Bioinformatics, 2017, 1–2
doi: 10.1093/bioinformatics/btx398
Advance Access Publication Date: 22 June 2017
Applications Note

OXFORD

Genome analysis

GfaPy: a flexible and extensible software library for handling sequence graphs in Python

Giorgio Gonnella* and Stefan Kurtz

ZBH – Center for Bioinformatics, MIN-Fakultät, Universität Hamburg, 20146 Hamburg, Germany

*To whom correspondence should be addressed.

Associate Editor: Bonnie Berger

Received on March 20, 2017; revised on June 8, 2017; editorial decision on June 12, 2017; accepted on June 13, 2017

Abstract

Summary: GFA 1 and GFA 2 are recently defined formats for representing sequence graphs, such as assembly, variation or splicing graphs. The formats are adopted by several software tools. Here, we present *GfaPy*, a software package for creating, parsing and editing GFA graphs using the programming language Python. *GfaPy* supports GFA 1 and GFA 2, using the same interface and allows for interconversion between both formats. The software package provides a simple interface for custom record types, which is an important new feature of GFA 2 (compared to GFA 1). This enables new applications of the format.

Availability and implementation: *GfaPy* is available open source at `https://github.com/ggonnella/gfapy` and installable via pip.

Contact: `gonnella@zbh.uni-hamburg.de`

Supplementary information: Supplementary data are available at *Bioinformatics* online.

1 Introduction

GFA (Graphical Fragment Assembly, `https://github.com/GFA-spec/GFA-spec`) are text-based tab-separated file formats allowing for the description of generic sequence graphs, such as different kinds of assembly graphs (de Bruijn, overlap and string graphs), variation graphs and gene splicing graphs. They are based on similar conventions as the SAM format, the *de facto* standard format for read mapping results. GFA is increasingly supported by several sequence analysis software tools including assemblers, e.g. ABySS (Simpson *et al.*, 2009), Readjoiner (Gonnella and Kurtz, 2012), Canu (Koren *et al.*, 2017) and Miniasm (`https://github.com/lh3/miniasm`), variation graph tools (vg, `https://github.com/vgteam/vg`) and visualization tools (Bandage, Wick *et al.* (2015)). All current GFA implementations write and read GFA 1, which was first defined by Heng Li in 2014 (`https://git.io/vyXr6`) and developed further by the community. Recently, initiated by Jason Chin, Richard Durbin and Gene Myers (`https://github.com/thegenemyers/DFA-SPEC`), a new version of the format (GFA 2) was collaboratively developed and released (`https://github.com/GFA-spec/GFA-spec/blob/master/GFA2.md`).

GFA 2 introduces several new features, which will extend the range of applications of the format. In particular, gap records allow representing scaffolding graphs. Fragment records represent the alignment of reads to contigs in an assembly. The support for generic alignments and a compact alignment representation (traces), makes GFA 2 useful for assembly of long noisy reads. GFA 2 simplifies the syntax for the definition of paths and allows specifying arbitrary subgraphs. Custom record types provide considerable flexibility for format extensions.

The combination of different programs and data sources in pipelines often requires editing and manipulating intermediate data. For example, when merging GFA files from different sources, renaming of records is necessary to avoid name clashes. For such an application, text substitution without regard to the underlying data representation is error prone and may lead to corrupted data. Furthermore, for sequence assembly tasks, additional information not available to the assembler can be systematically incorporated into the assembly graph to support its manual finishing.

In such contexts, the ability to programmatically access and traverse GFA graphs, ideally in commonly used scripting languages, is important. Here we present *GfaPy*, a software library implementing

```
import gfapy
g2 = gfapy.Gfa.from_file("a.gfa2") # read GFA from file
g2.header.TS                       # get TS tag from GFA header
for e in g2.edges:
  if not e.is_dovetail():          # remove all non-dovetail edges
    e.disconnect()
for s in g2.segments:
  if s.slen < 1000: s.disconnect() # remove all segments < 1kbp
g2.segment("s1").disconnect()      # remove segment s1 and all incident edges
g2.segment("s2").neighbours        # segments connected by dovetails to s2
g1 = g2.to_gfa1()                  # convert to GFA 1
g1.append('S\ts3\tACGTTTG')        # add segment s3 manually
g1.to_file("a.gfa1")               # write to file
```

Fig. 1. Examples of code using *GfaPy* for accessing GFA information

an application programming interface for creating, parsing, editing and writing GFA 1 and GFA 2 files. It is the first library which allows for comprehensive handling of GFA files using Python and the first publicly available implementation in any language fully supporting the GFA 2 specification.

2 Implementation and features

2.1 Python API for parsing, editing and writing GFA

The Python scripting language is popular in the bioinformatics community. Until now, the support of the GFA format in Python was limited. The only publicly available Python implementation of the GFA format is PyGFA (`https://github.com/pmelsted/pyGFA`). However, it only offers a limited functionality for editing GFA data and traversing a graph in GFA 1 format. Moreover, it cannot handle the GFA 2 format.

The authors previously presented RGFA (Gonnella and Kurtz, 2016), a Ruby library to handle GFA 1. It does not support GFA 2. Since Ruby is not as widely used as Python in the Bioinformatics community, we decided to continue our development concerning GFA in Python.

GfaPy is the first Python library offering a complete interface to parse files in the GFA formats. It provides a clear API for accessing the data in a GFA file and simple traversal of a GFA graph, as described in the user manual (`http://gfapy.readthedocs.io`). Figure 1 shows a simple example applying the *GfaPy* -API. In the Supplementary Material, we demonstrate that *GfaPy* is a significant advance over RGFA.

The core of the *GfaPy* implementation is independent of the concrete record types and the datatype of specific fields. It can therefore easily be modified to incorporate future changes in the GFA specification. To ensure stability and quality of the implementation, *GfaPy* is complemented with a comprehensive suite of over 350 unit tests.

2.2 Syntactic and semantic validation of GFA

gfalint (`https://github.com/sjackman/gfalint`) is a syntactic validator for both GFA 1 and GFA 2 files. However, a conforming syntax does not guarantee semantically valid GFA data. For example, an edge could point to a non-existing segment or use a position beyond the end of the segment sequence. To identify such issues, *GfaPy* provides both syntactic and semantic validation.

2.3 Support for GFA 2

GfaPy is the first software library with a complete interface to handle GFA 2 files. It includes operations to access the records data and traverse the graph, independent of the GFA version.

Current tools using the GFA format (except for *GfaPy* and gfalint) are based on GFA 1. However, tools employed in assembly pipelines for PacBio reads, are likely to adopt GFA 2. Combining these tools requires conversion between the two GFA formats. For such an application we provide a *GfaPy* -based command line script `gfapy-convert`.

2.4 Custom extensions to GFA

An important feature of GFA 2 is its support for custom extensions of the format. The first field in the GFA records contains a string defining the record type. According to the GFA 2 specification, a core parser may ignore all lines starting with a non-standard string, and these are reserved to be used for custom record types.

In *GfaPy*, these custom records can be accessed with a generic interface, which can be improved by the user via the implementation of an extension module. This specifies further requirements for the custom records. For example, specifying for each field a name and datatype (which can be user-defined) enables named access to the fields and validation of the data. If the custom records contain references to other lines, these can be resolved by *GfaPy*, thus fully integrating existing and custom line types.

In the Supplementary Material, we provide a description of how to write extensions, including an example applying *GfaPy* in a metagenomics context. In particular, we show how to integrate and use custom record types for representing taxa and assignments of contigs to taxa.

Acknowledgement

We thank Tim Weber for implementing parts of an earlier version of *GfaPy*.

Conflict of Interest: none declared.

References

Gonnella,G. and Kurtz,S. (2012) Readjoiner: a fast and memory efficient string graph-based sequence assembler. *BMC Bioinformatics*, **13**, 82.

Gonnella,G. and Kurtz,S. (2016) RGFA: powerful and convenient handling of assembly graphs. *PeerJ*, **4**, e2681.

Koren,S. *et al.* (2017) Canu: scalable and accurate long-read assembly via adaptive k-mer weighting and repeat separation. *Genome Res.*, **27**, 722–736.

Simpson,J.T. *et al.* (2009) ABySS: a parallel assembler for short read sequence data. *Genome Res.*, **19**, 1117–1123.

Wick,R.R. *et al.* (2015) Bandage: interactive visualization of de novo genome assemblies. *Bioinformatics*, **31**, 3350–3352.