What's new since ND24 - the shortlist



Fabian Brandt-Tumescheit, HU Berlin

NetworKit Day 2026, 10 November 2026

Updates since Apr 2024 (ND24, ~770 commits)



New Algorithms

Planarity (new module)

contrib

Left-Right Planarity Test
`planarity.LeftRightPlanarityCheck`

Graph & Distance

contrib

Floyd-Warshall (all-pairs shortest paths)
`distance.FloydWarshall`

contrib

Prim Minimum Spanning Forest
`graph.PrimMSF`

contrib

Bipartiteness check
`graphtools.isBipartite`

Edge scores

contrib

SimRank
`sparsification.SimRankScore`



planar or not? K_5 and $K_{3,3}$ are the two minimal non-planar graphs

New Algorithms

Flow

contrib

Dinic (maximum flow)
`flow.Dinic`

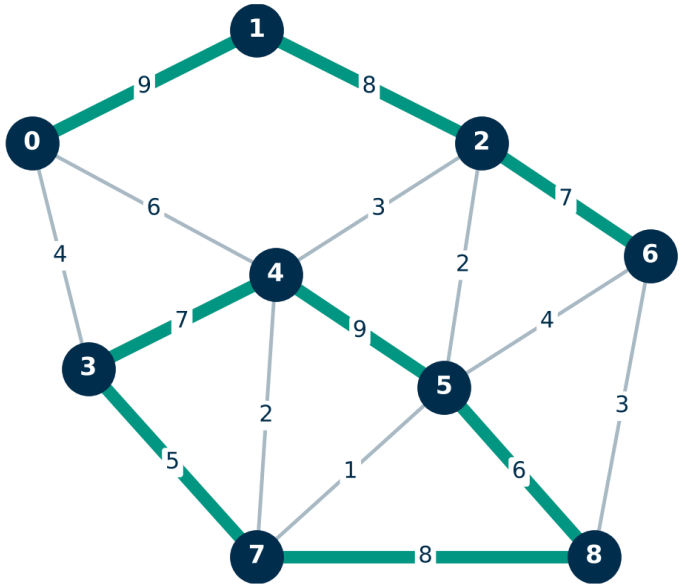
contrib

Successive Shortest Path (minimum-cost flow)
`flow.SuccessiveShortestPathMinCostFlow`

Matching

b-Suitor (½-approx. maximum weight b-matching)
`matching.BSuitorMatcher`

Dynamic b-Suitor
`matching.DynamicBSuitorMatcher`



b-matching with b = 2, computed with nk.matching.BSuitorMatcher



Improved Algorithms

Generators (parallel)

Barabási-Albert (parallel variant, Sanders & Schulz)
`generators.BarabasiAlbertGenerator`

R-MAT (parallel, > 2^{32} nodes)
`generators.RmatGenerator`

Chung-Lu (Alam et al.) and configuration model
`NetworKit::ChungLuGeneratorAlamEtAl`, `NetworKit::ConfigurationModel`

Numerics

LAMG solver: connected components solved in parallel
`NetworKit::Lamg`



LAMG solver: disconnected graphs in all setup functions
`NetworKit::Lamg`

Centrality



TopCloseness (faster)
`centrality.TopCloseness`

DynBetweennessOneNode (weighted graphs)
`centrality.DynBetweennessOneNode`

Community & Graph



Topological sort (custom node id map)
`graphtools.topologicalSort(G, nodeIdMap)`

Viz



Maxent-Stress (warm start)
`viz.MaxentStress(G, coordinates)`

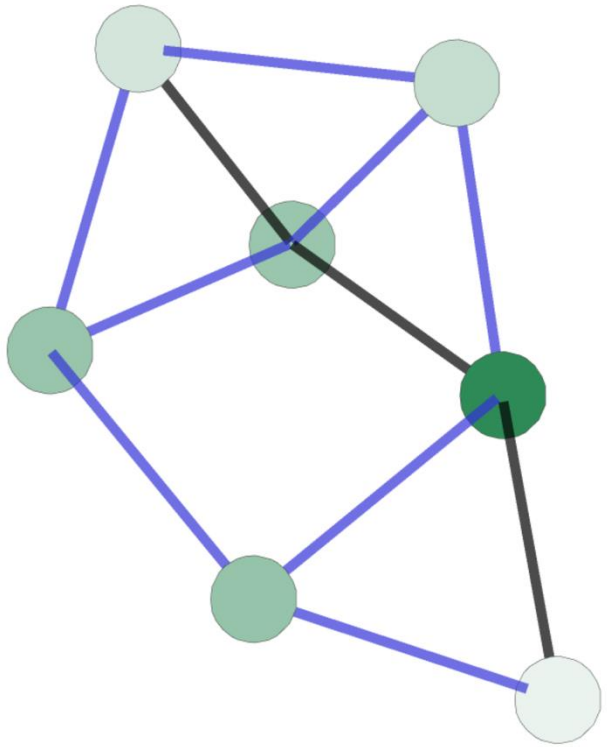
Improved Modules

vizbridges module `nk.vizbridges`

- Edge attributes as edge labels
- Scores mapped linearly to colors
- Maxent-Stress (3D): warm start with 3D coordinates

contrib

contrib



```
G.indexEdges()
att = G.attachEdgeAttribute("label", str)
for u, v in G.iterEdges():
    att[u, v] = f"{u}-{v}"
nk.vizbridges.widgetFromGraph(G, edgeAttributes=[("label", str)])
```



New Features

Ecosystem / Usability

Scores as NumPy array

```
bc = nk.centrality.Betweenness(G).run()
bc.scores(asarray=True)    # numpy.ndarray
bc.compactScores()         # existing nodes only
```



SciPy CSR → Graph

```
G = nk.GraphFromCsr(csr, directed=False, weighted=True)
```

COO keeps weights

```
G = nk.GraphFromCoo(scipy_coo, weighted=True)
```



int → float attributes

```
att = G.attachNodeAttribute("score", float)
att[0] = 1                 # int accepted
```

MatrixMarket, RB readers

```
G = nk.readGraph("matrix.mtx", nk.Format.MatrixMarket)
```

Other

- NumPy 2 support
- Wheels for Python 3.10 – 3.15
- C++20: GCC ≥ 10, Clang ≥ 11, MSVC ≥ 14.30 (TODO: Update after release)
- Stable and nightly documentation
- Nightly builds from master

3.10–3.15
Python wheels (11.2.2)

2.x
NumPy supported

C++20
language standard

```
pip install --upgrade --extra-index-url https://pypi.anaconda.org/networkit-nightly/simple networkit-nightly
```

Docs: networkit.github.io/dev-docs (stable) and networkit.github.io/dev-docs/nightly (nightly)



Under the hood

Code base

C++20

`concepts, std::span, std::ranges, std::string_view`

Tests and CI: many new unit tests, CI on forks, leak + coverage builds

`clang-format / clang-tidy 22`

Graph data structures

- Templated graph

  `NetworkKit::AdjListGraph<NodeT, EdgeWeightT>`

- Dynamic graph: DHB (experimental)

  `NetworkKit::DHBGraph`

GraphBuilder: parallel by default

`NetworkKit::GraphBuilder::completeGraph()`

Templated Graph

`Graph` = `AdjListGraph<node, edgeweight>` → existing code keeps working

"Why pay for 64-bit node ids if 2^{32} nodes are enough?"

NodeT (concept GraphNode)	EdgeWeightT (GraphEdgeWeight)
integral type, e.g. uint32_t	integral or floating point, e.g. float

```
using SmallGraph = NetworKit::AdjListGraph<uint32_t, float>;

SmallGraph G(1000, /*weighted=*/true);
G.addEdge(0, 1, 2.5f);
NetworKit::BreadthFirstSearch<SmallGraph> bfs(G, 0);
bfs.run();
```

Already templated

- graph + iterators
- BFS, SSSP, Floyd-Warshall
- topological sort, planarity
- Partition, Cover, UnionFind, BucketPQ

Next (#1415)

- readers and writers
- remaining algorithms
- lightweight graphs in Python

Templated Graph

Smaller types → smaller graph, better scalability

35–45 %
faster BFS
uint32_t / float

12–15 %
faster Floyd-Warshall
uint32_t

Preliminary benchmarks from proposal PR #1324; not a systematic evaluation.

- C++ only for now; Python keeps the 64-bit / double graph
- Migration continues algorithm by algorithm (issue #1415)

DHBGraph

based on: *van der Grinten, Predari, Willich. A Fast Data Structure for Dynamic Graphs Based on Hash-Indexed Adjacency Blocks. SEA 2022.*

DHB = Dynamic Hashed Blocks: a data structure for sparse dynamic graphs

"How can we update a graph fast without losing traversal speed?"

Adjacency blocks	Hash index
incident edges in blocks: cache-friendly traversal	fast insert/lookup, no duplicates

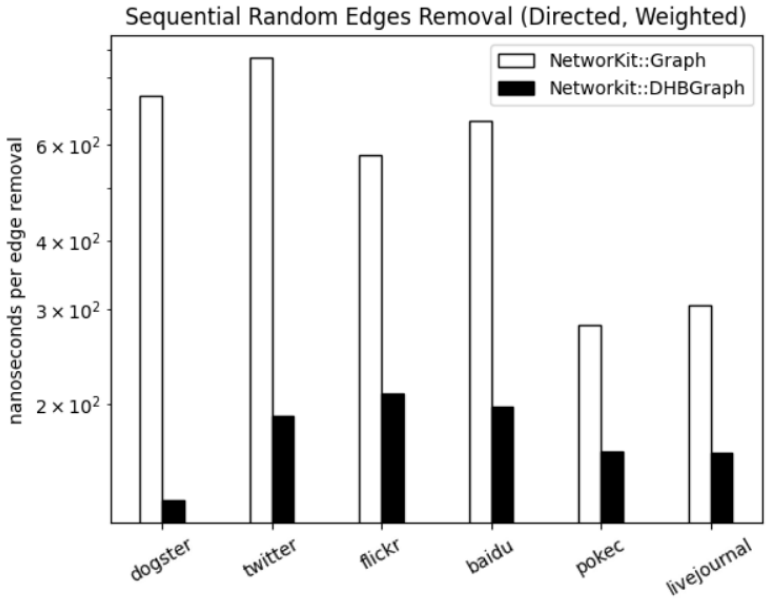
schematic



per vertex: a block of neighbors + hash index

DHBGraph

- Faster insertions, updates, deletions and traversal than competing dynamic graph data structures (paper)
- Only a small traversal overhead compared to static CSR (paper)
- New C++ class with an interface aligned to Graph: batch insertion, parallel iteration
- **Experimental**: not yet a drop-in replacement for Graph, not templated, no Python bindings yet




```
NetworKit::DHBGraph G(n);
G.addEdges(std::move(edges), /*do_update=*/true);
G.parallelForEdges([&](node u, node v) { /* ... */ });
```