Published in final edited form as:
J Open Source Softw. ; 4(33): . doi:10.21105/joss.01083.

GFAKluge: A C++ library and command line utilities for the Graphical Fragment Assembly formats

Eric T. Dawson[1,2,*], **Richard Durbin**[1]
[1]Human Genetics, Wellcome Trust Sanger Institute, Hinxton, CB10 1SA, UK
[2]Division of Cancer Epidemiology and Genetics, National Cancer Institute, Rockville, Maryland, 20850, USA

Abstract

Summary—GFA has emerged as a standard format for the exchange of genome assemblies and sequence graphs. To encourage further adoption in high-performance software we have developed an open-source C++ library for GFA and a set of utilities for summarizing and manipulating the format.

Availability—The `gfakluge` source code is freely available under the MIT license at https://github.com/edawson/gfakluge. It has been tested on both Mac OS X and Linux.

Introduction

FASTA has remained the dominant file format for representing genome assemblies for several decades. While it is a standard for representing sets of sequences, FASTA provides little information beyond the contig sequences themselves. Almost all modern assembly software generates assemblies using some form of sequence graph (which we take as including de Bruijn and overlap graphs) [Zerbino and Birney (2008); Simpson *et al.* (2009); Simpson and Durbin (2010)], but conversion to FASTA loses information contained in the graph about possible joins between contigs, and other potentially useful information available to the assembler. Proposed graph-friendly formats such as ASQG [ref] and FASTG [ref] have seen limited adoption. Sequence graphs have also shown promise for read mapping [(Erik Garrison *et al.*, vg: The variation graph toolkit)] and programs have emerged for visualizing graph structures for biological purposes [Wick *et al.* (2015)].

The GFA format was proposed to bring simple interoperability between programs for assembly, modification, and visualization while preserving the information in the assembly graph (Shaun Jackman *et al.* 2015, https://github.com/GFA-spec/GFA-spec). Parsers and format manipulators are available in ruby Gonnella and Kurtz (2016) and python (https://github.com/AlgoLab/pygfa), Diego Lobba, 2017), but the majority of assemblers are written in languages such as C or C++. Therefore where these programs read or write GFA they rely on hard-coded translators incorporated into their source code. This creates a substantial burden to supporting GFA, reducing the ease of modularising assembly code, and leads to

*To whom correspondence should be addressed. **Contact:** eric.t.dawson@gmail.com.

many small incompatibilities between output files that can be difficult to debug. We developed `gfakluge` as a freely-available, open-source C++ library for reading, writing and manipulating GFA with the aim that it will encourage community uptake of the GFA format in high-performance software, and the development of interoperable suites of assembly software.

1 Features and methods

1.1 Requirements

`gfakluge` requires a C++-11 compatible compiler (e.g. gcc4.8 or newer). It does not have any external dependencies.

1.2 Parsing GFA files

`gfakluge` parses text from the original GFA proposal ("version 0.1") and both the GFA1 and GFA2 specifications. The parser generates a set of C++ map containers using string keys and struct values. Structs are defined for each of the line types defined in the GFA spec. The parser supports both SAM-style and JSON tags and can easily be extended to include new or custom lines.

1.3 Converting between formats

`gfakluge` provides a high-level interface for conversion between GFA versions. A header containing the `VZ` tag is encouraged when parsing if the input file is not GFA 1.0. Setting the `GFAKluge` object's version before output is sufficient for converting formats if no modifications to the object's structures are necessary. The internal structures of the graph can be converted to be compatible with GFA 1.0 / 2.0 using the `gfa_1_ize()` and `gfa_2_ize()` methods in the `GFAKluge` class. Conversion between the various formats is currently useful, as many programs are using disparate versions of GFA. As uptake of flexible parsers like `GFAKluge` becomes more common the importance of being able to convert between the GFA versions will wane, as more tools will begin to operate on the most up-to-date specification.

1.4 Integrating GFA parsing into a larger program

We used `gfakluge` to implement GFA parsing in the variation graph toolkit `vg` [Garrison *et al.* (2016)]. Reading a GFA file of any version requires two lines of code. Converting from `vg`'s internal representation to valid GFA requires 39 lines of code, the majority of which is spent filling in individual struct fields with members from `vg`'s graph class.

1.5 Command line utilities

`gfakluge` provides a number of useful utilities for modifying GFA files as they pass between programs. A `gfa_stats` program computes basic assembly and graph statistics such as the number of nodes and edges, the N50/N90/L50/L90, and total sequence length. The `gfa_subset` utility can extract a subset of a GFA graph between two nodes. There are also utilities for sorting a GFA file by source node or by grouping lines of the same type together and for modifying the identifiers of multiple graphs and merging them into one ID space.

Acknowledgements

We would like to thank Shaun Jackman, Gene Myers, Jason Chin, Giorgio Gonnella and Heng Li for discussions about the GFA 1.0 and 2.0 formats.

Funding

ETD is supported by a Cambridge Trust - NIH Fellowship as part of the NIH Oxford Cambridge Scholars Program.

References

Garrison E, et al. vg : the variation graph toolkit. 2016:1–31.

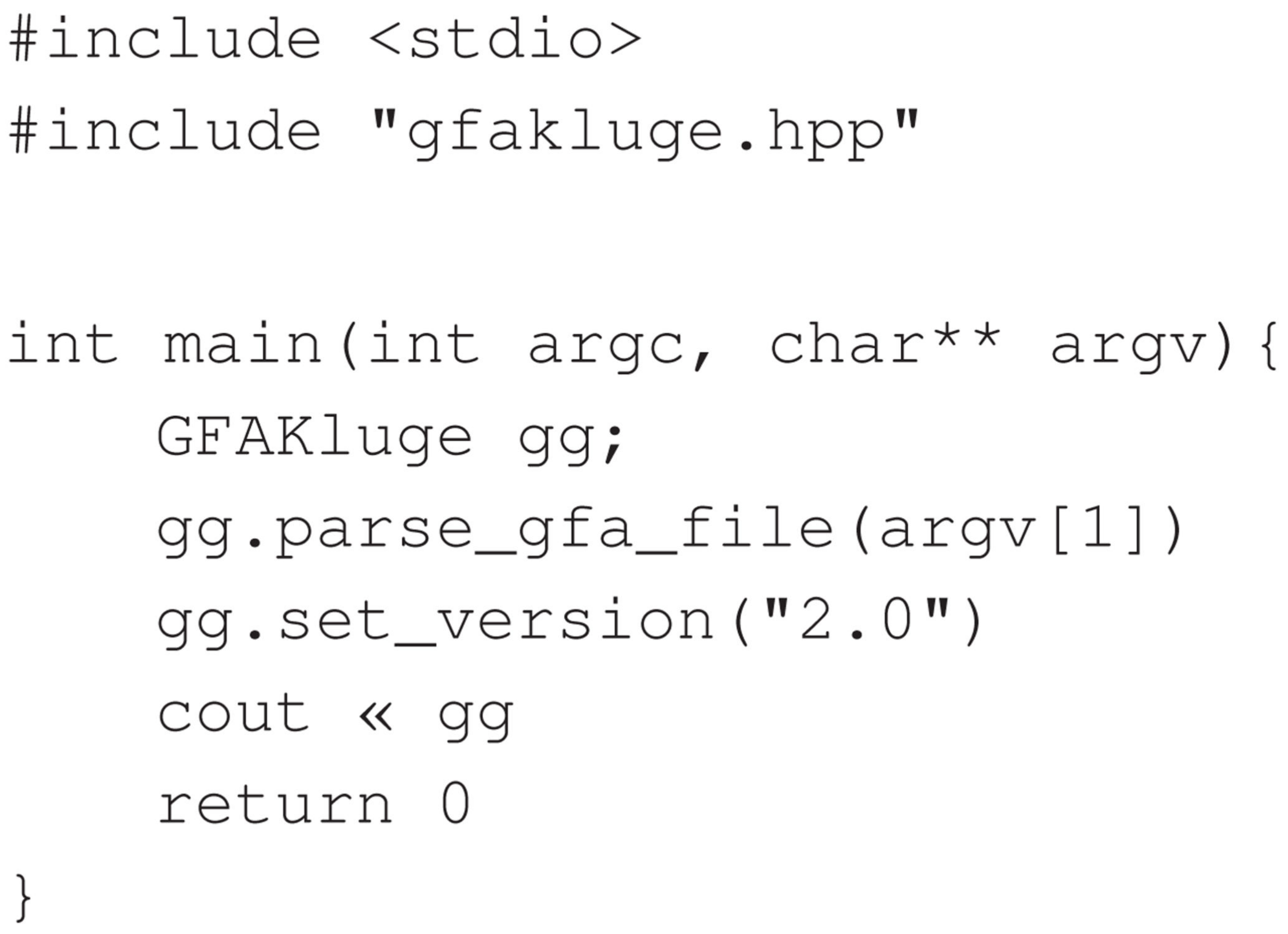
Gonnella G, Kurtz S. RGFA : powerful and convenient handling of assembly graphs. 2016:1–9.

Simpson JT, Durbin R. Efficient construction of an assembly string graph using the FM-index. Bioinformatics. 2010; 26(12):367–373.

Simpson JT, et al. ABySS : A parallel assembler for short read sequence data ABySS : A parallel assembler for short read sequence data. 2009:1117–1123.

Wick RR, et al. Bandage: Interactive visualization of de novo genome assemblies. Bioinformatics. 2015; 31(20):3350–3352. [PubMed: 26099265]

Zerbino DR, Birney E. Velvet: Algorithms for de novo short read assembly using de Bruijn graphs. Genome Research. 2008; 18(5):821–829. [PubMed: 18349386]



```
#include <stdio>
#include "gfakluge.hpp"


int main(int argc, char** argv){
    GFAKluge gg;
    gg.parse_gfa_file(argv[1])
    gg.set_version("2.0")
    cout « gg
    return 0
}
```

Fig. 1.

A basic C++ program for converting a GFA1.0 or 2.0 file to GFA2.0, which shows the high-level interface for format conversion.