SUTTA

Scoring-and-**U**nfolding **T**rimmed **T**ree **A**ssembler

Giuseppe Narzisi

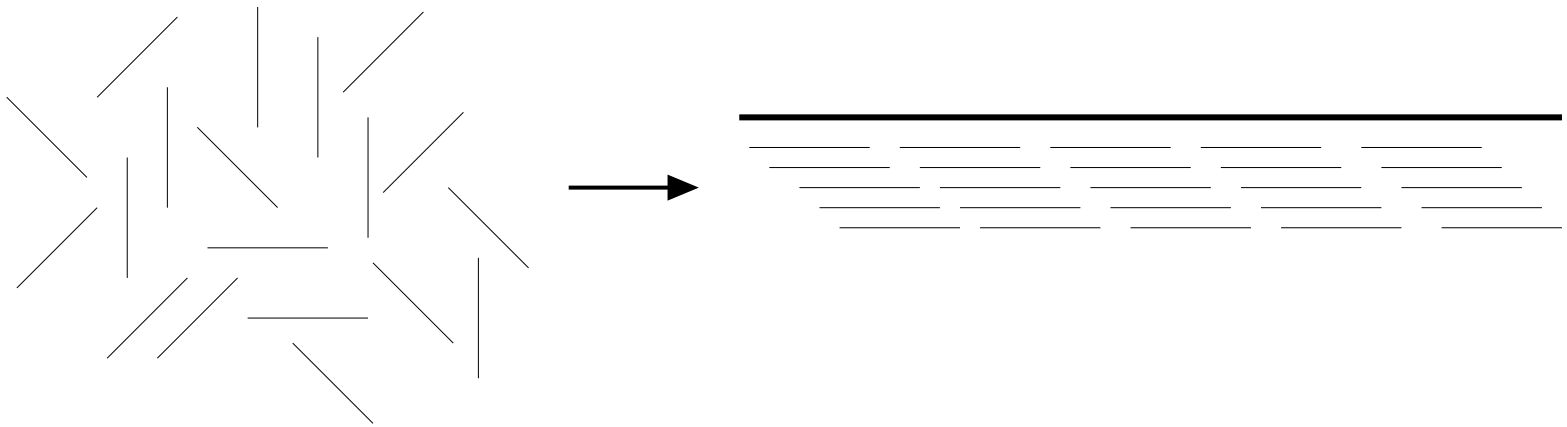
Courant Institute of Mathematical Sciences
New York University

March 2009

Outline

Shotgun Sequencing

- The DNA sequence of an organism is sheared into a large number of small fragments, the ends of the fragments are sequenced, then the resulting sequences are joined together using a computer program called *assembler*.



Basic Assumption

- Two sequence reads (two strings of letters produced by the sequencing machine) that share a same string of letters originated from the same place in the genome.
- Using such overlaps between the sequences, the assembler can join the sequences together in a manner similar to solving a jigsaw puzzle.

Assembly Modules

1. Overlap Detection/Pairing
2. Genome Assembly
3. Consensus Generation
4. Scaffolding
5. Assembly Validation
6. Finishing

AMOS

A Modular Open-Source assembler

- Bank
  - Central data-structure consisting of a collection of indexed files comprising assembly related objects (reads, inserts, overlaps, contigs, scaffolds, etc). Programs in the assembly pipeline communicate with each other using the bank as an intermediate storage space.
- Overlapper
  - Use "minimizers" technique for reducing the number of k-mers considered in the initial phase of overlapping by an order of magnitude.
- Consensus Computation
  - Parametric implementation of the Churchill-Waterman algorithm for computing the consensus base from a column in a multiple alignment of reads.
- Hawkeye visualizer
  - Facilitate inspection of large-scale assembly data minimizing the time needed to detect mis-assemblies and make accurate judgments of assembly quality.

Assembly Algorithms

- Greedy Algorithms (TIGR,Phrap,CAP3)
- Graph-based Algorithms (CELERA, EULER)
- ML Algorithms & Genetic Algorithms (Parsons et al.)
- Brute-Force Algorithms (SUTTA)

Shortest Superstring Problem

First approximation

Researchers first approximated the shotgun sequence assembly problem as one of finding the shortest common superstring of a set of sequences:

Definition (Shortest Superstring Problem)

Given a set of strings $\{s_1, s_2, \ldots, s_n\}$ *find the shortest string* T *such that* $\forall i$, s_i *is a substring of* T.

- This is an NP-hard problem!
- It does not correctly model the assembly problem. It can produce compression and other assembly errors that are associated with non-random structures in Eukaryotic genomes (e.g., repeated regions, rearrangements, segmental duplications).

Graph-Theoretical Approaches

- Overlap-Layout-Consensus
  - construct a graph in which nodes represent reads and edges indicate overlaps. The set of contigs is represented as the set of nonintersecting simple paths in the graph.
- Sequencing by Hybridization (Eularian Path)
  - Create a virtual SBH problem by breaking the reads into overlapping n-mers (an n-mer is a substring of length n). Build a deBruijn graph in which each edge is an n-mer and and the source and destination nodes are respectively the $n-1$ prefix and $n-1$ suffix of the corresponding n-mer. Find a path that uses all the edges (an Eulerian path).

Fragment Representation

Basic definitions

- A set of fragments/reads $F = \{f_1, f_2, \ldots, f_N\}$.
- Each fragments is represented as pairs of integers:

$$f_i = (s_i, e_i), i \in [1, |F|] \tag{1}$$

  where $1 \leq s_i, e_i \leq |R|$, and R is the *reconstructed* string.
- The following convention applies:
  - if $s_i \leq e_i$ then $R[s_i, e_i]$ is the substring of R covered by fragment f_i
  - if $e_i \leq s_i$ then $R[e_i, s_i]^c = R[s_i, e_i]$ is the substring of R^c covered by fragment f_i
- The order of s_i and e_i encodes the orientation of the fragment read in the layout (whether f_i was sampled from R or it complement strand R^c).
- We define the start-point sp_i and end-point ep_i of read f_i in the layout as follows:

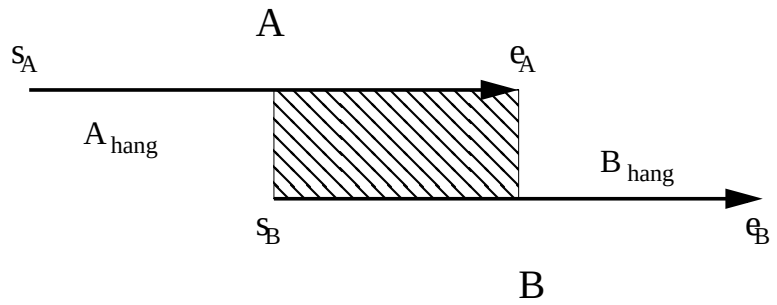
$$sp_i = \min\{s_i, e_i\}, \qquad ep_i = \max\{s_i, e_i\} \tag{2}$$

Overlap Representation

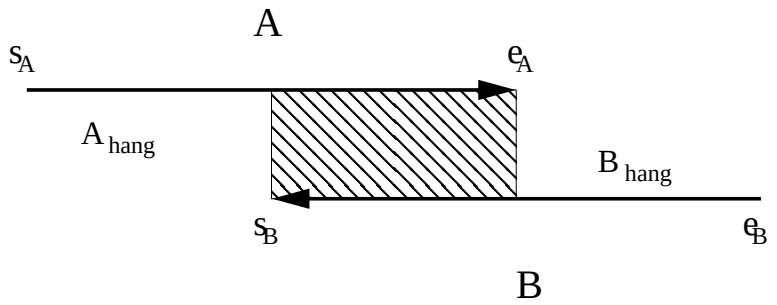
- The complete specification of an overlap π is given by specifying:
  1. the substrings $\pi.A[\pi.s_A, \pi.e_A]$ and $\pi.B[\pi.s_B, \pi.e_B]$
  2. the offsets from the left-most and right-most positions of the reads $\pi.A_{hang}$ and $\pi.B_{hang}$
  3. the relative directions of the two reads: Normal (N), Innie (I)
  4. a predicate $suffix_\pi(R)$ on a read R s.t.:

$$suffix_\pi(R) = \begin{cases} true \ \ iff \ \text{the suffix of } R \text{ participates in the overlap } \pi \\ false \ \ iff \ \text{the prefix of } R \text{ participates in the overlap } \pi \end{cases} \quad (3)$$

- The first read is always normalized in the forward orientation.



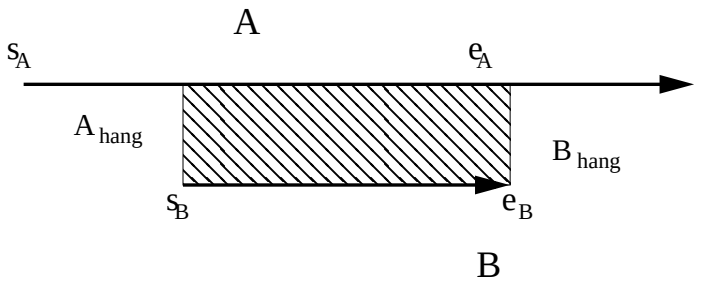
Normal



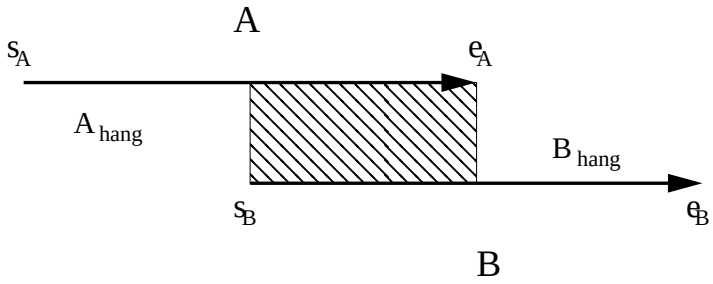
Innie

Overlap Taxonomy

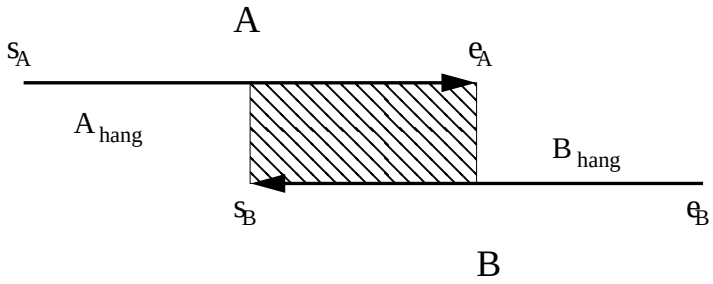
Containment



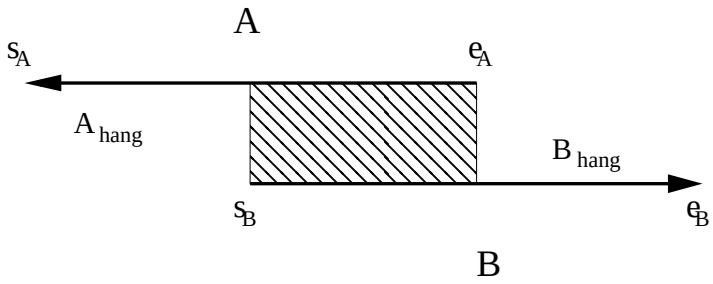
Regular dovetail



Suffix dovetail



Prefix dovetail



Layout Representation

- Let us define the layout L associated to a set of fragments $F = \{f_1, f_2, \ldots, f_N\}$ as follows:

$$L = f_1 \overset{\pi_1}{\rightleftharpoons} f_2 \overset{\pi_2}{\rightleftharpoons} f_3 \overset{\pi_3}{\rightleftharpoons} \cdots \overset{\pi_{N-1}}{\rightleftharpoons} f_N \tag{4}$$

  where there are no containments (contained reads can be initially removed and then added later after the layout has been created)

- A layout L is *consistent* if the following property holds:

$$\overset{\pi_{i-1}}{\rightleftharpoons} f_i \overset{\pi_{i+1}}{\rightleftharpoons} \quad iff \quad suffix_{\pi_{i-1}}(f_i) \neq suffix_{\pi_{i+1}}(f_i) \tag{5}$$

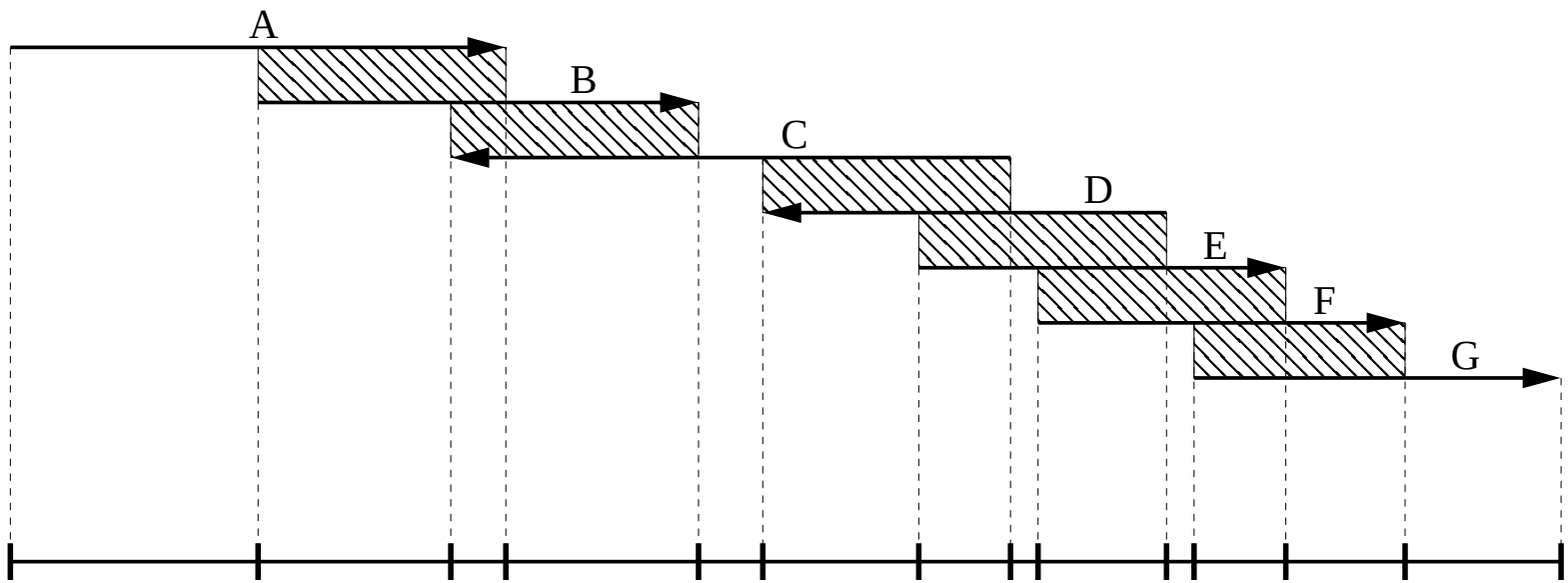
- The estimated start positions for each fragment is given by:

$$sp_1 = 1, \quad sp_i = sp_{i-1} + \pi_{i-1}.hang_{f_{i-1}} \quad if\ i > 1 \tag{6}$$

Layout Representation

Example

Layout for a set of fragments $F = \{A, B, C, D, E, F, G\}$ with a sequence of overlaps $\pi^N_{(A,B)}, \pi^I_{(B,C)}, \pi^N_{(C,D)}, \pi^I_{(D,E)}, \pi^N_{(E,F)}, \pi^N_{(F,G)}$



Min-length reconstruction theorem

Let us define the length of an overlap to be the average length of the two overlapping substrings:

$$length(\pi) = \frac{(|\pi.s_A - \pi.e_A| + |\pi.s_B - \pi.e_B|)}{2} \tag{7}$$

and let us define the length of a layout $|L|$ to be the sum of the lengths of its edges:

$$weight(L) = \sum_{\pi \in L} length(\pi) \tag{8}$$

then the following theorem holds:

Theorem (Tarhio & Ukkonen (88) and Turner (89))

A layout of maximum weight results in a reconstruction of minimum length

Min-length reconstruction theorem

Proof

First note that:

$$|L| = sp_n + |f_N| = \sum_{i=1}^{N-1} \pi_i.hang_{f_i} + |f_N| \tag{9}$$

using the fact that

1. $\pi_i.hang_{f_i} \approx |f_i| - length(\pi_i)$
2. $length(\pi_i) \approx |g|$ when π is a containment edge and g is the contained fragment

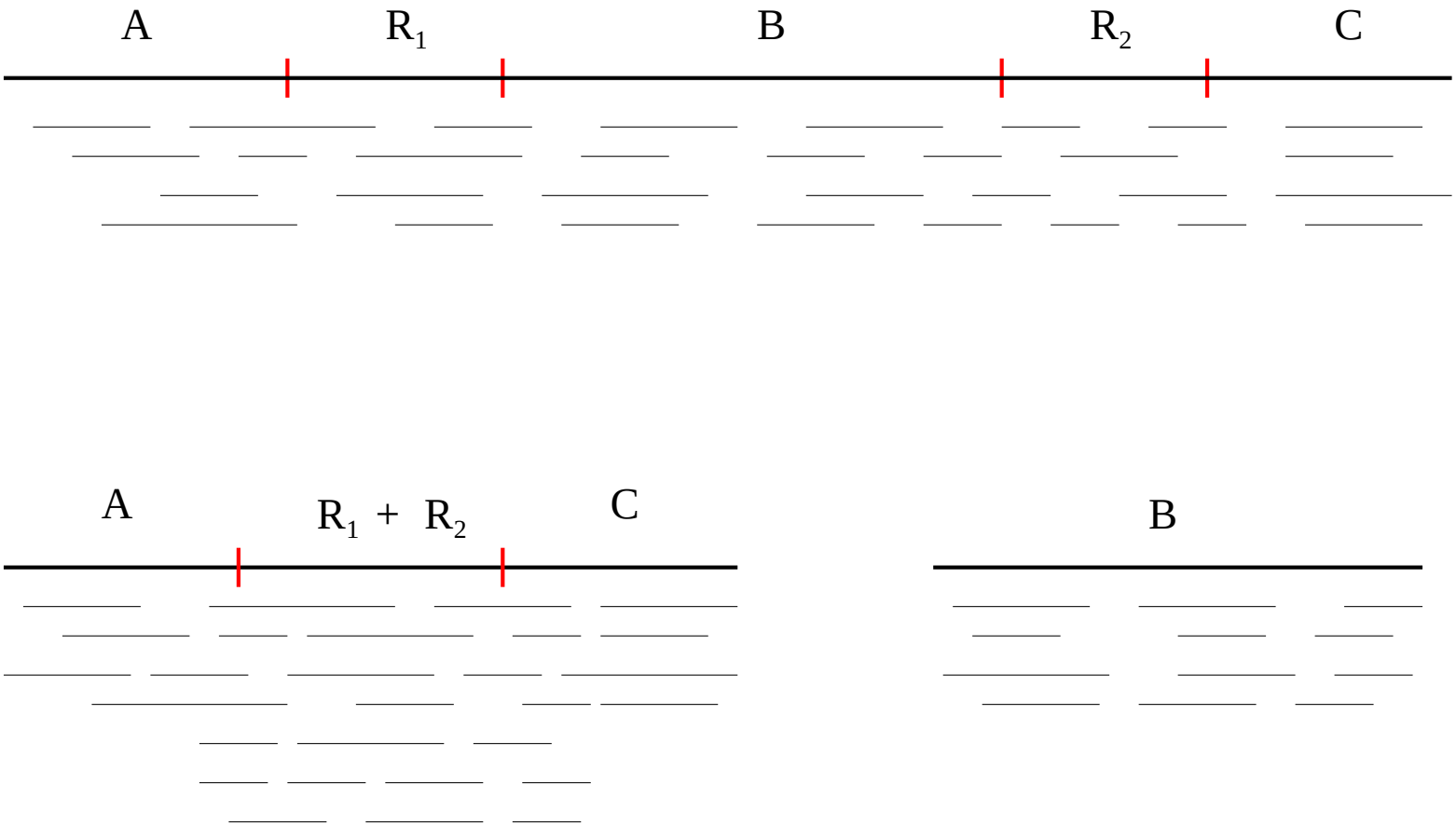
it follows that:

$$|L| = \sum_{f} |f| - \underbrace{\sum_{\pi} length(\pi)}_{weight(L)} \tag{10}$$

but the second sum is the weight of the layout. Thus maximizing weight minimizes length.

A few problems

- Although the previous theorem suggests to look for a reconstruction of maximum weight (minimum length), the reconstructed string can still have many errors due to *repeats*.



Genome Sequence Assembly problem

Definition (Genome Sequence Assembly Problem)

Given a set of fragments/reads $F = \{f_1, f_2, \ldots, f_n\}$ find a reconstruction R and a valid layout of the reads L such that:

- *The observed distribution of fragment reads start point, D_{obs}, has the minimum deviation from the source distribution D_{src}*
- *The distance between mated reads must be consistent with the size of the fragments generated.*
- *The observed distribution of restriction enzyme cutting point, C_{obs} is consistent with the distribution of experimental optical map data C_{src}.*
- *...*

This formulation assumes no errors in the fragments.

SUTTA

Main Ingredients

- Exhaustive
  - Not a greedy algorithm.
  - Avoid getting stuck with a locally best solution
- Use Branch-and-Bound (or Beam-Search) algorithm to improve algorithmic complexity.
  - Provide bounds and allow pruning of unpromising regions/directions.
  - Implement by "dove-tailing" between local (short sequence-reads) and global (long-range maps and haplotypic) information.
  - Tune heuristically (e.g., size of a priority queue) to get the best computational complexity and resource consumption for a specific error parameters and required accuracy
  - Exploit underlying 0-1 laws
  - Parallelize in a straight-forward way

SUTTA

The power of scoring

- Use a "score" function to choose the best global solution.
  - Achieve high accuracy
  - Model the "error processes" in the score, consisting of Bayesian likelihood and penalty functions
  - Use side-information (e.g., optical maps, mated pairs, base-content, homologous reference sequences, etc.) to sharpen the score function
  - Use empirical-Bayes method to decide the statistics (null-model, threshold, p-values, base- or sequence-quality)
  - Agnostic to the underlying technology, while being able to mix-and-match technologies

SUTTA

Pseudo-Code

Algorithm 1: SUTTA - pseudo code

Input: Set of N reads
Output: Set of contigs

```
1  F := ∅;                                   /* Forest of trees */
2  C := ∅;                                   /* Set of contigs */
3  B := ⋃_i^N R_i;                           /* All the available reads */
4  while (B ≠ ∅) do
5  |  R := B.getNextRead();
6  |  if ( !R.isUsed() && !R.isContained() ) then
7  |  |  DT := create_double_tree(R);
8  |  |  F.add(DT);
9  |  |  Contig CTG := create_contig(DT);
10 |  |  C.add(CTG);
11 |  |  CTG.layout();    /* Compute layout of the contig */
12 |  |  B := B \ {CTG.reads};          /* Remove used reads */
13 |  else
   |  |                  /* jump to next available read */
14 |  end
15 end
16 return C
```

Node expansion

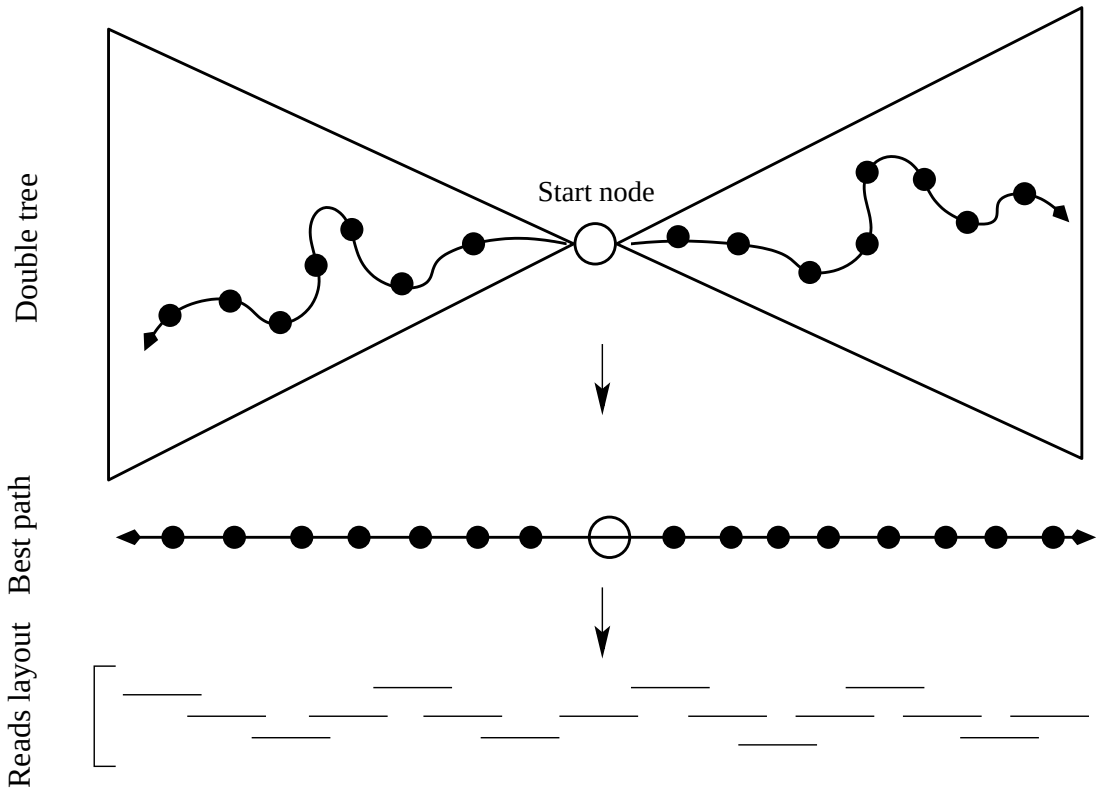
High-level Description

1. Start with a random read (It will be the root of a tree; Use only the read that has not been "used" in a contig yet, or that is not "contained")
2. Create RIGHT Tree: Start with an unexplored leaf node (a read) with the best score-value; Choose all its non-contained "right"-overlapping reads and expand the node by making them its children; Compute their scores. (Add the "contained" nodes along the way, while including them in the computed scores; Check that no read occurs repeatedly along any path of the tree). STOP when the tree cannot be expanded any further.
3. Create LEFT Tree: Symmetric to previous step.

Double Tree

Illustration

- The expand node routine is applied twice to generate LEFT and RIGHT trees for the start read.
- Next, the best LEFT path is concatenated with the root and the best RIGHT path to create a globally optimal contig.



Node expansion

Branch-and-Bound

Algorithm 2: Node expansion - Branch-and-Bound

Input: Start read $\mathcal{R}_0$, max queue size K
Output: Contig for the input read

```
1  T := ∅;                                          /* Set of leaves */
2  L := {(R_0, g(R_0))}; /* Set of live nodes (priority queue)
   */
3  while (L ≠ ∅) do
4  |   L := Prune(L, K);        /* Prune the queue to size K */
5  |   R_i := L.getNext();
6  |   L := L \ {R_i};
7  |   if (no reads align with R_i) then
8  |   |   T := T ∪ {R_i};                          /* R_i is a leaf */
9  |   else
10 |   |   Add contained reads to R_i;
   |   |         /* Branch on R_i generating R_{i_1}, R_{i_2}, ... R_{i_M} */
11 |   |   for (j=1 to M) do
12 |   |   |   L := L ∪ {(R_{i_j}, g(R_{i_j}))};
13 |   |   end
14 |   end
15 end
16 return max_{R_i ∈ T} {g(R_i)};
```

Node expansion

Beam-Search

Algorithm 3: Node expansion - Beam-Search

Input: Start read $\mathcal{R}_0$, max queue size K
Output: Contig for the input read

```
1  T := ∅;                                          /* Set of leaves */
2  L := {(R_0, g(R_0))};  /* Set of live nodes (FIFO queue) */
3  while (L ≠ ∅) do
4  |   Sort(L);  /* Sort live nodes based on their score */
5  |   L := Prune(L, K);      /* Prune the queue to size K */
6  |   for (i=1 to min(K, |L|)) do
7  |   |   R_i := L.getNext();
8  |   |   L := L \ {R_i};
9  |   |   if (no reads align with R_i) then
10 |   |   |   T := T ∪ {R_i};                    /* R_i is a leaf */
11 |   |   else
12 |   |   |   Add contained reads to R_i;
   |   |   |       /* Branch on R_i generating R_{i_1}, R_{i_2}, ... R_{i_M} */
13 |   |   |   for (j=1 to M) do
14 |   |   |   |   L := L ∪ {(R_{i_j}, g(R_{i_j}))};
15 |   |   |   end
16 |   |   end
17 |   end
18 end
19 return max_{R_i ∈ T} {g(R_i)};
```

Notes

- Use Branch-and-Bound (or Beam Search) to avoid exponential space and time complexity.
- Use depth-first search interval schemes to see if a read occurs repeatedly along a path.
- Only check right- or left-overlapping properties between two reads while expanding the root; checking just the *consistency* relation for the non-root node suffices.
- Caution must be taken in avoiding reads from the best right path to be included in any left path.
- Some book-keeping must be done to keep track of "used", "explored", "overlapping", and "contained" relationships.

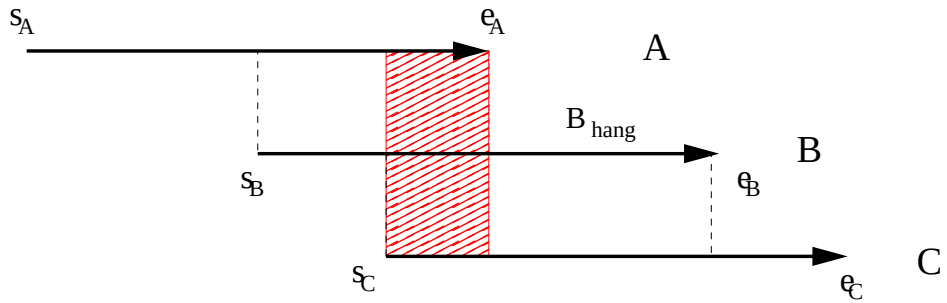
Computational Complexity

- Function of the Queue Size
  - Higher values clearly increase the computational time, however, due to the 0-1 law phenomena, optimal values can be chosen to reduce complexity maintaining the quality of results.
- Time to compute
  - Few seconds for influenza (12 Kbp) and zgene .
  - 20 minutes for Brucella suis.
  - 1hr for Wolbachia sp.
- Function of Score-Bounds
  - Strong bounds on the score function will allow to drastically reduce the search space (and the computational time) with minimum lost in quality.

Score Function

- Started with a very simple score function:
  - We used a "weighted transitivity" score that formulates the following intuition: if read *A* overlaps read *B*, and read *B* overlaps read *C*, we will score those overlaps strongly if in addition *A* and *C* also overlap. This implicitly assumes that the coverage is higher than 3.

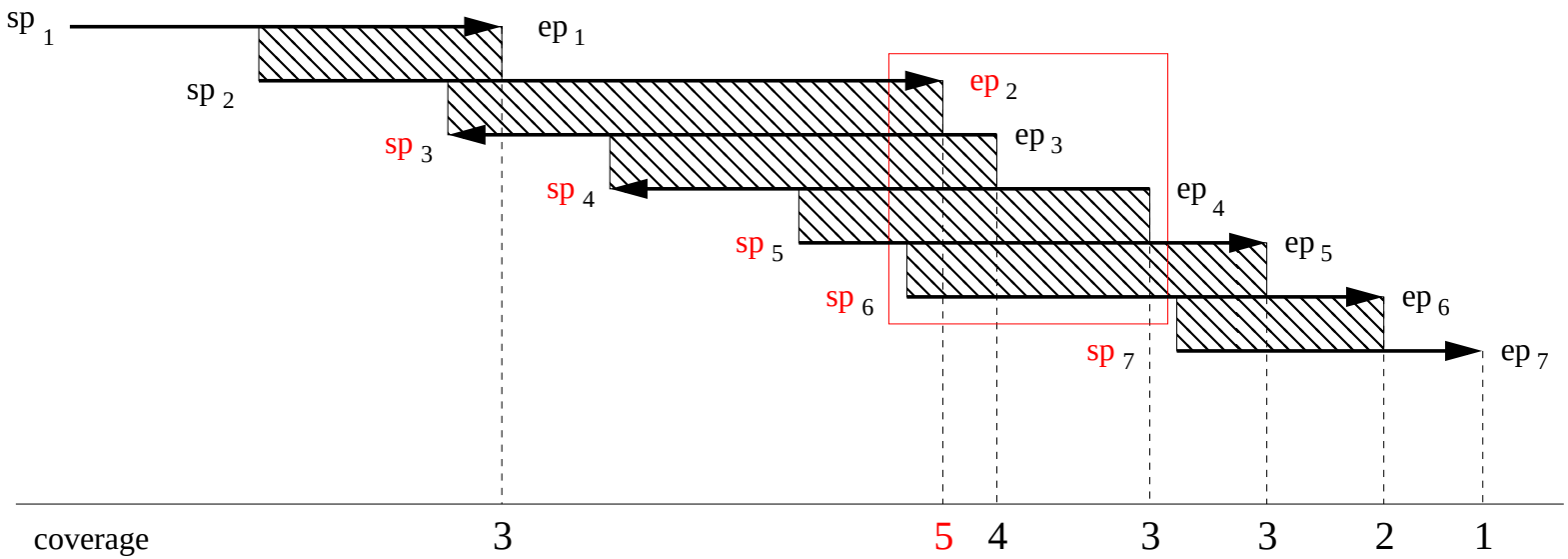
$$\texttt{if}(\pi_{(A,B)} \wedge \pi_{(B,C)})\texttt{than}\{S_{\pi_{(A,B,C)}} = S_{\pi_{(A,B)}} + S_{\pi_{(B,C)}} + (\pi_{(A,C)} ? S_{\pi_{(A,C)}} : 0)\} \quad (11)$$



  - A simple generalization for higher coverage is obvious.
- This score cannot resolve repeats or haplotypic variations. Solution: augment the score with information for optical map alignment or mated-pair distances to put an appropriate reward/penalty term.

Dynamic Coverage Score

- **Observation**: compressed (expanded) regions are characterized by an increase (decrease) in the depth of coverage compared to the expected average coverage of the shotgun process.
- **Idea**: penalize solutions whose observed coverage deviates from the expected coverage of the shotgun process.

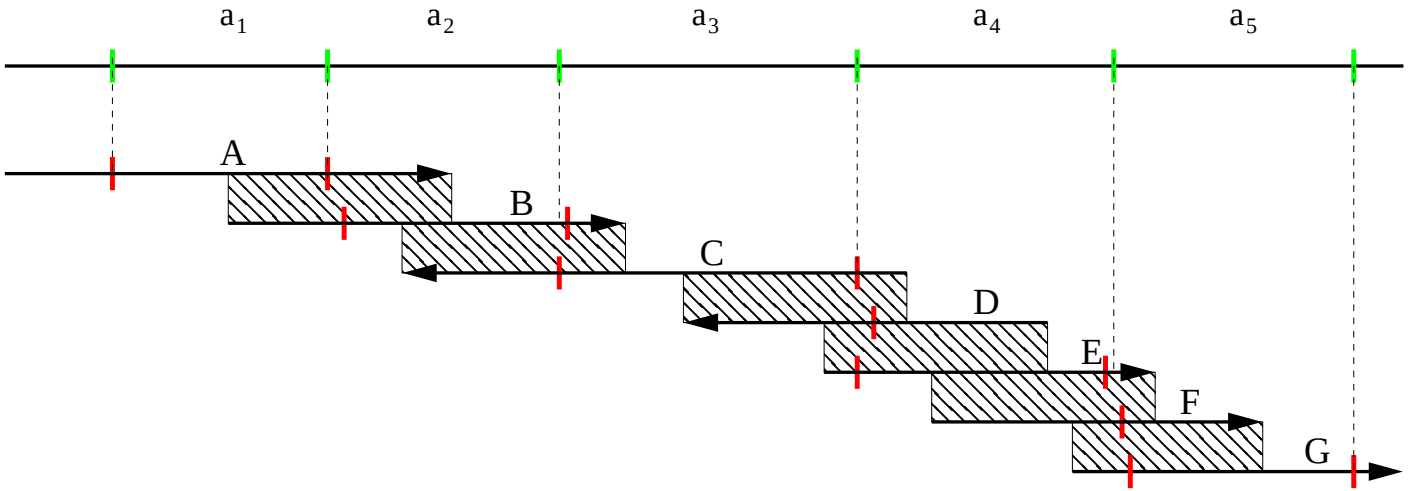


Optical Map Score

- **Observation**: restriction enzymes cut at precise locations in the genome. Let $< a_1, a_2, \ldots, a_n >$ be the *restriction map* obtained by a restriction enzyme digestion process.
- **Idea**: Organize the restriction sizes a_i into n-tuples. Build an hash table according to:

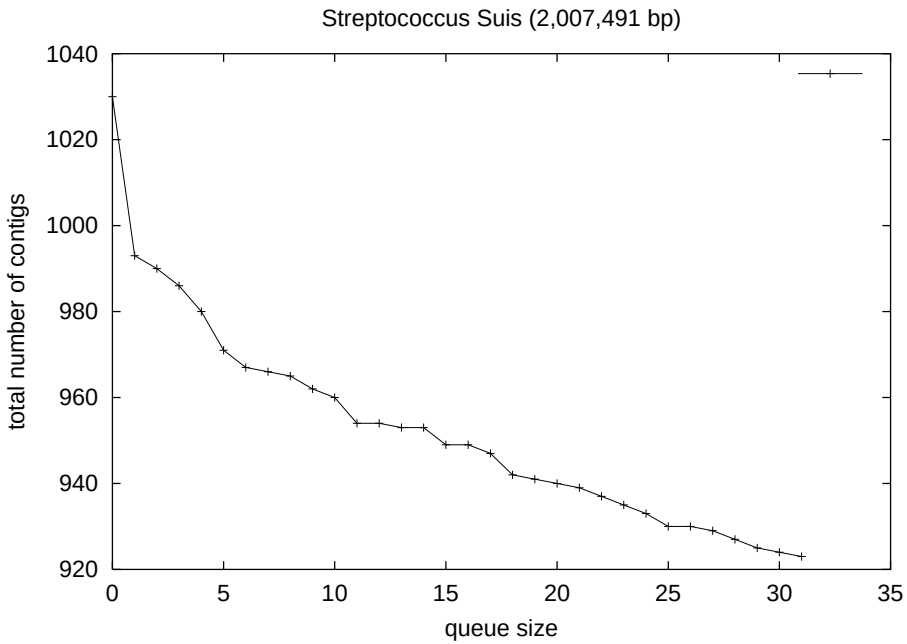
$$\mathcal{H}(a, b_1, \ldots, b_n) =< \left\lfloor \frac{b_1}{a} \times \alpha \right\rfloor, \left\lfloor \frac{b_2}{a} \times \alpha \right\rfloor, \ldots, \left\lfloor \frac{b_n}{a} \times \alpha \right\rfloor > \qquad (12)$$

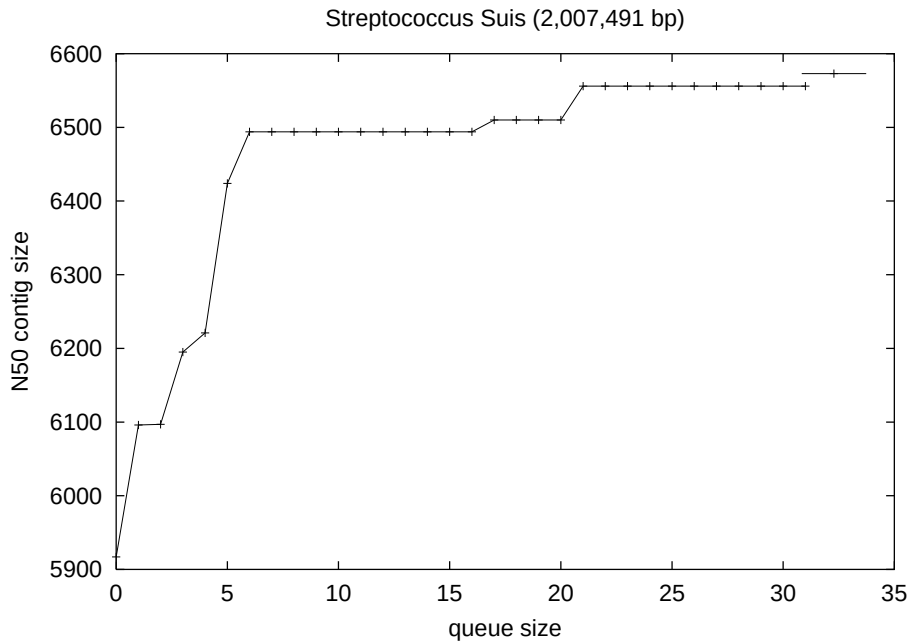
  store a in the corresponding slot (with possible collisions).
- Create an in-silico map of the candidate solution and score it according to the number of *hits* that its n-tuples have in the hash table.



Streptococcus Suis strain P1/7 - 2,007,491 bp

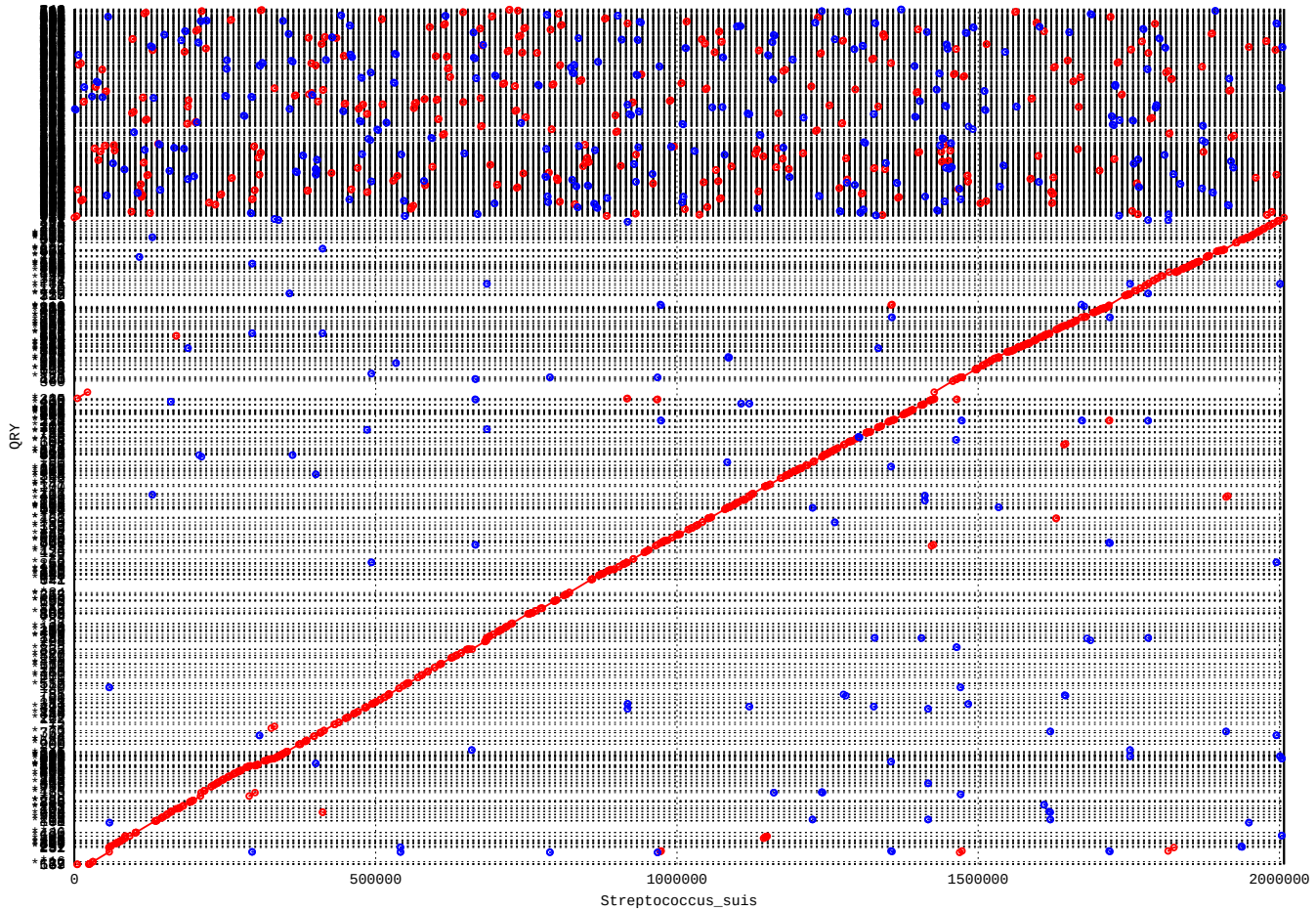
Queue size analysis





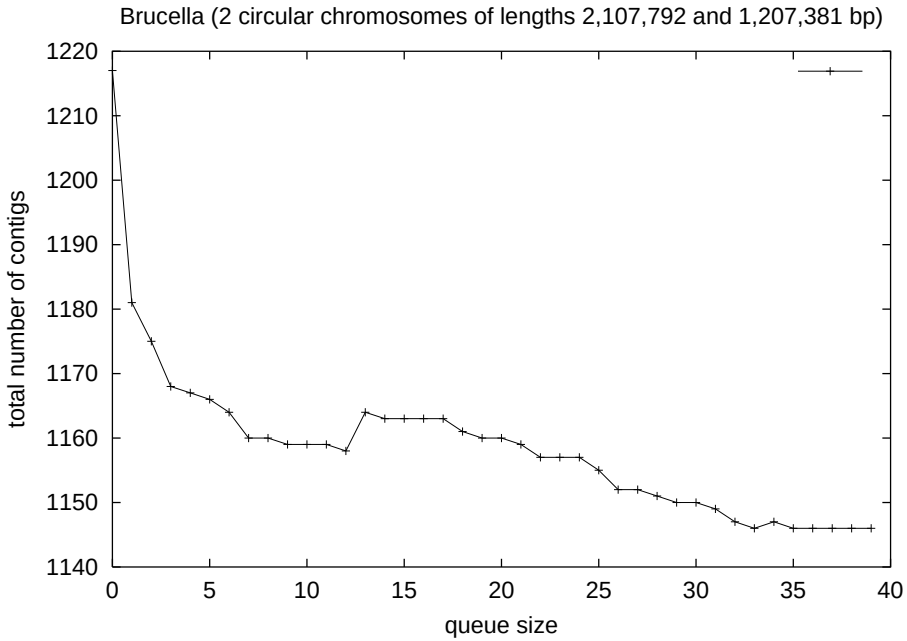
Streptococcus Suis strain P1/7 - 2,007,491 bp

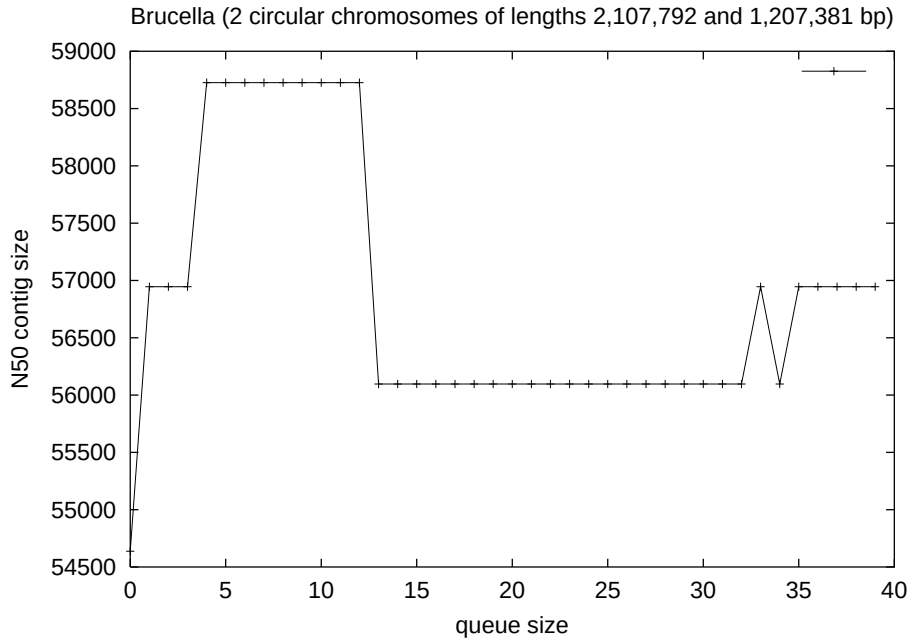
DotPlot



Brucella Suis
2 chromosomes of 2,107,792 and 1,207,381 bp

Queue size analysis

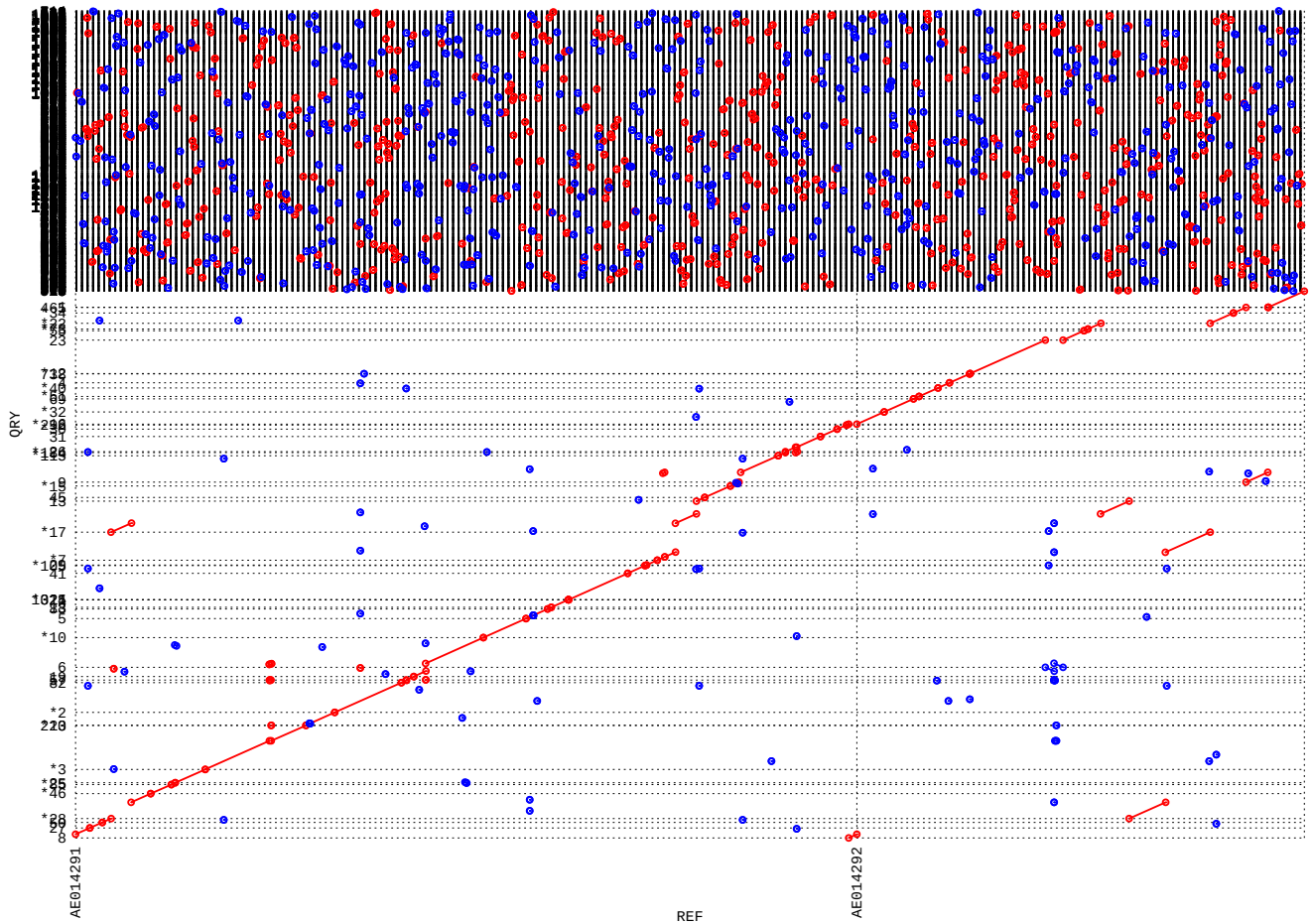




Brucella Suis

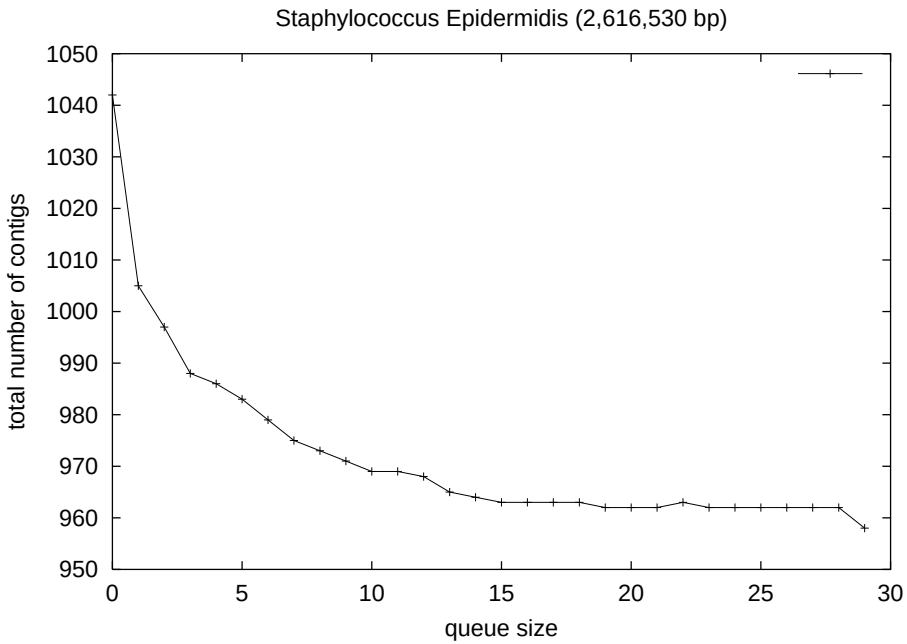
2 chromosomes of 2,107,792 and 1,207,381 bp

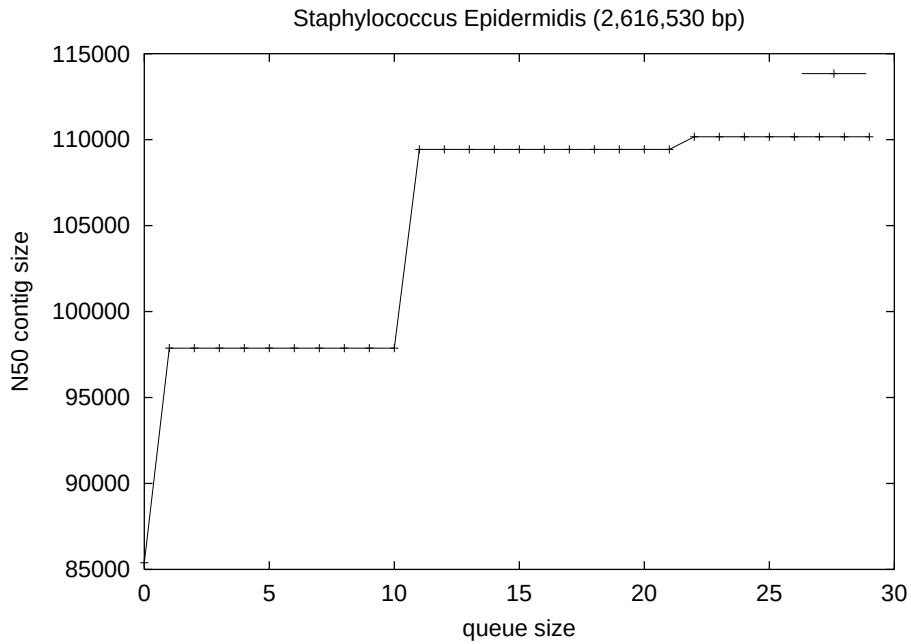
DotPlot



Staphylococcus Epidermidis - 2,616,530 bp

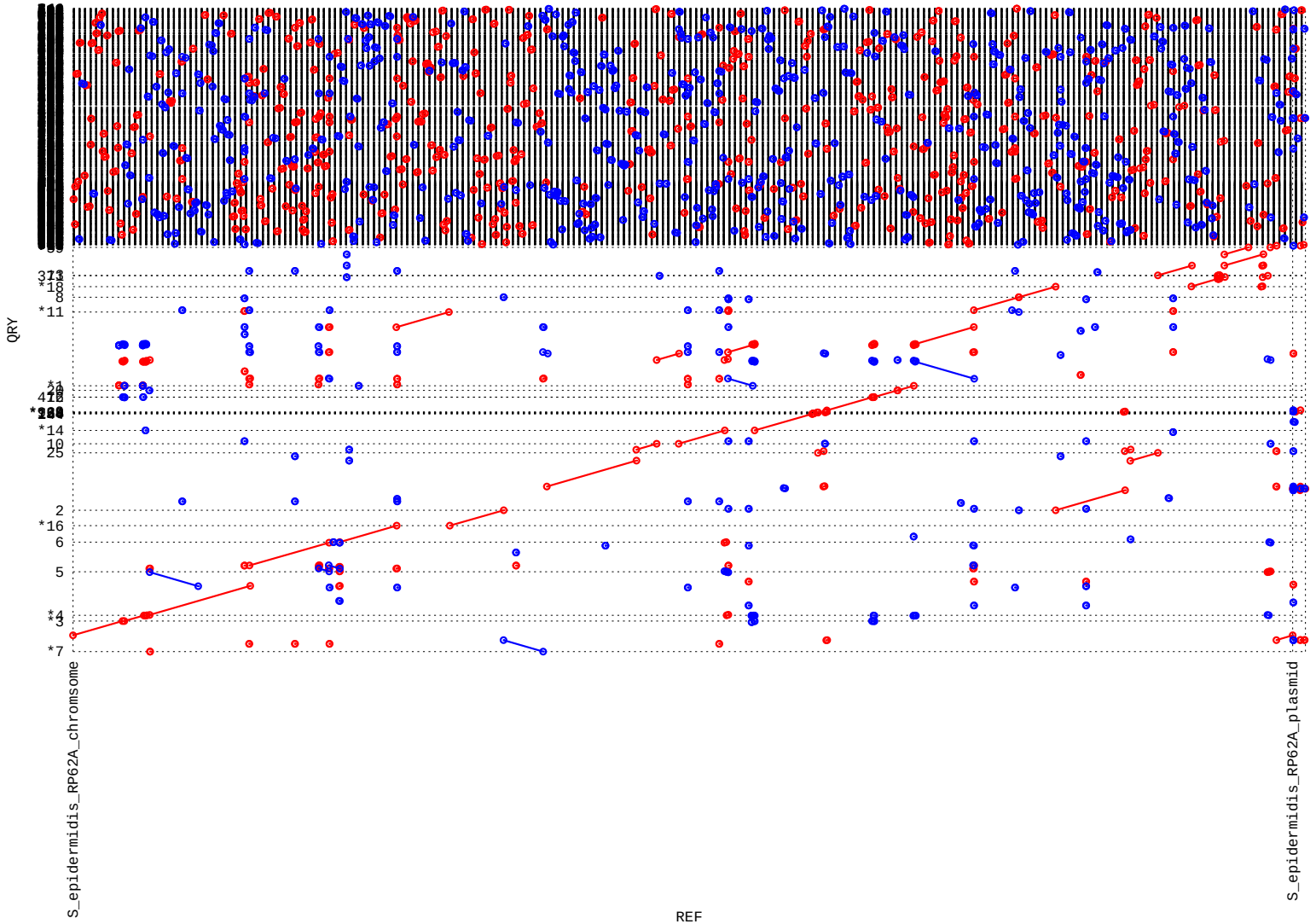
Queue size analysis





Staphylococcus Epidermidis - 2,616,530 bp

DotPlot



For Further Reading

Sun Kim, Haixu Tang and Elaine R. Mardis
Genome Sequencing Technology and Algorithms.
Artech House Publishers, 1 edition (October 31, 2007).

Myers EW.
Toward simplifying and accurately formulating fragment assembly.
J Comput Biol. 1995 Summer; 2(2):275-90.

Kececioglu and Myers.
Combinatorial algorithms for DNA sequence assembly.
Algorithmica (1995) vol. 13 (1-2) pp. 7-51

Adam M Phillippy et al.
Genome assembly forensics: finding the elusive mis-assembly.
Genome Biology (2008) vol. 9 (3) pp. R55