Genomorientierte Bioinformatik

-

BamFeatures

Malte Weyrich

December 2024

Das Sequenzieren in der Bioinformatik generiert Milliarden von *Reads* pro *Sample*, welche mit einem *Mapper* an das *Referenzgenom* aligniert werden. Diese Daten werden in einer *Sequence Alignment Map (SAM)* gespeichert und können zusätzlich in ein komprimiertes Format, einer *Binary Alignment Map (BAM)* Datei, umgewandelt werden. Anschließend können verschiedene Analysen auf den *BAM* Dateien durchgeführt werden. Die in diesem Report diskutierte *JAR* liest eine gegebene *paired-end RNA-seq BAM* Datei ein und errechnet verschiedene Features, die in einer *<tsv>* Datei gespeichert werden. Die JAR wurde auf drei verschiedenen *BAM* Dateien ausgeführt, um damit anschießend die *RPKM* Werte zu berechnen. Zudem wird die *JAR* anhand ihrer Laufzeit und Korrektheit analysiert.

---

1 — RNA-seq

Die Bezeichnung *RNA-seq* bezieht sich auf ein bestimmtes Sequenzierprotokoll der Bioinformatik bei dem möglichst alle exprimierten Transkripte mehrerer Samples mittels Hochdurchsatzsequenziergeräten wie *Illumina* verarbeitet und die resultierenden *Reads* in Ausgabe Dateien abgespeichert werden. Bei *Illumina* werden die extrahierten Transkripte mittels Ultraschall oder enzymatischer Fermentation in Fragmente mit ähnlicher Länge zerstückelt und mit Adaptersequenzen und Barcodes versehen. Darauf folgt eine *PCR* Amplifikation der Fragmente, um das Signal zu verstärken. Die amplifizierten Fragmente können nun im Sequenzierzyklus von *Illumina*, Base für Base, gelesen werden. Dabei wird jedoch nicht das ganze Fragment gelesen, sondern jeweils nur ca. $100BP$ (abhängig von Voreinstellung und Protokoll) beider Enden des Fragments (*paired-end sequencing*). Somit entstehen pro Fragment zwei *Reads*, ein forward *Read* und ein reverse *Read*, welche zu einem *ReadPair* zusammengefasst werden können. In dem Protokoll von *Illumina* werden zuerst alle *Reads* eines Endes aller Fragmente gemacht, dann wird das Fragment, **vereinfacht gesagt**, auf der *Flow Zelle* umgedreht (durch Replikation), wodurch die umgedrehten Fragmente jeweils das Komplement ihres ursprünglichen Fragments sind. Somit sollten die *Reads* eines *ReadPairs* immer auf entgegengesetzte Stränge *("+"/"-") "mappen"*. Ist bei einem Sequenzierexperiment die Ausgangskonfiguration der Fragmente bekannt, so handelt es sich um ein *strand specific* Experiment und man kann allen *Reads* einem festen Strang zuordnen, je nach dem, ob das Fragment in der Anfangskonfiguration vom *"-"* oder vom *"+"* Strang kam. Ein *Mapper* würde nun solche *ReadPairs* an einem *Referenzgenom mappen* und eine (oder mehrere) *SAM/BAM* Datei(en) erstellen, welche unter anderem die Koordinaten der alignierten *Reads* basierend auf dem *Referenzgenom* beinhalten. Die Alignment Daten dieser Datei können nun von der *BamFeatures JAR* weiter annotiert werden.

2 — Java Programm

```
Usage:
    java -jar bam.jar -bam <bamPath> -o <outputPath> -gtf <gtfPath> \\
                      [-frstrand <true/false>] [-lengths]
```

2.1. Argumente

Zusätzlich zu den vorausgesetzten Argumenten (*-bam*, *-o*, *-gtf*) können noch *-frstrand* und *-lengths* angegeben werden. Bei einem Strangpositiven Experiment (*-frstrand true*) ist der forward *Read* auf dem *"+"* und der reverse *Read* auf dem *"-"*. So kann die *JAR* die *Reads* korrekt zuordnen. Falls die Strangrichtung nicht angegeben ist, werden für beide *Reads* jeweils beide Stränge betrachtet. Die *-lengths* Option wird für die Berechnung der *RPKM* Werte benötigt. Ist diese Option gesetzt, so werden die für die Längennormalisierung benötigte Genlängen der kombinierten Exons jedes Gens berechnet und in einer *<tsv>* Datei gespeichert.

2.2. Logik

2.2.1 Erstellen der ReadPair Objekte

Die Einträge der *BAM* Datei werden der Reihe nach von einem *SAMFilereader* eingelesen. Da es sich um *paired-end* Daten handelt, muss für jeden *Read* sein zugehöriger *Mate* gefunden werden. Reads die nicht gepaart sind, oder nicht den Qualitätsanforderungen der Aufgabenstellung entsprechen, werden ignoriert. *Read Objekte* die zum ersten Mal vorkommen, werden in einer *HashMap<Id, Read> seenEntries* gespeichert und für jede Iteration wird überprüft, ob wir die *Read Id* bereits gesehen haben. Sobald wir zwei zusammengehörende *Reads* identifiziert haben, wird ein neues *ReadPair Objekt* erstellt. Dabei wird die Strangrichtung beim erstellen des Objekts berücksichtigt und die *Reads* jeweils nach *forward* und *reverse* kategorisiert. Zusätzlich werden die *AlignmentBlocks* beider Reads zu einem gemeinsamen *Regionvector meltedBlocks* und zwei einzelnen *Regionvectors* (*regionVecFw*, *regionVecRw*) verschmolzen. Dies ist notwendig für die anschließenden Berechnungen und fängt einige *Edge Cases* ab. Hat ein *Reads* z.B. $b_1, b_2 \in AlignmentBlocks$ und b_1.end $==$ b_2.start - 1, so werden $b_1, b_2 \rightarrow_{melt()} b_{neu}$, mit b_{neu}.start $==$ b_1.start $\wedge$ b_{neu}.end $==$ b_2.end

2.2.2 Berechnung der ReadPair Attribute

Als erstes werden die *igenes* und *cgenes* berechnet:

1. *cgenes* := $\{g \in \textit{cgenes} \mid g_{start} < fwRead_{start} \wedge g_{end} > rwRead_{end}\}$
2. *igenes* := $\{g \in \textit{igenes} \mid g_{start} >= fwRead_{start} \wedge g_{end} <= rwRead_{end}\}$

Für die Berechnung dieser Attribute verwenden wir ein verschachteltes Objekt *intervalTreeMap HashMap<String, HashMap<Boolean, IntervalTree<Gene>>>* verwendet, welches für jedes Chromosom die Gene nach ihrem Strang in Intervalbäumen abgespeichert hat (Strang ist entweder: [*true*|*false*|*null*]). Falls |*cgenes*| $== 0$ aber |*igenes*| > 0 wird das *ReadPair* verworfen und mit dem nächsten weiter gemacht, sind beide Mengen leer, so wird die kürzeste Distanz zu benachbarten Genen ausgerechnet und in *gdist* abgespeichert. Danach wird das *ReadPair* auf *split-inconsistency* überprüft (Algotithmus 1), d.h. falls es eine überlappende Region beider *Reads* gibt, müssen die potentiell implizierten Introns beider *Reads* übereinstimmen. Nach dem Aufruf von *getNsplit()* wird

Algorithm 1 getNsplit()

```
1:  Input: fw, rw                                          ▷ forward and reverse reads
2:  if |fw.Blocks| = 1 and |rw.Blocks| = 1 then            ▷ Checks if the reads imply introns
3:      return 0
4:  end if
5:  overlap = determineOverlap(fw, rw)                     ▷ Determine overlap region
6:  iFwRegions ← {}                                        ▷ Set for containing fw Introns
7:  iRwRegions ← {}                                        ▷ Set for containing rw Introns
8:  iRegions ← {}                                          ▷ Set for containing all Introns
9:  extractIntronsInOverlap(overlap.x1, overlap.x2, iFwRegions, iRegions, fw)
10: extractIntronsInOverlap(overlap.x1, overlap.x2, iRwRegions, iRegions, rw)
11: if |iRwRegions| ≠ |iFwRegions| then
12:     return -1                                          ▷ split-inconsistent
13: end if
14: if iRwRegions = iFwRegions then
15:     return |iRegions|                                  ▷ Return unique Introns in overlap
16: end if
17: return -1                                              ▷ split-inconsistent (default)
```

bei einem *return value* von -1 das *ReadPair* als *"split-inconsistent"* vermerkt und in die Ausgabedatei übernommen, ansonsten wird die Größe der Menge *iRegions* in der Variable *nsplit* gespeichert und das *ReadPair* weiter prozessiert.

Als nächstes wird das *ReadPair* anhand drei Kategorien annotiert 1:

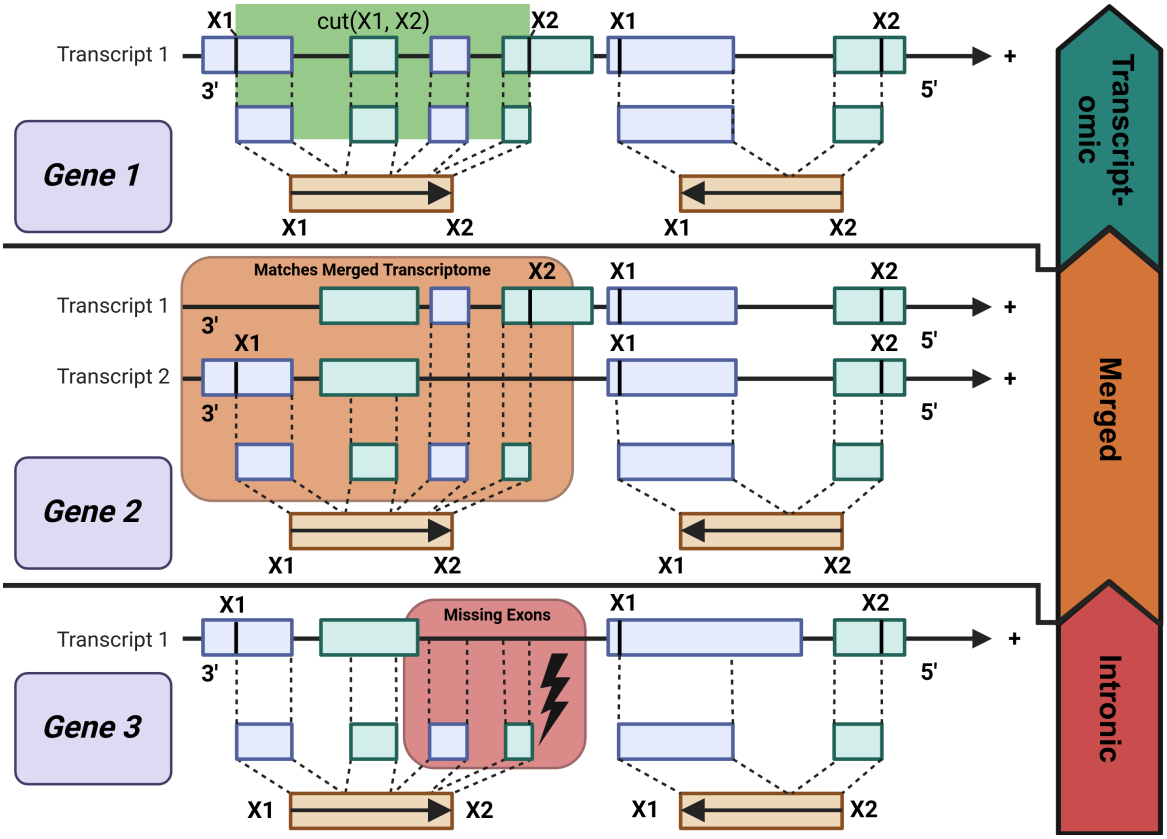

Abbildung 1 — Kategorisierung der *ReadPair*-Regionen in die Klassen *"Transcriptomic"*, *"Merged"* und *"Intronic"*, wobei eine Priorisierung nach der Reihenfolge *"Transcriptomic"* > *"Merged"* > *"Intronic"* erfolgt. Die *AlignmentBlocks* der *Reads* werden mit den Exons der Transkripte des inkludierenden Gens abgeglichen. Die Abbildung wurde mit BioRender 2024 erstellt.

Bei der Regions-Annotation wird als erstes die ***Transcriptomic*** Kategorie überprüft. Hierbei wird über alle Gene die das *ReadPair* inkludieren iteriert und für jeden *Read* des *ReadPairs* mit der Methode *cut(X1, X2)* zwei neuer *Regionvectors* aus jedem Transkript ausgeschnitten (Abbildung 1), wobei $X1$ der *Alignmentstart* und $X2$ das *Alignmentende* des momentanen *Reads* ist. Falls beide ausgeschnittenen *Regionvectors* aus dem Transkript gleich der geschmolzenen *Regionvectors* der *Reads* entsprechen, ist das *ReadPair* ***Transcriptomic*** und das entsprechende Transkript wird zusammen mit dem Gen in die Lösungsmenge genommen.

Für die ***Merged*** Kategorie reicht es, wenn die *Regionvectors* der *Reads* in dem geschmolzenen Transkriptom eines Gens enthalten sind (siehe Abbildung 1). Das geschmolzene Transkriptom wird im Algorithmus 2 berechnet und besteht aus allen annotierten Exons eines Gens. Wenn

Algorithm 2 Melt Exons into Regions

```
Input: transcriptList (list of transcripts containing exons)
allExons ← ∅
for each transcript in transcriptList do
    allExons.addAll(transcript.getExonList())        ▷ Add all exons from transcript to the list
end for
Sort allExons by start position                      ▷ Sort exons by their start positions
meltedRegions ← new TreeSet()                        ▷ Create a new set for melted regions
if allExons is not empty then
    first ← allExons.get(0)                          ▷ Get the first exon
    current ← new Region(first.getStart(), first.getStop()) ▷ Create a region for the first exon
    for i ← 1 to allExons.size() − 1 do
        exon ← allExons.get(i)
        if exon.getStart() ≤ current.getStop() + 1 then
            current.setStop(max(current.getStop(), exon.getStop()))   ▷ Extend the region if exons overlap or are adjacent
        else
            meltedRegions.add(current)               ▷ Add the completed region to the set
            current ← new Region(exon.getStart(), exon.getStop())     ▷ Start a new region
        end if
    end for
    meltedRegions.add(current)                       ▷ Add the last region
end if
return meltedRegions                                 ▷ Return the melted regions
```

ein *ReadPair* dieser Kategorie zugeteilt wird, wird das Gen in einer separaten Lösungsmenge gespeichert. Falls die oberen zwei Ansätze beide nicht zutreffen, so handelt es sich um ein ***Intronic*** *ReadPair*.

Nach der Regions-Annotation wird der *gcount* geupdated mit der Anzahl an Genen der plausibelsten Kategorie.

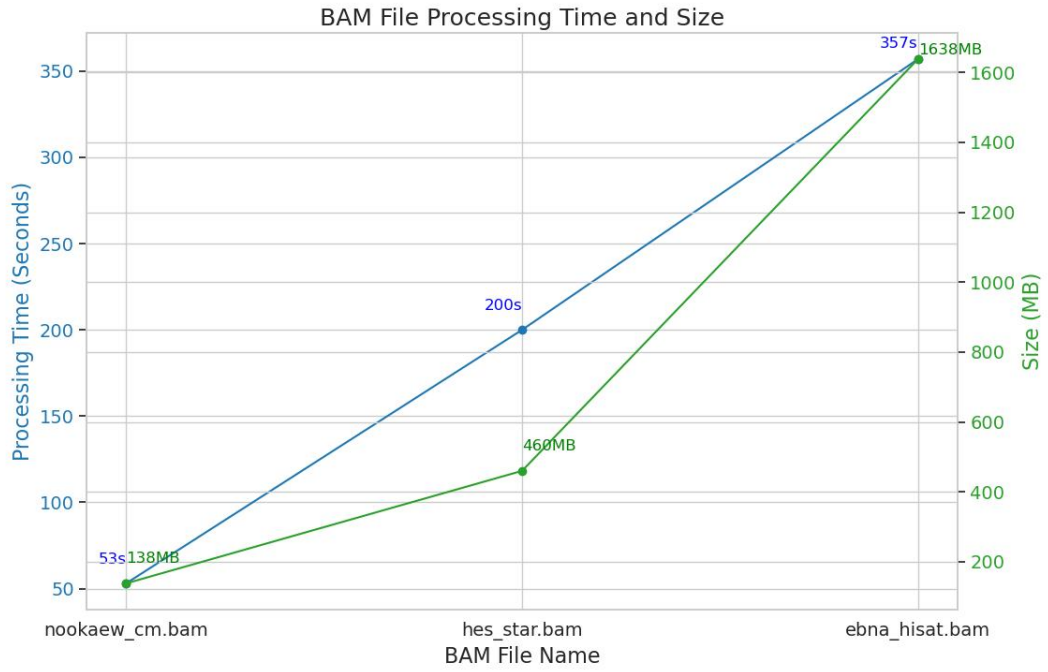

Abbildung 2 — Laufzeit der JAR auf den 3 vorgegebenen *BAMs* in Sekunden verglichen mit dem *BAM* Volumen in *MB*

2.3. Laufzeit

2.4. Korrektheit

2.5. Benchmarking

3 — Ergebnisse

References

BioRender. 2024. *BioRender - Biological Figure Creation Tool.* https://BioRender.com. Accessed: 2024-12-09. (Cited on page 5).

A — Appendix Section

hm

Text goes here