Graph Library: Overview

Document #:	**P3126r0**
Date:	2024-02-05
Project:	Programming Language C++
Audience:	Library Evolution
	SG19 Machine Learning
	SG14 Game, Embedded, Low Latency
	SG6 Numerics
Revises:	P1709r5

Reply-to:	Phil Ratzloff (SAS Institute)
	phil.ratzloff@sas.com
	Andrew Lumsdaine
	lumsdaine@gmail.com
Contributors:	Kevin Deweese
	Muhammad Osama (AMD, Inc)
	Jesun Firoz
	Michael Wong (Codeplay)
	Jens Maurer
	Richard Dosselmann (University of Regina)
	Matthew Galati (Amazon)

1 Getting Started

This paper is one of several interrelated papers for a proposed Graph Library for the Standard C++ Library. The Table 1 describes all the related papers.

Paper	Status	Description
P1709	Inactive	Original proposal, now separated into the following papers.
P3126	Active	**Overview**, describing the big picture of what we are proposing.
P3127	Active	**Background and Terminology** providing the motivation, theoretical background and terminology used across the other documents.
P3128	Active	**Algorithms** covering the initial algorithms as well as the ones we'd like to see in the future.
P3129	Active	**Views** has helpful views for traversing a graph.
P3130	Active	**Graph Container Interface** is the core interface used for uniformly accessing graph data structures by views and algorithms. It is also designed to easily adapt to existing graph data structures.
P3131	Active	**Graph Containers** describing a proposed high-performance `compressed_graph` container. It also discusses how to use containers in the standard library to define a graph, and how to adapt existing graph data structures.

Table 1: Graph Library Papers

Reading them in order will give the best overall picture. If you're limited on time, you can use the following guide to focus on the papers that are most relevant to your needs.

Reading Guide

— If you're **new to the Graph Library**, we recommend starting with the *Overview* paper (P3126) to understand focus and scope of our proposals.

— If you want to **understand the theoretical background** that underpins what we're doing, you should read the *Background and Terminology* paper (P3127).

— If you want to **use the algorithms**, you should read the *Algorithms* paper (P3128) and *Graph Containers* paper (P3131).

— If you want to **write new algorithms**, you should read the *Views* paper (P3129), *Graph Container Interface* paper (P3130) and *Graph Containers* paper (P3131). You'll also want to review existing implementations in the reference library for examples of how to write the algorithms.

— If you want to **use your own graph container**, you should read the *Graph Container Interface* paper (P3130) and *Graph Containers* paper (P3131).

2 Revision History

P3126r0

— Split from P1709r5. Added *Getting Started* section.

— Rewrite *Goals and Priorities* section to reflect the structure of the papers and to include a section on our *Future Roadmap*.

— Added *Notes and Considerations* section.

3 Overview

Graphs, used in ML and other **scientific** domains, as well as **industrial** and **general** programming, do **not** presently exist in the C++ standard. In ML, a graph forms the underlying structure of an **artificial neural**

network (ANN). In a **game**, a graph can be used to represent the **map** of a game world. In **business** environments, graphs arise as **entity relationship diagrams** (ERD) or **data flow diagrams** (DFD). In the realm of **social media**, a graph represents a **social network**.

All documents, taken as a whole for a Graph Library, proposes the addition of **graph algorithms, operators, views, adaptors**, the **graph container interface** and a **graph container implementation** to the C++ library to support **machine learning** (ML), as well as other applications. ML is a large and growing field, both in the **research community** and **industry**, that has received a great deal of attention in recent years. This documents presents an **interface** of the proposed algorithms, operators, adaptors, views, graph functions and containers.

4 Goals and Priorities

Because graphs and their algorithms cover a broad range of capabilities and implementations, we have defined a focused set of goals and priorities that will provide an initial set of useful functionality, as well as a sound foundation for future work.

- Provide a firm theoretical foundation for the library.
- Follow the separation of algorithms, ranges, views and containers established by the standard library.
- Include a rich enough set of algorithms for the library to be useful.
  - The syntax for an algorithm's implementation should be simple, expressive and easy to understand.
  - The ability to write high-performance algorithms should not be compromised.
  - Algorithms can expect vertices to be in a random access range with an integral vertex id initially.
- Include views for common traversals of a graph's vertices and edges that is concise and consistant without having to use a lower level interface.
  - Simple views for vertexlist, incidence edges on a vertes, neighbors of a vertex, and edges of a graph.
  - Complex views for depth-first search, breath-first search, and topological sort.
- A Graph Container Interface, used by Views and Algorithms, that provides a consistent interface for different graph data structures. The interface includes concepts, types, traits and functions and provides a similar role to the Ranges library for standard containers.
  - Descriptors for a consistent data model for vertex, edge and neighbor by views and edge lists.
  - Adjacency list, an outer range of vertices with an inner range of outgoing edges on each vertex.
    - Be able to use the algorithms and views with existing graph data structures using customization points.
    - Support for optional user-defined value types on an edge, vertex and/or the graph itself.
    - Support bipartite and multipartite graphs.
  - Edge list, which is a range of edge descriptors.
    - From an `edgelist` view.
    - From a user-defined range that uses `std::ranges::transform_view`.
- Provide one or more graph containers that can be used with the algorithms.
  - A high-performance `compressed_graph` container, based on the Compressed Sparse Row matrix.
  - The ability to create simple graph container from standard containers, e.g. `vector<vector<int>>`.

The design should not hinder the ability to extend the functionality to support expanded functionality identified in the future roadmap that follows.

4.1 Future Roadmap

The following are areas we'd like to see in future proposals, after the initial proposals are accepted. We endeavor to investigate these to assure the existing design will support them.

— Additional graph algorithms. The Graph Algorithms paper identifies tiers of algorithms we'd like to see added in the future, including parallel algorithms.

— Support for sparse vertex ids, implying the use of bi-directional containers such as `map` and `unordered_map` for vertices.

— Bi-directional graphs, where vertices have incoming and outgoing edges.

— Non-integral vertex ids.

— Constexpr graphs, where vertices and edges are stored in `std::array` or other constexpr-friendly container.

5 Examples

The following code demonstrates how a simple graph can be created as a range of ranges, using the standard containers.

```
std::vector<std::string> actors { "Tom Cruise", "Kevin Bacon", "Hugo Weaving",
                                  "Carrie-Anne Moss", "Natalie Portman", "Jack Nicholson",
                                  "Kelly McGillis", "Harrison Ford", "Sebastian Stan",
                                  "Mila Kunis", "Michelle Pfeiffer", "Keanu Reeves",
                                  "Julia Roberts" };

using G = std::vector<std::vector<int>>;
G costar_adjacency_list{
    {1, 5, 6}, {7, 10, 0, 5, 12}, {4, 3, 11}, {2, 11}, {8, 9, 2, 12}, {0, 1}, {7, 0},
    {6, 1, 10}, {4, 9}, {4, 8}, {7, 1}, {2, 3}, {1, 4} };

int main() {
  std::vector<int> bacon_number(size(actors));

  // 1 -> Kevin Bacon
  for (auto&& [uid,vid] : basic_sourced_edges_bfs(costar_adjacency_list, 1)) {
    bacon_number[vid] = bacon_number[uid] + 1;
  }

  for (int i = 0; i < size(actors); ++i) {
    std::cout << actors[i] << " has Bacon number " << bacon_number[i] << std::endl;
  }
}
```

`target_id(g,uv)` defines the required function to get a target_id for an edge in the graph `G`. Other functions can also be overridden to allow a developer to adapt their own graph data structures to the library.

6 What this proposal is not

The Graph Library proposal limits itself to adjacency graphs and edgelists only. An adjacency graph is an outer range of vertices with an inner range of outgoing edges on each vertex. An edgelist is a view of edges, which is either all the edges in the adjacency graph or a projection of a user-defined range.

Parallel versions of the algorithms are not included for several reasons. The executors proposal in P2300r5 [1] is expected to introduce new and better ways to do parallel algorithms beyond that used in the parallel STL algorithms and we would like to wait for finalization of that proposal before committing to parallel implementations. Secondly, many graph algorithms don't benefit from parallel implementations so there is less need to offer an implementation. Lastly, it will help limit the size of this proposal which is already looking to be large without it. It is expected that future proposals will be submitted for parallel graph algorithms.

Hypergraphs are not supported.

7 Impact on the Standard

This proposal is a pure **library** extension.

8 Interaction wtih Other Papers

Other than the papers identified as part of the Graph Libary, there is no interaction with other proposals to the standard.

9 Implementation Experience

The github github.com/stdgraph repository contains an implementation for this proposal.

10 Usage Experience

There is no current use of the library. There are plans to begin using it in 2024 in a commercial setting.

11 Deployment Experience

There is no current deployment experience of the library. There are plans for this to follow the usage experience.

12 Performance Considerations

The algorithms are being ported from NWGraph to the github.com/stdgraph implementation used for this proposal. Performance analysis from those algorithms can be found in the peer-reviewed papers for NWGraph [2, 3].

13 Prior Art

boost::graph has been an important C++ graph implementation since 2001. It was developed with the goal of providing a modern (at the time) generic library that addressed all the needs someone would want of a graph library. It is still a viable library used today, attesting to the value it brings.

However, boost::graph was written using C++98 in an "expert-friendly" style, adding many abstractions and using sophisticated tempate metaprogramming, making it difficult to use by a casual developer.

(Andrew is a co-author of boost::graph.)

NWGraph ([4] and [2]) was published in 2022 by Lumsdaine et al, bringing additional experience gained since creating boost::graph, to create a modern graph library using C++20 for its implementation that was more accessible to the average developer.

While NWGraph made important strides to introduce the idea of the graph as a range-of-ranges and implemented many important algorithms, there are some areas it didn't address that come a practical use in the field. For instance, it didn't have a well-defined API for graph data structures that could be applied to existing graphs, and there wasn't a uniform approach to properties.

This proposal takes the best of NWGraph, with previous work done for P1709 to define a Graph Container Interface, to provide a library that embraces performance, ease-of-use and the ability to use the algorithms and views on externally defined graph containers.

14 Alternatives

We're unaware of any other library that meets the same requirements and uses concepts and ranges from C++20.

15 Feature Test Macro

The `__cpp_lib_graph` feature test macro is recommended to represent all features in this proposal including algorithms, views, concepts, traits, types, functions and graph container(s).

16 Freestanding

We believe this library can be used in a freestanding C++ implementation.

17 Namespaces

Graph containers and their views and algorithms are not interchangeable with existing containers and algorithms. Additionally, there are some domain-specific terms that may clash with existing or future names, such as `degree` and `partition_id`. For these reasons, we recommend their own namespaces. The following assumption is used in this proposal.

`std::graph` and `std::graph::views`

Alternative locations include the following:

`std::ranges` and `std::ranges::views`

`std::ranges/graph` and `std::ranges::graph::views`

The advantage of these two options are that there would be no requirement to use the ranges:: prefix for things in the std::ranges namespace, a common occurance.

18 Notes and Considerations

There are some interesting observations that can be made about graphs and how they compare and contrast to the standard library that may not be obvious.

- The adjacency list, the primary data structure for this proposal, is a compound data structure of a range of ranges. This introduces a new form of container beyond a simple range.
- There is more than one possible value type, one each for edge, vertex and graph. Each is optional. This is in contrast to existing practice where the value type is the distinguishing difference between different containers, such as for `set` and `map`.
- Algorithms will often use views, though they can use the GCI functions when needed.
- Algorithms and Views often need to allocate memory internally to achieve their purpose. This is a departure from common practice in the standard.

There are other observations we've also discovered along the way that may not be obvious.

- Storing vertices in a `map` (bi-directional range) requires a different style of programming algorithms, compared to being kept in a `vector` (random access range). When using a `vector`, `edges(g,uid)` would normally be used without much thought. Using that with a `map` would incur a $\mathcal{O}(\log(V))$ cost. Instead, it will use vertex id once to get the vertex reference and then use `edges(g,uv)` . This is expected to result in overloading of existing algorithms based on the range type of container, distinguished with concepts.

Acknowledgements

Phil Ratzloff's time was made possible by SAS Institute.

Portions of *Andrew Lumsdaine's* time was supported by NSF Award OAC-1716828 and by the Segmented Global Address Space (SGAS) LDRD under the Data Model Convergence (DMC) initiative at the U.S. Department of Energy's Pacific Northwest National Laboratory (PNNL). PNNL is operated by Battelle Memorial Institute under Contract DE-AC06-76RL01830.

Michael Wong's work is made possible by Codeplay Software Ltd., ISOCPP Foundation, Khronos and the Standards Council of Canada.

Muhammad Osama's time was made possible by Advanced Micro Devices, Inc.

The authors thank the members of SG19 and SG14 study groups for their invaluable input.

References

[1] Dominiak, Evtushenko, Baker, Teodorescu, Howes, K. Shoop, M. Garland, E. Niebler, and B. Lelbach, “P2300r5 std::execution.” "https://www.open-std.org/jtc1/sc22/wg21/docs/papers/2022/p2300r5.html".

[2] A. Lumsdaine, L. D’Alessandro, K. Deweese, J. Firoz, T. Liu, S. McMillan, P. Ratzloff, and M. Zalewski, “Nwgraph: A library of generic graph algorithms and data structures in c++20.” "https://drops.dagstuhl.de/opus/volltexte/2022/16259/".

[3] A. Azad, M. M. Aznaveh, S. Beamer, M. P. Blanco, J. Chen, L. D’Alessandro, R. Dathathri, T. Davis, K. Deweese, J. Firoz, H. A. Gabb, G. Gill, B. Hegyi, S. Kolodziej, T. M. Low, A. Lumsdaine, T. Manlaibaatar, T. G. Mattson, S. McMillan, R. Peri, K. Pingali, U. Sridhar, G. Szarnyas, Y. Zhang, and Y. Zhang, “Evaluation of graph analytics frameworks using the gap benchmark suite,” in *2020 IEEE International Symposium on Workload Characterization (IISWC)*, pp. 216–227, 2020.

[4] A. Lumsdaine, L. D’Alessandro, K. Deweese, J. Firoz, T. Liu, S. McMillan, P. Ratzloff, and M. Zalewski, “Nwgraph library code.” "https://github.com/pnnl/NWGraph".

[5] J. G. Siek, L.-Q. Lee, and A. Lumsdaine, *The Boost Graph Library: User Guide and Reference Manual*. Addison-Wesley Professional, Dec. 2001.