Superbubbles

and their linear-time detection

Ritu Kundu

King's College London

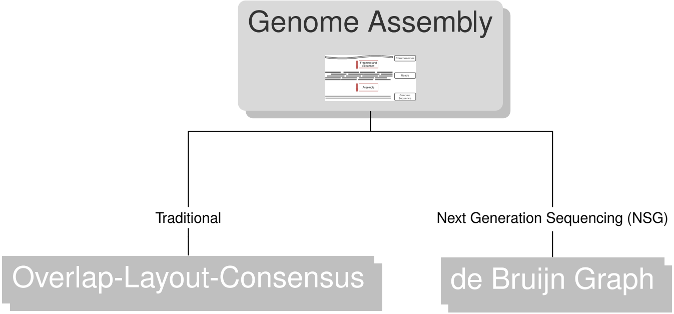
February 16, 2019

Joint work with
Ljiljana Brankovic, Costas S. Iliopoulos, Manal Mohamed, Solon P. Pissis, and Fatima Vayani

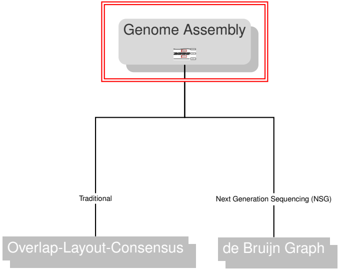
Outline

Outline

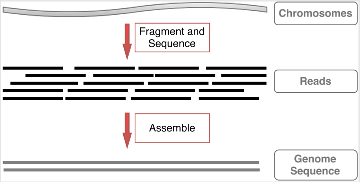
Motivation


Genome Assembly
Chromosomes
Fragment and Sequence
Reads
Assemble
Genome Sequence
Traditional
Next Generation Sequencing (NSG)
Overlap-Layout-Consensus
de Bruijn Graph

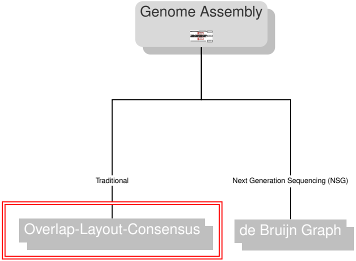
Motivation



Genome Assembly

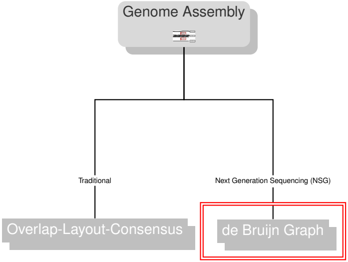


Motivation

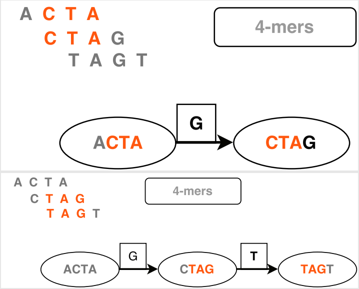


Overlap-Layout-Consensus

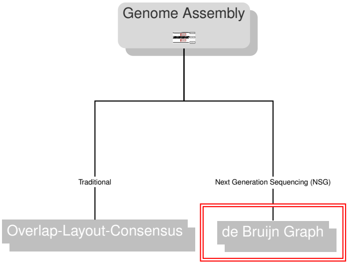
Motivation



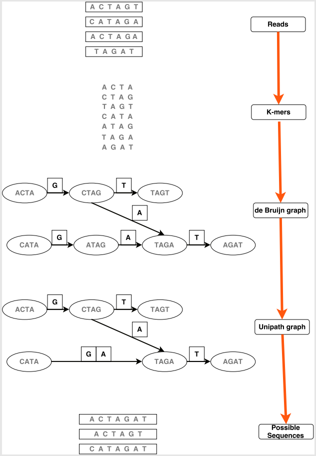
de Bruijn Graph



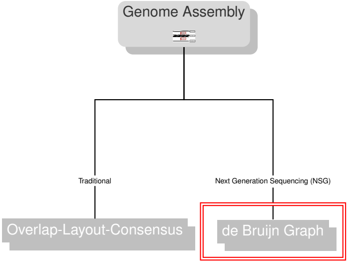
Motivation



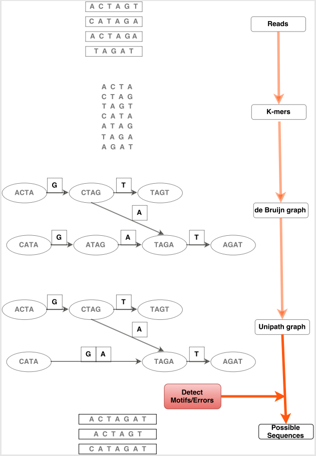
NSG



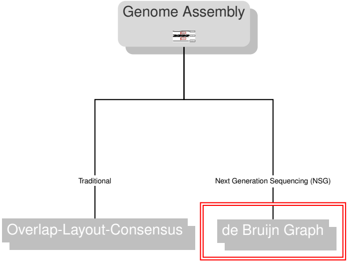
Motivation



NSG



Motivation

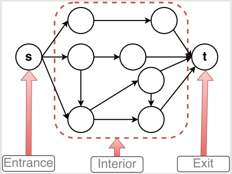


Motifs

- Tips
- Cross-links
- Bubbles
- More Complex Structures?
  Superbubbles

Superbubble

Superbubble: $\langle s,\ t \rangle$

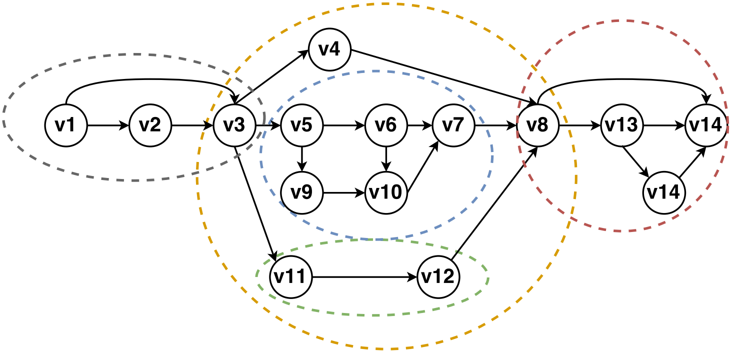


Definition [Onodera et al., 2013]

Let $G = (V, E)$ be a directed graph. For any ordered pair of distinct nodes s and t, $\langle s, t \rangle$ is called a *superbubble* if it satisfies the following:

- **reachability:** t is reachable from s;
- **matching:** the set of nodes reachable from s without passing through t is equal to the set of nodes from which t is reachable without passing through s;
- **acyclicity:** the subgraph induced by U is acyclic, where U is the set of nodes satisfying the matching criterion;
- **minimality:** no node in U other than t forms a pair with s that satisfies the conditions above;

Example



Background

$\mathcal{O}(nm)$-time algorithm [Onodera et al., 2013]

Topological sorting, starting from each vertex, to test if it is an entrance.

$\mathcal{O}(m \log m)$-time algorithm [Sung et al., 2015]

- Partition graphs into a set of sub-graphs -
  - subgraphs corresponding to each non-singleton strongly connected component
  - a subgraph corresponding to the set of all the nodes involved in singleton strongly connected components.
- Convert each subgraph into acyclic if it is cyclic.
- Find superbubbles in each of the subgraph.

Background

$\mathcal{O}(nm)$-time algorithm [Onodera et al., 2013]

Topological sorting, starting from each vertex, to test if it is an entrance.

$\mathcal{O}(m \log m)$-time algorithm [Sung et al., 2015]

- Partition graphs into a set of sub-graphs - → linear
  - subgraphs corresponding to each non-singleton strongly connected component
  - a subgraph corresponding to the set of all the nodes involved in singleton strongly connected components.
- Convert each subgraph into acyclic if it is cyclic. → linear
- Find superbubbles in each of the subgraph. → $\mathcal{O}(m \log m)$

Outline

2 Linear-Time Algorithm
- Properties
- Description

Properties of superbubbles

Lemma ([Onodera et al., 2013])

Any node can be the entrance (respectively exit) of at most one superbubble.

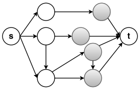
Lemma ([Sung et al., 2015])

Let G be a directed acyclic graph. We have the following two observations.
1) Suppose (p, c) is an edge in G, where p has one child and c has one parent, then $\langle p, c\rangle$ is a superbubble in G.
2) For any superbubble $\langle s, t\rangle$ in G, there must exist some parent p of t such that p has exactly one child t.

Lemma ([Brankovic et al., 2015])

For any superbubble $\langle s, t\rangle$ in a directed acyclic graph G, there must exist some child c of s such that c has exactly one parent s.

Properties of superbubbles



Lemma ([Onodera et al., 2013])

Any node can be the entrance (respectively exit) of at most one superbubble.

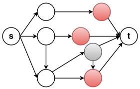
Lemma ([Sung et al., 2015])

Let G be a directed acyclic graph. We have the following two observations.
1) Suppose (p, c) is an edge in G, where p has one child and c has one parent, then $\langle p, c\rangle$ is a superbubble in G.
2) For any superbubble $\langle s, t\rangle$ in G, there must exist some parent p of t such that p has exactly one child t.

Lemma ([Brankovic et al., 2015])

For any superbubble $\langle s, t\rangle$ in a directed acyclic graph G, there must exist some child c of s such that c has exactly one parent s.

Properties of superbubbles



Lemma ([Onodera et al., 2013])

Any node can be the entrance (respectively exit) of at most one superbubble.

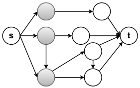
Lemma ([Sung et al., 2015])

Let G be a directed acyclic graph. We have the following two observations.
1) Suppose (p, c) is an edge in G, where p has one child and c has one parent, then $\langle p, c\rangle$ is a superbubble in G.
2) For any superbubble $\langle s, t\rangle$ in G, there must exist some parent p of t such that p has exactly one child t.

Lemma ([Brankovic et al., 2015])

For any superbubble $\langle s, t\rangle$ in a directed acyclic graph G, there must exist some child c of s such that c has exactly one parent s.

Properties of superbubbles



Lemma ([Onodera et al., 2013])

Any node can be the entrance (respectively exit) of at most one superbubble.

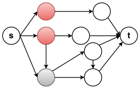
Lemma ([Sung et al., 2015])

Let G be a directed acyclic graph. We have the following two observations.
1) Suppose (p, c) is an edge in G, where p has one child and c has one parent, then $\langle p, c\rangle$ is a superbubble in G.
2) For any superbubble $\langle s, t\rangle$ in G, there must exist some parent p of t such that p has exactly one child t.

Lemma ([Brankovic et al., 2015])

For any superbubble $\langle s, t\rangle$ in a directed acyclic graph G, there must exist some child c of s such that c has exactly one parent s.

Properties of superbubbles



Lemma ([Onodera et al., 2013])

Any node can be the entrance (respectively exit) of at most one superbubble.

Lemma ([Sung et al., 2015])

Let G be a directed acyclic graph. We have the following two observations.
1) Suppose (p, c) is an edge in G, where p has one child and c has one parent, then $\langle p, c\rangle$ is a superbubble in G.
2) For any superbubble $\langle s, t\rangle$ in G, there must exist some parent p of t such that p has exactly one child t.

Lemma ([Brankovic et al., 2015])

For any superbubble $\langle s, t\rangle$ in a directed acyclic graph G, there must exist some child c of s such that c has exactly one parent s.

Abstract Description

Conceptual Idea

- **Input:** $G = (V, E)$, a Directed Acyclic Graph (DAG) where V and E are sets of vertices and edges resp.($|V| = n,\ |E| = m$)
- **Output:** Superbubbles in G
- **Assumption:** Single-source and single-sink *(if not, add dummy vertices)*
- **Work-Flow:**
  - Topologically order the vertices
  - Identify possible entrance and exit candidates: *Candidate-list*
  - Traverse candidate-list (in reverse topological order) to find superbubbles using subroutines:
    - *Report*
    - *Validate*

Detailed Description

Conceptual Idea

- Topological ordering
- Identify candidates
- Find superbubbles using subroutines:
  - *Report*
  - *Validate*

Detailed Description

Conceptual Idea

- Topological ordering
- Identify candidates
- Find superbubbles using subroutines:
  - *Report*
  - *Validate*

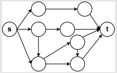
Topological-order

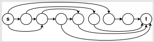
For every edge (a, b), $ord[a] < ord[b]$

TopologicalSort

Recursive DFS (Depth First Search)

Example





Lemma (4)

Given a directed graph $G = (V, E)$ containing a superbubble $\langle s, t \rangle$, a topological ordering obtained by TopologicalSort has the following properties.
- For all x such that $x \in U \setminus \{s, t\}$, ordD$[s]$ < ordD$[x]$ < ordD$[t]$
- For all y such that $y \notin U$, ordD$[y]$ < ordD$[s]$ or ordD$[y]$ > ordD$[t]$.

Detailed Description

Conceptual Idea

- Topological ordering
- Identify candidates
- Find superbubbles using subroutines:
  - *Report*
  - *Validate*

Candidate

A node v is an
- **exit candidate:** if it has at least one parent with exactly one child (out-degree 1)
- **entrance candidate:** if it has at least one child with exactly one parent (in-degree 1).

(From Lemmas 2 and 3)

Identifying Candidates

Check each node in V, in topological order, to identify whether it is an exit or an entrance candidate (or both).
-If both, add twice (First as entrance and then as exit).

- Maximum size: $2n$

- Candidates are added in topological order in Candidate-list.

Detailed Description

Conceptual Idea

- Topological ordering
- Identify candidates
- Find superbubbles using subroutines:
  - *Report*
  - *Validate*

What?

- Reports all the possible superbubbles (including the nested ones) between given *start* and *exit*
- Called for each exit candidate in decreasing order either by *main* routine or through a recursive call to identify a nested superbubble.

How?

- Checks the possible entrance candidates between given *start* and *exit* candidates starting with the nearest previous entrance candidate (to *exit*), using *Validate*.
  - If *valid*, report it and recursively find nested superbubbles.
  - Otherwise, mark the returned *alternative entrance candidate*

Detailed Description

Conceptual Idea

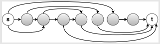
- Topological ordering
- Identify candidates
- Find superbubbles using subroutines:
  - *Report*
  - *Validate*

What?

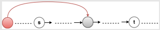
- Returns *start* itself, given *start* and *exit* is a valid superbubble.
- Otherwise returns an alternative possible entrance for *exit*.

How?

- Valid: For a valid superbubble $\langle s, t \rangle$, every $x \in U \backslash \{s, t\}$ has
  - t as its *topologically furthest* child. - s as its *topologically furthest* parent.



- Invalid:



Red Vertex is an *alternate entrance candidate* for the pair (s, t).

- Maintain arrays of topological furthest child and parent resp., for each vertex and using RMQ to verify the conditions for validity.

Detailed Description

Conceptual Idea

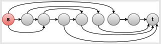
- Topological ordering
- Identify candidates
- Find super-bubbles using subroutines:
  - *Report*
  - *Validate*

What?

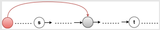
- Returns *start* itself, given *start* and *exit* is a valid superbubble.
- Otherwise returns an alternative possible entrance for *exit*.

How?

- Valid: For a valid superbubble $\langle s, t \rangle$, every $x \in U \backslash \{s, t\}$ has - t as its *topologically furthest* child. - s as its *topologically furthest* parent.



- Invalid:



Red Vertex is an *alternate entrance candidate* for the pair (s, t).

- Maintain arrays of topological furthest child and parent resp., for each vertex and using RMQ to verify the conditions for validity.

Detailed Description

Conceptual Idea

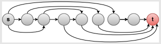
- Topological ordering
- Identify candidates
- Find super-bubbles using subroutines:
  - *Report*
  - *Validate*

What?

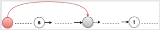
- Returns *start* itself, given *start* and *exit* is a valid superbubble.
- Otherwise returns an alternative possible entrance for *exit*.

How?

- Valid: For a valid superbubble $\langle s, t \rangle$, every $x \in U \backslash \{s, t\}$ has
  - t as its *topologically furthest* child. - s as its *topologically furthest* parent.



- Invalid:



Red Vertex is an *alternate entrance candidate* for the pair (s, t).

- Maintain arrays of topological furthest child and parent resp., for each vertex and using RMQ to verify the conditions for validity.

Detailed Description

Conceptual Idea

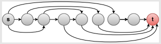
- Topological ordering
- Identify candidates
- Find superbubbles using subroutines:
  - *Report*
  - *Validate*

What?

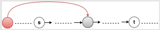
- Returns *start* itself, given *start* and *exit* is a valid superbubble.
- Otherwise returns an alternative possible entrance for *exit*.

How?

- Valid: For a valid superbubble $\langle s, t\rangle$, every $x \in U \setminus \{s, t\}$ has
  - t as its *topologically furthest* child. - s as its *topologically furthest* parent.



- Invalid:



Red Vertex is an *alternate entrance candidate* for the pair (s, t).

- Maintain arrays of topological furthest child and parent resp., for each vertex and using RMQ to verify the conditions for validity.

Detailed Description

Conceptual Idea

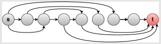
- Topological ordering
- Identify candidates
- Find super-bubbles using subroutines:
    - *Report*
    - *Validate*

What?

- Returns *start* itself, given *start* and *exit* is a valid superbubble.
- Otherwise returns an alternative possible entrance for *exit*.

How?

- Valid: For a valid superbubble $\langle s, t \rangle$, every $x \in U \backslash \{s, t\}$ has
  - t as its *topologically furthest* child. - s as its *topologically furthest* parent.



- Invalid:

Red Vertex is an *alternate entrance candidate* for the pair (s, t).

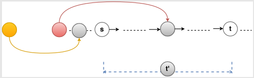
- Maintain arrays of topological furthest child and parent resp., for each vertex and using RMQ to verify the conditions for validity.

Detailed Description

Conceptual Idea

- Topological ordering
- Identify candidates
- Find super-bubbles using subroutines:
  - *Report*
  - *Validate*

Marking



- *Red Vertex* is an invalid entrance not only for the superbubble ending at *t* but also for all those ending at any other exit node (*t'*) between *s* and *t* for which *s* is not a valid entrance and which also has *Red Vertex* as an alternative entrance.
- Further, any candidate in the sequence of alternative entrance candidates following *Red Vertex* (*Orange Vertex* and so on) can not be a valid entrance for the superbubble ending at *t'*.
- *Marking*: to skip this sequence later.

Outline

3 Analysis
- Correctness
- Running Time

Correctness of the algorithm

Lemma (5)

Given s and t, the candidates for an entrance and an exit of a superbubble in G, respectively, subroutine Validate reports $\langle s, t\rangle$ if and only if $\langle s, t\rangle$ is a superbubble.

Lemma (6)

For a given exit candidate e, let x be the alternative entrance candidate returned by the subroutine VALIDATE(s, e). *Then any entrance candidate between x and e can not be a valid entrance for the superbubble ending at e.*

Theorem

Given a directed acyclic graph $G = (V, E)$, where $n = |V|$ and $m = |E|$, algorithm SUPERBUBBLE *correctly finds all superbubbles in G in decreasing order of the topological ordering of their exit nodes.*

Running Time of the algorithm

Lemma (7)

For the given exit and entrance candidates s and e_1 (respectively), let $mark[s]$ has been set to t_1 which later gets reset to t_2 while considering s with another exit candidate e_2. Then any exit candidate between s and e_2 can not reset $mark[s]$ to t_1 again.

- **Topological sorting:** $\mathcal{O}(n+m)$
- **Computing the candidates list:** $\mathcal{O}(n+m)$
- **All the list's operations:** constant time each, sums up to a linear cost $\mathcal{O}(n)$, as there are at most $2n$ candidates in the list.
- **Each call for *Validate*:** $\mathcal{O}(1)$ [RMQ]
- **Total calls to *Validate*:** $\mathcal{O}(n+m)$.
  - Either validates the entrance candidate ($\mathcal{O}(n)$)
  - Or marks the corresponding position ($\mathcal{O}(m)$)
    - Once 'marked', it will not be considered again in subsequent calls [from lemma (7)].
    - Marking is done every time an edge is found for the first time between a vertex (in between an entrance candidate and an exit candidate) and its topologically furthest parent.
- **Total time:** $\mathcal{O}(n+m)$

Outline

4 Summary

Summary

- A critical step of assembly algorithms utilizing de Bruijn graphs is to detect typical motif structures in the graph caused by sequencing errors and genome repeats, and filter them out; one such complex subgraph class is a so-called *superbubble*.
- A supperbubble $\langle s, t\rangle$ is equivalent to a single source, single sink, acyclic directed subgraph of G with source s and sink t, which does not have any cut nodes and preserves all in-degrees and out-degrees of nodes in $U \backslash \{s, t\}$, as well as the out-degree of s and in-degree of t.
- Given a directed acyclic graph $G = (V, E)$, where $n = |V|$ and $m = |E|$, all superbubbles in G can be identified in $\mathcal{O}(n + m)$-time.
- What's Next: More complex motifs

Paper appears in TCS (Open Access):
[L. Brankovica, C. S. Iliopoulos, R. Kundu, M. Mohamed, S. P. Pissis, F. Vayani: Linear-time superbubble identification algorithm for genome assembly].

References

- Brankovic, L., Iliopoulos, C. S., Kundu, R., Mohamed, M., Pissis, S. P., and Vayani, F. (2015).
  Linear-time superbubble identification algorithm for genome assembly.
  Theoretical Computer Science.

- Onodera, T., Sadakane, K., and Shibuya, T. (2013).
  Detecting superbubbles in assembly graphs.
  In *WABI*, pages 338–348.

- Sung, W., Sadakane, K., Shibuya, T., Belorkar, A., and Pyrogova, I. (2015).
  An O(m log m)-time algorithm for detecting superbubbles.
  IEEE/ACM Trans. Comput. Biology Bioinform., 12(4):770–777.

Thank You!