From Sequencing to Sequences:

Algorithms and Metrics for Accurate Genome Assembly

Bud Mishra

Courant Institute of Mathematical Sciences
New York University

&
Cold Spring Harbor Laboratory

~~~~

August 26, 2011
~~~~

Outline

1. **Introduction**
   - Genome Sequencing and Assembly
   - Issues and Challenges
2. **SUTTA assembler**
   - Algorithm details (Scoring, Pruning & Lookahead)
3. **Feature-Response Curve**
   - Motivation
   - Results
4. **Base-Calling and Assembly**
   - Re-sequencing (TotalReCaller)
   - Integrating Base-Calling and Assembly

Outline

1. **Introduction**
   - Genome Sequencing and Assembly
   - Issues and Challenges
2. **SUTTA assembler**
   - Algorithm details (Scoring, Pruning & Lookahead)
3. **Feature-Response Curve**
   - Motivation
   - Results
4. **Base-Calling and Assembly**
   - Re-sequencing (TotalReCaller)
   - Integrating Base-Calling and Assembly

Outline

1. **Introduction**
   - Genome Sequencing and Assembly
   - Issues and Challenges
2. **SUTTA assembler**
   - Algorithm details (Scoring, Pruning & Lookahead)
3. **Feature-Response Curve**
   - Motivation
   - Results
4. **Base-Calling and Assembly**
   - Re-sequencing (TotalReCaller)
   - Integrating Base-Calling and Assembly

DNA sequencing

(HiSeq 2000 from Illumina, Inc)

$\Downarrow$

200 GBp in 8 Days

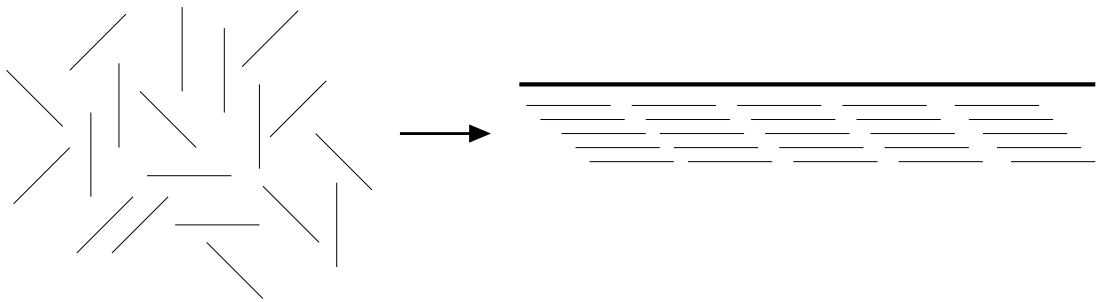
$\approx$ **50x coverage** of a human genome of **100Bp sequence reads**

$\Downarrow$

No human halpotypic genome assembly yet

Shotgun sequence assembly

- DNA sequence is sheared into a large number of small fragments.



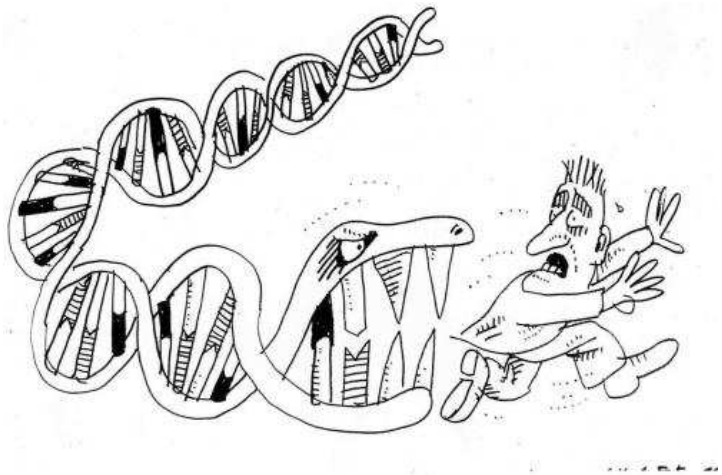
- **Assume**: If two sequence reads share the same string of letters (overlap), then they might have originated from the same genomic location.
- **Goal**: Join the sequences together using a computer program called assembler (similar to solving a jigsaw puzzle).
- Use long-range data to resolve complex genomic structures.

Outline

1. **Introduction**
   - Genome Sequencing and Assembly
   - Issues and Challenges
2. **SUTTA assembler**
   - Algorithm details (Scoring, Pruning & Lookahead)
3. **Feature-Response Curve**
   - Motivation
   - Results
4. **Base-Calling and Assembly**
   - Re-sequencing (TotalReCaller)
   - Integrating Base-Calling and Assembly

Why is de-novo sequence assembly so difficult?

1. $\mathcal{NP}$-complete: natural reduction to the *Shortest Superstring Problem* (easy for totally random DNA sequences).
2. Genomic structures: repeated regions, rearrangements, segmental duplications etc.
3. Sequencing-Technology Dependent: algorithms must change to accommodate changes to read-length or nature and availability of long-range information.



The Sense of the Approximation

A wicked problem in search for a correct solution

Definition (Wicked Problem)

A **wicked** problem is a problem that is difficult or impossible to solve because of *incomplete*, *contradictory*, and *changing* requirements that are often difficult to recognize.

Incomplete, contradictory, changing requirements = genome structure

$\Downarrow$

Not complete and biologically correct mathematical formulation!

$\Downarrow$

Difficult to have a *sense of the approximation* of the sequence relative to the true sequence as they are being assembled

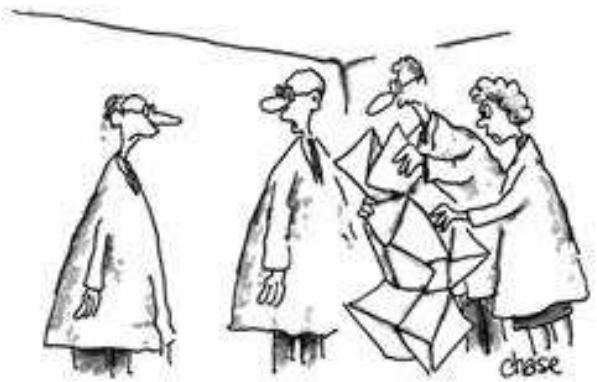
Genome Sequencing – History & Accuracy

Did we solve the problem?

- **1995**: *Haemophilus Influenzae* - 1.8 Mbp, $\sim$ 30h.
- **2000**: *Drosophila* - 120 Mbp, $\sim$ week.
- **2001**: 1st Human Genome draft - 3 billion bp (genotypic), cost: $3 billion!.

How well did we do?

- High rates of misassembly.
  [Semple, *Bioinformatics for Geneticists*, 2003]
- "Revolution Postponed: Why the Human Genome Project Has Been Disappointing"
  [Stephen S. Hall, *Scientific American*, 2010]
- Need for Quality Assessment! $\Rightarrow$ Assemblathon (but only very recently, 2011).



Why did we not try to do better?

- "Since the problem is $\mathcal{NP}$-hard (shortest superstring), any efficient reconstruction procedure must resort to heuristics." [Kececioglu and Myers, *Algorithmica*, 1995].

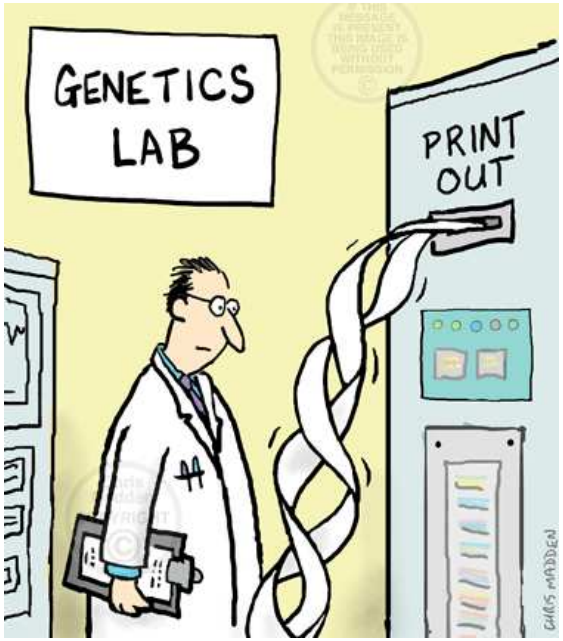
List of current Sequence Assemblers

Name	Read Type	Algorithm	Reference
SUTTA	long & short	B&B	(Narzisi and Mishra, 2010)
Arachne	long	OLC	(Batzoglou et al., 2002)
CABOG	long & short	OLC	(Miller et al., 2008)
Celera	long	OLC	(Myers et al., 2000)
Edena	short	OLC	(Hernandez et al., 2008)
Minimus (AMOS)	long	OLC	(Sommer et al., 2007)
Newbler	long	OLC	454/Roche
CAP3	long	Greedy	(Huang and Madan, 1999)
PCAP	long	Greedy	(Huang et al., 2003)
Phrap	long	Greedy	(Green, 1996)
Phusion	long	Greedy	(Mullikin and Ning, 2003)
TIGR	long	Greedy	(Sutton et al., 1995)
ABySS	short	SBH	(Simpson et al., 2009)
ALLPATHS	short	SBH	(Butler et al., 2008)
ALLPATHS-LG	short	SBH	(Gnerre et al., 2010)
Contrail	short	SBH	(Schatz M. et al., 2010)
Euler	long	SBH	(Pevzner et al., 2001)
Euler-SR	short	SBH	(Chaisson and Pevzner, 2008)
Ray	long & short	SBH	(Boisvert et al., 2010)
SOAPdenovo	short	SBH	(Li et al., 2010)
Velvet	long & short	SBH	(Zerbino and Birney, 2008)
PE-Assembler	short	Seed-and-Extend	(Nuwantha and Sung, 2010)
QSRA	short	Seed-and-Extend	(Bryant et al., 2009)
SHARCGS	short	Seed-and-Extend	(Dohm et al., 2007)
SHORTY	short	Seed-and-Extend	(Hossain et al., 2009)
SSAKE	short	Seed-and-Extend	(Warren et al., 2007)
Taipan	short	Seed-and-Extend	(Schmidt et al., 2009)
VCAKE	short	Seed-and-Extend	(Jeck et al., 2007)

Issues and Challenges

with current assemblers

- Sequencing Technology dependent.
- Difficult to integrate other bio-technologies (e.g., optical maps).
- Validation as a post-process.



Challenges of new sequencing technology:

- Short read lengths (up to 500 bps).
- Lots of data (requires distributed systems).

Need for novel and more flexible assembly platforms

Outline

1. Introduction
   - Genome Sequencing and Assembly
   - Issues and Challenges
2. **SUTTA assembler**
   - Algorithm details (Scoring, Pruning & Lookahead)
3. Feature-Response Curve
   - Motivation
   - Results
4. Base-Calling and Assembly
   - Re-sequencing (TotalReCaller)
   - Integrating Base-Calling and Assembly

Goals

1. **Better formulation** $\Rightarrow$ constrained optimization.
2. **Better algorithms** $\Rightarrow$ more accurate.
3. **More flexibility** $\Rightarrow$ easy to integrate data from other bio-technologies.
4. **Simultaneous validation** $\Rightarrow$ using score functions.

De Novo Genome Assembly

"An assembler must either "guess" (often incorrectly) the correct genome from among a large number of alternatives (a number that grows exponentially with the number of repeats in the genome) or restrict itself to assembling only the non-repetitive segments of the genome, thereby producing a fragmented assembly."
[Pop and Salzberg, **Trends in Genetics**, 2008]

- We promote an approach with the following features:
  1. Exhaustive search (not greedy).
  2. Prune implausible overlays quickly (Branch-and-Bound).
  3. Score-functions: combine different structural properties (e.g., transitivity, coverage, physical maps, etc).
  4. Independent of the particular technology.

Outline

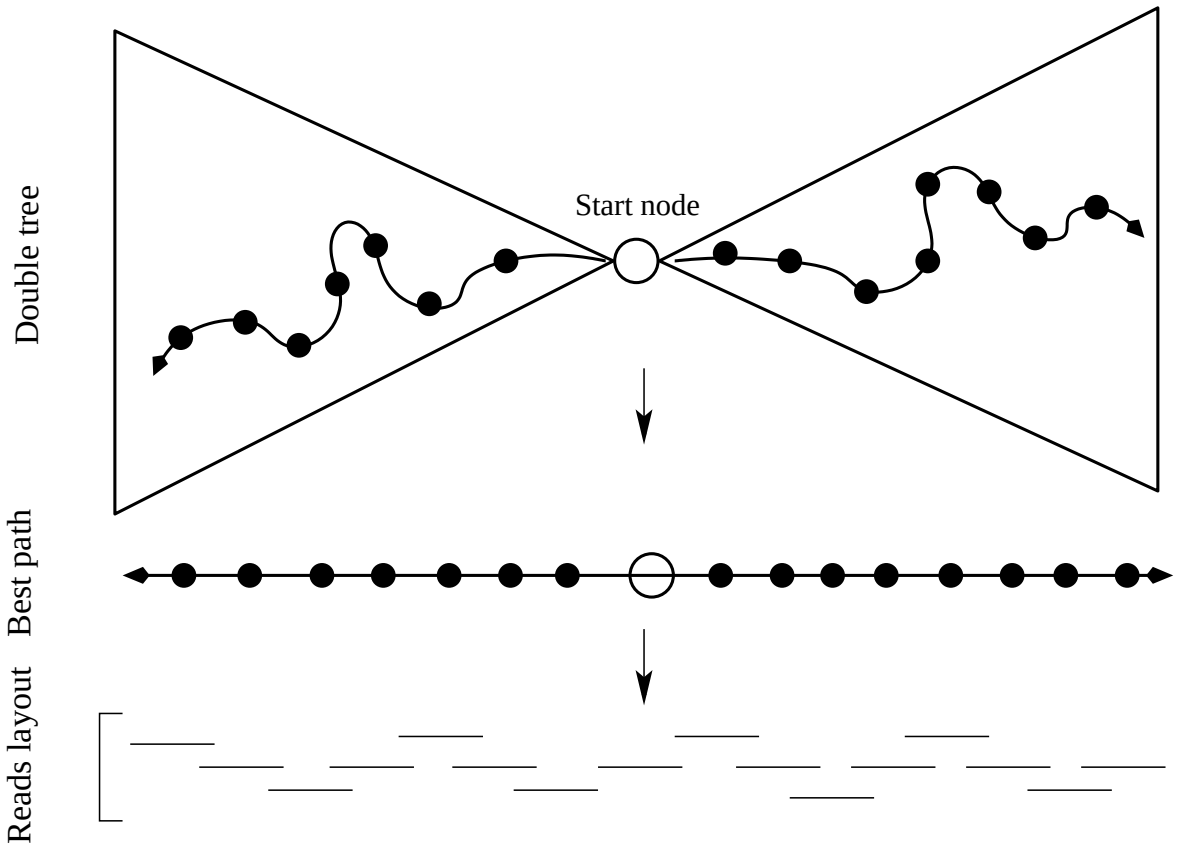
1 Introduction
- Genome Sequencing and Assembly
- Issues and Challenges

2 SUTTA assembler
- Algorithm details (Scoring, Pruning & Lookahead)

3 Feature-Response Curve
- Motivation
- Results

4 Base-Calling and Assembly
- Re-sequencing (TotalReCaller)
- Integrating Base-Calling and Assembly

SUTTA
Illustration

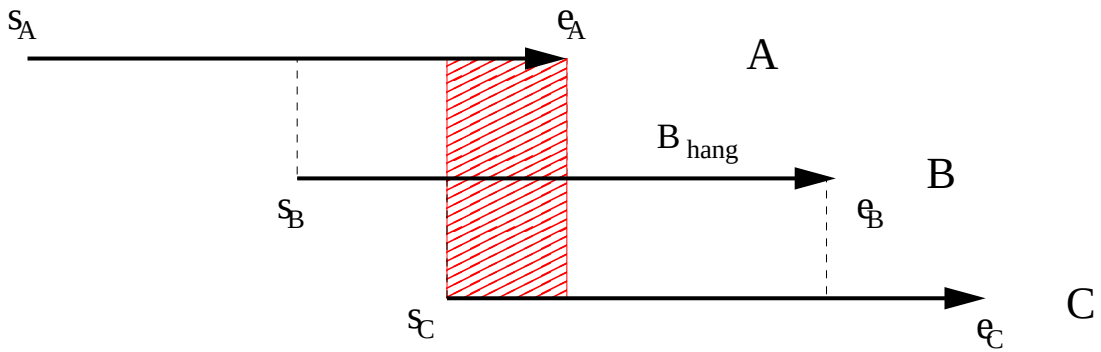


1. Generate LEFT and RIGHT trees for the start read.
2. Best LEFT path is concatenated with the root and the best RIGHT path to create a globally optimal contig.

Overlap Score
(Weighted transitivity)

- **Idea**: 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.

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

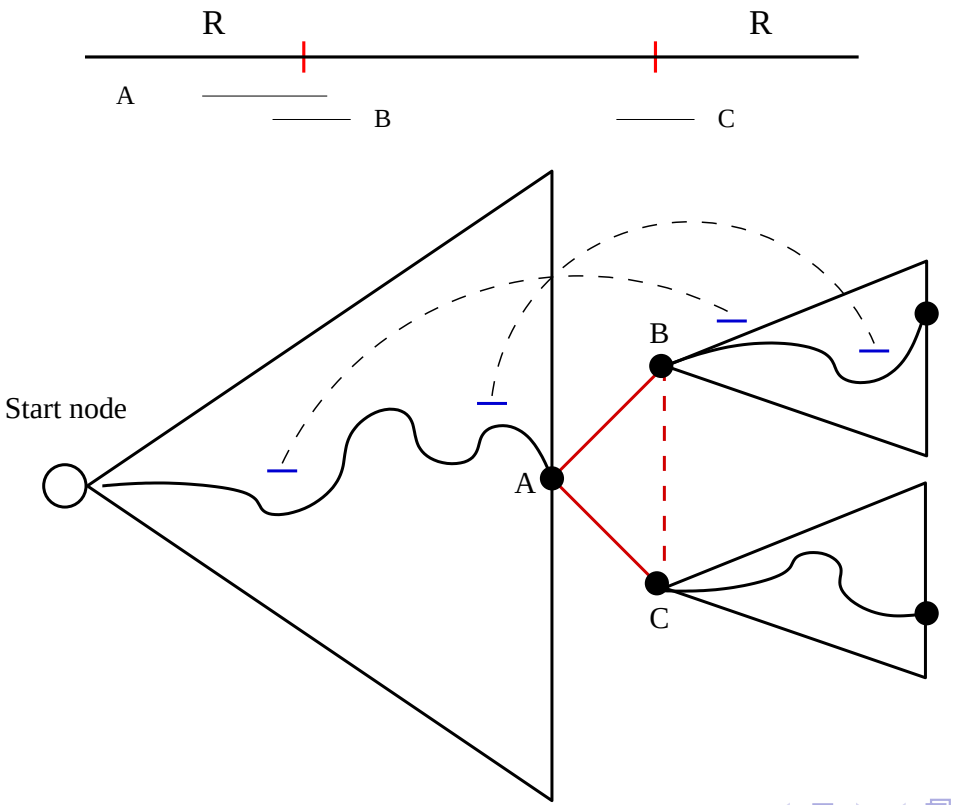


- This score cannot resolve repeats or haplotypic variations. **Solution**: augment the score with information for mate-pairs distances or optical map alignment to put an appropriate reward/penalty term.

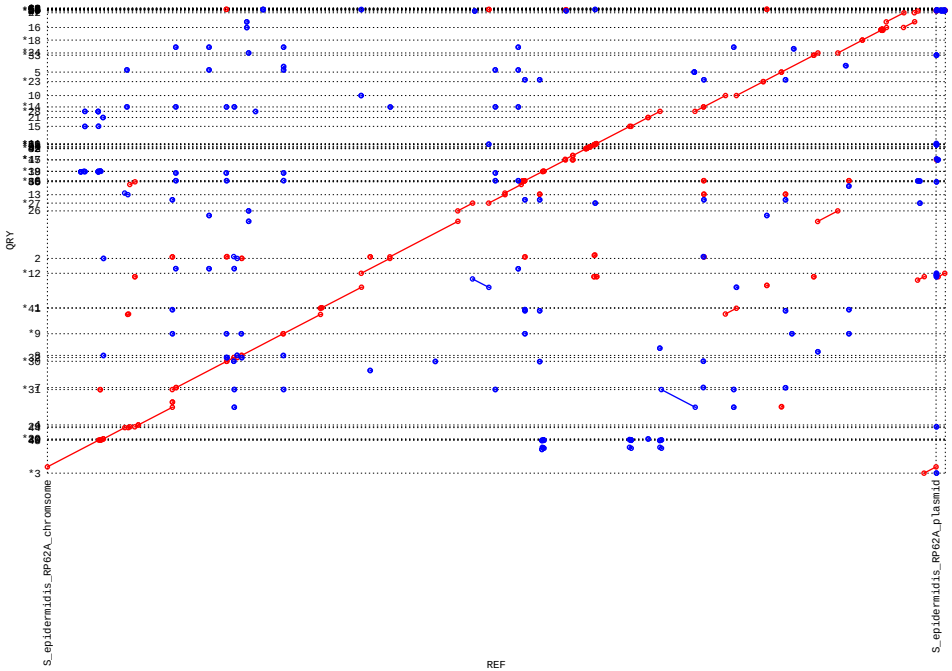
Lookahead

How to resolve repeats

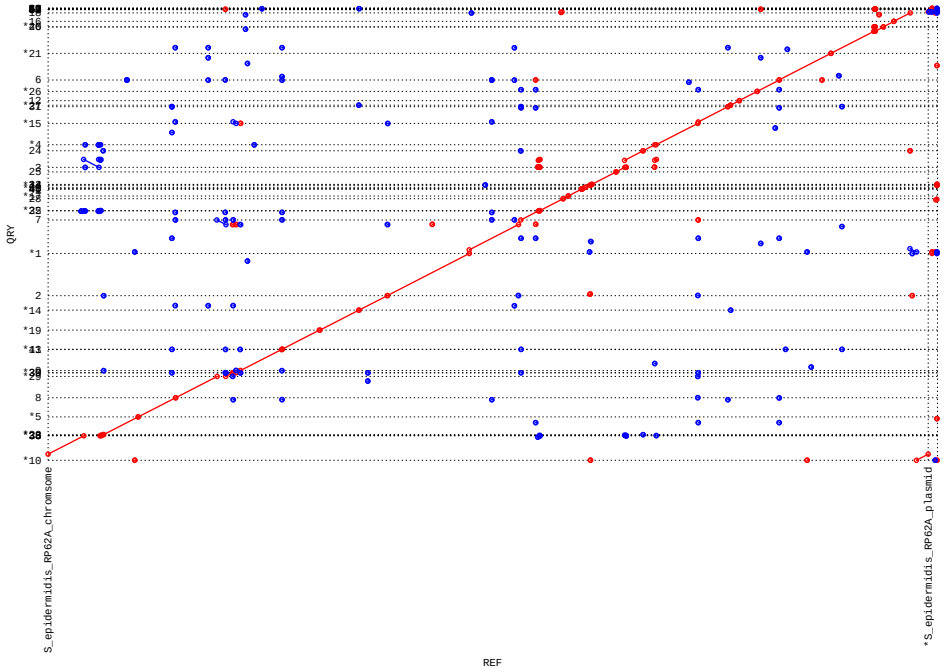
- **Scenario**: A potential repeat boundary between reads *A*, *B* and *C*. Read *A* overlaps both reads *B* and *C*, but *B* and *C* do not overlap each other.
- **Observation**: No decision can be made at this point on which read to keep/prune.
- **Idea**: Chose between reads *A* and *B* based on how well the mate-pairs (or other long-range data) in their subtree satisfy the length constraints.



Staphylococcus Epidermidis - 2,616,530 bp
(SUTTA DotPlot)



No lookahead



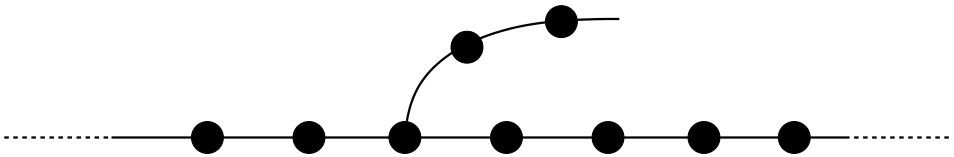
With lookahead

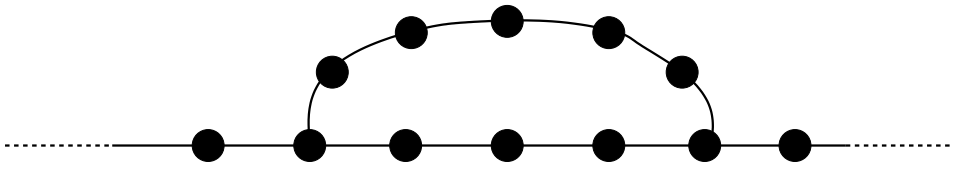
Num. of reads: 60, 761; Avg read length: 900.2; Coverage: 19.9X

Lookahead

How to resolve dead-ends and bubbles

The lookahead procedure can easily handle dead-ends and bubbles.





Dead-ends: short branches of overlaps that extend only for very few steps (associated with base errors located close to the read ends).

Strategy: prune all the branches that are shorter than W_{de}.

Bubbles: false branches that reconnect after a small number of steps (caused by single nucleotide difference carried by a small subset of reads).

Strategy: check if branches converge after W_{bb} steps. Keep the branch with higher coverage.

Outline

1. Introduction
   - Genome Sequencing and Assembly
   - Issues and Challenges
2. SUTTA assembler
   - Algorithm details (Scoring, Pruning & Lookahead)
3. **Feature-Response Curve**
   - Motivation
   - Results
4. Base-Calling and Assembly
   - Re-sequencing (TotalReCaller)
   - Integrating Base-Calling and Assembly

Outline

1 Introduction
- Genome Sequencing and Assembly
- Issues and Challenges

2 SUTTA assembler
- Algorithm details (Scoring, Pruning & Lookahead)

3 Feature-Response Curve
- Motivation
- Results

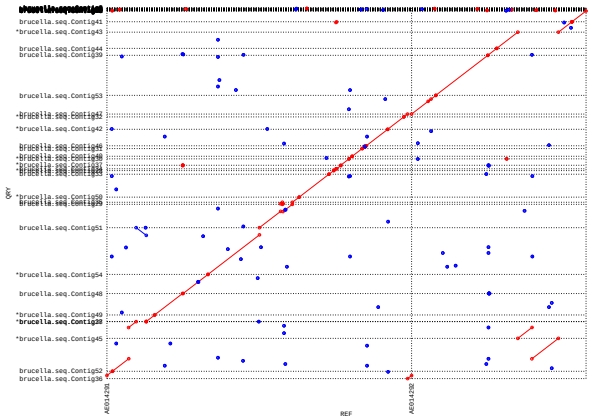
4 Base-Calling and Assembly
- Re-sequencing (TotalReCaller)
- Integrating Base-Calling and Assembly

The need for Quality Assessment
(motivation)

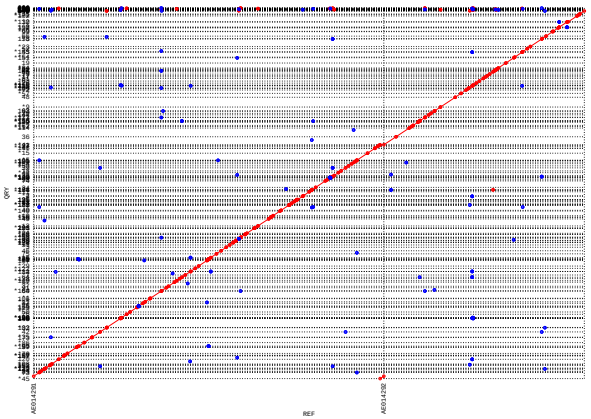
Definition (N50)

Given M contigs of size $c_1, c_2, \ldots, c_M$, N50 is defined as the largest number L such that the combined length of all contigs of length $\geq L$ is at least 50% of the total length of all contigs.

- **Problem**: emphasizes only size, without capturing quality!



Few very long contigs $\Rightarrow$ useless if mis-assembled.



Many short contigs $\Rightarrow$ too short for annotation efforts.

- Other metric: **count** the number of mis-assembled contigs by alignments to the reference genome (if available). **Problem**: error types are not weighted accordingly.

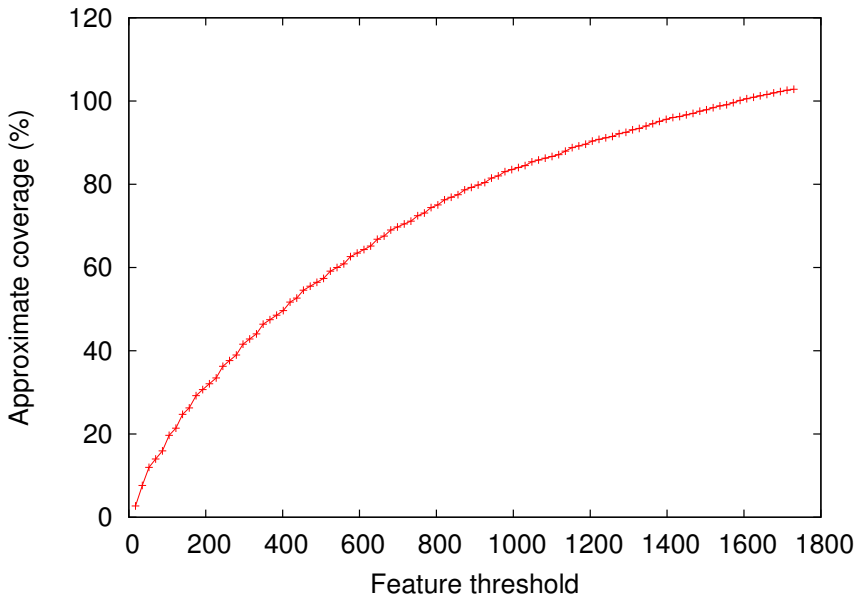
Feature-Response Curve

Goal: *evaluate the structural properties of the contigs and of the reads arranged in the layout.*

- The Feature-Response curve characterizes the sensitivity (coverage) of the sequence assembler as a function of its discrimination threshold (number of features/errors).

Features include:

- (M) mate-pair orientations and separations,
- (K) repeat content by k-mer analysis,
- (C) depth-of-coverage,
- (P) correlated polymorphism in the read alignments, and
- (B) read alignment breakpoints to identify structurally suspicious regions of the assembly.

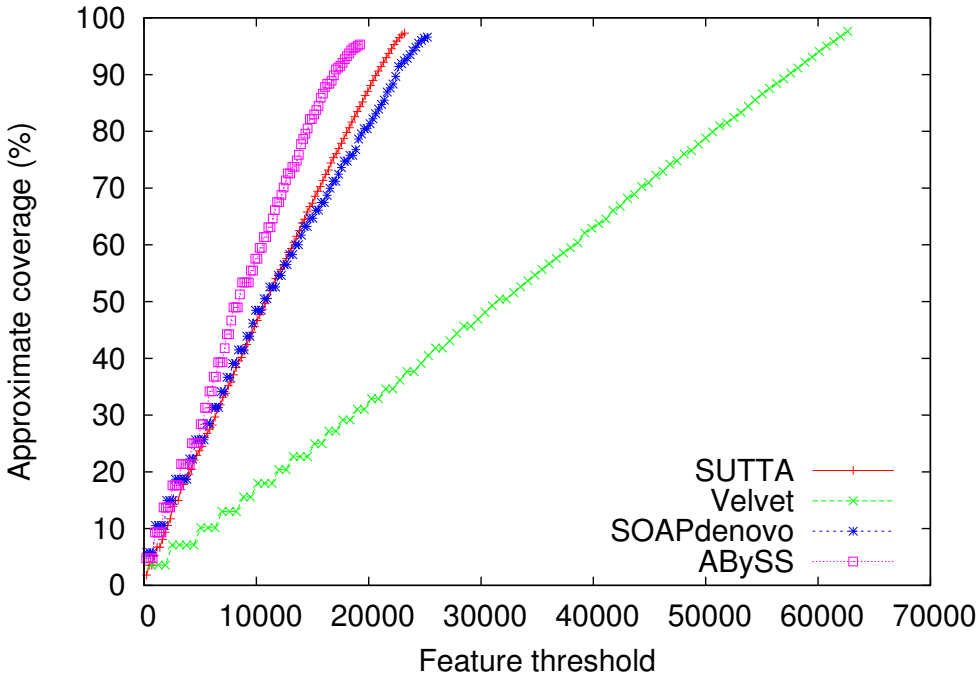


Outline

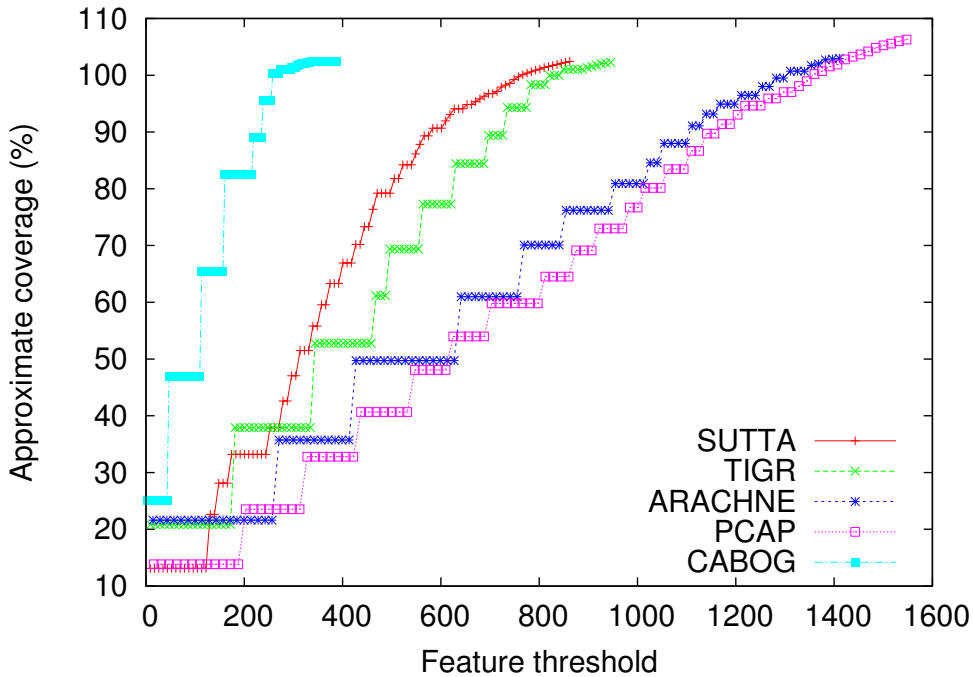
1. Introduction
   - Genome Sequencing and Assembly
   - Issues and Challenges
2. SUTTA assembler
   - Algorithm details (Scoring, Pruning & Lookahead)
3. **Feature-Response Curve**
   - Motivation
   - Results
4. Base-Calling and Assembly
   - Re-sequencing (TotalReCaller)
   - Integrating Base-Calling and Assembly

Feature-Response curve Results

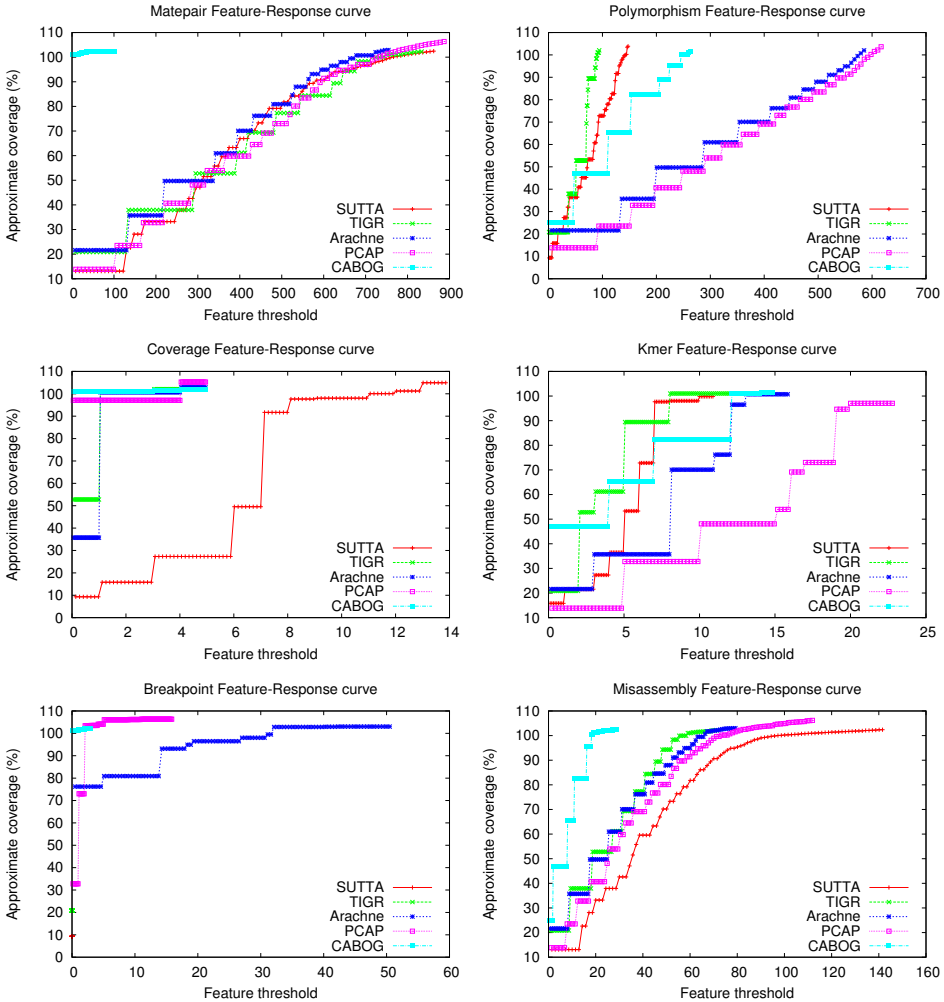
Escherichia coli 4.6 Mbp



Staphylococcus epidermidis 2.6 Mbp



Experimental Comparison
(De Novo Genome Assembly Results)



Feature-Response curve comparison by feature type

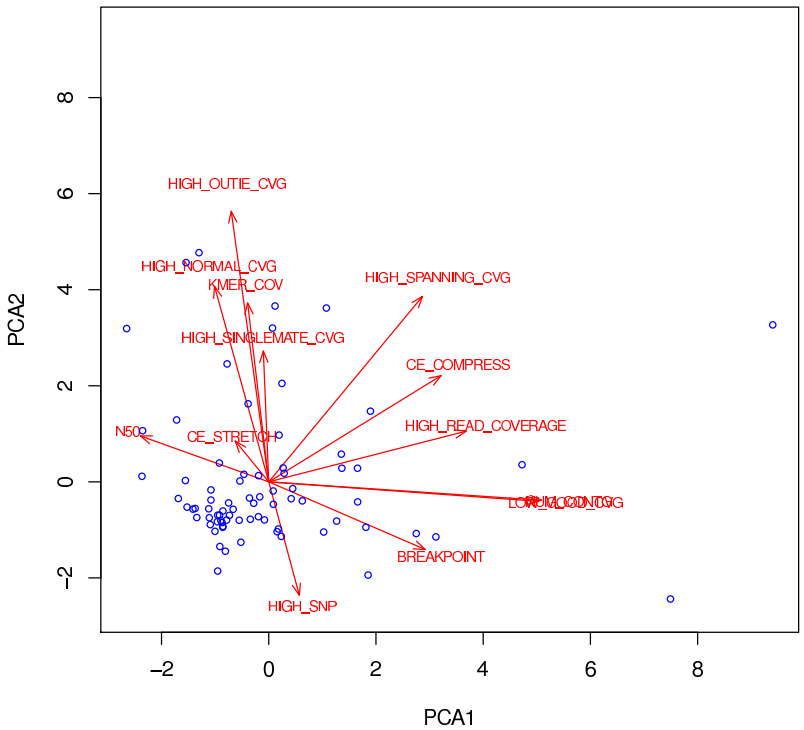
- **7 different genomes** (Bacterial and Human).
- **Simulated** and **real** data.
- **16 different sequence assemblers** (both for old Sanger and next-generation Illumina sequencing technology).
- **All the generally accepted assembly paradigms** (Greedy, OLC, SBH, Seed-and-Extend, and B&B).

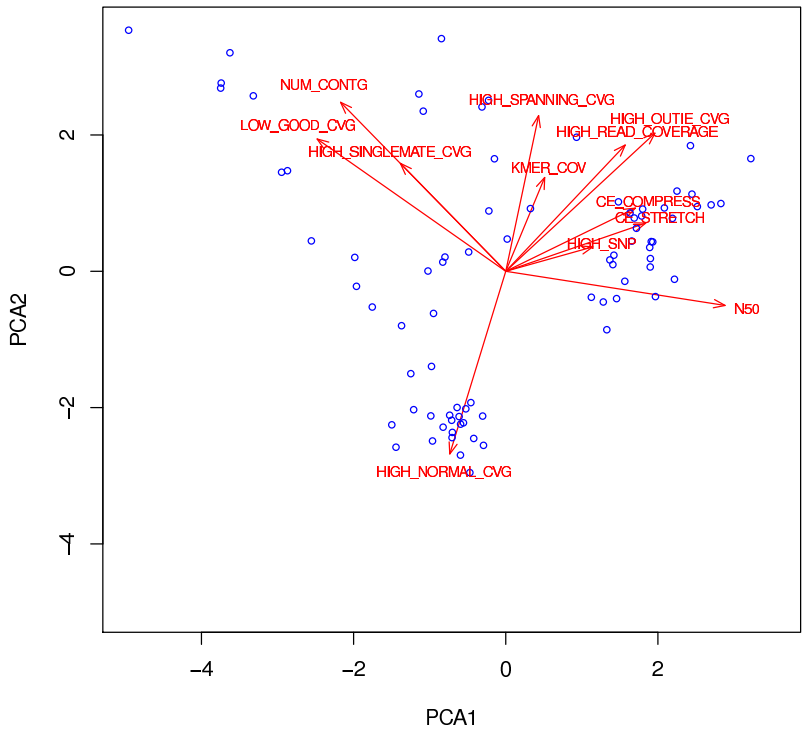
⇓

Quality and performance of the existing assemblers varies dramatically!

PCA and ICA

1. PCA to remove **redundant features**.
2. ICA to select the most **independent** (i.e. important) **features**.



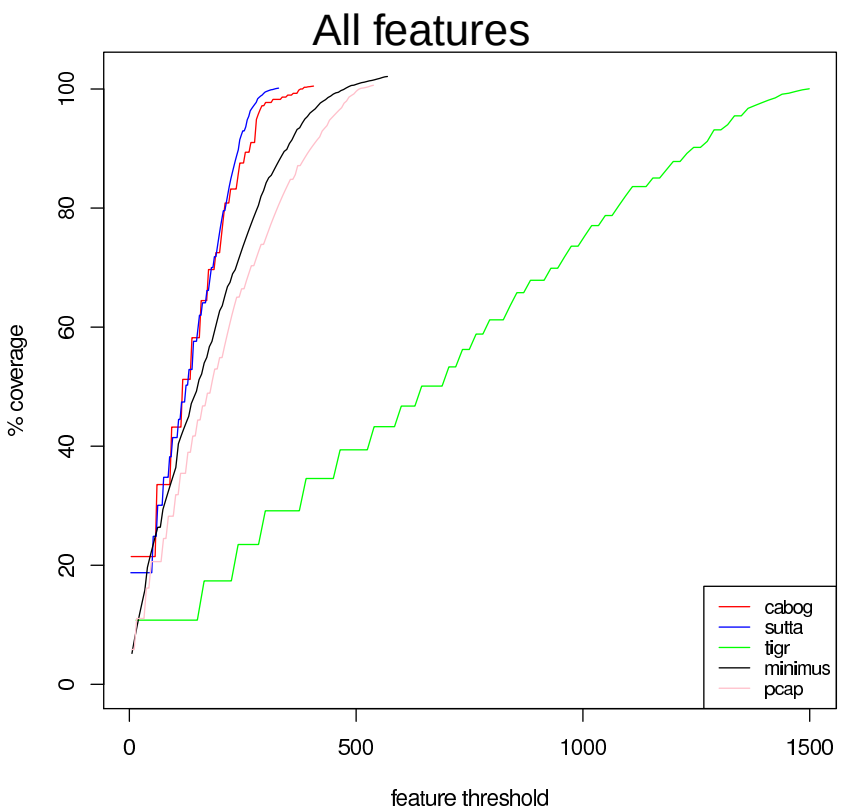


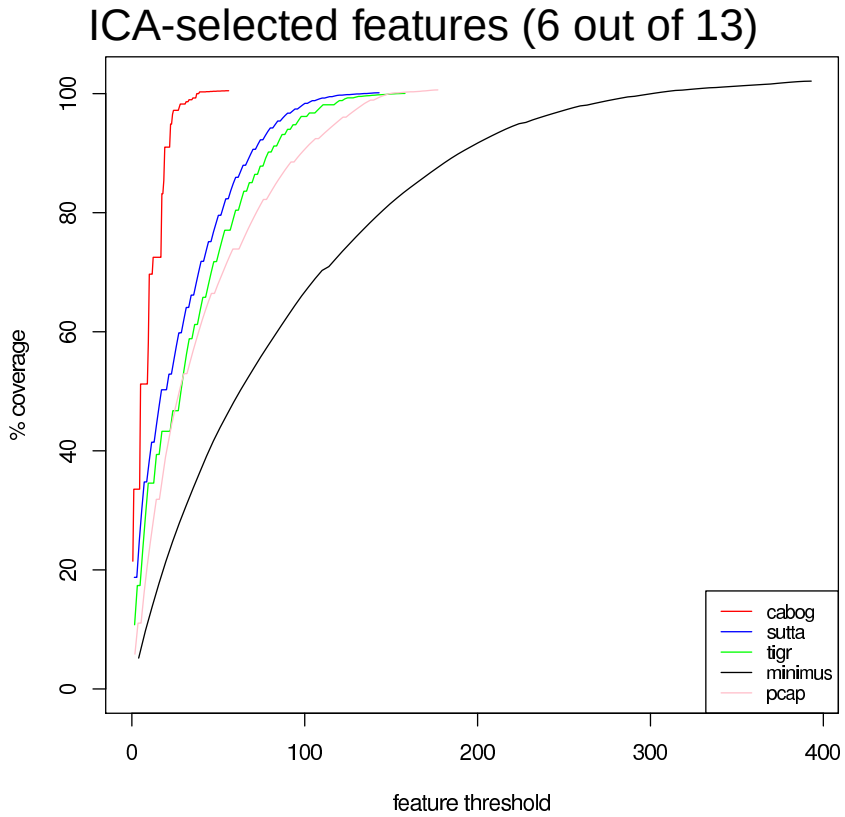
Long reads: 21 organisms with length from ∼11 Kbp to ∼8 Mbp assembled with 5 assemblers. A total of 84 assemblies were used in the analysis.

Short reads: 5 datasets with coverages ranging from 30× to 130× assembled with 5 assemblers. A total of 82 assemblies were used in the analysis.

FRCurve on ICA-selected features

Comparing 5 assemblers on the *Brucella suiss* dataset





- **Observation**: focus on the most informative features leads to better analysis.

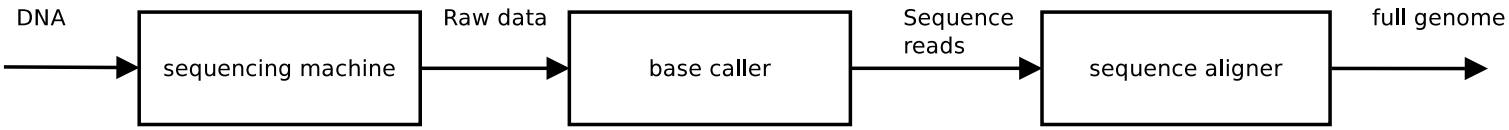
Outline

1. Introduction
   - Genome Sequencing and Assembly
   - Issues and Challenges
2. SUTTA assembler
   - Algorithm details (Scoring, Pruning & Lookahead)
3. Feature-Response Curve
   - Motivation
   - Results
4. **Base-Calling and Assembly**
   - Re-sequencing (TotalReCaller)
   - Integrating Base-Calling and Assembly

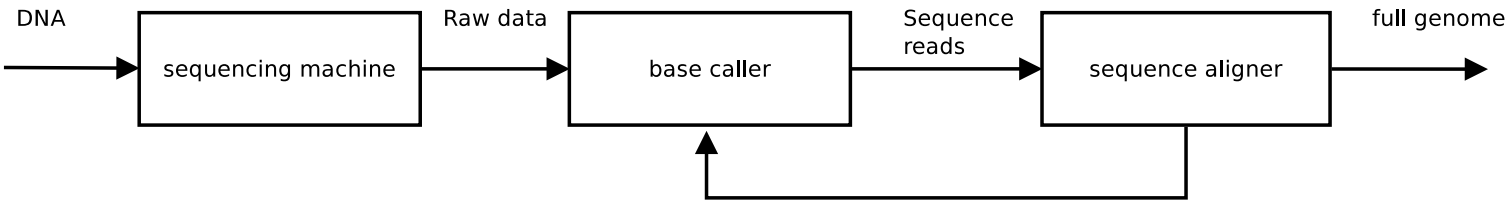
Outline

1. Introduction
   - Genome Sequencing and Assembly
   - Issues and Challenges
2. SUTTA assembler
   - Algorithm details (Scoring, Pruning & Lookahead)
3. Feature-Response Curve
   - Motivation
   - Results
4. **Base-Calling and Assembly**
   - Re-sequencing (TotalReCaller)
   - Integrating Base-Calling and Assembly

Re-sequencing
Motivation



Conventional *re-sequencing* pipeline

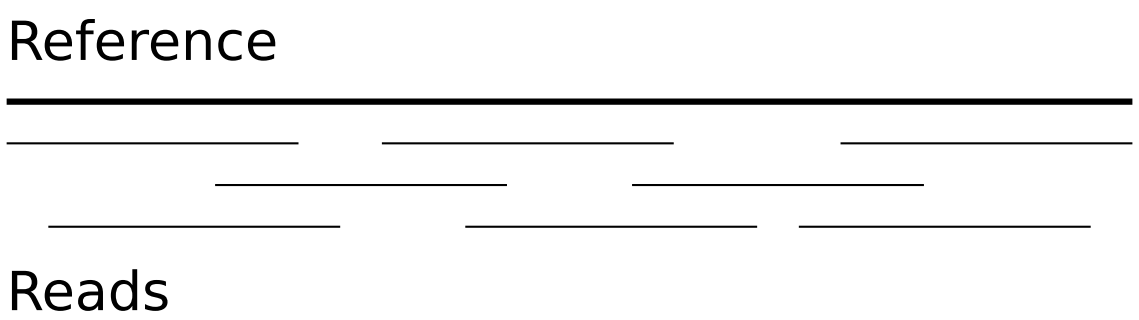


Proposed *re-sequencing* pipeline

Idea: Performing both base-calling and alignment in a single unified step.

TotalReCaller

(joint work with Fabian Menges)

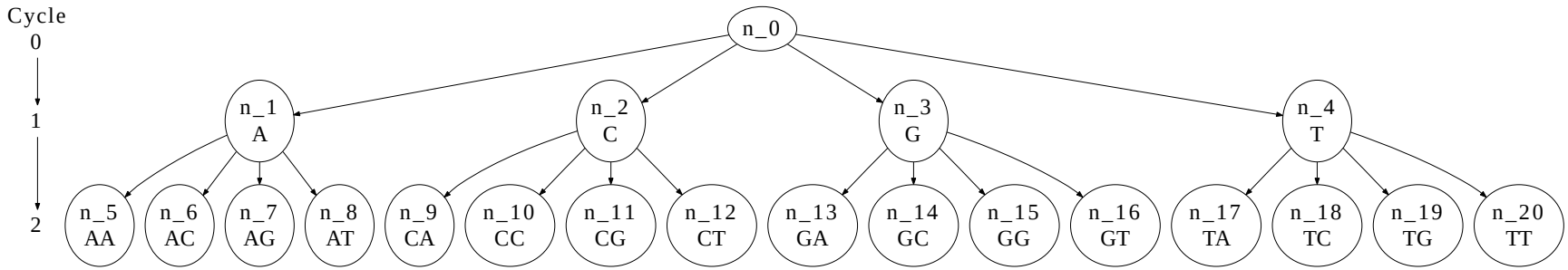


A **Beam Search** algorithm combining information from:

- raw sequencing data (intensities) and
- alignment to a reference genome (Based on Burrows-Wheeler transform and Ferragina-Manzini search).

$\Longrightarrow$ Combined base-calling and base-by-base alignment.

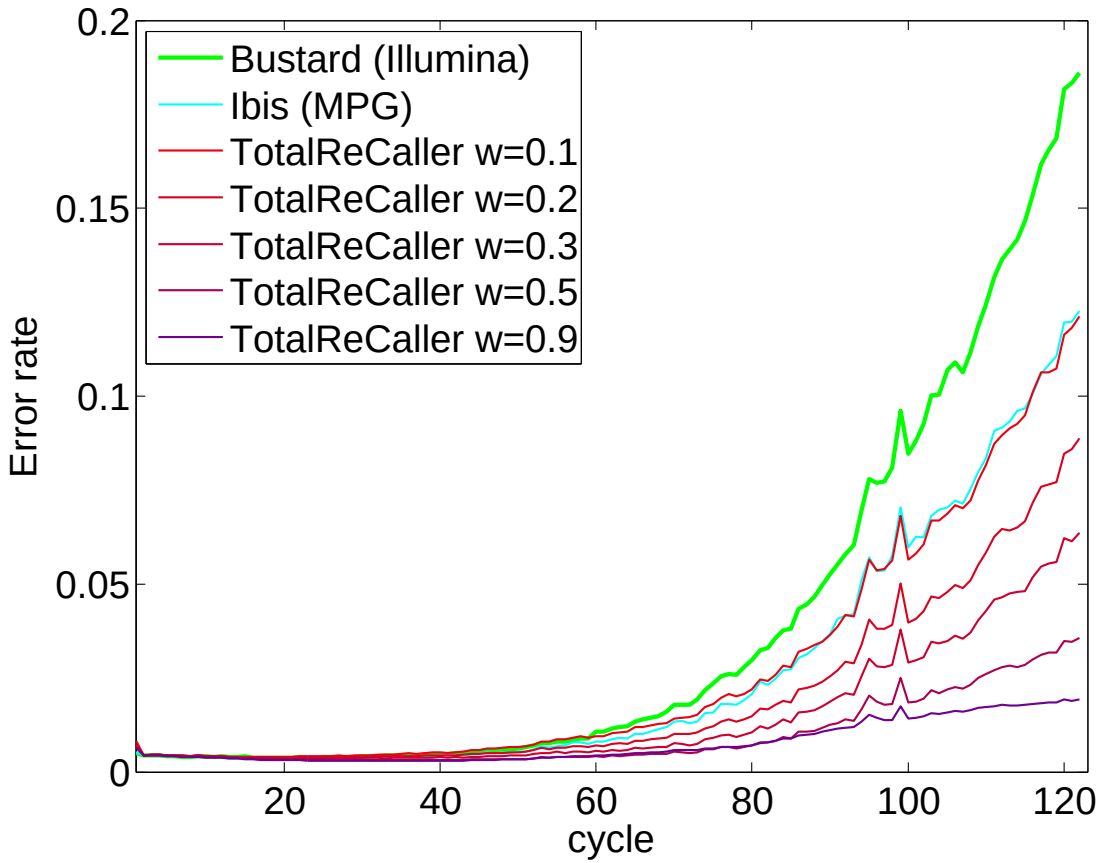
Base-calling by building a tree



- **Branch:** For each sequence in the solution space N_{k-1} all four possible successor sequences are generated;
- **Bound:** Each sequence in N_k is evaluated according to the score function g (combining intensity and alignment information);
- **Pruning:** All but the best (highest score) $l \in \mathbb{N}$ sequences are pruned, thus reducing the size of N_k to $|N_k| = l$.

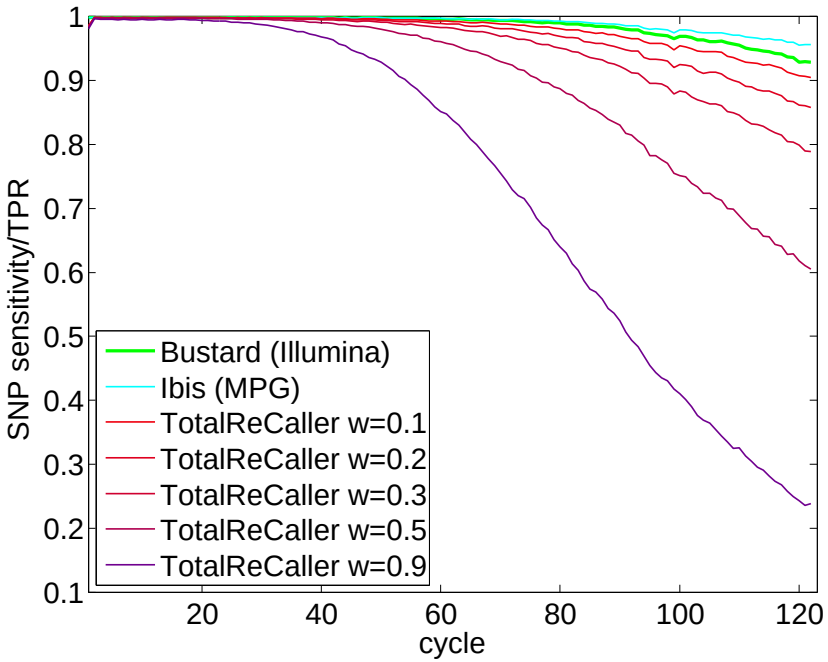
Results: E.Coli

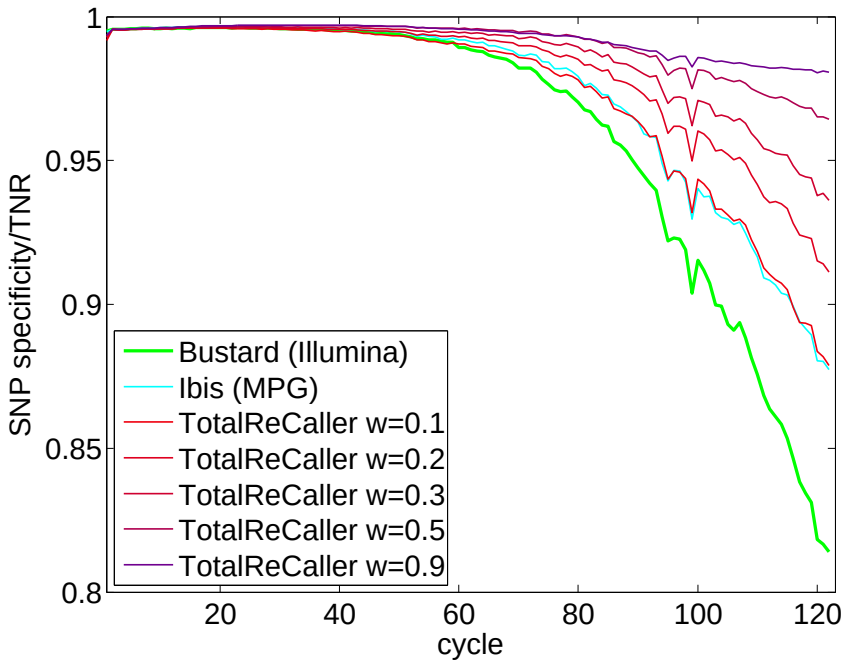
Error rate



Results: E.Coli

SNPs specificity and sensitivity





Tradeoff between *specificity* and *sensitivity*.

Outline

1. Introduction
   - Genome Sequencing and Assembly
   - Issues and Challenges
2. SUTTA assembler
   - Algorithm details (Scoring, Pruning & Lookahead)
3. Feature-Response Curve
   - Motivation
   - Results
4. **Base-Calling and Assembly**
   - Re-sequencing (TotalReCaller)
   - Integrating Base-Calling and Assembly

Synergy: TotalReCaller + SUTTA

Integrating Base-Calling, Error Correction and Assembly

De Novo assembly pipeline to take advantage of both SUTTA and TotalReCaller capabilities:

1. DRAFT ASSEMBLY: generate with SUTTA a draft assembly using the available reads.

2. BASE-CALLING & ERROR CORRECTION: given the reads intensity files and the draft assembly (generated in step 1), run TotalReCaller to generate a new set of reads with higher accuracy.

3. SEQUENCE ASSEMBLY: Run SUTTA on the new set of reads generated in step 2 to create an improved assembly.

TotalReCaller + SUTTA pipeline results

E. coli 125bp reads (Illumina Genome Analyzer II)

Assembler	#correct	#errors (μ kbp)	#ctgs$\geq$10K (kbp)	N50 (kbp)	Max (kbp)	Mean (kbp)	Cov. all (%)	Cov. correct (%)
SUTTA	339	49 (13.8)	147 (37.9%)	24.1	105.6	11.6	97.4	**82.7**
SUTTA (draft)	168	21 (20.9)	100 (52.9%)	54.6	221.5	24.1	98.2	**88.6**
SUTTA (ref.)	154	25 (31.4)	86 (48.0%)	71.7	141.6	25.4	98.2	**81.3**
SOAPdenovo (ctg)	245	80 (18.6)	52 (42.3%)	35.7	100.1	14.1	98.4	66.3
SOAPdenovo (scaf)	106	17 (99.6)	53 (45.3%)	117.6	312.5	37.1	99.3	61.9
ABySS	92	13 (80.9)	54 (49.5%)	134.4	312.5	40.7	102.9	79.7
Velvet	126	60 (32.1)	100 (53.8%)	54.8	148.8	24.5	98.5	56.9

Table: Assembly results (contigs) for *E. coli* (4.6 Mbp) dataset (100X 125bp reads from one lane of Genome Analyzer II).

Correct contig = align to the reference genome along the whole length with at least 95% base similarity

SUTTA achieves higher coverage using correctly assembled contigs.

TotalReCaller + SUTTA pipeline results

E. coli 125bp reads (Illumina Genome Analyzer II)

Assembler	#correct	#errors (μ kbp)	#ctgs$\geq$10K (kbp)	N50 (kbp)	Max (kbp)	Mean (kbp)	Cov. all (%)	Cov. correct (%)
SUTTA	339	49 (13.8)	147 (37.9%)	24.1	105.6	11.6	97.4	**82.7**
SUTTA (draft)	168	21 (20.9)	100 (52.9%)	54.6	221.5	24.1	98.2	**88.6**
SUTTA (ref.)	154	25 (31.4)	86 (48.0%)	71.7	141.6	25.4	98.2	**81.3**
SOAPdenovo (ctg)	245	80 (18.6)	52 (42.3%)	35.7	100.1	14.1	98.4	66.3
SOAPdenovo (scaf)	106	17 (99.6)	53 (45.3%)	117.6	312.5	37.1	99.3	61.9
ABySS	92	13 (80.9)	54 (49.5%)	134.4	312.5	40.7	102.9	79.7
Velvet	126	60 (32.1)	100 (53.8%)	54.8	148.8	24.5	98.5	56.9

Table: Assembly results (contigs) for *E. coli* (4.6 Mbp) dataset (100X 125bp reads from one lane of Genome Analyzer II).

Correct contig = align to the reference genome along the whole length with at least 95% base similarity

SUTTA achieves higher coverage using correctly assembled contigs.

Conclusion
(Contributions)

1. A new sequence assembler (**SUTTA**) based on the **branch-and-bound** method that allows to perform *assembly and validation concurrently*.
2. New metric that more faithfully captures the **trade-off** between assembly quality and contiguity (**Feature-Response curve**).
3. A new Base-Caller (**TotalRecaller**) that has the ability to concurrently perform base-calling, alignment, error correction and SNP detection.

Feature works

SUTTA:

- Scaling to large assembly projects $\Rightarrow$ distributed computing.
- Optical Maps integration $\Rightarrow$ *dovetailing* between short and long range data.
- Haplotypic (Human) genome assembly?

FRCurve:

- Increase specificity $\Rightarrow$ reduced number of false-positive features.
- New specialized features $\Rightarrow$ other technologies (not just mate-pairs).

Patents & Publications

Publications:

- NARZISI G. and MISHRA B.: *Scoring-and-Unfolding Trimmed Tree Assembler: Concepts, Constructs and Comparisons.* **Bioinformatics**, Oxford Journals, 2010.
- NARZISI G. and MISHRA B.: *Comparing De Novo Genome Assembly: The Long and Short of It.* **PLoS ONE**, April 2011.
- MENGES F., NARZISI G. and MISHRA B. *TotalReCaller: Improved Accuracy and Performance via Integrated Alignment & Base-Calling*, **Bioinformatics**, Oxford Journals, 2011.

Patents:

- *Methods, Computer-Accessible Medium, and Systems for Score-Driven Whole-Genome Shotgun Sequence Assembly.* Filed: February, 2009.
- *Methods, Computer-Accessible Medium, and Systems for Base-Calling and Alignment.* Filed: April, 2009.

Acknowledgment

- Dr. Giuseppe Narzisi (CSHL) – SUTTA/FRC
- Engg. Fabian Menges (NYU) – TotalReCaller/SUTTA
- Dr. Andreas Witzel (NYU) – SUTTA
- Francesco Vezzi (Applied Genomics Institute) – PCA & ICA feature analysis

- Dr. Christian Haudenschild (Illumina, Inc) – Sequencing data
- Prof. Alberto Policriti (University of Udine) – Sequencing data

THE END !

Appendix

Supplementary slides

Appendix

Sequence Assembly Problem

Definition (Sequence Assembly Problem (*SAP*))

*Given a collection of fragment reads $F = \{r_i\}_{i=1}^{N}$ and a tolerance level (error rate) ϵ, find a reconstruction R whose layout $L = \langle r_{j_1} \overset{\pi_1}{\rightleftharpoons} r_{j_2} \overset{\pi_2}{\rightleftharpoons} \cdots \overset{\pi_{N-1}}{\rightleftharpoons} r_{j_N} \rangle$ is ϵ-**valid**, **consistent** and such that the following set of properties (oracles) are satisfied :*

1. **(Overlap-Constraint (O))** *The cumulative overlap score O of the layout L is optimized:*

$$O(L) = \sum_{\substack{(r_i, r_j) \in L \\ \pi(r_i, r_j)}} S_O(r_i, r_j)$$

2. **(Mate-Pair-Constraint (MP))** *The cumulative mate-pair score S_{MP} of the distance between reads in the layout L is consistent with the mate-pair constraints:*

$$MP(L) = \sum_{\substack{(r_i, r_j) \in L \\ (r_i \leftrightarrow r_j)}} S_{MP}(r_i, r_j)$$

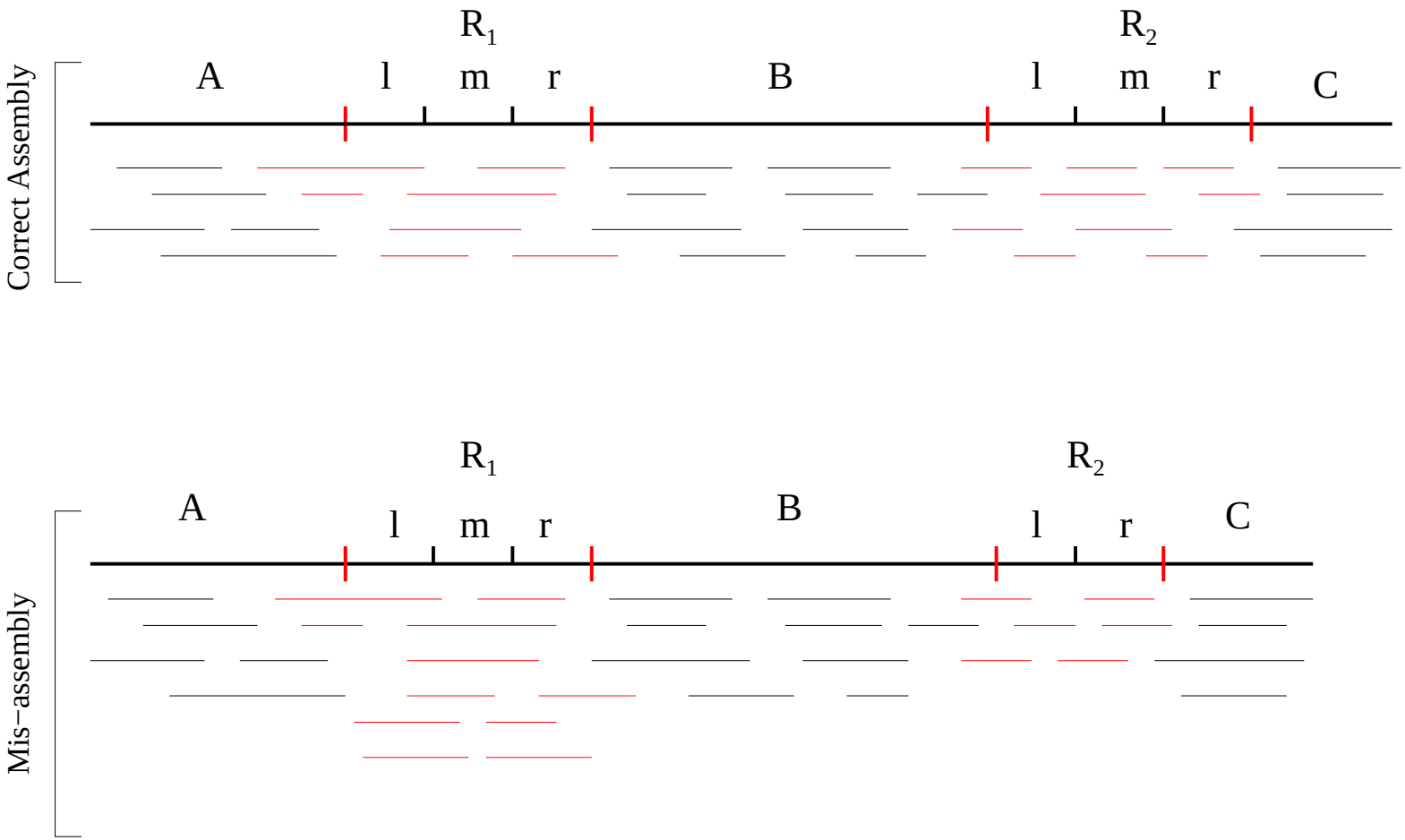
3. **(Optical-Map-Constraint (OM))** *The observed distribution of restriction enzyme sites in the layout L, $C_{obs} = \langle a_1, a_2, \ldots, a_n \rangle$, is consistent with the distribution of experimental optical map data $C_{src} = \langle b_1, b_2, \ldots, b_n \rangle$ (obtained by a restriction enzyme digestion process).*

We propose an algorithmic approach that can combine and use **concurrently** all the oracles while searching the optimal layout.

Appendix

Repeats

- If we look for a reconstruction of minimum length, the reconstructed string can have many errors due to repeats.



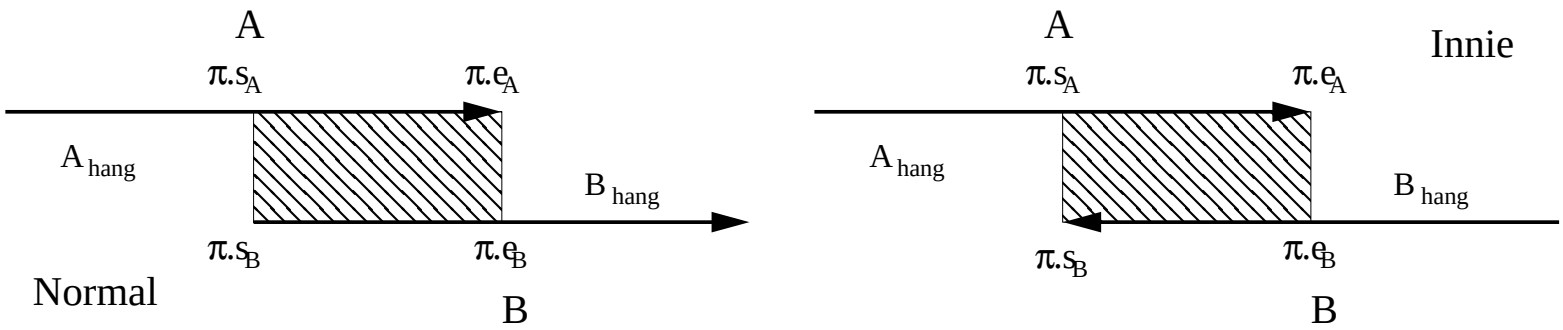
Fragments and Overlaps

Fragments:

- A set of fragments/reads $F = \{f_1, f_2, \ldots, f_N\}$, s.t. $f_i \in \{A, C, G, T\}^*$.
- Each fragment is represented as pairs of integers $f_i = (s_i, e_i), i \in [1, |F|]$ where $1 \leq s_i, e_i \leq |R|$, and R is the reconstructed string (the order of s_i and e_i encodes the orientation of the fragment).

Overlaps:

- Use Smith-Waterman algorithm to compute the best alignment between a pair of strings.



- predicate $suffix_\pi(f)$ on a fragment f s.t.:

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

Layout Representation

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

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

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

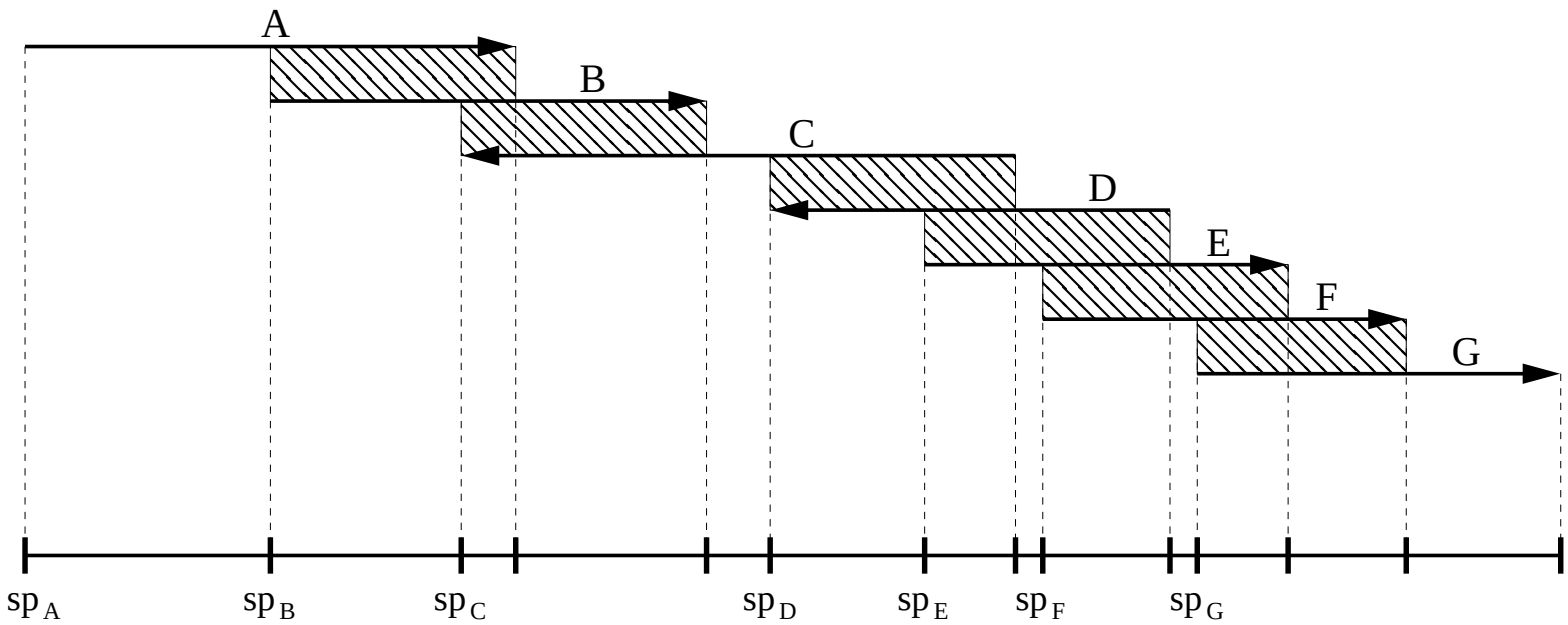
Definition (Consistency Property)

A layout L is **consistent** *if the following property holds for* $i = 2, \ldots, N-1$*:*

$$\overset{\pi_{i-1}}{\rightleftharpoons} f_i \overset{\pi_i}{\rightleftharpoons} \quad \underline{\text{iff}} \quad suffix_{\pi_{i-1}}(f_i) \neq suffix_{\pi_i}(f_i)$$

Layout
(Illustration)

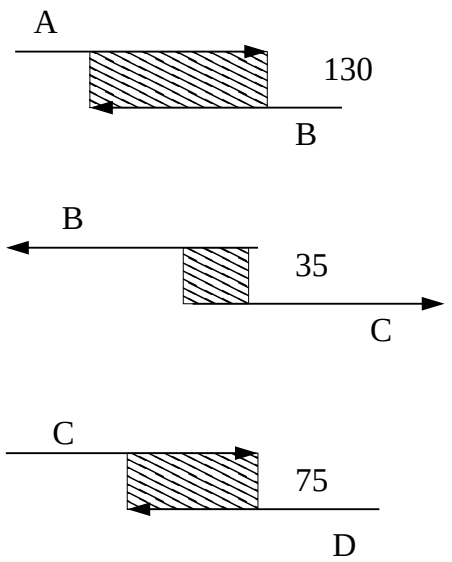
Layout for a set of fragments $F = \{A, B, C, D, E, F, G\}$ with sequence of overlaps $\pi^N_{(A,B)}, \pi^I_{(B,C)}, \pi^N_{(C,D)}, \pi^I_{(D,E)}, \pi^N_{(E,F)}, \pi^N_{(F,G)}$

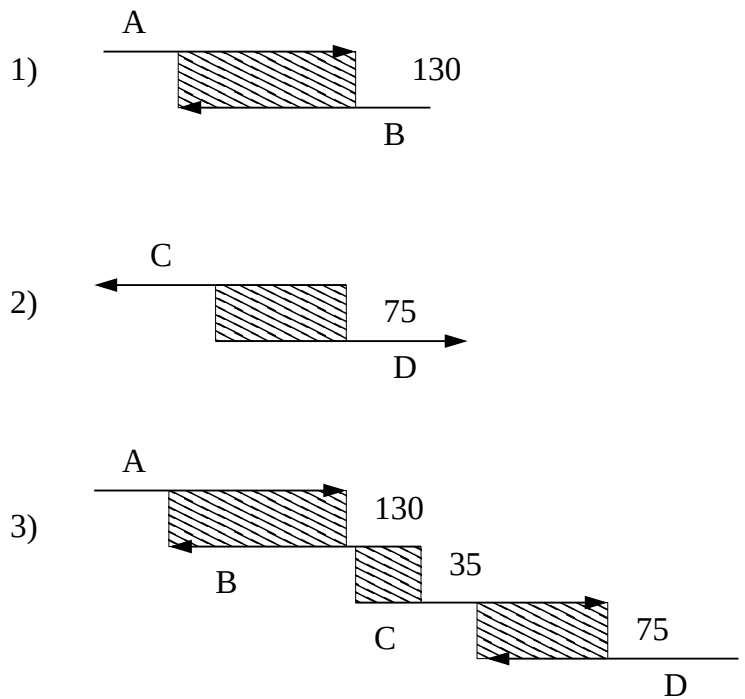


Appendix

Greedy Strategy
(TIGR 1995, Phrap 1996, CAP3 1999)

1. Pick the highest scoring overlap.
2. Merge the two fragments (add this new sequence to the pool of sequences).
3. Heuristically correct regions of the overlay in some plausible manner (whenever possible).
4. Regions that do not yield to these error-correction heuristics are abandoned as irrecoverable and shown as gaps.
5. Repeat until no more merges can be done.

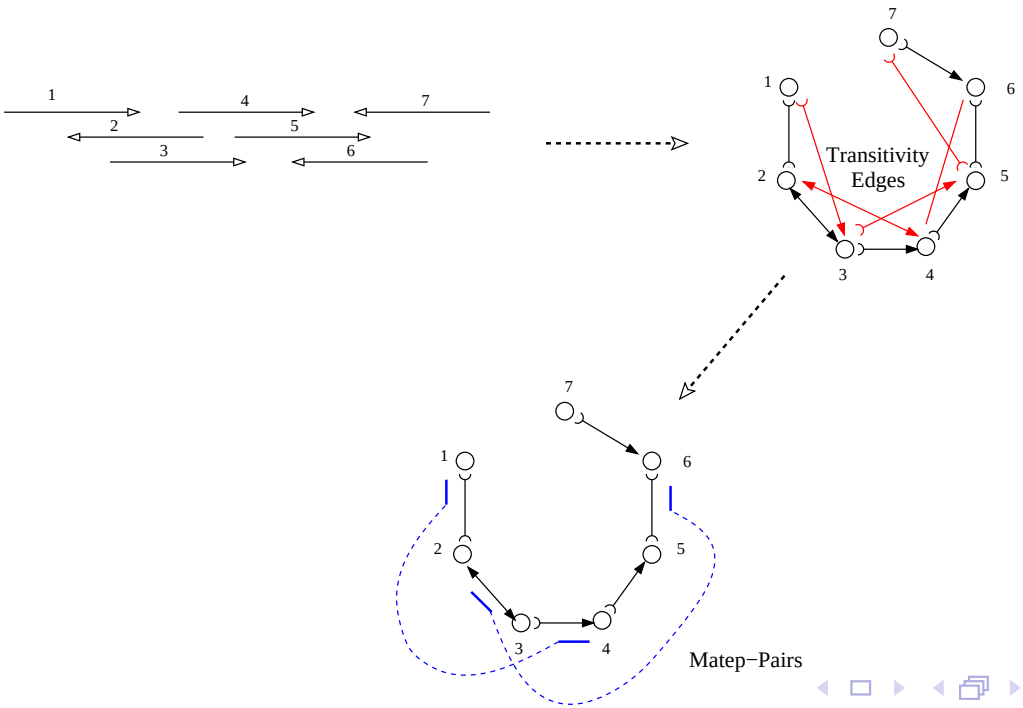




Appendix

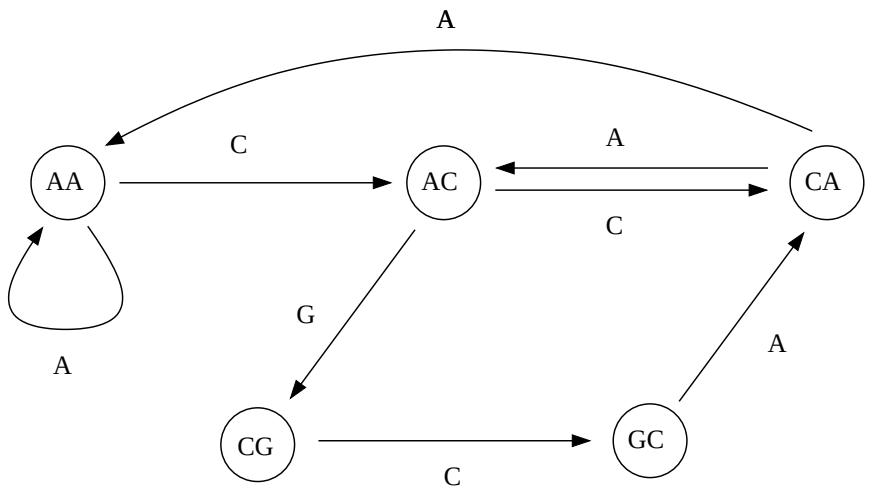
Overlap-Layout-Consensus
(CELERA 2000, Minimus 2007)

- **Idea:** Construct a graph in which nodes represent reads and edges indicate overlaps.
- **Goal:** Need to solve an Hamiltonian path !
- **Strategy**:
  1. Remove contained and transitivity edges.
  2. Collapse "unique connector" overlaps (chordal subgraph with no conflicting edges).
  3. Use mate-pairs to connect and order the contigs.
- Contigs correspond to nonintersecting simple paths in the reduced graph.



Sequencing by Hybridization
(EULER 2001, Velvet 2008)

- **Idea**: Break 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 the source and destination nodes are respectively the $n-1$ prefix and $n-1$ suffix of the corresponding n-mer.
- **Idela Goal**: find a path that uses all the edges (an Eulerian path) $\rightarrow$ linear time algorithm
- **Real Goal**: Eulerian-superpath: given an Eulerian graph and a sequence of paths, find an Eulerian path in the Eulerian graph that contains all these paths as sub-paths ($\mathcal{NP}$-hard).
- **Note**: If no n-mer appears more than once in the genome then there exist at least one Eulerian path.
- **Problem**: Errors in the data can introduce many erroneous edges !



DeBruijn graph for the list $L = \{AAA, AAC, ACA, CAC, CAA, CGC, GCG\}$. The Euler path is:

SUTTA Pseudocode

Algorithm 1: SUTTA - pseudo code

Input: Set of N reads
Output: Set of contigs

```
1  B := ∅;                                   /* Forest of D-trees */
2  C := ∅;                                   /* Set of contigs */
3  F := ⋃_i^N {r_i};    /* All the available reads/fragments */
4  while (F ≠ ∅) do
5      r := F.getNextRead();
6      if ( ¬isUsed(r) ∧ ¬isContained(r) ) then
7          DT := create_double_tree(r);
8          B := B ∪ {DT};
9          Contig CTG := create_contig(DT);
10         C := C ∪ {CTG};
11         CTG.layout();                  /* Compute contig layout */
           F := F \ {CTG.reads};          /* Remove used reads */
12     end
13 end
14 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.

Node expansion
(Branch-and-Bound)

Algorithm 2: Node expansion

Input: Start read r_0, max queue size K, percentage T of top ranking solutions, dead-end depth W_{de}, bubble depth W_{bb}, mate-pair depth W_{mp}

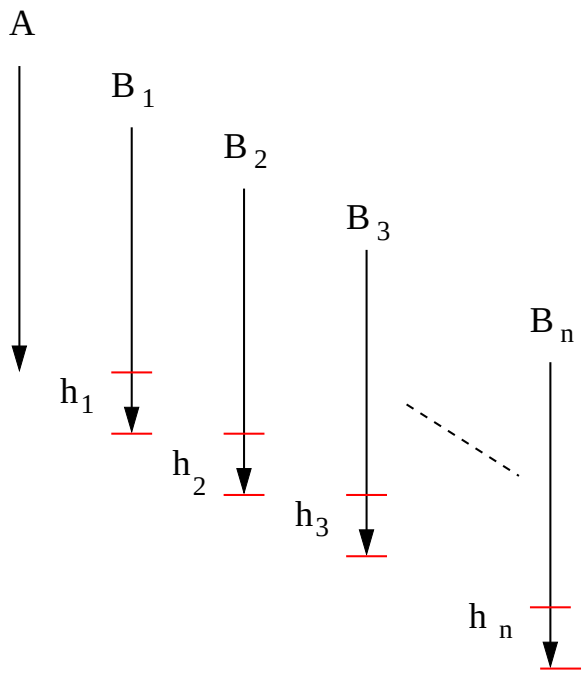
Output: Best scoring leaf

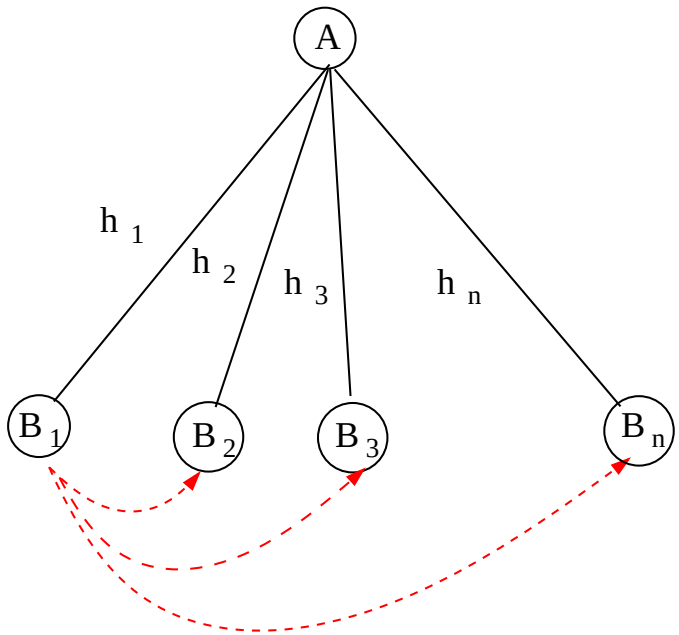
```
1  V := ∅;                                          /* Set of leaves */
2  L := {(r_0, g(r_0))};              /* Live nodes (priority queue) */
3  while (L ≠ ∅) do
4  |   L := Prune(L, K, T);                        /* Prune the queue */
5  |   r_i := L.popNext();              /* Get the best scoring node */
6  |   E := Extensions(r_i);                      /* Possible extensions */
7  |   E^(1) := Transitivity(E, r_i);            /* Transitivity pruning */
8  |   E^(2) := DeadEnds(E^(1), r_0, W_de);      /* Dead-end pruning */
9  |   E^(3) := Bubbles(E^(2), r_0, W_bb);       /* Bubble pruning */
10 |   E^(4) := MatePairs(E^(3), r_0, W_mp);     /* Mate pruning */
11 |   if (|E^(4)| == 0) then
12 |   |   V := V ∪ {r_i};                        /* r_i is a leaf */
13 |   else
14 |   |   for (j=1 to |E^(4)|) do
15 |   |   |   L := L ∪ {(r_j, g(r_j))};
16 |   |   end
17 |   end
18 end
19 return max_{r_i ∈ V} {g(r_i)} ;
```

Appendix

Transitivity pruning

- **Observation**: do not waste time expanding nodes that (due to *transitivity*) will be explored at the next level in the tree.
- **Idea**: delay expansion of the "last" node/read involved in a transitivity relation.





Strategy for selecting next sub-problem

Best First Search (BeFS): always select among the live subproblems the one with best score.

Depth First Search (DFS): always select among the live subproblems the one deepest in the tree.

- **Combined strategy**: Use DFS as overall search strategy and BeFS when choice is to be made between nodes at the same level.
- **Implementation**: priority queue with precedence relation between two nodes x and y:

$$x \prec y \text{ iff } \begin{cases} depth(x) > depth(y) \\ or \\ depth(x) = depth(y) \wedge score(x) > score(y) \end{cases}$$

- Because BeFS is applied locally at each level the score is optimized concurrently.

Zig-Zag function

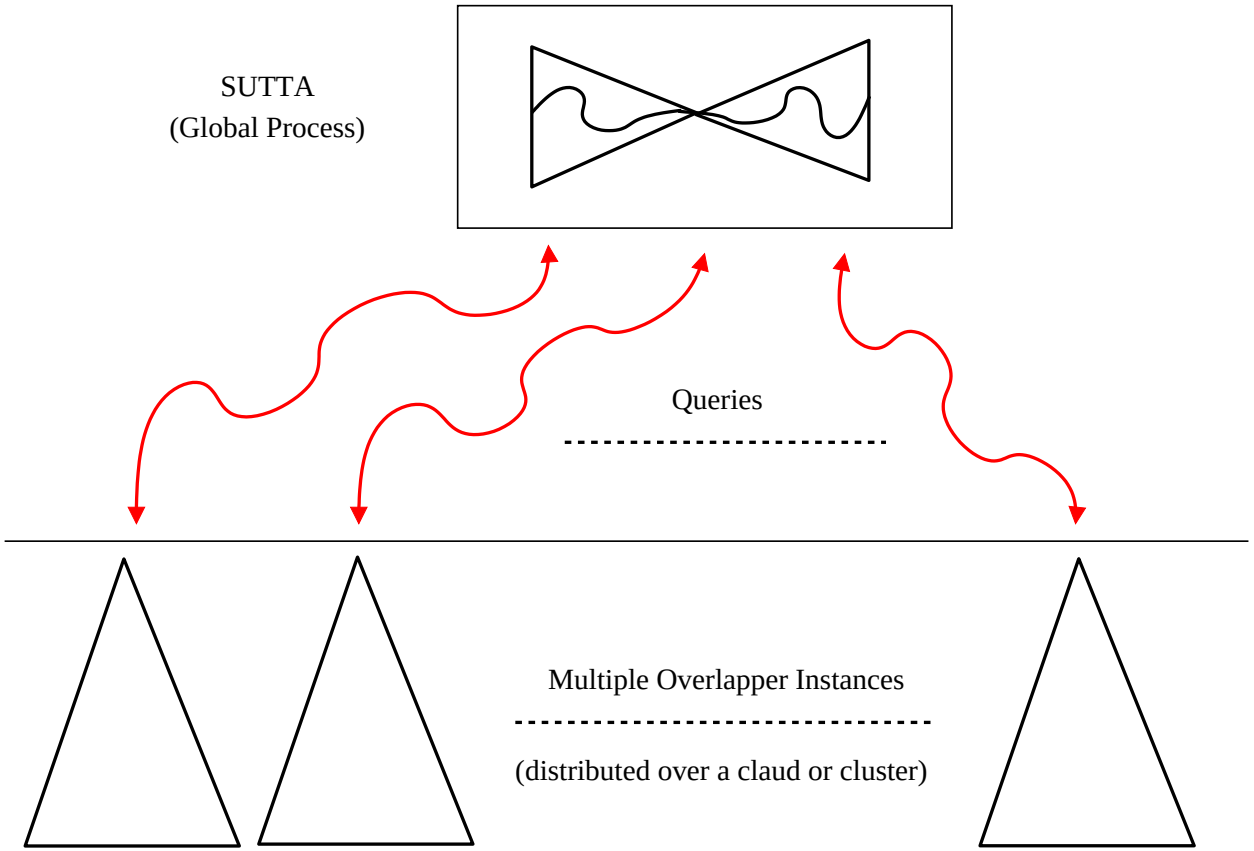
Problem: *given the set of overlaps O for a set of reads F, find the overlap (or set of overlaps) for a pair of reads (r_1, r_2) (if one exists).*

- **Naive strategy:** takes time $O(n^2)$ where $n = |O|$.
- **Graph approach:** takes time $O(l)$ where l is the size of the longest adjacency list in the graph.
- **Fast approach:** hashing!

$$H(x,y) = \frac{(x+y)(x+y-1)}{2} + (1-y) \tag{1}$$

$|H(x,y)| \leq c$, where c is function of the read size, genome structure and overlap strategy (Smith-Waterman, exact match,etc.).

Distributed Computing Approach



Distribute overlapper over a cloud computing environments (e.g., Amazon Elastic Compute Cloud). SUTTA as global single process requesting the overlap information to the multiple distributed overlapper instances.

Linear filter for intensities

Crosstalk, fading and lagging

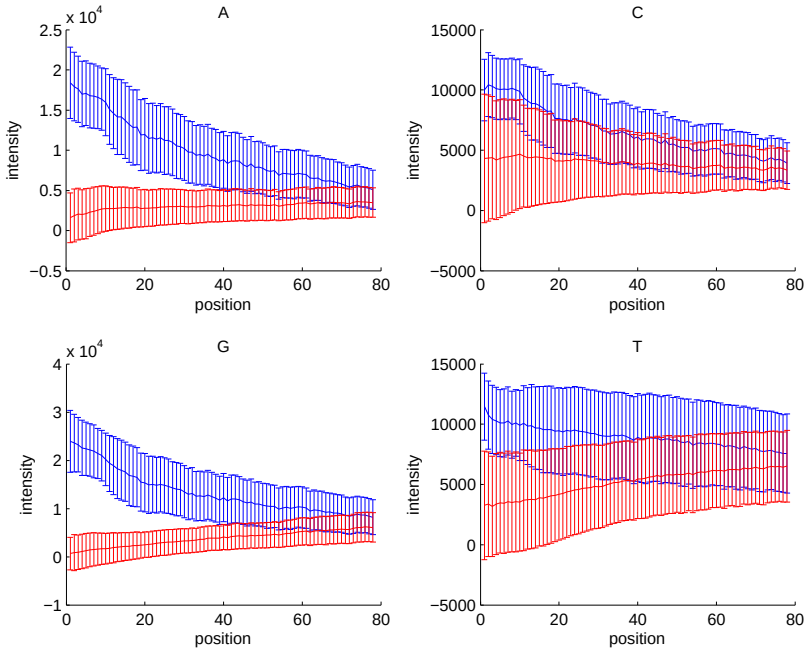
- Cycle: $k \in \mathbb{N}$:
- **Input:** Raw intensities: $\boldsymbol{I}_k = \begin{pmatrix} I_A^k & I_C^k & I_G^k & I_T^k \end{pmatrix}^\top$
- Crosstalk matrix: $\boldsymbol{A}_k \in \mathbb{R}^{4\times 4}$
- Lagging matrix: $\boldsymbol{\Upsilon}_k \in \mathbb{R}^{4\times 4}$
- **Output:** Filtered intensities:

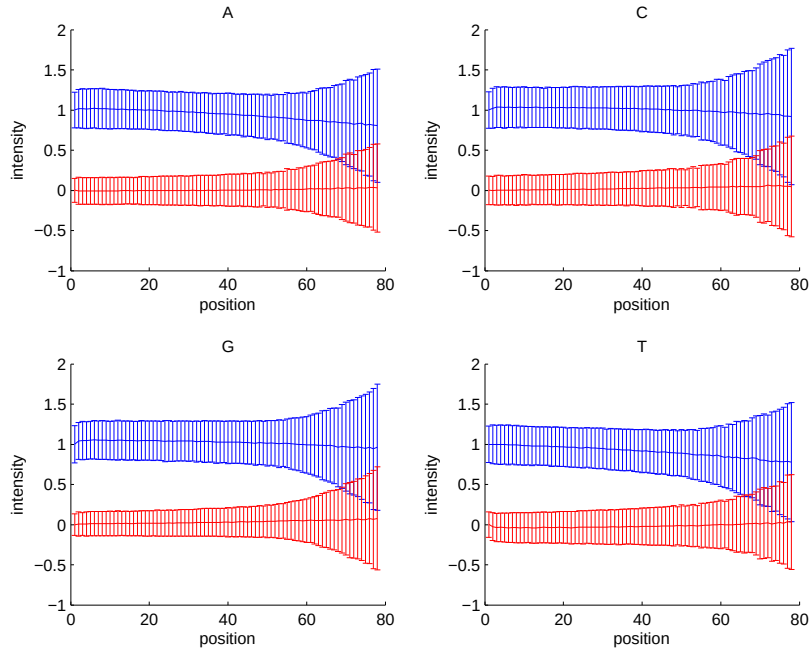
$$\begin{pmatrix} \boldsymbol{X}_{k-1} \\ \boldsymbol{X}_k \end{pmatrix} = \underbrace{\begin{pmatrix} \boldsymbol{A}_{k-1} & 0 \\ \boldsymbol{\Upsilon}_k & \boldsymbol{A}_k \end{pmatrix}^{-1}}_{\boldsymbol{G}_k \in \mathbb{R}^{8\times 8}} \cdot \begin{pmatrix} \boldsymbol{I}_{k-1} \\ \boldsymbol{I}_k \end{pmatrix}$$

Appendix

Raw and Filtered Intensities

Result



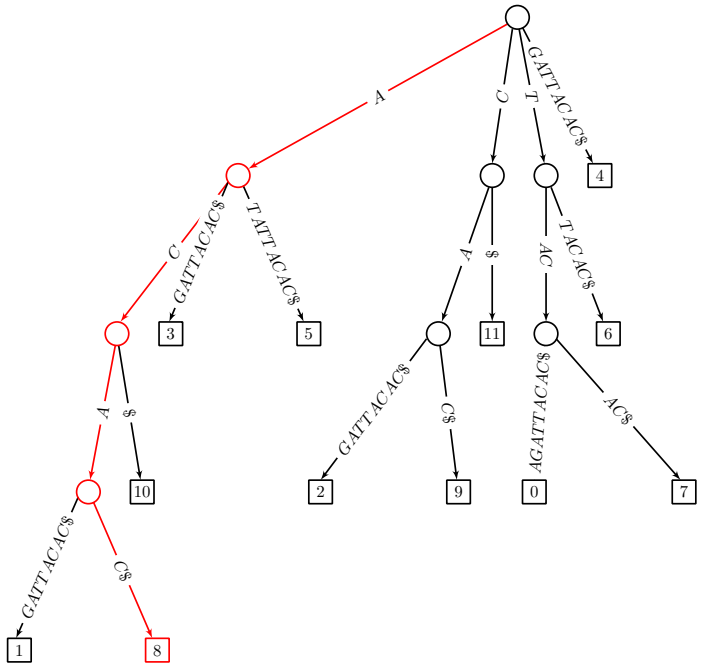


Appendix

Base-by-base alignment

Based on Burrows-Wheeler transform and Ferragina-Manzini search (like Bowtie, BWA, SOAP2, etc.). Closely related to Suffix trees:

Example: Reference $T =$ "TACAGATTACAC$"



Score function

$$
\begin{aligned}
P_k(B \mid \boldsymbol{X}_k) &= \frac{P_k(\boldsymbol{X}_k \mid B)P_k(B)}{P_k(\boldsymbol{X}_k)} \qquad \text{with } B \in \{A, C, G, T\} \\
&= \frac{P_k(\boldsymbol{X}_k \mid B)P_k(B)}{P_k(\boldsymbol{X}_k \mid B)P_k(B) + P_k(\boldsymbol{X}_k \mid \neg B)P_k(\neg B)} \\
&= \frac{1}{1 + \frac{P_k(\boldsymbol{X}_k \mid \neg B)P_k(\neg B)}{P_k(\boldsymbol{X}_k \mid B)P_k(B)}} \\
&= \frac{1}{1 + \underbrace{\frac{P_k(\boldsymbol{X}_k \mid \neg B)}{P_k(\boldsymbol{X}_k \mid B)}}_{\text{Intensities}} \cdot \underbrace{\frac{P_k(\neg B)}{P_k(B)}}_{\text{Sequence alignment}}}
\end{aligned}
$$