

EasyLocal – Benchmarks

A C++23 framework for local search and metaheuristics

Contents

Benchmarks	3
How it was measured	3
EasyLocal 3 v3.4.1 versus EasyLocal 4 main-4fc56ca	5
Infrastructure of EasyLocal 4 main-4fc56ca	10
Compilers and architectures of EasyLocal 4 main-4fc56ca	13

Benchmarks

EasyLocal 4 is measured at every release against EasyLocal 3 (the [v3.4.1 tag](#) of `easylocal-legacy`) and on its own infrastructure. The numbers are produced in [easylocal-benchmarks](#) on a GitHub-hosted runner (Ubuntu, GCC 16, Release); each EasyLocal release starts a run.

- **EasyLocal 3 versus EasyLocal 4.** The three example problems (TSP with 2-opt, Assignment with job reassignment, Exam Timetabling with exam moves) are written in both frameworks with the same cost functions, delta evaluations and neighborhood orders, and searched from the same initial solutions. The TSP is searched a second time with the union of the 2-opt and the swap neighborhoods (EasyLocal 3 `SetUnion`, EasyLocal 4 `neighborhood_union`), which measures the cost of a compound neighborhood in both frameworks. Each is measured with delta evaluations for all its cost components, for some of them, and for none (the TSP with 2-opt has a single component and a single neighborhood, so no mixed mode; the union has the delta of the 2-opt moves only); the trajectories are the same, only the speed changes. Both frameworks are measured at every release, in the same job on the same machine.
- **Infrastructure.** Neighborhood traversal, runner-level search and tracing overhead (`infrastructure/` of `easylocal-benchmarks`).
- **Compilers and architectures.** The EasyLocal 4 side of the comparison built by several toolchains, on Linux `x86_64` and `ARM64`, macOS `ARM64` and Windows `x86_64`: each platform in a job of its own, its toolchains alternating run by run.

The speed-up compares the times per evaluation, because the two frameworks may explore different trajectories: EasyLocal 3 steepest descent breaks ties at random, and simulated annealing uses each framework's random numbers. Costs are means over the seeds.

A trajectory also decides which moves are evaluated, which a cost per evaluation does not correct: a scan that restarts at the first move of a union, enumerated child after child, reaches almost only the first child, so a delta on that child alone looks nearly free. First descent is therefore scanned the same way on both sides here, from a move drawn at random; what the library's own `FirstImprovement` costs, which restarts at the first move, is measured by the infrastructure benchmarks below.

How it was measured

- **Where:** GitHub Actions, GitHub-hosted runner (image `ubuntu26 20260927.149.1`).
- **Versions:** EasyLocal 4 `main-4fc56ca` (`4fc56ca`); EasyLocal 3 `v3.4.1`.
- **Benchmarks:** `easylocal-benchmarks cee7b2`.
- **Date:** 2026-10-11.
- **Jobs:** each part runs in a job of its own, one after the other, possibly on a different machine; EasyLocal 3 and EasyLocal 4 always share theirs.

	EasyLocal 3 vs 4			Neighborhood			Trace		
Operating system	Ubuntu 26.04.1 LTS, kernel 7.0.0-1012- azure			Ubuntu 26.04.1 LTS, kernel 7.0.0-1012- azure			Ubuntu 26.04.1 LTS, kernel 7.0.0-1012- azure		
Architecture	x86_64, AMD EPYC 7763 64-Core Proces- sor, 4 logical CPUs			x86_64, AMD EPYC 7763 64-Core Proces- sor, 4 logical CPUs			x86_64, AMD EPYC 9V45 96-Core Proces- sor, 4 logical CPUs		
Virtual machine	Microsoft (full vir- tualization), size Standard_D4ads_v5			Microsoft (full vir- tualization), size Standard_D4ads_v5			Microsoft (full vir- tualization), size Standard_D4ads_v7		
Caches	L1d 64 KiB (2 in- stances), L1i 64 KiB (2 instances), L2 1 MiB (2 instances), L3 32 MiB (1 instance)			L1d 64 KiB (2 in- stances), L1i 64 KiB (2 instances), L2 1 MiB (2 instances), L3 32 MiB (1 instance)			L1d 96 KiB (2 in- stances), L1i 64 KiB (2 instances), L2 2 MiB (2 instances), L3 32 MiB (1 instance)		
Frequency	3238 MHz nominal			3230 MHz nominal			4211 MHz nominal		
Memory	15.6 GiB			15.6 GiB			15.6 GiB		
Instruction sets	sse4_2, avx, avx2, fma, bmi2, sha_ni, aes			sse4_2, avx, avx2, fma, bmi2, sha_ni, aes			sse4_2, avx, avx2, fma, bmi2, avx512f, avx512bw, avx512vl, avx512_vnni, sha_ni, aes		
CPU steal during the run	0.0% (busy 26%)			0.0% (busy 50%)			0.0% (busy 42%)		
Compiler	x86_64-linux-gnu-g+ +-16 (Ubuntu 16-20260322+11b(ubuntu16.0.1 20260322 (experimental) [trunk r16-8246- g569ace1fa50]; lib- stdc++			x86_64-linux-gnu-g+ +-16 (Ubuntu 16-20260322+11b(ubuntu16.0.1 20260322 (experimental) [trunk r16-8246- g569ace1fa50]; lib- stdc++			x86_64-linux-gnu-g+ +-16 (Ubuntu 16-20260322+11b(ubuntu16.0.1 20260322 (experimental) [trunk r16-8246- g569ace1fa50]; lib- stdc++		
Build	cmake version 4.4.3, Ninja, Release, C++23			cmake version 4.4.3, Ninja, Release, C++23			cmake version 4.4.3, Ninja, Release, C++23		

Each instance, algorithm and delta mode is run on every seed of the matrix, 3 times per framework; a time is the median of the repetitions, then the mean over the seeds. The two frameworks run in the same job, alternating run by run, from the same initial solutions. The neighborhood benchmarks report the median of 3 trials, the tracing ones of 5. CPU steal is the share of the virtual machine’s CPU time that the hypervisor gave to other machines on the same host while the part ran: the higher it is, the more the neighbours disturbed the measurement.

Measured on shared virtual machines

The benchmarks run as a GitHub Actions workflow on a GitHub-hosted runner: a virtual machine on shared cloud hardware, whose speed depends on the load of the other machines on the same host, on the CPU model the runner gets and on its frequency at the time. The same run repeated

can differ by several percent, and occasionally by more. EasyLocal 3 and EasyLocal 4 are measured in the same job, alternating run by run, so their ratios are more reliable than the absolute times; compare absolute times between versions only with care, and read the results as indicative, not as measurements on dedicated hardware.

EasyLocal 3 v3.4.1 versus EasyLocal 4 main-4fc56ca

Measured on 2026-10-11, EasyLocal 4 main-4fc56ca against EasyLocal 3 v3.4.1, both frameworks on the same machine, alternating run by run. Times are medians over the repetitions, then means over the seeds; speed-up is the ratio of the times per evaluation (higher is better for EasyLocal 4).

- **Instances**, generated by `el3_vs_el4/generate.py`; each is searched from 3 random initial solutions, one per seed:
 - `tsp-300`: TSP, 300 cities at random integer coordinates in a 1000×1000 square (rounded Euclidean distances); cost tour length; 2-opt moves;
 - `tsp-union-300`: TSP, 300 cities at random integer coordinates in a 1000×1000 square (rounded Euclidean distances), searched with the union of the 2-opt and the swap neighborhoods; cost tour length; the union of the 2-opt and the swap neighborhoods (EL3 `SetUnion`, EL4 `neighborhood_union`), drawn with equal bias and enumerated one after the other;
 - `assignment-1000x20`: Assignment, 1000 jobs of demand 1 to 20 on 20 machines of equal capacity, 97% of the total demand (so some overload cannot be avoided); cost $1000 \times \text{total overload} + \text{load imbalance}$; job reassignment moves;
 - `exam-400x20`: Exam Timetabling, 400 exams in 20 timeslots, 4000 pairs of conflicting exams, each shared by 1 to 10 students; cost $1000 \times \text{student conflicts} + 10 \times \text{consecutive exams} + \text{timeslot load}$; exam moves.
- **Algorithms**, each framework's own:
 - steepest descent (EL3 `SteepestDescent`, EL4 `BestImprovement`), until a local optimum;
 - first descent, scanned the same way on both sides (EL3 `FirstDescent`; EL4 a first improvement of the driver, over cursor explorers that derive from the examples' own), each iteration scanning from a move drawn at random and wrapping around, until a local optimum. The library's own `FirstImprovement` restarts at the first move instead, which walks another trajectory and evaluates another mix of moves: on a union, enumerated child after child, it reaches almost only the first child, so a delta on that child alone looks nearly free. It is measured by the infrastructure benchmarks, not here, where the two frameworks must do the same work;
 - simulated annealing with a geometric schedule (EL3 `SimulatedAnnealing`, EL4 `SimulatedAnnealing<Classic>`);
 - the schedule of simulated annealing, the same in both: `tsp` starts at 100, cools by 0.95 every 20000 proposals and ends at 0.5; `tsp-union` starts at 100, cools by 0.95 every 20000 proposals and ends at 0.5; `assignment` starts at 50, cools by 0.95 every 2000 proposals and ends at 0.5; `exam` starts at 500, cools by 0.95 every 2000 proposals and ends at 1.
- **Speed-up**: $\text{ns/eval of EasyLocal 3} / \text{ns/eval of EasyLocal 4}$; in bold when EasyLocal 4 is faster.
- **Cost**: the final cost, mean over the seeds (lower is better). The two frameworks may follow different trajectories (see above), so the costs may differ even where the speed is the same.
- **Time**: the seconds of the search alone, median over the repetitions, mean over the seeds. It depends on how many evaluations the trajectory takes, and the first descents of the two frameworks, scanning from a random move and from the first one, may take very different numbers of them. Compare times within a framework, and ns/eval across them.
- **ns/eval**: the time divided by the evaluations of solutions and moves.

Speed-up of EasyLocal 4 over EasyLocal 3, by delta mode.

Delta mode all

Every cost component has a delta evaluation.

Instance	Algo- rithm	Speed- up	Cost EL3	Cost EL4	Time EL3 (s)	Time EL4 (s)	ns/eval EL3	ns/eval EL4
as- signment-1000x20	steepest descent	1.05 ×	302 032.7	302 050	1.226	1.139	845.2	806.3
as- signment-1000x20	first descent	1.04 ×	302 040.7	302 047.7	0.021	0.020	879.3	846.8
as- signment-1000x20	simu- lated an- neal- ing	1.09 ×	302 007	302 006.3	0.175	0.160	970.7	887.0
exam-400x20	steepest descent	1.04 ×	8 504.7	8 456	0.770	0.720	344.8	331.1
exam-400x20	first descent	1.24 ×	8 374	8 295.3	0.026	0.025	343.4	276.3
exam-400x20	simu- lated an- neal- ing	1.23 ×	8 021.3	8 011.3	0.099	0.080	404.2	328.1
tsp-300	steepest descent	4.60 ×	14 255.3	14 109.3	0.690	0.152	45.6	9.9
tsp-300	first descent	5.16 ×	14 392.3	14 351	0.049	0.013	45.9	8.9
tsp-300	simu- lated an- neal- ing	2.64 ×	13 557	13 573.3	0.248	0.094	119.4	45.2
tsp-union-300	steepest descent	1.73 ×	14 515.7	14 447.7	1.722	0.996	58.8	33.9
tsp-union-300	first descent	3.03 ×	14 780.3	14 456.7	0.110	0.031	58.7	19.4
tsp-union-300	simu- lated an- neal- ing	1.66 ×	13 820.3	13 693.7	0.307	0.185	147.8	88.8

Delta mode mixed

Some cost components have a delta evaluation, the others are evaluated on a copy of the solution with the move applied; in the union of neighborhoods, the delta of the 2-opt moves only.

Instance	Algo- rithm	Speed- up	Cost EL3	Cost EL4	Time EL3 (s)	Time EL4 (s)	ns/eval EL3	ns/eval EL4
as- signment-1000x20	steepest descent	1.74 ×	302 032.7	302 050	2.267	1.266	1563.3	896.1
as- signment-1000x20	first descent	1.69 ×	302 040.7	302 047.7	0.037	0.021	1512.3	896.5
as- signment-1000x20	simu- lated anneal- ing	1.78 ×	302 007	302 006.3	0.294	0.165	1631.1	915.8
exam-400x20	steepest descent	1.86 ×	8 504.7	8 456	1.318	0.691	589.6	317.7
exam-400x20	first descent	1.78 ×	8 374	8 295.3	0.045	0.030	587.7	329.6
exam-400x20	simu- lated anneal- ing	1.75 ×	8 021.3	8 011.3	0.163	0.093	667.8	382.3
tsp-union-300	steepest descent	2.02 ×	14 515.7	14 447.7	21.631	10.717	738.4	365.1
tsp-union-300	first descent	1.97 ×	14 780.3	14 456.7	1.595	0.698	849.7	431.6
tsp-union-300	simu- lated anneal- ing	1.95 ×	13 820.3	13 693.7	1.725	0.883	829.1	424.7

Delta mode none

No delta evaluation: every move is applied to a copy of the solution and its cost computed from scratch.

Instance	Algo- rithm	Speed- up	Cost EL3	Cost EL4	Time EL3 (s)	Time EL4 (s)	ns/eval EL3	ns/eval EL4
as- signment-1000x20	steepest descent	2.03 ×	302 032.7	302 050	3.404	1.633	2346.9	1155.9
as- signment-1000x20	first descent	2.07 ×	302 040.7	302 047.7	0.059	0.028	2433.8	1174.0
as- signment-1000x20	simu- lated an- neal- ing	1.95 ×	302 007	302 006.3	0.437	0.224	2428.3	1243.8
exam-400x20	steepest descent	2.00 ×	8 504.7	8 456	32.475	15.801	14534.3	7261.1
exam-400x20	first descent	2.02 ×	8 374	8 295.3	1.058	0.625	13929.6	6907.0
exam-400x20	simu- lated an- neal- ing	2.00 ×	8 021.3	8 011.3	3.420	1.706	14016.6	6992.1
tsp-300	steepest descent	2.01 ×	14 255.3	14 109.3	21.841	10.999	1441.9	717.7
tsp-300	first descent	2.00 ×	14 392.3	14 351	1.530	1.047	1437.5	718.7
tsp-300	simu- lated an- neal- ing	2.01 ×	13 557	13 573.3	3.167	1.572	1522.5	755.8
tsp-union-300	steepest descent	2.01 ×	14 515.7	14 447.7	42.036	20.921	1435.0	712.7
tsp-union-300	first descent	2.03 ×	14 780.3	14 456.7	2.699	1.145	1438.1	708.0
tsp-union-300	simu- lated an- neal- ing	1.96 ×	13 820.3	13 693.7	3.171	1.618	1524.6	777.7

Infrastructure of EasyLocal 4 main-4fc56ca

Runner-level search

Median ns per evaluation over the trials, per variant; the fastest in bold.

A stock runner, First Improvement or Best Improvement, on two problems of the EasyLocal examples. The variants are the ways the runner reaches the moves of the neighborhood.

- **Domains:**
 - assignment: Assignment, 96 jobs on 8 machines, job reassignment moves;
 - tsp: TSP, 64 cities, 2-opt moves;
 - tsp-union: TSP, 64 cities, the union of the 2-opt and the swap neighborhoods (neighborhood_union), each child with its own tour-length delta.
- **Algorithms:**
 - first-improvement: `runners::FirstImprovement`, which applies the first improving move of each scan;
 - best-improvement: `runners::BestImprovement`, which scans the whole neighborhood and applies its best move.
- **Variants:**
 - raw-cursor: the algorithm written by hand, calling `first_move` and `next_move` of a cursor explorer of the benchmark (the one the examples had until EasyLocal 4.0.0-alpha.1) directly, without a range and outside the runner framework: the reference the others are measured against;
 - cursor-range: the stock runner iterating `easylocal::moves`, which turns the `first_move/next_move` cursor of the same explorer into a lazy range holding one move, without materializing the neighborhood: the difference from raw-cursor is the cost of the range and the runner;
 - coroutine-range: the stock runner on the example's explorer, whose `moves()` is a coroutine (`easylocal::generator`, which is `std::generator` where the standard library provides it), as a user would write it;
 - union: the neighborhood_union of the two neighborhoods, whose iterator and delta bindings dispatch on the variant that is its move (tsp-union only);
 - child-after-child: the same moves, of the same type and in the same order, from one child and then the other, with one delta that visits the move and calls the delta of the child it comes from: the union's own machinery written by hand (tsp-union only).

Runner-level search, ns per evaluation.

Domain	Algo- rithm	child-after- child (ns/ eval)	coroutine- range (ns/ eval)	cursor- range (ns/ eval)	raw-cursor (ns/eval)	union (ns/ eval)
assign- ment	best-im- prove- ment	–	37.00	31.64	27.74	–
assign- ment	first-im- prove- ment	–	36.30	31.51	27.13	–
tsp	best-im- prove- ment	–	9.20	5.90	4.90	–
tsp	first-im- prove- ment	–	8.96	5.60	4.46	–
tsp- union	best-im- prove- ment	34.23	–	–	–	30.87
tsp- union	first-im- prove- ment	30.77	–	–	–	27.77

Neighborhood traversal

Median ns per move over the trials, per variant; the fastest in bold.

The moves of a whole neighborhood, visited without a runner.

- **Domains:**

- assignment-medium: Assignment, 256 jobs on 32 machines, job reassignment moves;
- tsp-medium: TSP, 256 cities, 2-opt moves;
- tsp-union-medium: TSP, 256 cities, the union of the 2-opt and the swap neighborhoods (neighborhood_union), enumerated one child after the other.

- **Workloads:**

- traversal: reads only the fields of each move, so the time is the cost of producing the moves;
- light-consumer: also reads the data of the solution and of the instance the move touches (demands and capacities, distances), as a minimal evaluation would.

- **Variants:**

- cursor: a cursor explorer of the benchmark, with first_move and next_move (the one the examples had until EasyLocal 4.0.0-alpha.1), iterated through easylocal::moves;
- coroutine-custom: moves() as a coroutine of a minimal generator of the benchmark, like the easylocal::generator of a standard library without std::generator;
- coroutine-std: moves() as a std::generator, where the standard library provides it;
- union: the neighborhood_union of the two neighborhoods, whose iterator holds a variant of its children's and visits it to advance and to build each move;
- child-after-child: the same moves, of the same type and in the same order, enumerated by one child and then the other without that iterator: what the union's dispatch costs is the difference.

Neighborhood traversal, ns per move.

Domain	Work-load	child-after-child (ns/ move)	coroutine-custom (ns/ move)	coroutine-std (ns/ move)	cursor (ns/ move)	union (ns/ move)
assignment-medium	light-consumer	–	4.14	4.88	2.76	–
assignment-medium	traversal	–	3.83	4.11	2.00	–
tsp-medium	light-consumer	–	6.10	6.56	4.53	–
tsp-medium	traversal	–	4.43	4.14	1.30	–
tsp-union-medium	light-consumer	22.74	–	–	–	6.07
tsp-union-medium	traversal	13.19	–	–	–	3.65

Tracing overhead

Median ns per evaluation of a First Improvement run, without a tracer and with each tracer; the slowdown is their ratio.

A First Improvement run on an Assignment instance of 96 jobs on 8 machines, traced by each tracer; “without” is the same run with no tracer argument.

• Tracers:

- ▶ explicit-null: `trace::null_tracer` passed explicitly; it should cost nothing;
- ▶ counting: a tracer that only counts the events: the cost of producing them;
- ▶ memory: `trace::memory_recorder`, every event kept in memory;
- ▶ binary-buffered-discard: `trace::binary_recorder` (the ELTR binary format) writing to a stream that discards the bytes: encoding and buffering without input/output;
- ▶ binary-buffered-discard-visited: the same, on a problem with a solution hash, which records a `solution_visited` event with the hash of each solution the search moves to;
- ▶ binary-buffered-discard-without-visits: the same problem, with `solution_visited` left out at compile time (`trace::without<event::solution_visited>`); it should cost what binary-buffered-discard costs;
- ▶ binary-async-producer-discard: `trace::async_binary_recorder`, which hands the events to a writer thread, writing to a discarding stream; the time of the search until it ends;
- ▶ binary-async-total-discard: the same, also waiting for the writer to finish;
- ▶ binary-buffered-file: `trace::binary_recorder` writing a temporary file (without `fsync`);
- ▶ binary-async-producer-file: `trace::async_binary_recorder` writing a temporary file; the time of the search until it ends;
- ▶ binary-async-total-file: the same, also waiting for the writer to finish.

Tracer	ns/eval without	ns/eval with	Slow-down
explicit-null	21.03	20.85	0.99×
counting	21.03	20.69	0.98×
memory	21.03	85.60	4.07×
binary-buffered-discard	21.03	38.81	1.85×
binary-buffered-discard-visited	21.03	40.54	1.93×
binary-buffered-discard-without-visits	21.03	39.09	1.86×
binary-async-producer-discard	21.03	41.29	1.96×
binary-async-total-discard	21.03	41.30	1.96×
binary-buffered-file	21.03	50.98	2.42×
binary-async-producer-file	21.03	48.50	2.31×
binary-async-total-file	21.03	48.55	2.31×

ELTR cost encoding

Median ns per encoded event, by the layout of the cost.

A `move_evaluated` event, which carries the current and the candidate cost, encoded one million times in the ELTR binary format; bytes/event is the size of the encoded event.

• Costs:

- `scalar-i64`: a 64-bit integer;
- `lexicographic-2xi64`: a `cost::lexicographic` of two 64-bit integers;
- `hierarchical-2xi64-plus-i64`: a `cost::hierarchical` with that lexicographic cost as hard cost and a 64-bit integer as soft cost.

Cost	ns/event	bytes/event
scalar-i64	6.65	41
lexicographic-2xi64	7.19	57
hierarchical-2xi64-plus-i64	8.51	73

Compilers and architectures of EasyLocal 4 main-4fc56ca

The EasyLocal 4 driver of the comparison above, built by several toolchains and run on the same instances, algorithms and seeds, in delta mode `all`. Each platform runs in a job of its own, on its own machine, and its toolchains alternate run by run, so the times of one platform are comparable with each other; the platforms are different (shared) machines, so their absolute times are not. The descents follow the same trajectory with every toolchain; simulated annealing may not, as the standard libraries draw random numbers differently, hence ns/eval rather than times.

	Linux x86_64	Linux ARM64	macOS ARM64
Operating system	Ubuntu 26.04.1 LTS, kernel 7.0.0-1012-azure	Ubuntu 26.04.1 LTS, kernel 7.0.0-1012-azure	macOS 26.6.2 (25G83), kernel 25.6.0
Architecture	x86_64, AMD EPYC 9V74 80-Core Processor, 4 logical CPUs	aarch64, Neoverse-N2, 4 logical CPUs	arm64, Apple M1 (Virtual) (VirtualMac2,1), 3 logical CPUs
Virtual machine	Microsoft (full virtualization), size Standard_D4ads_v6	–	–
Caches	L1d 64 KiB (2 instances), L1i 64 KiB (2 instances), L2 2 MiB (2 instances), L3 32 MiB (1 instance)	L1d 256 KiB (4 instances), L1i 256 KiB (4 instances), L2 4 MiB (4 instances), L3 128 MiB (1 instance)	L1d 128 KiB, L2 12288 KiB
Frequency	3717 MHz nominal	–	–
Memory	15.6 GiB	15.6 GiB	7.0 GiB
Instruction sets	sse4_2, avx, avx2, fma, bmi2, avx512f, avx512bw, avx512vl, avx512_vnni, sha_ni, aes	aes	–
CPU steal during the run	0.0% (busy 25%)	0.0% (busy 25%)	–
Build	cmake version 4.4.3, Ninja, Release, C++23	cmake version 4.4.3, Ninja, Release, C++23	cmake version 4.4.3, Ninja, Release, C++23

Linux x86_64

Measured on 2026-10-11.

Toolchain	Compiler	Standard library	Flags
gcc16	gcc 16.0.1 20260322 (experimental) [trunk r16-8246-g569ace1fa50]	libstdc++ 20260322	-O3 -DNDEBUG
clang23.1.3 (trunk)	clang 23.1.3 (trunk) 20260922084409+67f4a076a097-1_exp120260922084409+67f4a076a097-1	libstdc++ 20260322	-stdlib=libstdc++ --gcc-install-dir=/usr/lib/gcc/x86_64-linux-gnu/16 -O3 -DNDEBUG
clang23.1.3 (trunk)	clang 23.1.3 (trunk) 20260922084409+67f4a076a097-1_exp120260922084409+67f4a076a097-1	libc++ 20260322	-stdlib=libc++ -O3 -DNDEBUG

Instance	Algorithm	gcc16 (ns/eval)	clang23-libstdc++ (ns/eval)	clang23-libc++ (ns/eval)
assignment-1000x20	steepest descent	696.1	1153.4	1165.8
assignment-1000x20	first descent	704.9	1154.9	1151.5
assignment-1000x20	simulated annealing	739.1	1341.5	1366.5
exam-400x20	steepest descent	236.4	230.5	231.1
exam-400x20	first descent	240.1	226.4	321.8
exam-400x20	simulated annealing	287.8	273.0	287.1
tsp-300	steepest descent	7.1	6.3	5.9
tsp-300	first descent	6.1	4.8	5.2
tsp-300	simulated annealing	34.0	51.6	75.5
tsp-union-300	steepest descent	28.4	10.2	27.0
tsp-union-300	first descent	16.3	10.8	12.6
tsp-union-300	simulated annealing	71.9	70.7	115.6

Time per evaluation relative to gcc16, geometric mean over the rows (below 1 is faster): clang23-libstdc++ 1.00×, clang23-libc++ 1.23×.

Linux x86_64: time per evaluation relative to gcc16.

Linux ARM64

Measured on 2026-10-11.

Toolchain	Compiler	Standard library	Flags
gcc16	gcc 16.0.1 20260322 (experimental) [trunk r16-8246-g569ace1fa50]	libstdc++ 20260322	-O3 -DNDEBUG
clang23-libstdc++ (experimental)	clang 23.1.3 (trunk r16-8246-g569ace1fa50)	libstdc++ 20260322	-std=libstdc++ --gcc-install-dir=/usr/lib/gcc/aarch64-linux-gnu/16 -O3 -DNDEBUG
clang23-libc++	clang 23.1.3 (trunk r16-8246-g569ace1fa50)	libc++ 20260322	-std=libc++ -O3 -DNDEBUG

Instance	Algorithm	gcc16 (ns/eval)	clang23-libstdcxx (ns/eval)	clang23-libcxx (ns/eval)
assign-ment-1000x20	steepest descent	2154.8	1331.6	1316.8
assign-ment-1000x20	first descent	2972.6	1379.6	1403.8
assign-ment-1000x20	simulated annealing	3300.0	1520.3	1539.5
exam-400x20	steepest descent	286.4	280.4	278.3
exam-400x20	first descent	280.7	274.2	276.6
exam-400x20	simulated annealing	315.4	311.2	325.2
tsp-300	steepest descent	8.4	8.7	7.4
tsp-300	first descent	7.6	5.7	6.1
tsp-300	simulated annealing	35.1	39.4	62.9
tsp-union-300	steepest descent	13.5	12.0	14.9
tsp-union-300	first descent	11.0	11.2	14.8
tsp-union-300	simulated annealing	48.5	57.2	89.7

Time per evaluation relative to gcc16, geometric mean over the rows (below 1 is faster): clang23-libstdcxx 0.83×, clang23-libcxx 0.94×.

Linux ARM64: time per evaluation relative to gcc16.

macOS ARM64

Measured on 2026-10-11.

Toolchain	Compiler	Standard library	Flags
appleclang	appleclang 21.0.0 (clang-2100.1.1.101)	libc++ 210106	-03 -DNDEBUG

Instance	Algorithm	appleclang (ns/eval)
assignment-1000x20	steepest descent	1477.5
assignment-1000x20	first descent	1498.7
assignment-1000x20	simulated annealing	1725.8
exam-400x20	steepest descent	324.4
exam-400x20	first descent	311.8
exam-400x20	simulated annealing	363.6
tsp-300	steepest descent	7.8
tsp-300	first descent	4.5
tsp-300	simulated annealing	64.8
tsp-union-300	steepest descent	12.8
tsp-union-300	first descent	11.5
tsp-union-300	simulated annealing	87.4

```

<script> (function () { function scheme() { var dark = document.body.getAttribute("data-md-color-scheme") === "slate"; var text = getComputedStyle(document.body).color; var grid = dark ? "rgba(255,255,255,0.15)" : "rgba(0,0,0,0.12)"; return { font: { color: text, family: "inherit" }, grid: grid }; } function draw() { var theme = scheme(); document.querySelectorAll(".benchmark-chart").forEach(function (element) { var figure = JSON.parse(element.getAttribute("data-figure")); var layout = figure.layout; layout.paper_bgcolor = "rgba(0,0,0,0)"; layout.plot_bgcolor = "rgba(0,0,0,0)"; layout.font = theme.font; ["xaxis", "yaxis"].forEach(function (axis) { layout[axis] = layout[axis] || {}; layout[axis].grid-color = theme.grid; layout[axis].zerolinecolor = theme.grid; }); (layout.shapes || []).forEach(function (shape) { shape.line = shape.line || {}; shape.line.color = theme.font.color; }); Plotly.react(element, figure.data, layout, { responsive: true, displaylogo: false, modeBarButtonsToRemove: ["select2d", "lasso2d", "autoScale2d"] }); }); if (!document.querySelector(".benchmark-chart")) { return; } var script = document.createElement("script"); script.src = "https://cdn.plot.ly/plotly-basic-4.1.1.min.js"; script.onload = function () { draw(); new MutationObserver(draw).observe( document.body, { attributes: true, attributeFilter: ["data-md-color-scheme"] }); }); document.head.appendChild(script); }); </script>

```