de Bruijn project

Sergey Nurk, Anton Bankevich and Nikolay Vyahhi

April 19, 2011

1 Common Classes

We have two classes for Sequences (all use 2 bits per nucleotide) in `src/common/`:

- `Seq<size_t size>` – immutable ACGT-sequence with compile-time length (= template argument `size`) and zero memory overhead.
- `Sequence` — immutable ACGT-sequence with run-time length and some constant memory overhead. It stores sequence in helper class `SequenceData` with reference counting. Usually both straight and reverse-complement sequence share the same `SequenceData` object. Substring is O(1) operation for `Sequence`.

2 Graph Construction

First we construct usual de Bruijn node-graph with:

- K-mers as nodes,
- (K+1)-mers as edges.

for some compile-time fixed K.

Comment: Yes, it'll be better to make it runtime-length later.

It's `DeBruijn<int size>` class in `src/debruijn/debruijn.hpp`, where `size` is length of K-mer (i.e. `size` = K).

In practice, `DeBruijn` class stores only nodes with information about incoming and outgoing edges — map from K-mer to some `DeBruijn::Data`. Now it's `std::tr1::unordered_map` (standard C++0x hashmap), later it'll probably be replaced with Cuckoo hashmap (now in `src/common/cuckoo.hpp`). Cuckoo have higher load factor (90-95%, hence very memory-effective), much slower `insert` but much faster `find`.

- K-mer as `Seq<size>`.

- `DeBruijn::Data` stores 8 `size_t` counters for 4 incoming and 4 outgoing edges.

  Comment #1: Since we don't need coverage here, we can reduce memory to only 1 *byte (*$4 + 4$ *bits) instead of* $8 * 4 = 32$ *bytes for counters.*

  Comment #2: Even if we need counters, we can mostly use `char` *instead of* `size_t` *here.*

3 Graph Simplification / Condensing

We observed that it's more effective to construct new (condensed) graph from `DeBruijn` graph than to try to condense `DeBruijn` itself. So new `edge_graph::EdgeGraph` (`src/debruijn/edge_graph.hpp` is constructed from `DeBruijn` graph. It's done by usual traversal and accumulating edge-sequence while we're on a simple path (path without branchings).

Every `edge_graph::Edge` stores:

- it's `Sequence`,
- end `Vertex`,
- `size_t` coverage – how much times (K+1)-mers from this edge was observed in original reads. It's additive, so we can simply join two edges and sum their coverages. This coverage will be filled during Read Threading (below).

Every `edge_graph::Vertex` stores:

- `vector<Edge*>` of outgoing edges.
- `Vertex*` – reverse-complement `Vertex`.

Also old `DeBruijn` graph isn't waste of memory. From now it acts as the index for new `EdgeGraph`. It maps K-mers to edges which contains this K-mer. *Comment: Is it already implemented or only in plans?*

4 Read Threading

After graph construction and simplification all original reads are threaded over the resulting `EdgeGraph` using `SimpleReadThreader::ThreadRead()` (`src/debruijn/utils.hpp`). By that we calculate `Edge::coverage` for all edges in `EdgeGraph`.

Comment: Now this function uses `SimpleIndex` *from* `utils.hpp`*, but it can be substituted by* `DeBruijn` *later.*

5 Smart Iterators

`EdgeGraph` have 2 types of smart iterators (`src/debruijn/utils.hpp`):

- `SmartEdgeIterator<Graph, Comparator>` – to iterate over all edges.
- `SmartVertexIterator<Graph, Comparator>` – to iterate over all vertices.

They are used for all graph operation below (e.g. tips clipping and bulges removal).

This iterators are smart not because they have such class-name, but because edges/vertices insert/delete operators don't invalidate them. Hence this iterators can be used to traverse the graph while dynamically change it (e.g. for tips clipping and bulges removal).

Smart iterators are template-classes, so they work with any `Graph` that supports some operations (*TODO: write which ones!*) and with any `Comparator` (`less` for edges/vertices for ordering traversal, e.g. from lower coverage to higher).

This iterators use standard `std::set` to store, order and update edges/vertices which should be traversed. It requires additional ammount of memory equal to $O(\#edges)/O(\#vertices)$. But since `Sequence`s are major memory-consumers, this overhead is not sufficient.

6 Tips Clipping (TC)

Using `SmartEdgeIterator` we traverse graph from smaller edge-length to bigger (`src/debruijn/tip_clipper.hpp`). We delete edge (tip) and his reverse-complementary edge if:

- It's end-vertex have no outgoing edges. We can skip check of start-vertex here since we traverse both straight edge and it's reverse-complementary edge during TC.
- It has length < `DEFAULT_MAX_TIP_LENGTH` (50 now).
- It has coverage < `DEFAULT_COVERAGE_BOUND` (1000 now, like infinity).
- It has relative coverage < `DEFAULT_RELATIVE_COVERAGE_BOUND` (2.0 now). It means that tip's coverage is at most `DEFAULT_RELATIVE_COVERAGE_BOUND` times (max coverage over all edges outgoing from the same vertex).

We condense (join) new simple path after the tip was clipped only if:

- There is only one more edge outgoing from the same start-vertex.

- This only edge is also a tip.

It's required for proper clipping of this another tip.

Any other case of condensing is not necessary to be handled on-line during the TC (and even can lead to quadratic complexity in worst case because of too much `Sequence` joins). So we condense any other simple pathes only after all tips were clipped.

Comment: Now all parameters are `#defines`*, but it's better to make them tunable.*

7 Bulges Removal (BR)

Method `BulgeRemover::RemoveBulges()` (from `src/debruijn/bulge_remover.hpp`) walks through the graph and removes bulges. The edge is bulge if:

- It's edge (not path!).
- Difference between this edge's length and the length of other alternative path (between it's start-vertex and end-vertex) is less then δ. Now $\delta = 3$ (hardcoded. oh... should be as parameter).
- It's coverage is less then 1.1 * (alternative path's coverage). (hardcoded value too).
- It's coverage is some fixed coverage *(TODO: isn't in code now).*

After every bulge removal all new simple pathes are condensed (they can be formed only near start-vertex and end-vertex, not deeply inside alternative path).

TODO: Transfer coverage from bulge to alternative path.
TODO: Transfer edge-pair distance from bulge to alternative path.

8 Mate Pairs (Edge-Pairs) Distance

TODO: Handle it, not implemented now.

9 Evaluation

All Quake's filtered reads were filtered to first 100 kilo-basepairs of E.Coli. I.e. we try to assemble only first 100kbp of E.Coli (for simplicity). Also, for evaluation runs we use only 1st read from every mate pair. It's total of 306174 error-corrected reads with 100 or less basepairs each.

9.1 Log 100k

(first column is seconds from the starting time, ≈ 9 minutes total):

```
310 INFO d.edge_graph (edge_graph_tool.hpp:95) - Edge graph construction tool started
311 INFO d.edge_graph (edge_graph_tool.hpp:53) - Constructing DeBruijn graph
14928 INFO d.edge_graph (edge_graph_tool.hpp:55) - DeBruijn graph constructed
14928 INFO d.edge_graph (edge_graph_tool.hpp:61) - Condensing graph
23146 INFO d.edge_graph (edge_graph_tool.hpp:64) - Graph condensed
23146 INFO d.edge_graph (edge_graph_tool.hpp:66) - Counting coverage
32289 INFO d.edge_graph (edge_graph_tool.hpp:69) - Coverage counted
32289 INFO d.edge_graph (edge_graph_tool.hpp:22) - Counting stats
32289 INFO d.edge_graph (edge_graph_tool.hpp:27) - Vertex count=790; Edge count=836
32289 INFO d.edge_graph (edge_graph_tool.hpp:28) - Stats counted
32319 INFO d.edge_graph (edge_graph_tool.hpp:34) - Writing graph 'edge_graph' to file edge_graph.dot
32911 INFO d.edge_graph (edge_graph_tool.hpp:36) - Graphraph 'edge_graph' written to file edge_graph.dot
32911 INFO d.edge_graph (edge_graph_tool.hpp:75) - Clipping tips
33114 INFO d.edge_graph (edge_graph_tool.hpp:79) - Tips clipped
33114 INFO d.edge_graph (edge_graph_tool.hpp:22) - Counting stats
33114 INFO d.edge_graph (edge_graph_tool.hpp:27) - Vertex count=118; Edge count=164
33114 INFO d.edge_graph (edge_graph_tool.hpp:28) - Stats counted
33146 INFO d.edge_graph (edge_graph_tool.hpp:34) - Writing graph 'no_tip_graph' to file tips_clipped.dot
33180 INFO d.edge_graph (edge_graph_tool.hpp:36) - Graphraph 'no_tip_graph' written to file tips_clipped.dot
33180 INFO d.edge_graph (edge_graph_tool.hpp:85) - Removing bulges
33563 INFO d.edge_graph (edge_graph_tool.hpp:88) - Bulges removed
33563 INFO d.edge_graph (edge_graph_tool.hpp:22) - Counting stats
33563 INFO d.edge_graph (edge_graph_tool.hpp:27) - Vertex count=70; Edge count=92
33563 INFO d.edge_graph (edge_graph_tool.hpp:28) - Stats counted
33598 INFO d.edge_graph (edge_graph_tool.hpp:34) - Writing graph 'no_bulge_graph' to file bulges_removed.dot
33602 INFO d.edge_graph (edge_graph_tool.hpp:36) - Graphraph 'no_bulge_graph' written to file bulges_removed.dot
33602 INFO d.edge_graph (edge_graph_tool.hpp:115) - Tool finished
```

Figure 1: Edge Graph 100k after TC

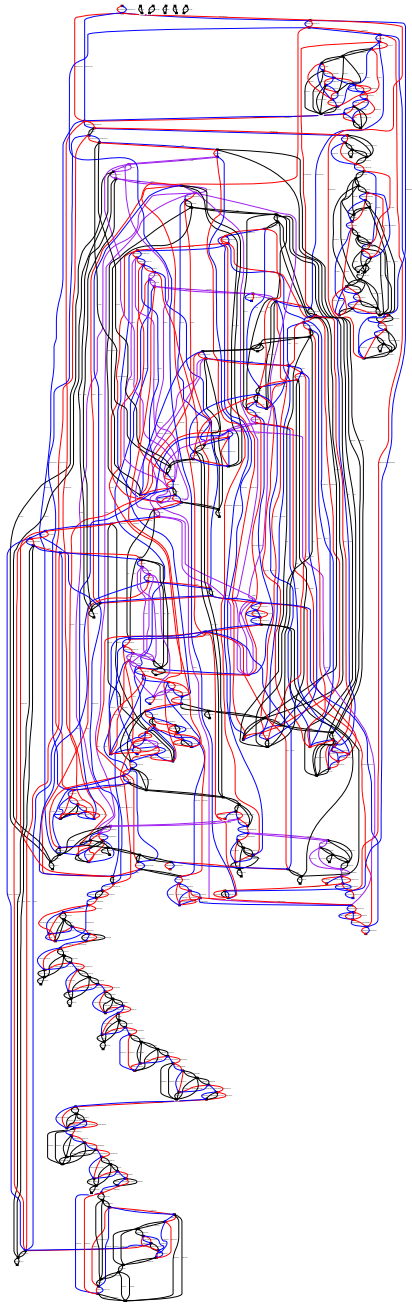

Figure 2: Edge Graph 100k after TC+BR