Genomorientierte Bioinformatik - Report

ExonSkipping

Malte Weyrich

October 2024

Exon Skipping Splicing Events (*ES-SE*) beschreiben, wie co- oder posttranslational manche Exons eines Transkripts durch das Spleißosom herausgeschnitten oder übersprungen werden, während in anderen Transkripten des selben Gens, diese weiterhin Teil der mRNA bleiben. Die *ES-SE* lassen sich anhand von **Gene Transfer Format** (*gtf*) files, also Genom Annotations Dateien ablesen und analysieren. Im Folgenden wird ein Programm zur Erkennung von allen *ES-SE* innerhalb eines Genoms anhand seiner Logik, Laufzeit und Ergebnisse Analysiert, wobei nur *ES-SE* berücksichtigt werden, die protein-kodierende Transkripte betreffen. Das Programm wurde auf allen verfügbaren *gtf* Dateien in $/mnt/biosoft/praktikum/genprakt/gtfs/$ ausgeführt.

---

1 — ES-SE Definition

In einem Gen kann jedes Transkript jeweils mehrere *ES-SE* haben. Ein *ES-SE* involviert immer jeweils mindestens eine **Splice Variant** (*SV*) und einen **Wild Type** (*WT*). Beide dieser Begriffe beziehen sich auf Transkripte eines Gens G. Ein *SV* ist ein Transkript T_{SV}, welches ein Intron I mit Startposition I_S und Endposition I_E besitzt, was gleichzeitig bedeutet, dass es in T_{SV} zwei Exons A, B gibt, die I flankieren. Somit endet A bei $I_S - 1 = A_E$ und B startet bei $I_E + 1 = B_S$. Zudem ist die Position $B_{pos} - A_{pos} = 1$, wobei sich A_{pos} auf die Position von Exon A relativ gesehen zu allen anderen Exons von T_{SV} bezieht. Ein *WT* wäre nun ein weiteres Transkript T_{WT} des selben Gens G, welches ebenfalls zwei Exons C, D besitzt mit $C_{pos} < D_{pos}$, wobei $C_E = I_S - 1$ und $D_S = I_E + 1$, jedoch gilt für C, D: $D_{pos} - C_{pos} \geq 1$. Dies bedeutet, dass die Exons von T_{WT} zwischen C und D in T_{SV} herausgespleißt wurden.

2 — Java Programm

2.1. Logik

Der Workflow der *JAR* lässt sich in drei Schritte Aufteilen:

I **Einlesen der *gtf* Datei und Initialisierung der Datenstruktur:**

Zum einlesen wird die *gtf* Datei zuerst nach relevanten Zeilen gefiltert, denn für uns sind momentan nur Zeilen relevant, die in der 3. Spalte entweder *"exon"* oder *"CDS"* stehen haben. Hierbei wird vermieden, die Methode *String.split("\t")* zu verwenden. Stattdessen wird in einem *for loop* jedes Zeichen einzeln betrachtet. Dabei werden Zeilen die mit einem *"#"* anfangen direkt übersprungen. Für alle anderen Zeilen werden die Anzahl der *tabs* gezählt und nach dem zweiten *tab*, werden alle darauf folgenden Zeichen zu einem *String* zusammen konkateniert, bis der dritte *tab* erreicht wurde. Falls der entstandenen String $\in$ {"exon", "CDS"}, wird die Zeile einer *ArrayList<String>* hinzugefügt, ansonsten wird mit der nächsten Zeile weiter gemacht. Diese Liste enthält am Ende alle relevanten Zeilen. Jede der relevanten Zeilen werden nun mit *String.split("\t")* in ein *String[] mainComponents* geschrieben. Die *attributeColumn* wird aus *mainComponents* extrahiert (auch als ein *String[]* Names *attributes*), indem man *mainComponents[mainComponents.length - 1]* am *";"* splitted.

Mit diesen zwei Komponenten pro Zeile wird als erstes die *gene_id* abgespeichert und überprüft, ob wir eine neue *gene_id* erreicht haben. Falls ja, wird ein neues Gen erstellt. Für die darauf folgenden Zeilen wird überprüft, ob wir ein neues Transkript erreicht haben. Neue Transkripte werden in einer *ArrayList<Transkript>* des dazugehörigen Gens abgespeichert. Transkripte wiederum besitzen eine *ArrayList<CodingDnaSequence> cdsList* und zwei *HashMap<Integer, CodingDnaSequence> cdsStartIndices*, *cdsEndIndices*. Die Transkripte werden mit den dazugehörigen *CodingDnaSequence*'s befüllt, wobei für jede erstellte *CodingDnaSequence* die Start- und Endposition in den jeweiligen *HashMap*'s als Key auf das erstellte Objekt verweisen. Zudem wird mit einer Zählvariable *int cdsCount* die Position der *CodingDnaSequence*'s innerhalb des Transkripts in dem *CodingDnaSequence* Objekt gespeichert.

II **Generieren der *ES-SE***

Zum generieren der *ES-SE* werden als erstes für alle in dem Genom abgespeicherten Gene, die dazugehörigen *Introns* errechnet und in einem *HashSet<Introns>* innerhalb des Gens abgespeichert. Dafür werden alle Transkripte eines Gens und deren *CodingDnaSequence*'s angeschaut. Die Introns werden dann mit jeweils zwei *CodingDnaSequence*'s berechnet (Bei Genen die sich auf dem *"-"* Strang befinden, müssen zuerst die die *cdsList*'s aller Transkripte invertiert werden und die Positionen der *CodingDnaSequence*'s neu berechnet werden. Das ist später relevant für die Identifikation der *WT*'s):

```
for (Transcript transcript : transcripts) {
    for (int i = 0; i < transcript.getCdsList().size() - 1; i++) {
        int intronStart = transcript.getCdsList().get(i).getEnd() + 1;
        int intronEnd = transcript.getCdsList().get(i + 1).getStart() - 1;
        Intron intron = new Intron(intronStart, intronEnd);
        introns.add(intron);
    }
}
```

III **Erstellen der *<out>.tsv* Datei**

3 — Ergebnisse

A — Appendix Section

hm

Text goes here