Experimental Report: Gate Puzzle Solvers

Algorithms: A1 (UCS), A2 (UCS + Duplicate Checking), A3 (Iterated Width)
Datasets: capability1–13, impassable1–3

Introduction & Objective

Goal. Evaluate three search strategies for the Gate puzzle: uniform-cost search (A1), uniform-cost search with duplicate detection (A2), and Iterated Width (A3). We benchmark them on two suites: 13 small-to-medium 'capability' maps and 3 hard 'impassable' maps.

Research questions. (1) How do the algorithms compare in time and space? (2) Does novelty-based pruning (A3) meaningfully reduce the number of generated/expanded nodes? (3) What IW width is typically sufficient to reach a solution?

Methods (A1 / A2 / A3)

A1 — Uniform-Cost Search (UCS). Standard best-first expansion by path cost (unit step costs). No global visited set; completeness holds on finite spaces but repeated states make it inefficient.

A2 — UCS + duplicate checking. Adds a global visited structure keyed by a packed map encoding. This prevents re-expansions of previously seen states, usually reducing generated/expanded nodes at the expense of auxiliary memory.

A3 — Iterated Width (IW). Performs width-limited novelty searches: IW(1), IW(2), ... until a goal is found. A state is expanded only if it introduces at least one novel atom (or k-tuple for IW(k)). We bit-pack the map and check novelty in per-width radix trees. This aggressively prunes redundant states; in our runs, solutions were typically found at small widths (mean≈1.69, median≈2).

Experimental Setup

Environment. All runs were compiled with -O2 -DNDEBUG and executed on the same machine. Each run reports: solution path (if any), execution time, expanded and generated nodes, duplicate count, auxiliary memory usage, number of pieces, solution length, empty spaces, IW width at solution (A3), and nodes/sec.

Datasets. We used 13 'capability' puzzles (short solutions, designed to expose basic capabilities) and 3 'impassable' puzzles (long solutions with large branching). For fair comparison, we ran each puzzle with each algorithm and consolidated outputs into a single CSV (see Appendix).

Results — Interpretation

Coverage. On capability maps A1 solved 10 / 13, A2 solved 13 / 13, and A3 solved 13 / 13. On the three impassable maps A3 solved all 3, while A1/A2 failed on some cases due to time or memory limits.

Efficiency. Generated-node and execution-time curves show a consistent pattern: A2 improves over A1 by avoiding revisits, and A3 prunes even more aggressively using novelty. On small capability maps A1 can still be competitive in time; however, as depth/branching grows, A3 becomes the clear winner.

IW widths. The width required to solve is modest (mean≈1.69, median≈2), suggesting that solutions can be discovered without exploring the full combinatorial state space.

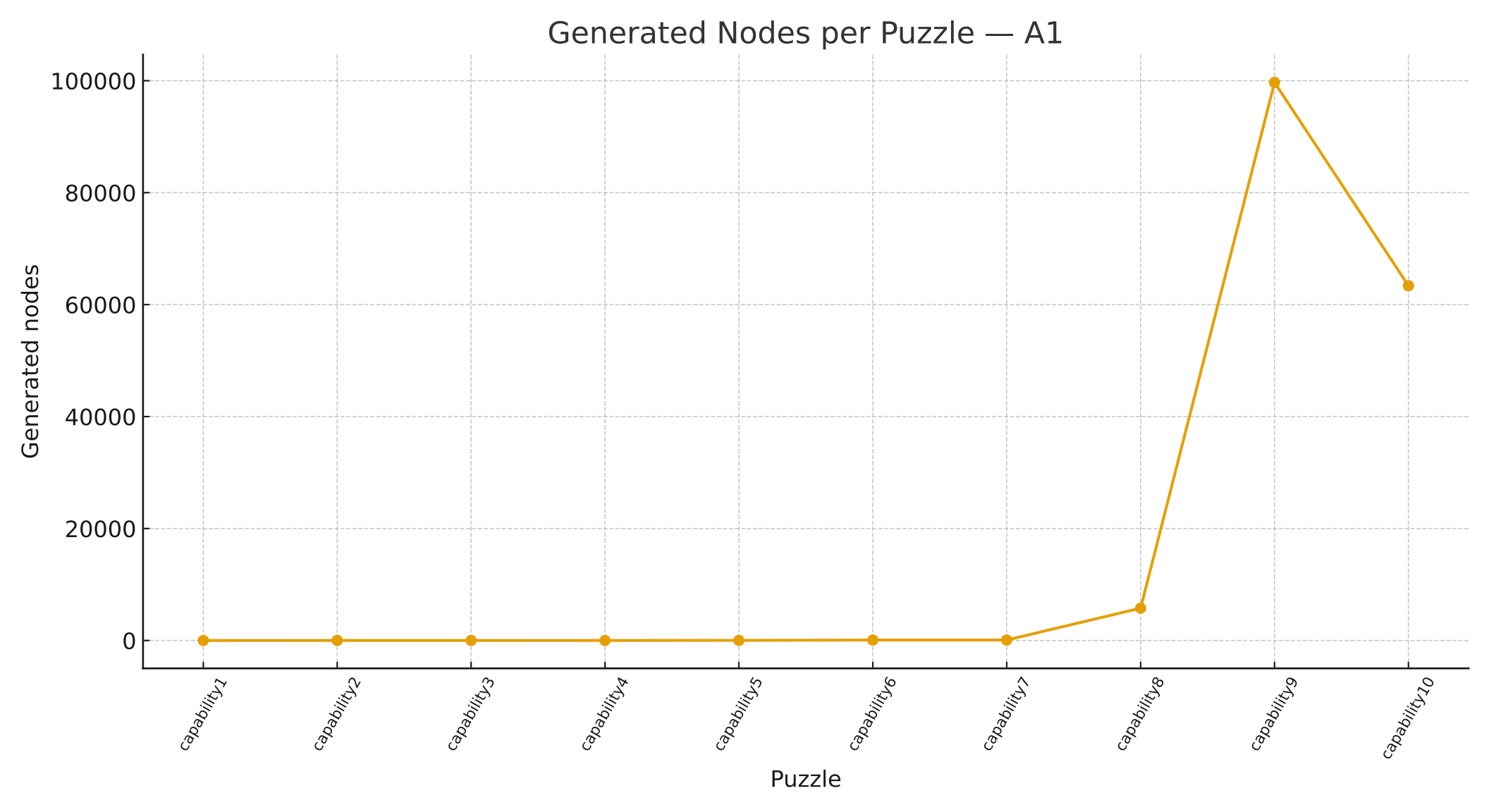
Threats to Validity

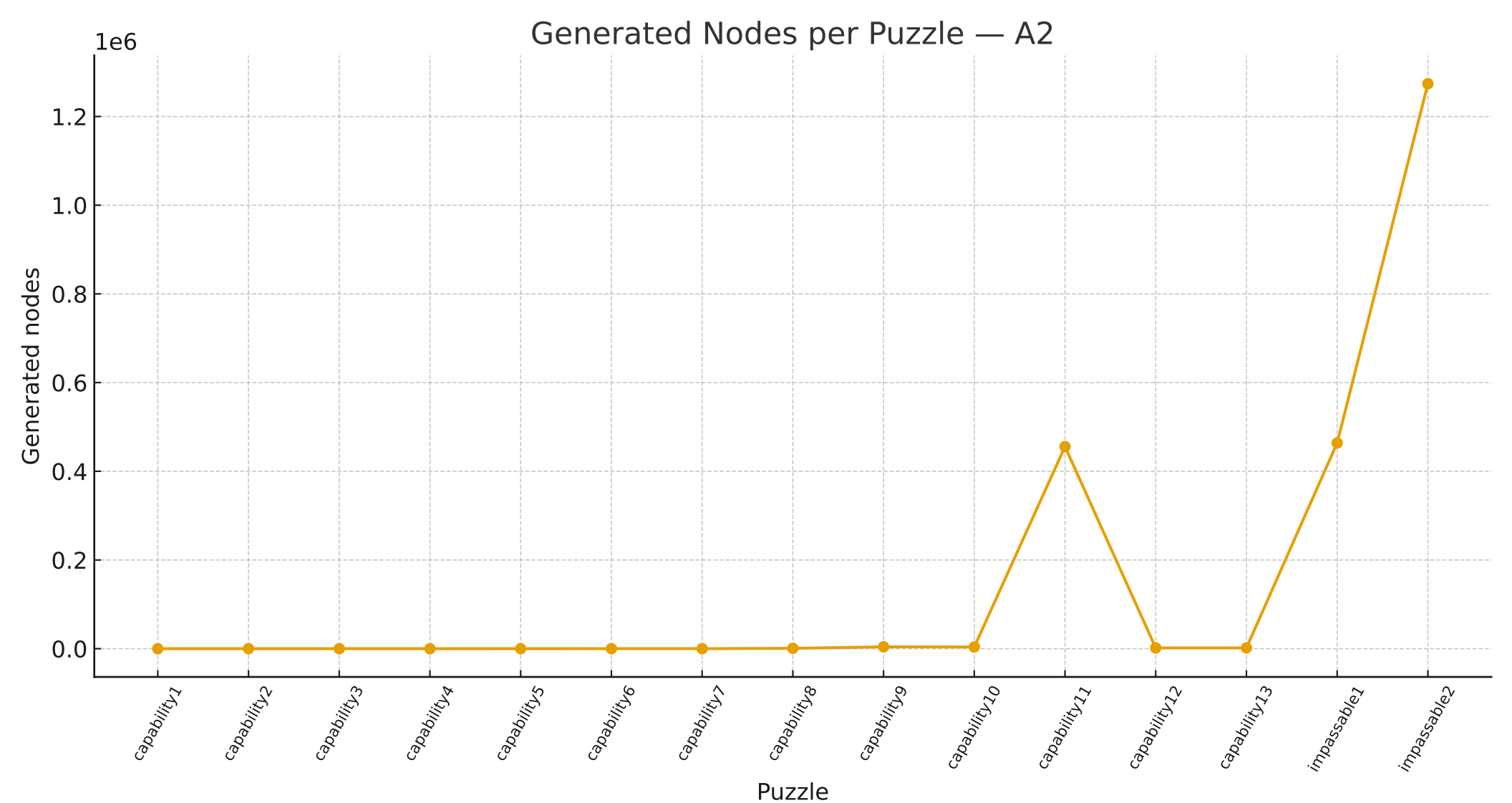
- Hardware variability. All methods ran on the same machine to reduce noise, but OS scheduling and caches still introduce minor variance.
- Heuristic/encoding choices. A3 depends on the atomization/packing and move ordering; A2's hash design affects memory/runtime.
- Completeness vs. speed. A3 is not complete for fixed width; in practice we increase width until a solution is found.

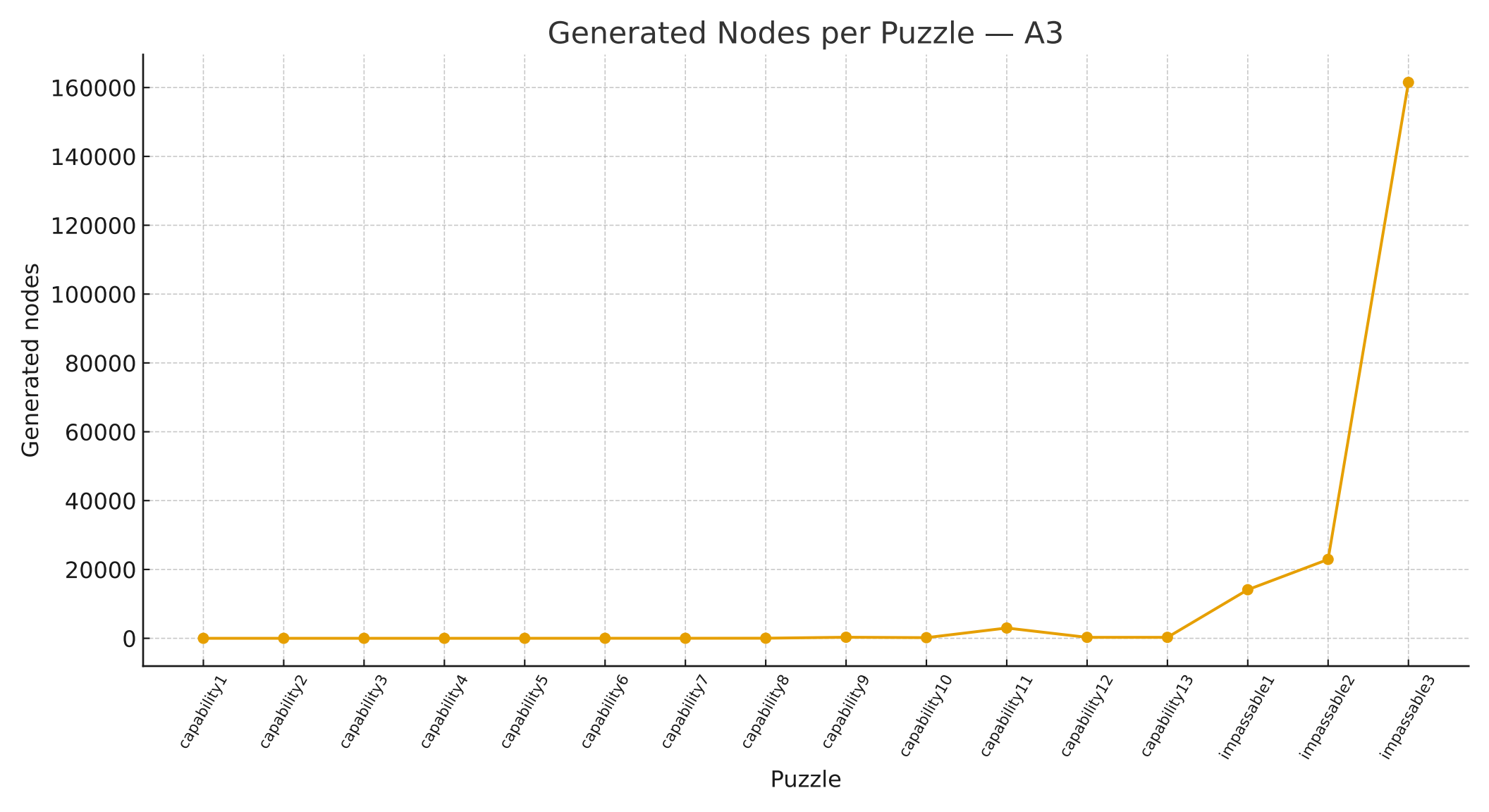
Conclusion & Takeaways

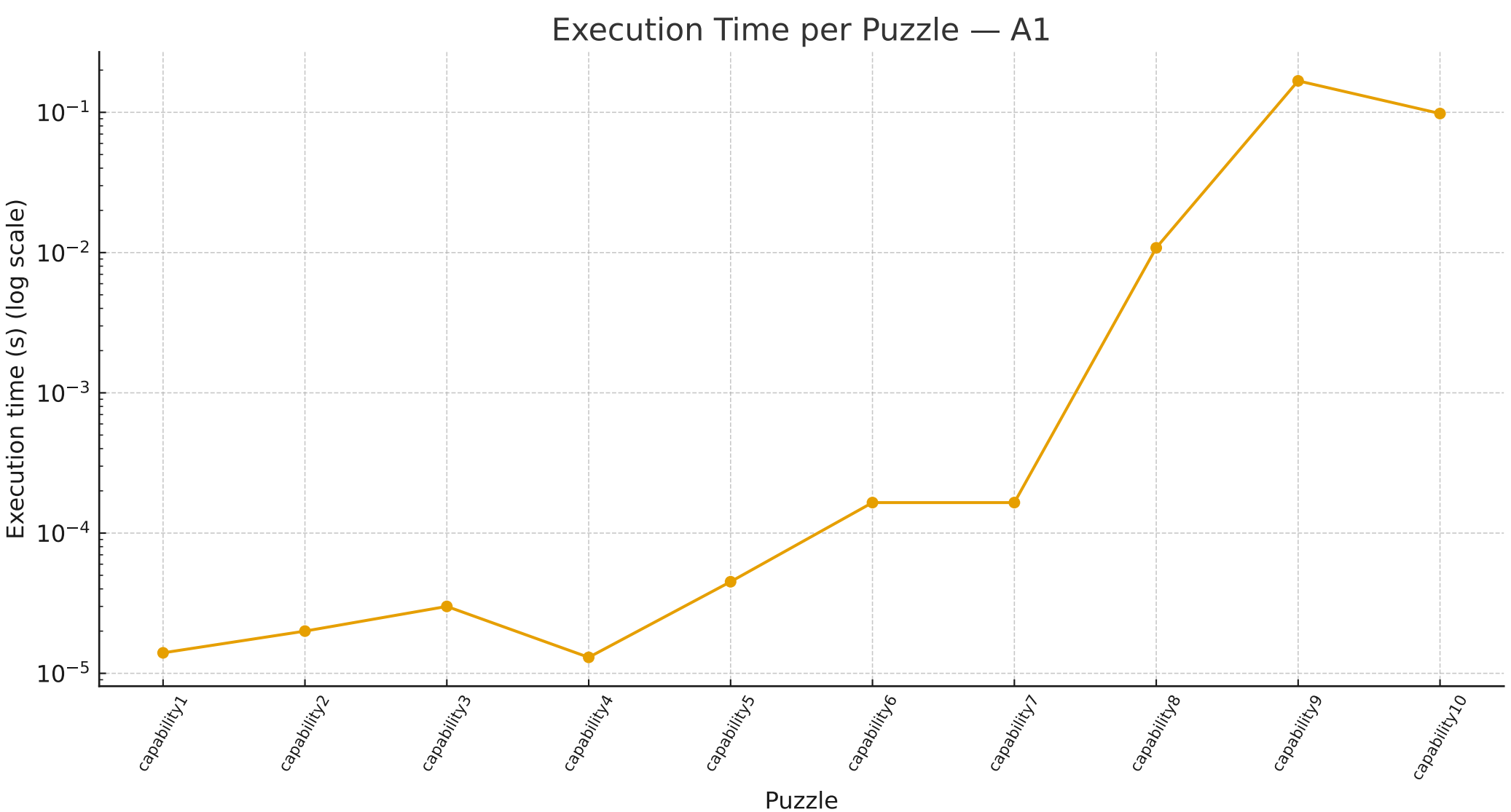
A3 (Iterated Width) is the most robust choice on medium-to-large Gate puzzles. It solved all 'impassable' maps quickly at small widths, while A1 and A2 struggled. A2 is a solid baseline when novelty machinery is unavailable; A1 remains useful on tiny puzzles due to minimal overhead.

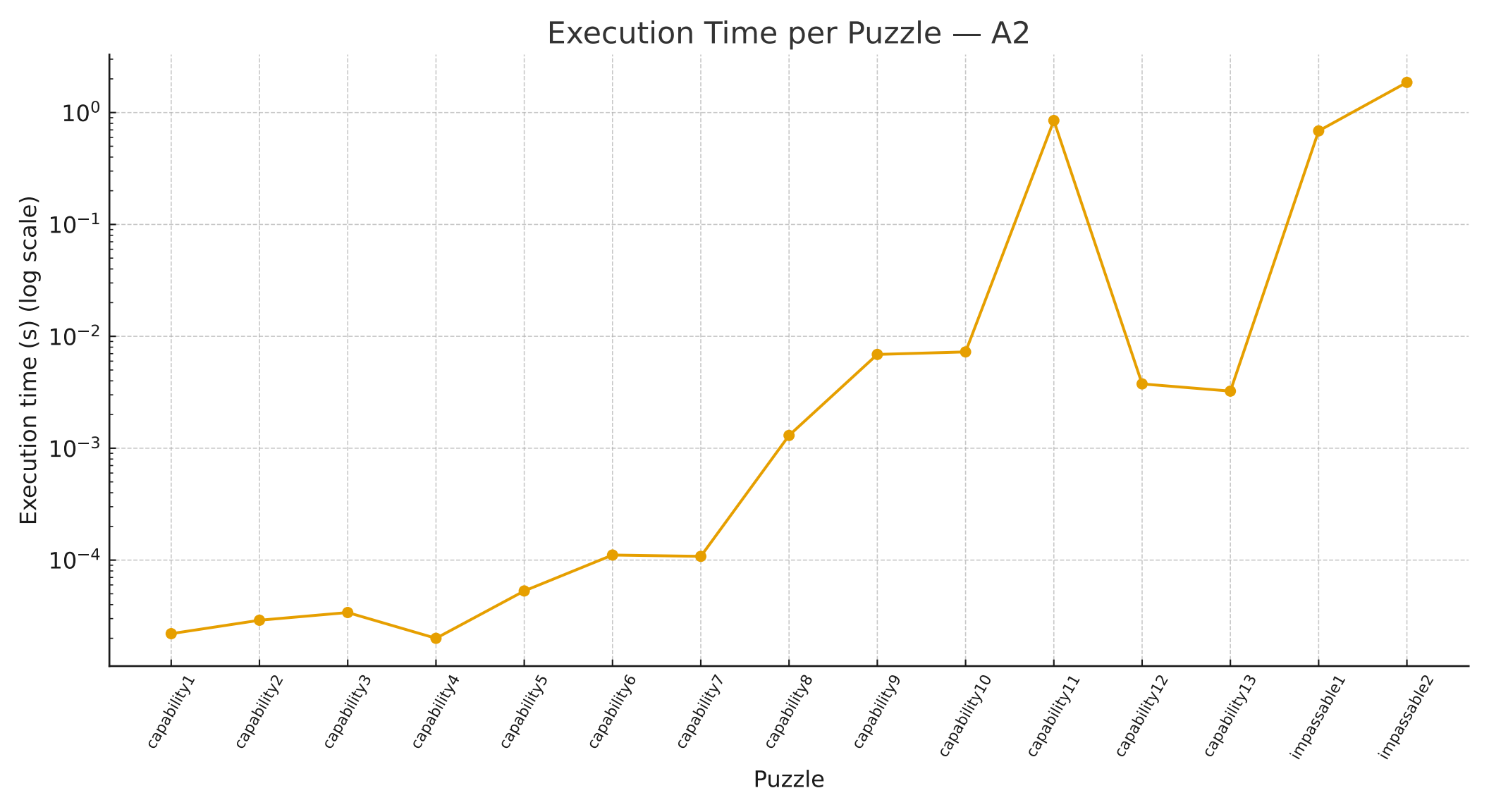
Future work. Explore better atomization for novelty, hybrid UCS+IW schedules, and parallel novelty checking.

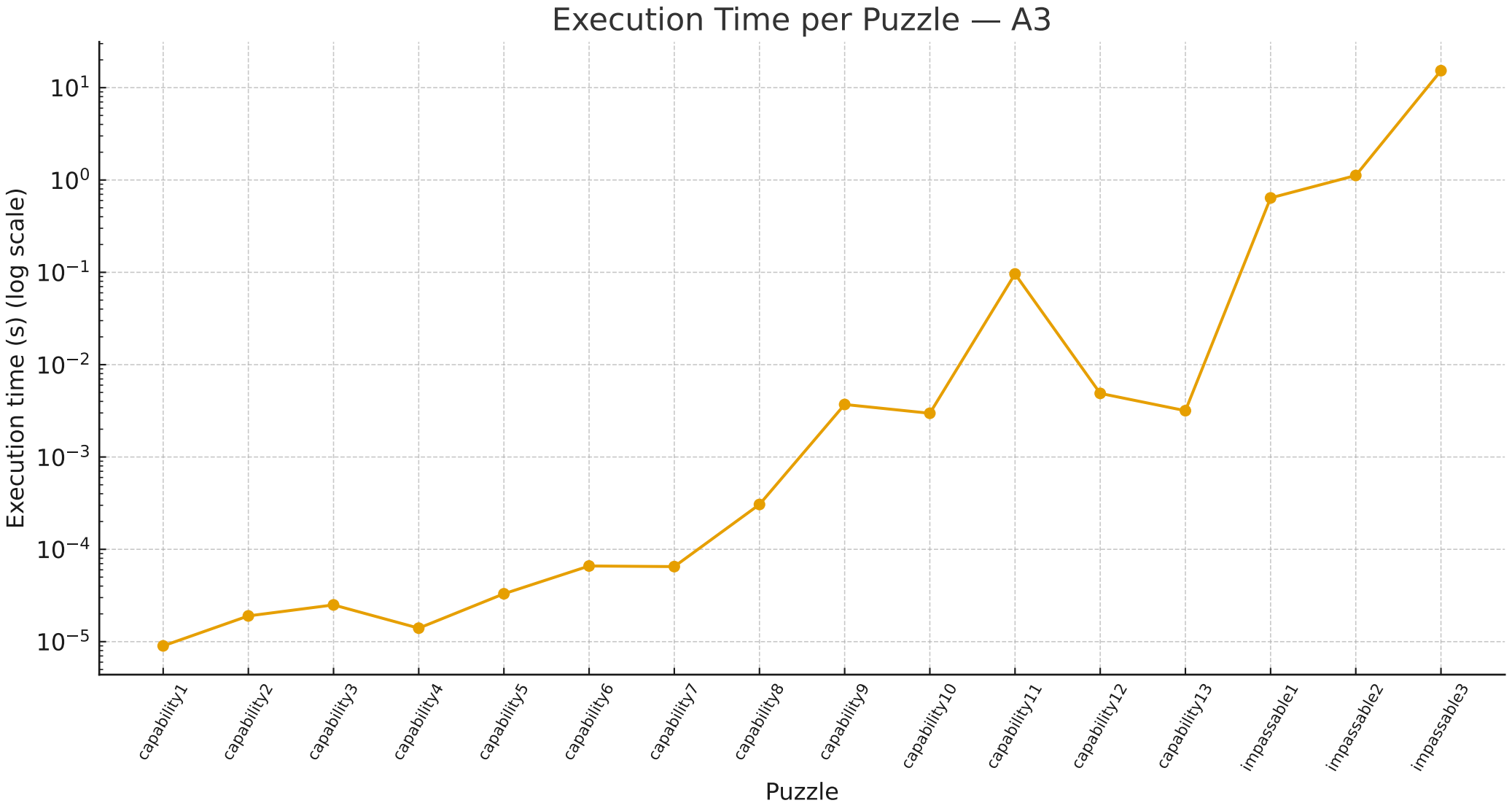

Generated Nodes per Puzzle — A1
Generated nodes
100000
80000
60000
40000
20000
0
capability1
capability2
capability3
capability4
capability5
capability6
capability7
capability8
capability9
capability10
Puzzle

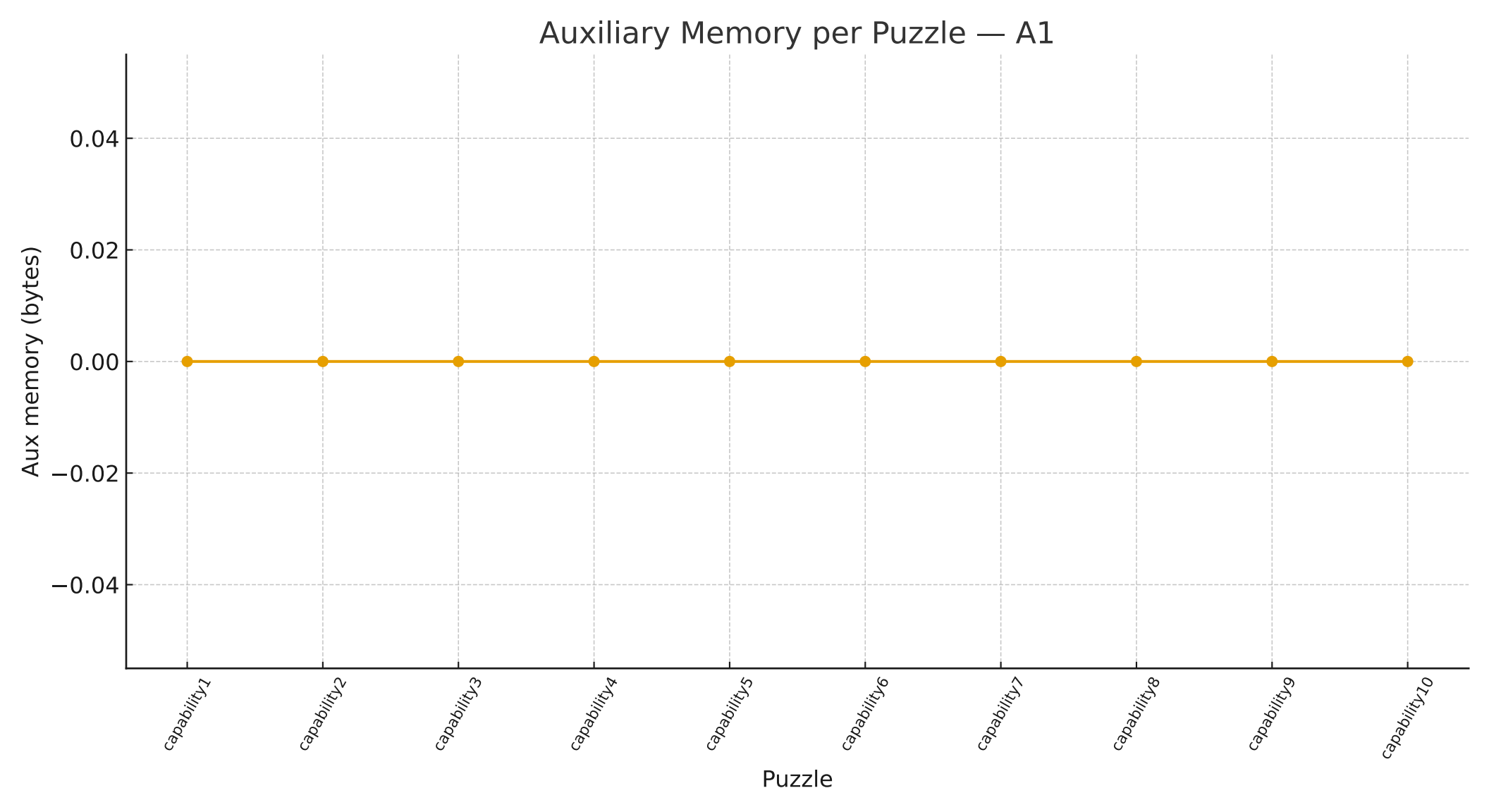

Generated Nodes per Puzzle — A2
1e6
1.2
1.0
0.8
0.6
0.4
0.2
0.0
Generated nodes
capability1
capability2
capability3
capability4
capability5
capability6
capability7
capability8
capability9
capability10
capability11
capability12
capability13
impassable1
impassable2
Puzzle

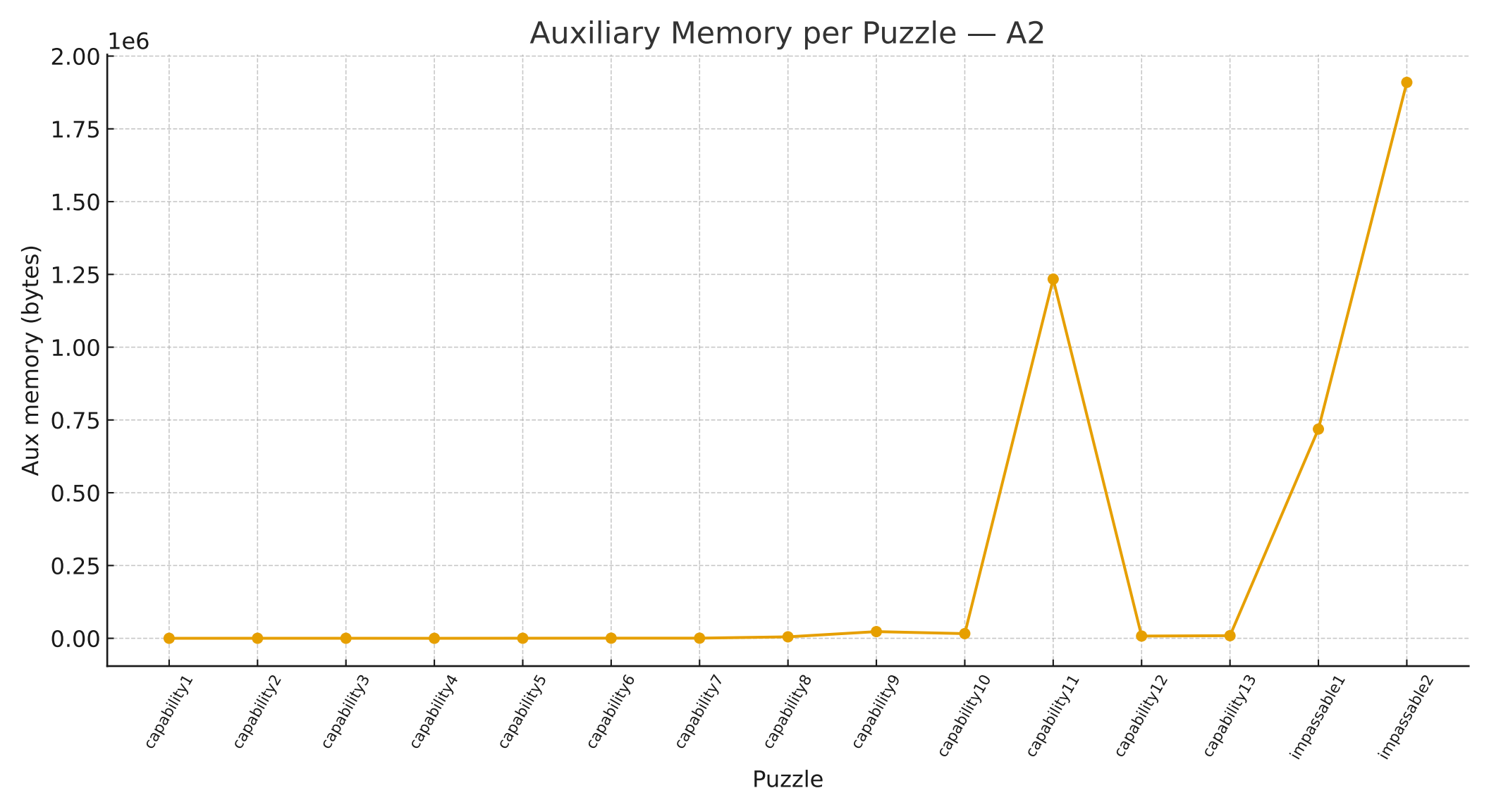

Generated Nodes per Puzzle — A3
Generated nodes
0
20000
40000
60000
80000
100000
120000
140000
160000
capability1
capability2
capability3
capability4
capability5
capability6
capability7
capability8
capability9
capability10
capability11
capability12
capability13
impassable1
impassable2
impassable3
Puzzle

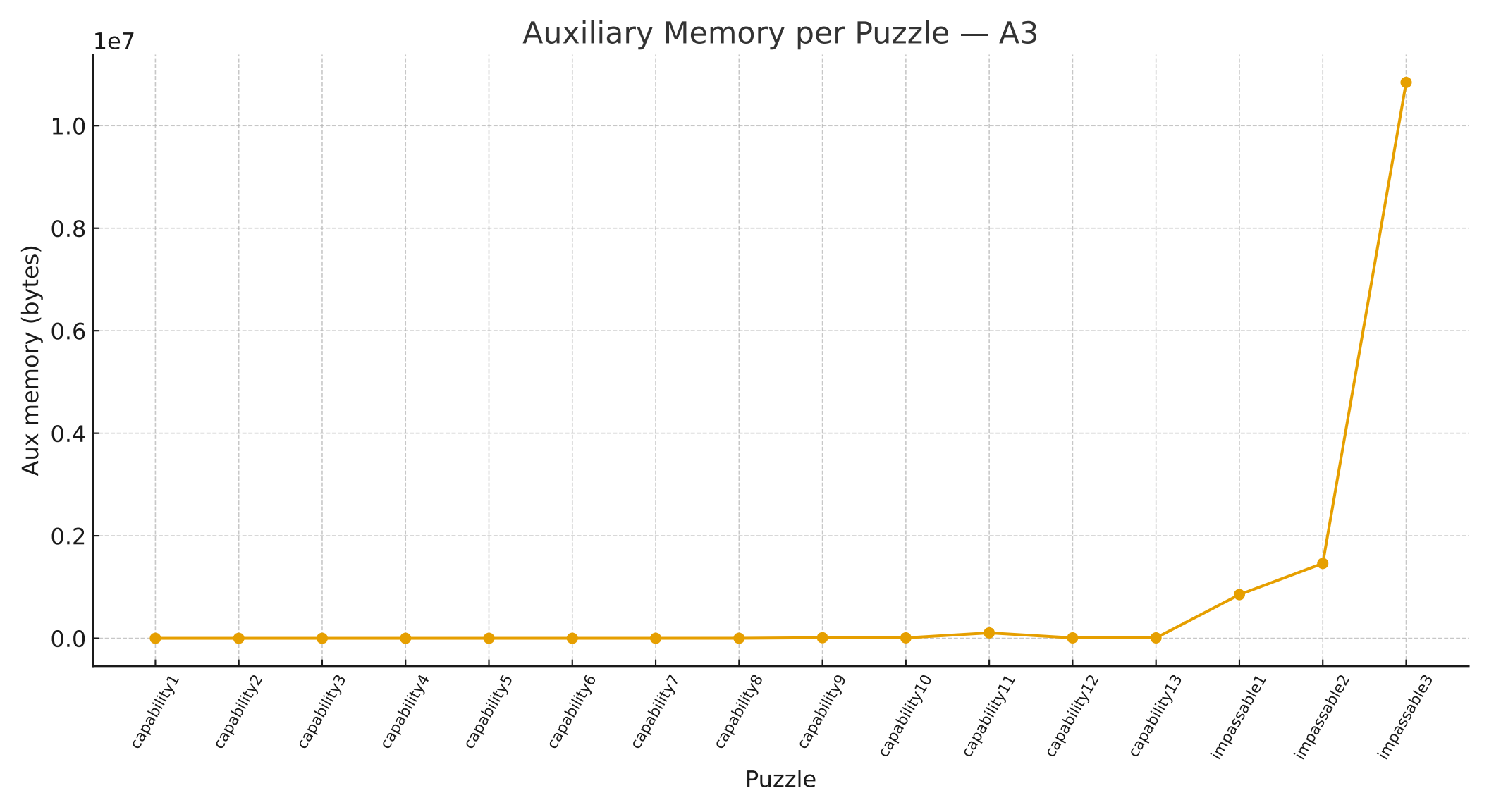

Execution Time per Puzzle — A1
Execution time (s) (log scale)
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
capability1
capability2
capability3
capability4
capability5
capability6
capability7
capability8
capability9
capability10
Puzzle

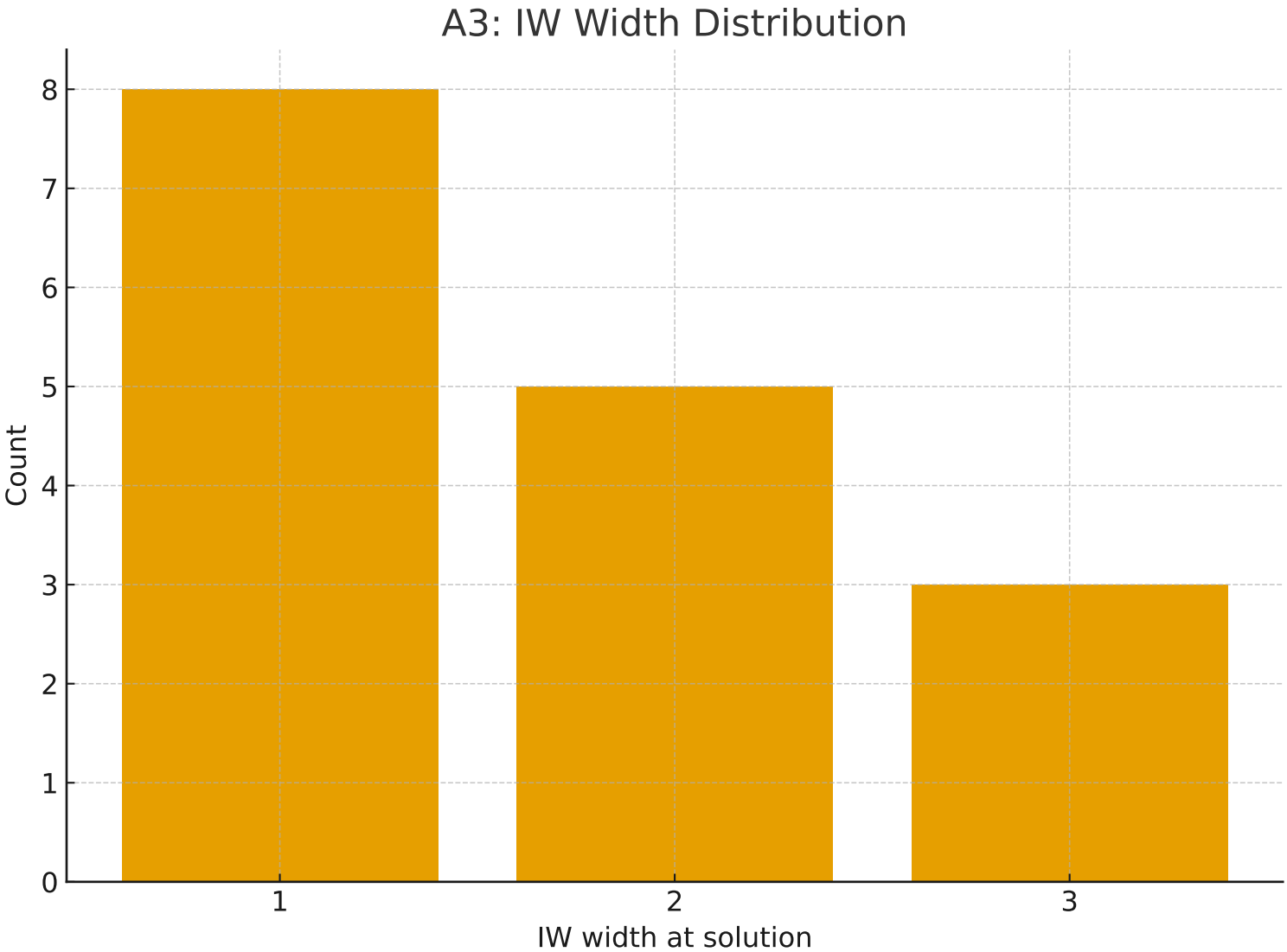

Execution Time per Puzzle — A2
Execution time (s) (log scale)
10^{0}
10^{-1}
10^{-2}
10^{-3}
10^{-4}
capability1
capability2
capability3
capability4
capability5
capability6
capability7
capability8
capability9
capability10
capability11
capability12
capability13
impassable1
impassable2
Puzzle


Execution Time per Puzzle — A3
Execution time (s) (log scale)
10^{1}
10^{0}
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
capability1
capability2
capability3
capability4
capability5
capability6
capability7
capability8
capability9
capability10
capability11
capability12
capability13
impassable1
impassable2
impassable3
Puzzle


Auxiliary Memory per Puzzle — A1
Aux memory (bytes)
0.04
0.02
0.00
−0.02
−0.04
capability1
capability2
capability3
capability4
capability5
capability6
capability7
capability8
capability9
capability10
Puzzle


Auxiliary Memory per Puzzle — A2
1e6
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
Aux memory (bytes)
capability1
capability2
capability3
capability4
capability5
capability6
capability7
capability8
capability9
capability10
capability11
capability12
capability13
impassable1
impassable2
Puzzle


Auxiliary Memory per Puzzle — A3
1e7
1.0
0.8
0.6
0.4
0.2
0.0
Aux memory (bytes)
capability1
capability2
capability3
capability4
capability5
capability6
capability7
capability8
capability9
capability10
capability11
capability12
capability13
impassable1
impassable2
impassable3
Puzzle


A3: IW Width Distribution
Count
IW width at solution
0
1
2
3
4
5
6
7
8
1
2
3

Appendix: All algorithms results (page 1/2) — without 'solution_path'

puzzle	algo	time_s	expanded	generated	duplicated	aux_mem	pieces	steps	empty	iw	nps	map
capability1	A1	1.4e-05	2	4	0	0	1	1	34	2	213269.695	algo1_c1
capability1	A2	2.2e-05	2	4	0	50	1	1	34	1	136770.783	algo2_c1
capability1	A3	9e-06	1	1	0	0	1	1	34	1	226719.135	algo3_c1
capability2	A1	2e-05	3	8	0	0	1	1	36	2	199728.762	algo1_c2
capability2	A2	2.9e-05	3	8	0	148	1	1	36	1	137518.164	algo2_c2
capability2	A3	1.9e-05	1	2	0	17	1	1	36	1	106184.911	algo3_c2
capability3	A1	3e-05	3	8	0	0	1	1	36	2	133152.508	algo1_c3
capability3	A2	3.4e-05	3	8	0	148	1	1	36	1	117323.189	algo2_c3
capability3	A3	2.5e-05	1	2	0	17	1	1	36	1	79891.505	algo3_c3
capability4	A1	1.3e-05	2	4	0	0	1	1	34	2	233016.889	algo1_c4
capability4	A2	2e-05	2	4	0	17	1	1	34	1	149796.571	algo2_c4
capability4	A3	1.4e-05	1	1	0	0	1	1	34	1	142179.797	algo3_c4
capability5	A1	4.5e-05	6	20	0	0	1	2	34	2	155344.593	algo1_c5
capability5	A2	5.3e-05	6	20	5	278	1	2	34	1	132252.829	algo2_c5
capability5	A3	3.3e-05	2	5	0	115	1	2	34	1	91180.522	algo3_c5
capability6	A1	0.000165	21	80	0	0	1	3	34	2	133344.925	algo1_c6
capability6	A2	0.000111	13	48	20	504	1	3	34	1	126009.133	algo2_c6
capability6	A3	6.6e-05	7	12	8	342	1	3	34	1	121135.134	algo3_c6
capability7	A1	0.000165	21	80	0	0	1	3	34	2	133344.925	algo1_c7
capability7	A2	0.000108	13	48	20	504	1	3	34	1	129625.289	algo2_c7
capability7	A3	6.5e-05	7	12	8	342	1	3	34	1	123361.882	algo3_c7
capability8	A1	0.010816	723	5,776	0	0	2	4	32	3	66937.268	algo1_c8
capability8	A2	0.001301	95	752	353	5,075	2	4	32	2	73800.070	algo2_c8
capability8	A3	0.000306	21	28	85	926	2	4	32	1	71921.035	algo3_c8

Appendix: All algorithms results (page 2/2) — without 'solution_path'

puzzle	algo	time_s	expanded	generated	duplicated	aux_mem	pieces	steps	empty	iw	nps	map
capability9	A1	0.167703	8,308	99,684	0	0	3	5	30	4	49545.949	algo1_c9
capability9	A2	0.00689	350	4,188	1,736	22,983	3	5	30	3	50942.964	algo2_c9
capability9	A3	0.003706	174	307	883	11,429	3	5	30	2	47220.998	algo3_c9
capability10	A1	0.09808	3,962	63,376	0	0	4	5	24	5	40405.824	algo1_c10
capability10	A2	0.007259	240	3,824	1,074	16,067	4	5	24	4	33199.569	algo2_c10
capability10	A3	0.002975	114	189	518	8,465	4	5	24	2	38655.631	algo3_c10
capability11	A2	0.848907	28,479	455,648	210,235	1,233,868	4	13	27	4	33549.023	algo2_c11
capability11	A3	0.096115	2,932	2,992	21,290	105,808	4	19	27	2	30515.568	algo3_c11
capability12	A2	0.003756	219	1,744	788	7,732	2	11	30	2	58572.228	algo2_c12
capability12	A3	0.004885	248	279	889	8,236	2	11	30	2	50972.800	algo3_c12
capability13	A2	0.003238	221	1,760	853	9,036	2	8	31	2	68561.629	algo2_c13
capability13	A3	0.003173	230	279	848	8,209	2	8	31	2	72799.175	algo3_c13
impassable1	A2	0.685462	19,329	463,872	84,945	718,688	6	46	17	6	28199.959	algo2_i1
impassable1	A3	0.639493	13,330	14,119	53,702	852,026	6	47	17	3	20846.202	algo3_i1
impassable2	A2	1.86097	53,080	1,273,896	230,560	1,909,627	6	64	17	6	28523.311	algo2_i2
impassable2	A3	1.11938	22,322	22,921	87,172	1,460,751	6	65	17	3	19942.273	algo3_i2
impassable3	A3	15.2841	159,158	161,502	687,400	10,843,957	8	93	14	3	10413.363	algo3_i3