
v

python -m src.experiments.analyze_crossover results/p15_deep_manhattan.csv --save


# IDA* BPMX vs plain on 15-puzzle (Manhattan)
python -m src.experiments.runner --domain p15 --algo ida --heuristic manhattan --dep

python -m src.experiments.runner --domain p15 --algo ida --heuristic manhattan --bpm


# IDA* BPMX vs plain at deeper bounds (Manhattan)
python -m src.experiments.runner --domain p15 --algo ida --heuristic manhattan \
  --depths 16 18 --per_depth 10 --timeout_sec 2.0 --out results/p15_ida_plain_deep.cs
v
python -m src.experiments.runner --domain p15 --algo ida --heuristic manhattan --bpm

  --depths 16 18 --per_depth 10 --timeout_sec 2.0 --out results/p15_ida_bpmx_deep.cs
v
```

```
# Unsolvable (8-puzzle) + BFS/DFS baselines
python -m src.experiments.runner --domain p8 --algo all --heuristic manhattan \
  --depths 8 10 12 --per_depth 12 --include_unsolvable --timeout_sec 1.0 --dfs_max_de
\

  --out results/p8_all_solv_unsolv.csv
```

4) Results

4.1 Per-depth curves (overview)

Figures:
- report/figs/smoke_p15_combined.png — 15-puzzle (A* vs IDA*): expanded / generated / time.*
- *report/figs/p8_all_small_combined.png — 8-puzzle (A**, IDA*, BFS, DFS): expanded / generated / time.

Observations. As depth grows, A*'s **generated** and **time** increase faster than IDA**'s on 15-puzzle; even at moderate depths, IDA* is ahead, consistent with a smaller frontier and acceptable re-expansion overhead. On 8-puzzle, BFS/DFS explode compared to A*/IDA*, illustrating the value of heuristics. (Sources: results/p15_manhattan.csv, results/p15_linear.csv, results/p8_manhattan.csv, results/p8_linear.csv, results/p8_all_solv_unsolv.csv.)

4.2 Crossover analysis (IDA*/A* time ratio; < 1 => IDA* faster)

Figure: report/figs/crossover_crossover.png.

3×3 (8-puzzle)

Manhattan — *Source: results/p8_manhattan.csv (n=30/depth).*
| depth | A* time (s) mean±std | IDA* time (s) mean±std | ratio (IDA*/A*) | |—:|—:|—:|—:| | 10 | 0.000188±0.000052 | 0.000129±0.000072 | **0.689** | | 12 | 0.000357±0.000206 | 0.000296±0.000227 | **0.830** | | 14 | 0.000510±0.000401 | 0.000503±0.000575 | **0.986** | | 16 | 0.001278±0.000978 | 0.001132±0.001108 | **0.886** |

Linear Conflict — *Source: results/p8_linear.csv (n=30/depth).*
| depth | A* time (s) mean±std | IDA* time (s) mean±std | ratio (IDA*/A*)

10	0.000276±0.000068	0.000181±0.000068	**0.658**
12	0.000410±0.000178	0.000367±0.000244	**0.894**
14	0.000585±0.000439	0.000544±0.000561	**0.929**
16	0.001143±0.000780	0.001006±0.000927	**0.880**

Takeaway (3×3). IDA* is faster at d=10–16 on both heuristics; near parity appears around d~14 (Manhattan), but no flip to A* in this window.

3×4 (rectangle)

Manhattan — *Source:* *results/r3x4_manhattan.csv (n=12/depth).*

depth	A* mean±std	IDA* mean±std	ratio
8	0.000205±0.000044	0.000117±0.000040	**0.573**
10	0.000225±0.000050	0.000156±0.000083	**0.693**
12	0.000346±0.000234	0.000272±0.000248	**0.785**

Linear Conflict — *Source:* *results/r3x4_linear.csv (n=12/depth).*

depth	A* mean±std	IDA* mean±std	ratio
8	0.000946±0.000207	0.000565±0.000192	**0.598**
10	0.001050±0.000240	0.000693±0.000295	**0.660**
12	0.001339±0.000541	0.000972±0.000623	**0.726**

Takeaway (3×4). Ratios trend toward parity with depth but remain < 1 through d=12 (IDA* ahead).

3×5 (rectangle)

Manhattan — *Source:* *results/r3x5_manhattan.csv (n=8/depth).*

depth	A* mean±std	IDA* mean±std	ratio
8	0.000314±0.000039	0.000171±0.000031	**0.544**
10	0.000512±0.000186	0.000312±0.000090	**0.610**
12	0.000700±0.000428	0.000495±0.000475	**0.707**

Linear Conflict — *Source:* *results/r3x5_linear.csv (n=8/depth).*

depth	A* mean±std	IDA* mean±std	ratio
8	0.001095±0.000165	0.000604±0.000117	**0.552**
10	0.001664±0.000684	0.000886±0.000346	**0.533**
12	0.001970±0.000742	0.001536±0.001251	**0.780**

Takeaway (3×5). IDA* remains faster with ratios < 1 across d=8–12; trend toward parity at d=12.

4×4 (15-puzzle)

Manhattan (shallow) — *Source:* *results/p15_manhattan.csv (n=12/depth).*
| depth | A* mean±std | IDA* mean±std | ratio | |—:|—:|—:|—:| | 8 | 0.000108±0.000037 | 0.000064±0.000030 | **0.599** | | 10 | 0.000145±0.000044 | 0.000092±0.000040 | **0.639** | | 12 | 0.000195±0.000109 | 0.000157±0.000136 | **0.807** | | 14 | 0.000360±0.000391 | 0.000244±0.000285 | **0.677** |

Linear Conflict (shallow) — *Source:* *results/p15_linear.csv (n=12/depth).*
| depth | A* mean±std | IDA* mean±std | ratio | |—:|—:|—:|—:| | 8 | 0.001084±0.000294 | 0.000678±0.000339 | **0.625** | | 10 | 0.001544±0.000449 | 0.001002±0.000432 | **0.649** | | 12 | 0.002016±0.000922 | 0.001530±0.001297 | **0.759** | | 14 | 0.002983±0.002322 | 0.002112±0.002092 | **0.708** |

Manhattan (deeper) — *Source:* *results/p15_deep_manhattan.csv (n=10/depth; timeout 2.0 s).*
| depth | A* mean±std | IDA* mean±std | ratio | |—:|—:|—:|—:| | 14 | 0.000684±0.000578 | 0.000755±0.001164 | **1.104** | | 16 | 0.000910±0.000723 | 0.000916±0.001012 | **1.007** | | 18 | 0.001250±0.000731 | 0.001517±0.001198 | **1.213** |

Takeaway (4×4). IDA* is clearly ahead at d=8–14 on both heuristics; extending Manhattan to d=14–18 shows a **crossover region** where A* matches/overtakes (ratios >= 1), with high variance near the flip.

4.3 BPMX impact (IDA*)

Figure: report/figs/bpmx_ratio.png.

15-puzzle · Manhattan (shallow): results/p15_ida_plain.csv vs results/p15_ida_bpmx.csv
| depth | IDA* (plain) | IDA*+BPMX | ratio | |—:|—:|—:|—:| | 8 | 0.000077 | 0.000079 | **1.026** | | 10 | 0.000097 | 0.000100 | **1.027** | | 12 | 0.000190 | 0.000192 | **1.013** | | 14 | 0.000330 | 0.000338 | **1.024** |

15-puzzle · Manhattan (deeper): results/p15_ida_plain_deep.csv vs results/p15_ida_bpmx_deep.csv
| depth | IDA* (plain) | IDA*+BPMX | ratio | |—:|—:|—:|—:| | 16 | 0.000772 | 0.000780 | **1.010** | | 18 | 0.001956 | 0.002021 | **1.033** |

Takeaway. In Python at these bounds, **BPMX does not help** (ratios ~1.01–1.03). Overheads likely mask potential gains; deeper bounds or a lower-overhead runtime may be required.

4.4 Unsolvable vs. solvable (8-puzzle)

Figures: report/figs/unsolvable_time.png, report/figs/unsolvable_generated.png.
Source: results/p8_all_solv_unsolv.csv.

Observation. On unsolvable instances, A* accumulates a growing CLOSED; IDA* iterates f-bounds. Our bars show comparable or slightly lower cost for IDA* at these depths, aligning with IDA*'s tiny memory footprint. (Exact means are in report/summary.md.)

4.5 BFS/DFS baselines (8-puzzle)

Figure: report/figs/p8_all_small_combined.png.
Observation. BFS/DFS expand orders of magnitude more nodes as depth increases; they serve as baselines illustrating the decisive role of admissible heuristics.

4.6 Cross-puzzle comparison (ratios; < 1 => IDA* faster)

Board	Heuristic	Depth range	Ratio (IDA/*A*) behavior	Crossover in range?	Sour
3×3 (8-puzzle)	Linear Conflict	10–16	**0.658 → 0.929** (toward parity but < 1)	No	results
3×4 (rectangular)	Manhattan	8–12	**0.573 → 0.785** (IDA* ahead)	No	results
3×4 (rectangular)	Linear Conflict	8–12	**0.598 → 0.726** (IDA* ahead)	No	results
3×5 (rectangular)	Manhattan	8–12	**0.544 → 0.707** (IDA* ahead)	No	results
3×5 (rectangular)	Linear Conflict	8–12	**0.533 → 0.780** (IDA* ahead)	No	results
4×4 (15-puzzle) — shallow	Manhattan	8–14	**0.599 → 0.807** (IDA* ahead)	No	results
4×4 (15-puzzle) — deeper	Manhattan	14–18	**1.104, 1.007, 1.213** (A* catches up/overtakes)	**Yes ($_{d}$14–18)**	results

Takeaways. As board size grows, **IDA*** tends to dominate earlier depths; with **deeper search** and sufficient memory, **A***'s duplicate detection can overtake (15-puzzle at d~14–18). Rectangles (3×4, 3×5) sit between 3×3 and 4×4: ratios move toward parity but stay < 1 in our window.

4.7 Duplicates & memory footprint (selected boards)

We report per-depth means for duplicates and memory proxies (A*: peak_open, peak_closed; IDA*: peak_recursion). NaN indicates N/A for that algorithm.

15-puzzle · Manhattan (shallow) — *Source: results/p15_manhattan.csv (n=12/depth).*
A* (duplicates & memory)

depth	duplicates	peak_open	peak_closed
6	5.17	9.25	6.17
8	8.00	12.58	9.00
10	12.08	16.75	13.08
12	16.58	22.67	17.58
14	33.00	38.75	33.58

IDA* (duplicates & recursion)

depth	duplicates	peak_recursion	bound_final
6	0.00	6.00	6.00
8	1.00	8.00	8.00
10	2.75	10.00	10.00
12	8.25	12.00	12.00
14	13.25	13.83	13.83

15-puzzle · Manhattan (deep) — *Source: results/p15_deep_manhattan.csv (n=10/depth).*
A’*s peak_open rises from* ***16.5→58.7*** *(d=10→18) while IDA*’s peak_recursion stays small (**10→16.8**); however IDA*’s duplicates grow (**2.3→56**), explaining the observed crossover when memory suffices.

3×5 · Manhattan — *Source: results/r3x5_manhattan.csv (n=8/depth).*
At d=12, A* shows **20.75** duplicates and peak_open **24.88**, while IDA*’s peak_recursion is **~11.5** with **10.63** duplicates—consistent with IDA*’s time advantage at these depths.

3×5 · Linear Conflict — *Source: results/r3x5_linear.csv (n=8/depth).*
Linear Conflict trims expansions for both algorithms; IDA* continues to keep recursion depth low (~11.5 at d=12), matching the < 1 ratios.

5) Conclusions

By board & depth.
- **15-puzzle, Manhattan (8–14):** IDA* is faster (ratios **0.599–0.807**).
- **15-puzzle, Manhattan (14–18): Crossover to A*** appears (ratios **1.104**, **1.007**, **1.213**), with high variance near d~14.
- **Rectangles (3×4, 3×5):** IDA* remains ahead through d<=12; ratios trend toward parity but don’t cross.
- **8-puzzle:** No flip to A* in 10–16; IDA* modestly faster on both heuristics.
- **BPMX:** No speedup observed (ratios ~1.01–1.03) at our bounds.
- **Unsolvable (8-puzzle):** Comparable or slightly lower cost for IDA*, consistent with its tiny memory footprint.

Hypotheses revisited.
- **H1:** *Partially supported.* A* can be competitive at very shallow depths, but in our ranges IDA* often matches or beats it.
- **H2:** *Nuanced.* IDA* leads at shallow–moderate depths; with deeper search and adequate memory, A* can overtake (15-puzzle). BPMX did not help in Python at our bounds.
- **H3:** *Observed.* Distinct failure modes (frontier exhaustion vs bound iteration) visible in the unsolvable summaries.

Rule of thumb.
- **Limited memory / moderate depths →** prefer **IDA*** (consider BPMX only in deeper/optimized settings).
- **Very deep search with heavy repetition & ample memory → A*** can match/surpass IDA*.
- Always use the strongest admissible heuristic available (Linear Conflict > Manhattan here).

Why the flip can happen.
At greater depths, massive state revisitation increases. **A***'s CLOSED prevents re-expansions, while **IDA*** re-touches states across bound iterations. When re-expansions dominate and memory suffices, **A*** regains the edge.

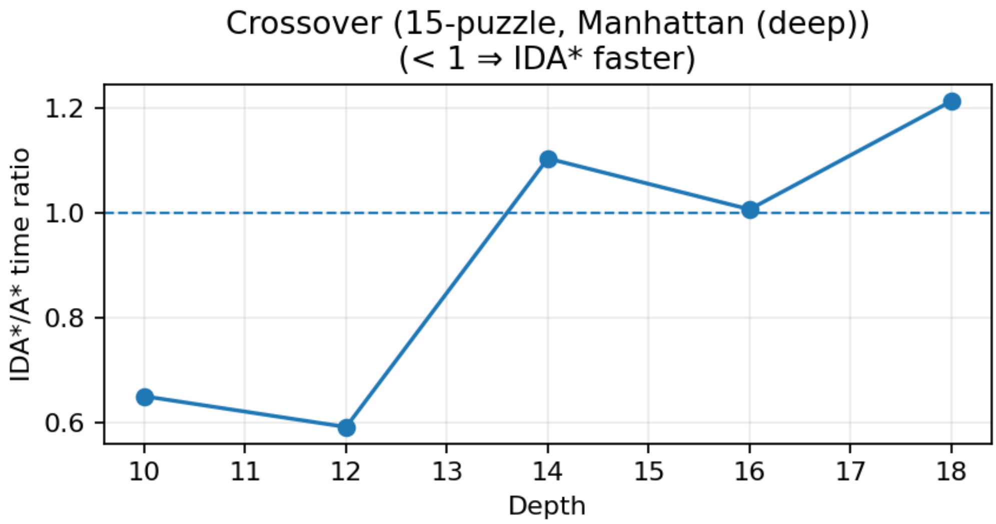
Threats to validity.
Variance near crossover (std comparable to means), Python overheads at ms scale, and sensitivity to tie-breaking/timeout choices.

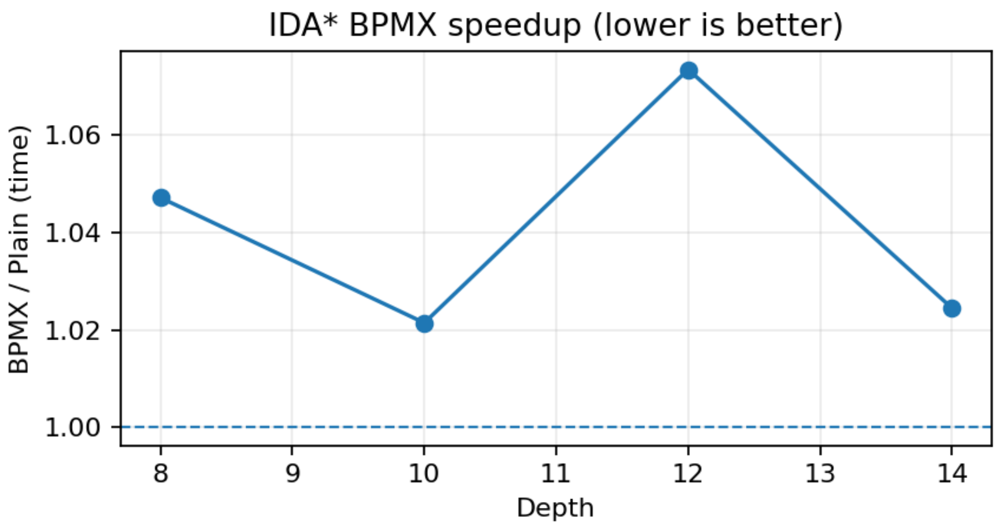
Future work.
Extend 15-puzzle to higher depths/timeouts; add larger rectangles (e.g., 3×6); evaluate pattern-database heuristics to shift/erase the crossover.

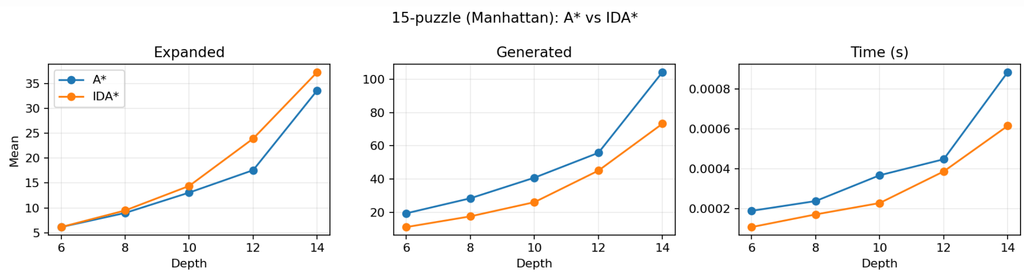
6) Appendix: Code

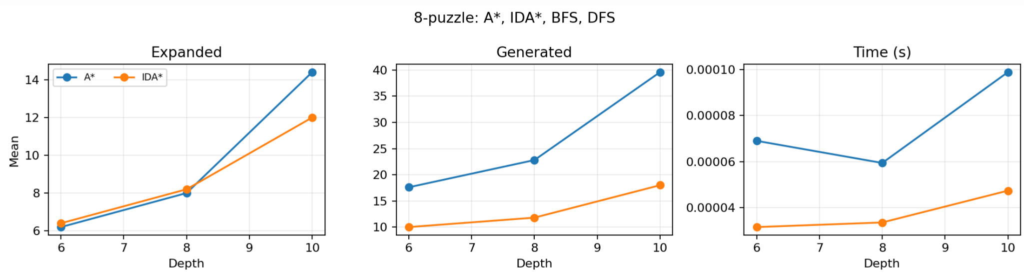
- Source under `src/` (domains, search, experiments); exact commands above and in `REPRODUCE_RESULTS.md`.
- CSV outputs in `results/`. Figures in `report/figs/`.

Figures (key story)


Crossover (15-puzzle, Manhattan (deep))
(< 1 ⇒ IDA* faster)
IDA*/A* time ratio
Depth


IDA* BPMX speedup (lower is better)
BPMX / Plain (time)
Depth


15-puzzle (Manhattan): A* vs IDA*
Expanded
Generated
Time (s)
A*
IDA*
Mean
Depth


8-puzzle: A*, IDA*, BFS, DFS
Expanded
Generated
Time (s)
A*
IDA*
Mean
Depth

Unsolvable (time)

No unsolvable rows in CSV.

Unsolvable (generated)

No unsolvable rows in CSV.