Reference Documentation for ODGI 0.3.0

Table of Contents

1. odgi (1)

1.1. NAME

odgi - dynamic succinct variation graph tool

1.2. SYNOPSIS

odgi build -g graph.gfa -o graph.og

odgi stats -i graph.og -S

odgi sort -i graph.og - graph.sorted.og -p bSnSnS

1.3. DESCRIPTION

Odgi is a set of of tools that manipulate variation graphs. TODO Explain data structure. TODO Link to vg paper. TODO Link to handle graph paper. TODO Explain purpose. TODO Give summary of the capabilities of the set of tools.

1.4. COMMANDS

Each command has its own man page which can be viewed using e.g. **man odgi_build.1**. Below we have a brief summary of syntax and sub-command description.

odgi build [**-g, --gfa**=*FILE*] [**-o, --out**=*FILE*] [*OPTION*]…
The odgi build(1) command constructs a succinct variation graph from a GFA. Currently, only GFA1 is supported. For details of the format please see https://github.com/GFA-spec/GFA-spec/blob/master/GFA1.md.

odgi stats [**-i, --idx**=*FILE*] [*OPTION*]…
The odgi stats(1) command produces statistics of a variation graph. Among other metrics, it can calculate the #nodes, #edges, #paths and the total nucleotide length of the graph. Various histogram

summary options complement the tool. If [**-B, --bed-multicov**=*BED*] is set, the metrics will be produced for the intervals specified in the BED.

odgi sort [**-i, --idx**=*FILE*] [**-o, --out**=*FILE*] [*OPTION*]…
The odgi sort(1) command sorts a succinct variation graph. Odgi sort offers a diverse palette of sorting algorithms to determine the node order:

- A topological sort: A graph can be sorted via breadth-first search (BFS) or depth-first search (DFS). Optionally, a chunk size specifies how much of the graph to grab at once in each topological sorting phase. The sorting algorithm will continue the sort from the next node in the prior graph order that has not been sorted, yet. The cycle breaking algorithm applies a DFS sort until a cycle is found. We break and start a new DFS sort phase from where we stopped.
- A random sort: The graph is randomly sorted. The node order is randomly shuffled from Mersenne Twister pseudo-random generated numbers.
- A sparse matrix mondriaan sort: We can partition a hypergraph with integer weights and uniform hyperedge costs using the Mondriaan partitioner.
- A 1D linear SGD sort: Odgi implements a 1D linear, variation graph adjusted, multi-threaded version of the Graph Drawing by Stochastic Gradient Descent algorithm. The force-directed graph drawing algorithm minimizes the graph’s energy function or stress level. It applies stochastic gradient descent (SGD) to move a single pair of nodes at a time.
- An eades algorithmic sort: Use Peter Eades' heuristic for graph drawing.

Sorting the paths in a graph my refine the sorting process. For the users' convenience, it is possible to specify a whole pipeline of sorts within one parameter.

1.5. BUGS

Refer to the **odgi** issue tracker at https://github.com/vgteam/odgi/issues.

1.6. AUTHORS

Erik Garrison from the University of California Santa Cruz wrote the whole **odgi** tool. Despite small code contributions, Simon Heumos from the Quantitative Biology Center Tübingen wrote **odgi pathindex**, **odgi panpos**, **odgi server**, and the documentation.

1.7. RESOURCES

Project web site: https://github.com/vgteam/odgi

Git source repository on GitHub: https://github.com/vgteam/odgi

GitHub organization: https://github.com/vgteam

Discussion list / forum: https://github.com/vgteam/odgi/issues

1.8. COPYING

2. odgi build(1)

2.1. NAME

odgi_build - construct a dynamic succinct variation graph

2.2. SYNOPSIS

odgi build [**-g, --gfa**=*FILE*] [**-o, --out**=*FILE*] [*OPTION*]…

2.3. DESCRIPTION

The odgi build(1) command constructs a succinct variation graph from a GFA. Currently, only GFA1 is supported. For details of the format please see https://github.com/GFA-spec/GFA-spec/blob/master/GFA1.md.

2.4. OPTIONS

2.4.1. Graph Files IO

-g, --gfa=*FILE*

GFA1 file containing the nodes, edges and paths to build a dynamic succinct variation graph from.

-o, --out=*FILE*

Write the dynamic succinct variation graph to this file. A file ending with *.og* is recommended.

2.4.2. Graph Sorting

-s, --sort

Apply a general topological sort to the graph and order the node ids accordingly. A bidirected adaptation of Kahn’s topological sort (1962) is used, which can handle components with no heads or tails. Here, both heads and tails are taken into account.

2.4.3. Processing Information

-p, --progress

Print progress updates to stdout.

-d, --debug

Verbosely print graph information to stderr. This includes the maximum node_id, the minimum node_id, the handle to node_id mapping, the deleted nodes and the path metadata.

--trace

Include backtrace information when reporting errors.

-v, --verbose

Verbosely print processing information to stderr, including debug-level log messages.

-w, --warnings

Turn on script warnings (applies to executed code).

-t, --timings

Print timings report to stderr (time to read, parse, and convert).

2.4.4. Program Information

-h, --help

Print a help message for **odgi build**.

2.5. EXIT STATUS

0

Success.

1

Failure (syntax or usage error; parameter error; file processing failure; unexpected error).

2.6. BUGS

Refer to the **odgi** issue tracker at https://github.com/vgteam/odgi/issues.

2.7. AUTHORS

odgi build was written by Erik Garrison.

3. odgi stats(1)

3.1. NAME

odgi_stats - metrics describing variation graphs

3.2. SYNOPSIS

odgi stats [**-i, --idx**=*FILE*] [*OPTION*]…

3.3. DESCRIPTION

The odgi stats(1) command produces statistics of a variation graph. Among other metrics, it can calculate the #nodes, #edges, #paths and the total nucleotide length of the graph. Various histogram summary options complement the tool. If [**-B, --bed-multicov**=*BED*] is set, the metrics will be produced for the intervals specified in the BED.

3.4. OPTIONS

3.4.1. Graph Files IO

-i, --idx=*FILE*

File containing the succinct variation graph to create statistics from. The file name usually ends with *.og*.

3.4.2. Summary Options

-S, --summarize

Summarize the graph properties and dimensions. Print to stdout the #nucleotides, #nodes, #edges and #paths of the graph.

-b, --base-content

Describe the base content of the graph. Print to stdout the #A, #C, #G and #T of the graph.

-C, --coverage

Provide a histogram of path coverage over bases in the graph. Print three tab-delimited columns to stdout: **type**, **cov**, **N**. **type** is one of *full* or *uniq* and determines if the histogram corresponds to the full graph or only to a unique paths graph. **cov** implies the #paths. **N** implies the #nucleotides.

-V, --set-coverage

Provide a histogram of coverage over unique set of paths. Print two tab-delimited columns to stdout: **cov**, **sets**. **cov** implies #nucleotides. **sets** lists the unique set of paths in a comma separated list. Sets with a **cov** of one and no paths in **sets** are listed, too.

-M, --multi-coverage

Provide a histogram of coverage over unique multiset, the combination with possible repetition of paths. Print two tab-delimited columns to stdout: **cov**, **sets**. **cov** implies #nucleotides. **sets** lists the unique multisets of paths in a comma separated list. Multisets with a **cov** of one and no paths in **sets** are listed, too.

3.4.3. BED Interval

-B, --bed-multicov=***BED***

For each BED entry, provide a table of path coverage over unique multisets of paths in the graph. Each unique multiset of paths overlapping a given BED interval is described in terms of its length relative to the total interval, the number of path traversals and unique paths involved in these traversals.

3.4.4. Threading

-t, --threads=***N***

Number of threads to use.

3.4.5. Program Information

-h, --help

Print a help message for **odgi stats**.

3.5. EXIT STATUS

0

Success.

1

Failure (syntax or usage error; parameter error; file processing failure; unexpected error).

3.6. BUGS

Refer to the **odgi** issue tracker at https://github.com/vgteam/odgi/issues.

3.7. AUTHORS

odgi stats was written by Erik Garrison.

4. odgi sort(1)

4.1. NAME

odgi_sort - sort a variation graph

4.2. SYNOPSIS

odgi sort [**-i, --idx**=*FILE*] [**-o, --out**=*FILE*] [*OPTION*]…

4.3. DESCRIPTION

The odgi sort(1) command sorts a succinct variation graph. Odgi sort offers a diverse palette of sorting algorithms to determine the node order:

- A topological sort: A graph can be sorted via breadth-first search (BFS) or depth-first search (DFS). Optionally, a chunk size specifies how much of the graph to grab at once in each topological sorting phase. The sorting algorithm will continue the sort from the next node in the prior graph order that has not been sorted, yet. The cycle breaking algorithm applies a DFS sort until a cycle is found. We break and start a new DFS sort phase from where we stopped.
- A random sort: The graph is randomly sorted. The node order is randomly shuffled from Mersenne Twister pseudo-random generated numbers.
- A sparse matrix mondriaan sort: We can partition a hypergraph with integer weights and uniform hyperedge costs using the Mondriaan partitioner.
- A 1D linear SGD sort: Odgi implements a 1D linear, variation graph adjusted, multi-threaded version of the Graph Drawing by Stochastic Gradient Descent algorithm. The force-directed graph drawing algorithm minimizes the graph's energy function or stress level. It applies stochastic gradient descent (SGD) to move a single pair of nodes at a time.
- An eades algorithmic sort: Use Peter Eades' heuristic for graph drawing.

Sorting the paths in a graph my refine the sorting process. For the users' convenience, it is possible to specify a whole pipeline of sorts within one parameter.

4.4. OPTIONS

4.4.1. Graph Files IO

-i, --idx=*FILE*

File containing the succinct variation graph to sort. The file name usually ends with *.og*.

-o, --out=*FILE*

Write the sorted dynamic succinct variation graph to this file. A file ending with *.og* is recommended.

-s, --sort-order=*FILE*

File containing the sort order. Each line contains one node identifier.

4.4.2. Topological Sorts

-b, --breadth-first

Use a (chunked) breadth first topological sort.

-B, --breadth-first-chunk=*N*

Chunk size for breadth first topological sort. Specify how many nucleotides to grap at once in each BFS phase.

-z, --depth-first

Use a (chunked) depth first topological sort.

-Z, --depth-first-chunk=*N*

Chunk size for the depth first topological sort. Specify how many nucleotides to grap at once in each DFS phace.

-w, --two-way

Use a two-way topological algorithm for sorting. It is a maximum of head-first and tail-first topological sort.

-n, --no-seeds

Don’t use heads or tails to seed topological sort.

-c, --cycle-breaking

Use a cycle breaking sort.

4.4.3. Random Sort

-r, --random

Randomly sort the graph.

4.4.4. Mondriaan Sort

-m, --mondriaan

Use the sparse matrix diagonalization to sort the graph.

-N, --mondriaan-n-parts=*N*

Number of partitions for the mondriaan sort.

-E, --mondriaan-epsilon=*N*

Set the epsilon parameter for the mondriaan sort.

-W, --mondriaan-path-weight

Weight the mondriaan input matrix by the path coverage of edges.

4.4.5. 1D Linear SGD Sort

-S, --linear-sgd

Apply 1D linear SGD algorithm to sort the graph.

-O, --sgd-bandwidth=*sgd-bandwidth*

Bandwidth of linear SGD model. The default value is *1000*.

-Q, --sgd-sampling-rate=*sgd-sampling-rate*

Sample pairs of nodes with probability distance between them divided by the sampling rate. The default value is *20*.

-K, --sgd-use-paths

Use the paths to structure the distances between nodes in SGD.

-T, --sgd-iter-max=*sgd_iter-max*

The maximum number of iterations for the linear SGD model. The default value is *30*.

-V, --sgd-eps=*sgd-eps*

The final learning rate for the linear SGD model. The default value is *0.01*.

-C, --sgd-delta=*sgd-delta*

The threshold of the maximum node displacement, approximately in base pairs, at which to stop SGD.

4.4.6. Eades Sort

-e, --eades

Use eades algorithm.

4.4.7. Path Sorting Options

-L, --paths-min

Sort paths by their lowest contained node identifier.

-M, --paths-max

Sort paths by their highest contained node identifier.

-A, --paths-avg

Sort paths by their average contained node identifier.

-R, --paths-avg-rev

Sort paths in reverse by their average contained node identifier.

-D, --path-delim=*path-delim*

Sort paths in bins by their prefix up to this delimiter.

4.4.8. Pipeline Sorting

-p, --pipeline=*STRING*

Apply a series of sorts, based on single character command line arguments given to this command. The default sort is *s*. The reverse sort would be specified via *f*.

4.4.9. Additional Parameters

-d, --dagify-sort

Sort on the basis of a DAGified graph.

-O, --Optimize

Use the MutableHandleGraph::optimize method to compact the node identifier space.

4.4.10. Threading

-t, --threads=*N*

Number of threads to use for parallel sorting in SGD. Only specify this argument in combination with **-S, --linear-sgd**. No multi-threading support for any other sorting algorithm.

4.4.11. Processing Information

-P, --progress

Print sort progress to stdout.

4.4.12. Program Information

-h, --help

Print a help message for **odgi sort**.

4.5. EXIT STATUS

0

Success.

1

Failure (syntax or usage error; parameter error; file processing failure; unexpected error).

4.6. BUGS

Refer to the **odgi** issue tracker at https://github.com/vgteam/odgi/issues.

4.7. AUTHORS

odgi sort was written by Erik Garrison.