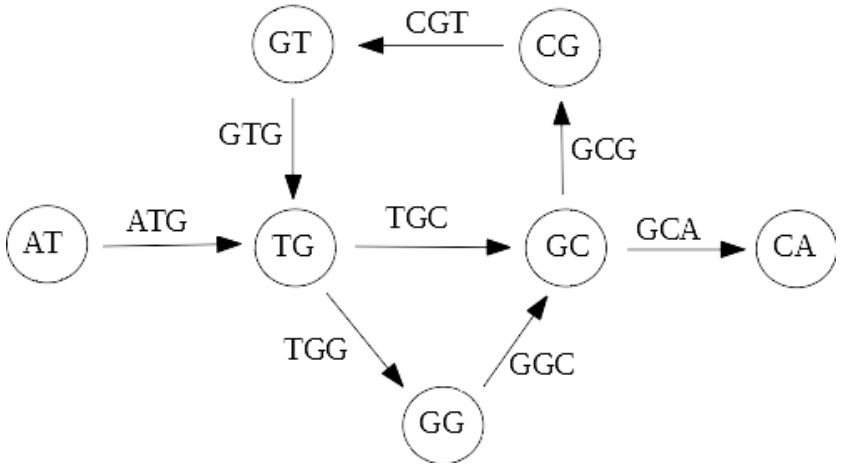
Bioinformatik

Sequenzassemblierung mit de Bruijn Graphen

Uwe Menzel, 2014

uwe.menzel@matstat.org

www.matstat.org



Nicolaas Govert de Bruijn

Nicolaas Govert de Bruijn

Born	9 July 1918 The Hague
Died	17 February 2012 (aged 93) Nuenen
Nationality	Dutch
Fields	Mathematics

http://de.forvo.com/word/nicolaas_govert_de_bruijn/

Aussprache auf Niederländisch [nl] Zurück zu Niederländisch

Aussprache von **viool** (Weiblich aus Niederlande)

0 Stimmen Gut Schlecht Zu Favoriten hinzufügen Als MP3 herunterladen

de Bruijn benutzte Graphen für eine abstrakte Aufgabe: Konstruiere den kürzesten (zirkulären) String, der alle möglichen Strings einer gegebenen Länge k als Substrings enthält.

Die Idee, de Bruijn Graphen zur Sequenzassemblierung zu benutzen, enstand erst 1989 im Zusammenhang mit der Sequenzierung durch Hybridisation[1] (SBH).

[1]Pevzner, P.A., J. Biomol. Struct. Dyn. 7, 63-73 (1989)

Warum brauchen wir neue Algorithmen zur Sequenz-Assemblierung?

- Neue Technologien (Next Generation Sequencing, **NGS**) machen de-novo Assemblierung von ganzen Genomen möglich.
- Dabei ensteht eine sehr **grosse Anzahl kurzer Fragmente**, die assembliert werden müssen. Typisch: 1 Milliarde Fragmente mit je 100 bp[1].
- Die traditionelle Overlap-Layout-Consensus Methode ist dann nicht mehr praktikabel, denn es wären sehr viele Alignments zwischen Sequenzierungs-fragmenten ("Reads") zu machen.
- [1] Siehe Anhang, Tabelle Illumina, SOLiD, Roche 454

$$\frac{n \cdot (n-1)}{2} \sim \mathcal{O}\left(n^2\right) \qquad n = \text{Anzahl der Reads}$$

Plan für die Vorlesung

- **Grundidee** des Assemblierens mit Hilfe von de Bruijn Graphen verstehen
- Im Mittelpunkt soll ein kleines **Beispiel** stehen: die Assemblierung eines Genomes “von Hand”.

- Theorie wird “on-the-fly” erklärt.

Sequenzierungsprojekt für heute:

Minigenom: ATGGCGTGCA

Das Genom ist **unbekannt**, wir haben nur 2 Sequenzierungsfragmente ("**Reads**"):

ATGGCG
CGTGCA

Bakterium mit dem **kleinsten Genom** auf der Erde (Stand 2006):

- *Carsonella ruddii* mit 159.662 Nukleotiden
- Nakabachi A., *et al*. *Science*, 314 . 267 (2006).

1. Sequenzierungsfragmente in k-mere zerlegen

Idee: Fragmente in noch kleinere Teile zerlegen! (Verrückt!! ... Warum nur?)
(siehe Anhang: "Warum das Puzzle noch weiter zerschneiden?")

Wir zerlegen die Fragmente in überlappende k-Tupel ("k-mers")
Die Zahl k gibt die Länge der Teile an. (meros = Teil, griechisch)

Sei $k = 3$. Wir zerschneiden also unsere Reads in (überlappende) 3-Tupel:

ATGGCG wird zu:
ATG
TGG
GGC
GCG

CGTGCA wird zu:
CGT
GTG
TGC
GCA

Wir erhalten eine Anzahl von k-Tupeln, welche nun das Genom repräsentieren.
Die Gesamtheit dieser k-Tupel bezeichen wir als **Spektrum**:

- ATG, TGG, GGC, GCG, CGT, GTG, TGC, GCA

2. Präfixe und Suffixe des Spektrums finden

Hier noch einmal das gefundene **Spektrum**:

- ATG, TGG, GGC, GCG, CGT, GTG, TGC, GCA

Wir brauchen noch die **Präfixe** und **Suffixe** eines jeden k-Tupels des Spektrums:

- Präfix: die ersten $(k-1)$ Zeichen: der Präfix von ATG ist AT
- Suffix: die letzten $(k-1)$ Zeichen: der Suffix von ATG ist TG
- Präfix und Suffix sind also $(k-1)$ -Tupel.

Wir brauchen **nur unikale** Präfixe und Suffixe (keine Verdopplungen).

Die unikalen Präfixe und Suffixe zu dem obigen Spektrum sind:

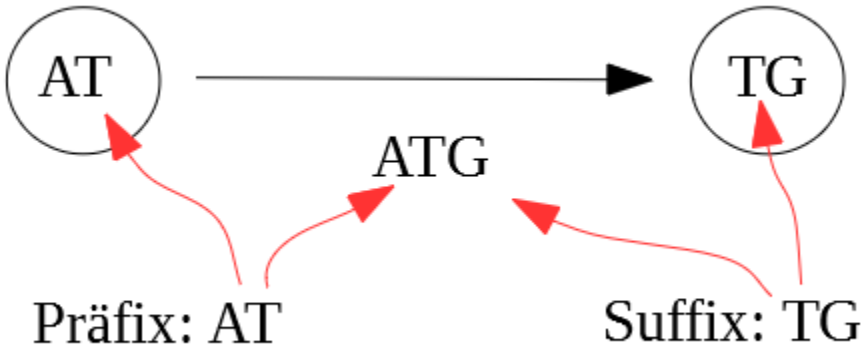
- AT, TG, GG, GC, CG, GT, CA

3. Knoten und Kanten des Graphen definieren

Nun wird ein Graph konstruiert, der jeweils die Präfixe und Suffixe eines jeden k-Tupels miteinander verbindet:

- **Knoten** (nodes, vertices): die Präfixe und Suffixe aller k-Tupel
- **Kanten** (edges): die k-Tupel selbst

Beispiel: 2 Knoten, die durch eine Kante miteinander verbunden sind:

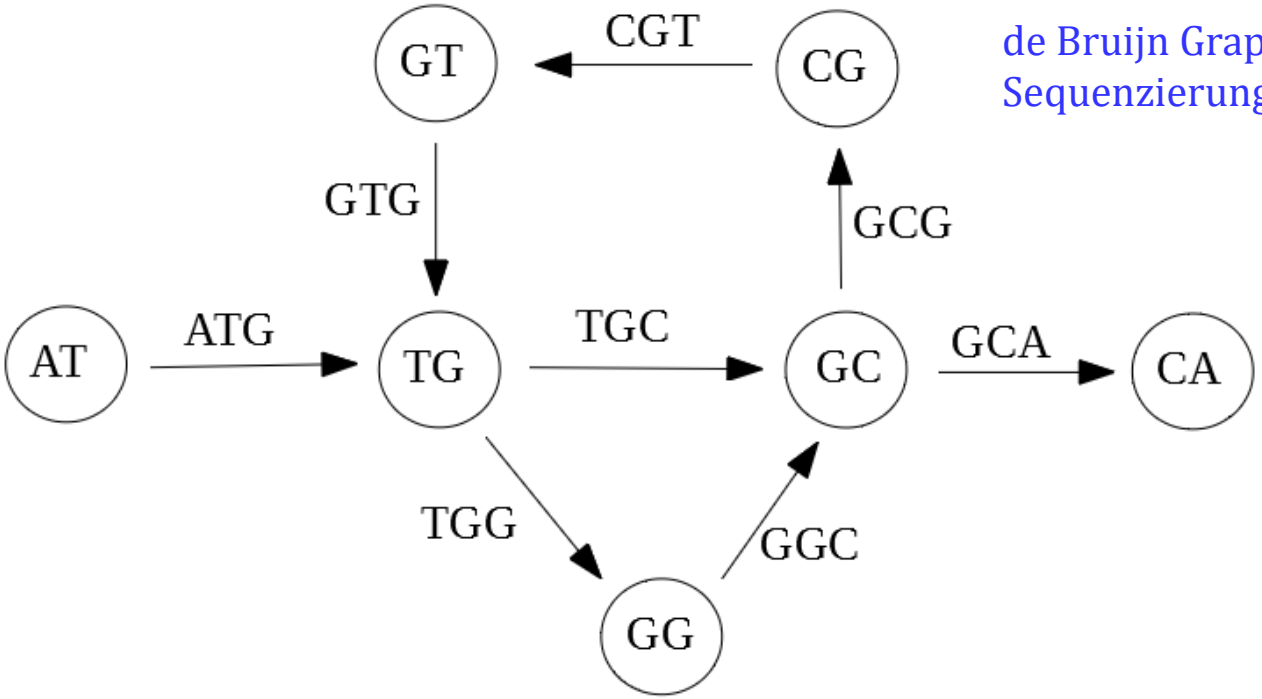


Es ensteht ein gerichteter Graph, denn die Richtung AT → TG ist durch den 3-Tupel ATG vorgegeben.

4. de Bruijn Graph konstruieren

Der de Bruijn Graph wird nun mit Hilfe der k-Tupel und $(k-1)$-Tupel konstruiert:

- Hier noch einmal die ***k*-Tupel**: ATG, TGG, GGC, GCG, CGT, GTG, TGC, GCA
  → werden zu **Kanten**
- Hier noch einmal die $(k-1)$ **-Tupel**: AT, TG, GG, GC, CG, GT, CA
  → werden zu **Knoten**

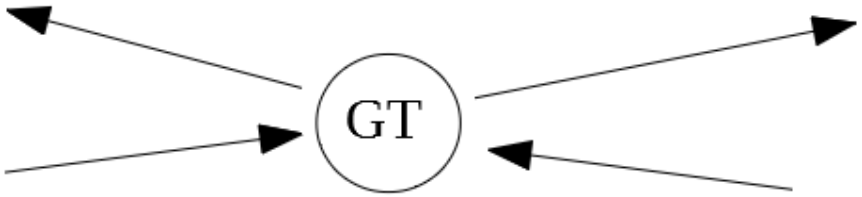


de Bruijn Graph für unser Sequenzierungsprojekt

5. Eulerpfad im Graphen finden

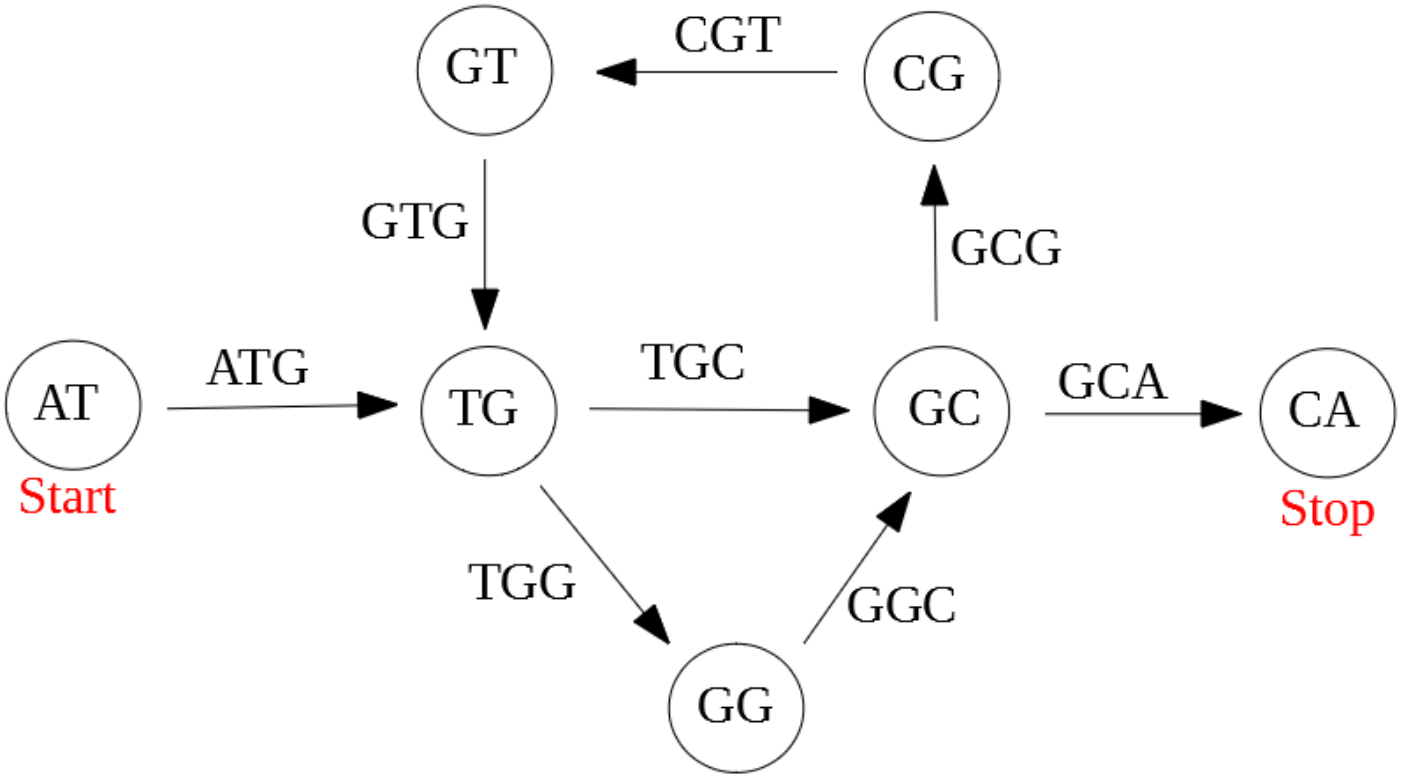
- **Eulerpfad:** ein Pfad, der jede **Kante** eines Graphen genau einmal durchläuft
- **Euler-Theorem**: ein (verbundener) gerichteter Graph hat genau dann einen Eulerpfad, wenn in jedem Knoten die Anzahl der einlaufenen Kanten gleich der Anzahl der auslaufenden Kanten ist (Der Graph ist "**balanciert**").
- Mögliche Ausnahme: Startknoten und Endknoten (bei nicht-zyklischen Pfaden)

Indegree = Outdegree



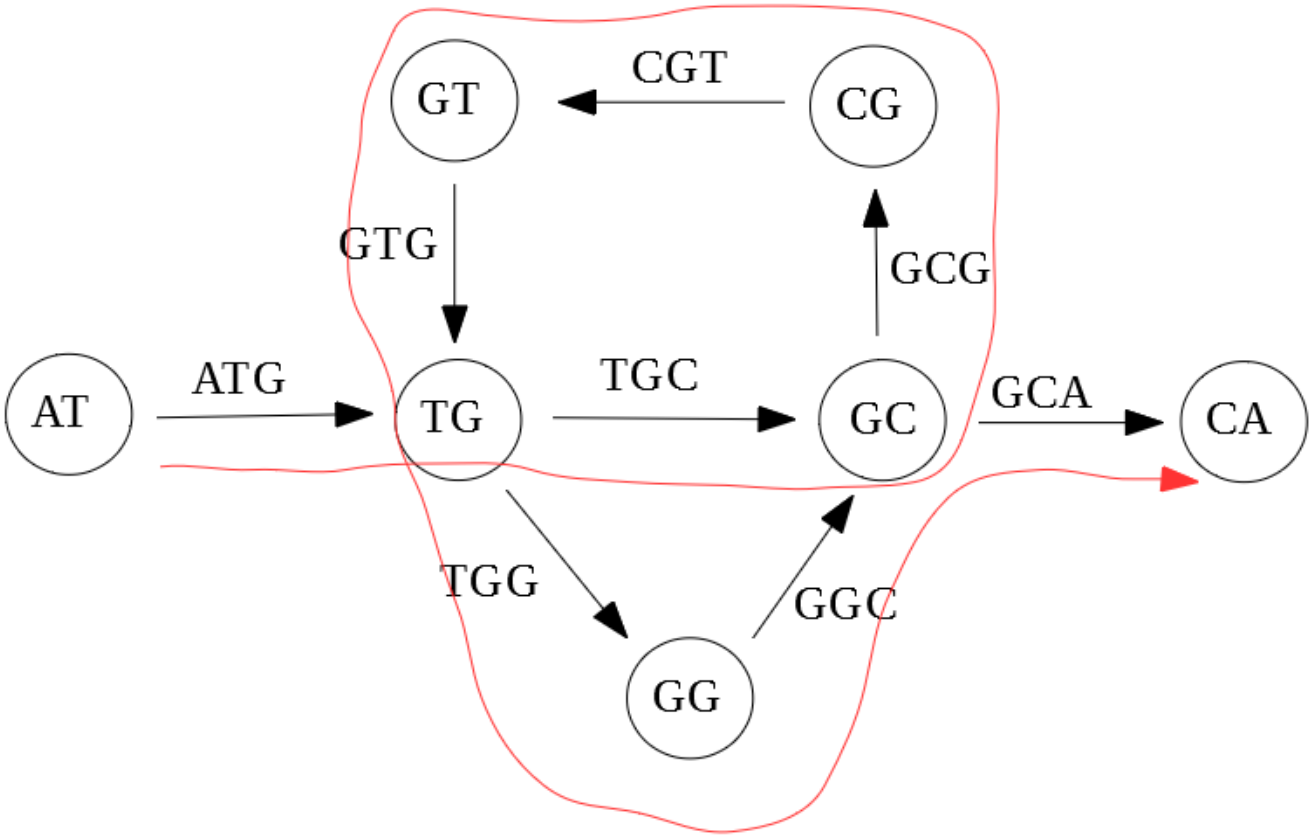
In den nach obiger Vorschrift konstruierten Graphen sind immer Eulerpfade auffindbar (Warum?)
(siehe Anhang: "Warum ist ein nach unserer Vorschrift konstruierter Graph immer ein Eulerscher?"

Indegree = Outdegree



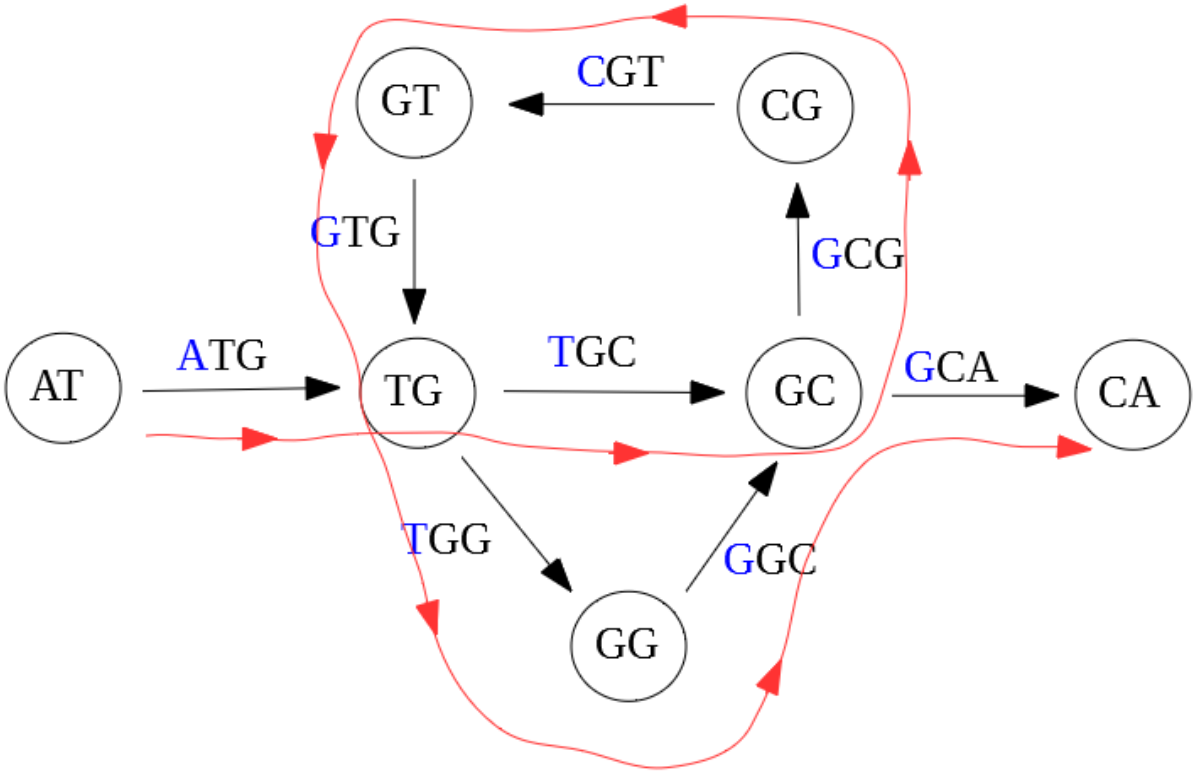
Ist diese Bedingung erfüllt, lässt sich immer (relativ leicht) ein Eulerpfad finden.
(siehe Anhang: Konstruktion eines Eulerschen Pfades)

Eulerpfad für unseren Graphen



6. Sequenz aus dem Eulerpfad rekonstruieren

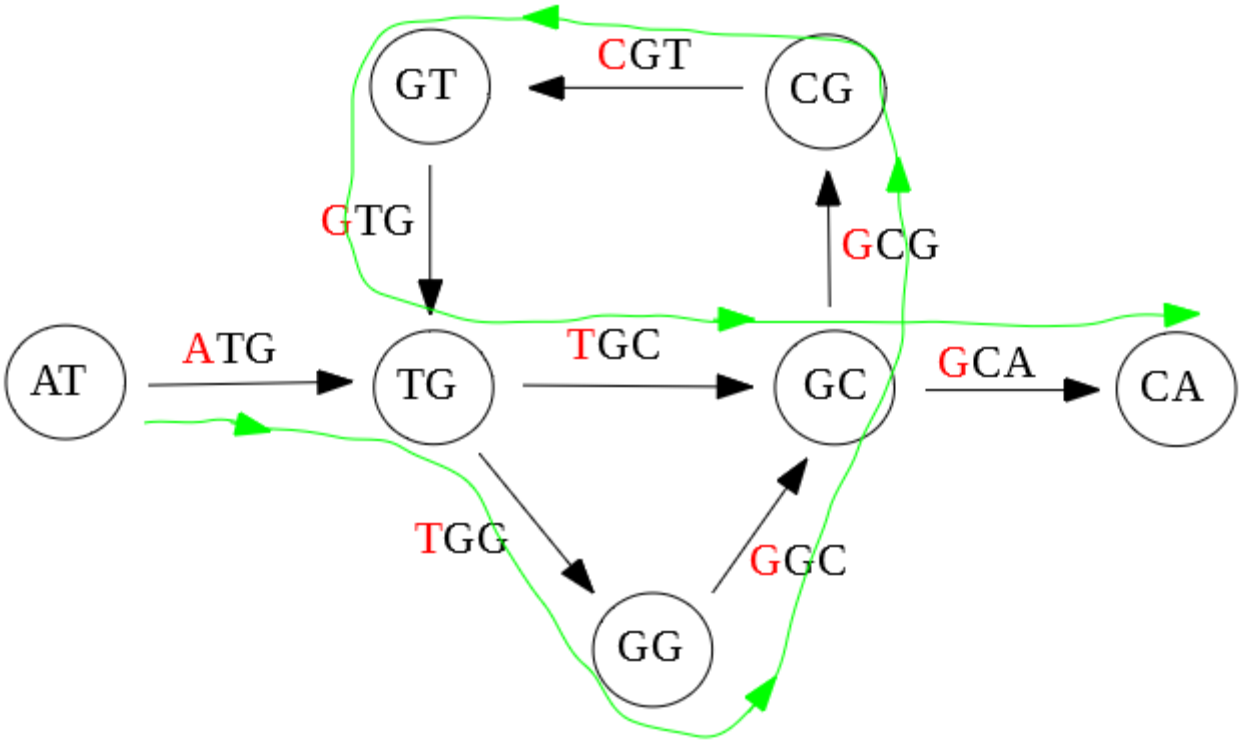
Eulerpfad entlangfahren und jeweils das erste Nukleotid mitnehmen:



Wir erhalten die Sequenz ATGCGTGGCA (den letzten Knoten dazunehmen)

7. Noch ein Eulerpfad für unseren Graphen

Vorsicht! In einem Graphen kann es mehrere Eulerpfade geben:



Wir erhalten die Sequenz ATGGCGTGCA (den letzten Knoten dazunehmen)

8. Welcher Eulerpfad ist der Richtige?

Wir erhielten die Sequenz ATGCGTGGCA

… und die Sequenz ATGGCGTGCA

Minigenom: ATGGCGTGCA

Die richtige Sequenz kann durch Vergleich mit den Reads gefunden werden:

ATGGCG

CGTGCA

Die rauhe Wirklichkeit

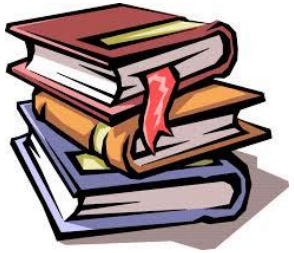
Das war ein schönes, einfaches Beispiel. Natürlich ist die Wirklichkeit sehr viel rauher:

- **Sequenzierungsfehler** … täuschen falsche Reads vor
- Ungleichmäßige **Coverage** … lassen Genomregionen unentdeckt
- DNA ist **double-stranded**
- **Repeats** … machen Assemblierung mehrdeutig
  - z.B: Assembliere ACGTACGT mit obiger Methode
  - (siehe Anhang: "Assemblierung eines Genoms mit Repeats")

Es wurden eine Reihe **heuristischer Methoden** entwickelt, um diese Probleme zu überwinden (siehe Literaturverzeichnis).

Heuristik (altgr. εὑρίσκω *heurísko* „ich finde"; von εὑρίσκειν *heuriskein* ‚auffinden', ‚entdecken') bezeichnet die Kunst, mit begrenztem Wissen (unvollständigen Informationen) und wenig Zeit zu guten Lösungen zu kommen.

Literatur



1. Compeau, Pevzner, Tesler: **How to apply de Bruijn graphs to genome assembly**. Nat Biotechnol. 2011 Nov 8; 29(11):987-91. - Sehr gute Einführung und Review
2. Idury, Waterman: **A new algorithm for DNA sequence assembly.** J Comput Biol. 1995 Summer;2(2):291-306. Ansatz: Imitiere Fragment-Assemblierungs-Problems als ein SBH ("Sequencing by Hybridizytion")-Problem.
3. Chu, Lu, Liu, Lee, Li, Shih: **Assembler for de novo assembly of large genomes.** Proc Natl Acad Sci U S A. 2013 Sep 3;110(36):E3417-24. Welche Software für welches Assemblierungs-Problem?
4. Pevzner , Tang, Tesler: **De novo repeat classification and fragment assembly**. Genome Res. 2004 Sep;14(9):1786-96. - Heuristische Methoden zur Behandlung von Repeats und Sequenzierungsfehlern in de Bruijn Graphen.
5. Pevzner, Tang, Waterman: **An Eulerian path approach to DNA fragment assembly.** Proc Natl Acad Sci U S A. 2001 Aug 14;98(17):9748-53. "Meilenstein", Programm EULER.
6. Zerbino, Birney: **Velvet: algorithms for de novo short read assembly using de Bruijn graphs.** Genome Res. 2008 May;18(5):821-9. Ein anderes Programm zur Assemblierung mittels de Bruijn Graphen.

Software

Name	Link	Plattform
EULER-SR	http://euler-assembler.ucsd.edu/portal/ ?	obsolet?
Velvet	www.ebi.ac.uk/~zerbino/velvet/	64-bit Linux, Mac OS X
ALLPATHS	www.broadinstitute.org/software/allpaths-lg/blog/	(Source)
ABySS	www.bcgsc.ca/platform/bioinfo/software/abyss	(Source)
SOAPdenovo	http://soap.genomics.org.cn/soapdenovo.html	64-bit Linux, Source

Eulerpfade sind leicht zu finden, selbst in sehr großen Graphen.
(siehe Anhang: “Konstruktion eines Eulerschen Pfades”)

- Algorithmus von **Fleury**
- Algorithmus von **Hierholzer**

Laufzeit skaliert mit der Anzahl der Kanten: $t \sim \mathcal{O}(n)$

Software

Name	Link	Plattform
EULER-SR	http://euler-assembler.ucsd.edu/portal/ ?	obsolet?
Velvet	www.ebi.ac.uk/~zerbino/velvet/	64-bit Linux, Mac OS X
ALLPATHS	www.broadinstitute.org/software/allpaths-lg/blog/	(Source)
ABySS	www.bcgsc.ca/platform/bioinfo/software/abyss	(Source)
SOAPdenovo	http://soap.genomics.org.cn/soapdenovo.html	64-bit Linux, Source

Eulerpfade sind leicht zu finden, selbst in sehr großen Graphen.
(siehe Anhang: “Konstruktion eines Eulerschen Pfades”)

- Algorithmus von **Fleury**
- Algorithmus von **Hierholzer**

Laufzeit skaliert mit der Anzahl der Kanten: $t \sim \mathcal{O}(n)$

Anhang

Sequenzassemblierung mit de Bruijn Graphen

Uwe Menzel, 2014

uwe.menzel@matstat.org

www.matstat.org

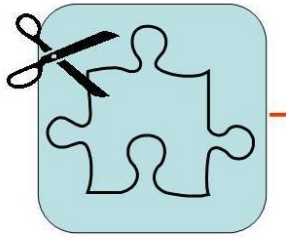
Warum das Puzzle noch weiter zerschneiden?

Es funktioniert!

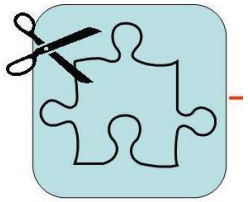
- Pevzner, Tang, Waterman: **An Eulerian path approach to DNA fragment assembly.** Proc Natl Acad Sci USA, 98(17):9748-53 (2001).
- "**EULER**" macht weniger Assemblierungsfehler mit einem realen Genom als andere Programme mit einem fehlerfreien (durch "Schreddern" erhaltenes) Genom.

Es ist unwahrscheinlich, dass z.B. Reads der Länge 100 aus einem Sequenzierungsprojekt auch alle 100-Tupel des Genomes repräsentieren. Es ist wahrscheinlicher, dass z.B. nach Zerlegung 51-Tupel ein fast vollständiges Spektrum aller 51-Tupel des Genoms vorliegt.

- Compeau, Pevzner, Tesler: **How to apply de Bruijn graphs to genome assembly.** Nat Biotechnol. 29(11):987-91 (2011).



Warum das Puzzle noch weiter zerschneiden?



Beispiel: unser Sequenzierungsprojekt

Minigenom: ATGGCGTGCA (Länge 10 bp)

Anzahl der 6-Tupel im "Genom": $10 - 6 + 1 = 5$

Durch unsere Reads werden jedoch nur 2 von 5 möglichen 6-Tupeln repräsentiert:

ATGGCG

CGTGCA

Aus den Reads erhalten wir die folgenden 3-Tupel ("3-mers"):

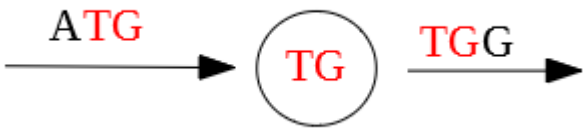
- ATG, TGG, GGC, GCG, CGT, GTG, TGC, GCA

Anzahl der 3-Tupel im "Genom": $10 - 3 + 1 = 8$

- **Wir haben also alle im Genom vorhandenen 3-Tupel in unserem Spektrum!**

Warum ist ein nach unserer Vorschrift konstruierter Graph immer ein Eulerscher?

ATGGCG wird zu:
ATG
TGG
GGC
GCG



- Die k-Tupel überlappen sich alle bis auf das erste und das letzte Nukloetid.
- Jeder Knoten ist Suffix eines k-Tupels und zugleich Präfix des darauf folgenden k-Tupels.
- Damit hat jeder Knoten für jeden einlaufenden Pfeil auch einen auslaufenden.
- Ausnahmen bilden nur "Rand-Reads", die nicht genügend mit anderen Reads überlappen. Der Graph ist dann nicht verbunden, es entstehen mehrere Contigs.
- Sowohl der "indegree" als auch der "outdegree" geben an, wie oft das $(k-1)$-Tupel im Genom vorhanden ist.

Eulers Theorem für gerichtete Graphen

Ein verbundener gerichteter Graph hat dann und nur dann einen Eulerschen Zyklus wenn er balanziert ist.

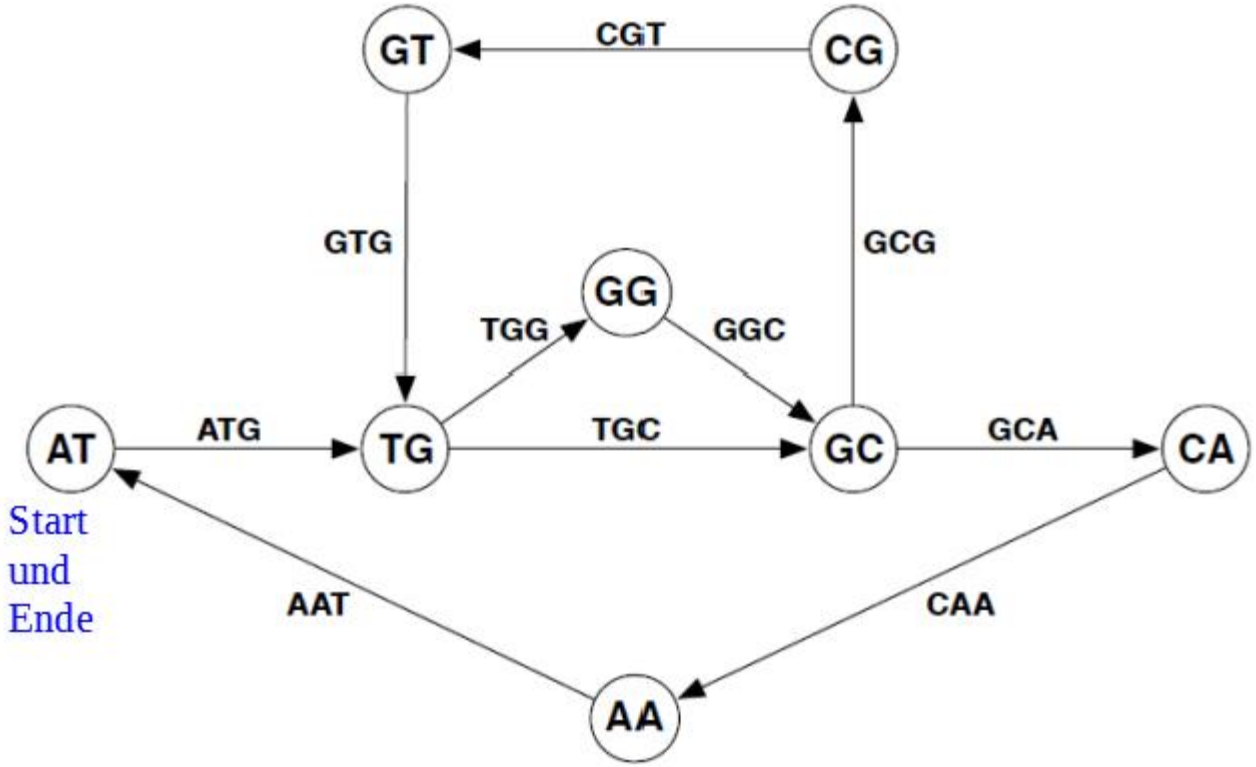
Die Bedingung der Balanziertheit ist notwendig und hinreichend:

a) notwendig: Um einen Eulerschen Zyklus zu haben, muss der Graph balanziert sein.
b) hinreichend: Wenn der Graph balanziert ist, kann immer ein Eulerscher Zyklus gefunden werden.

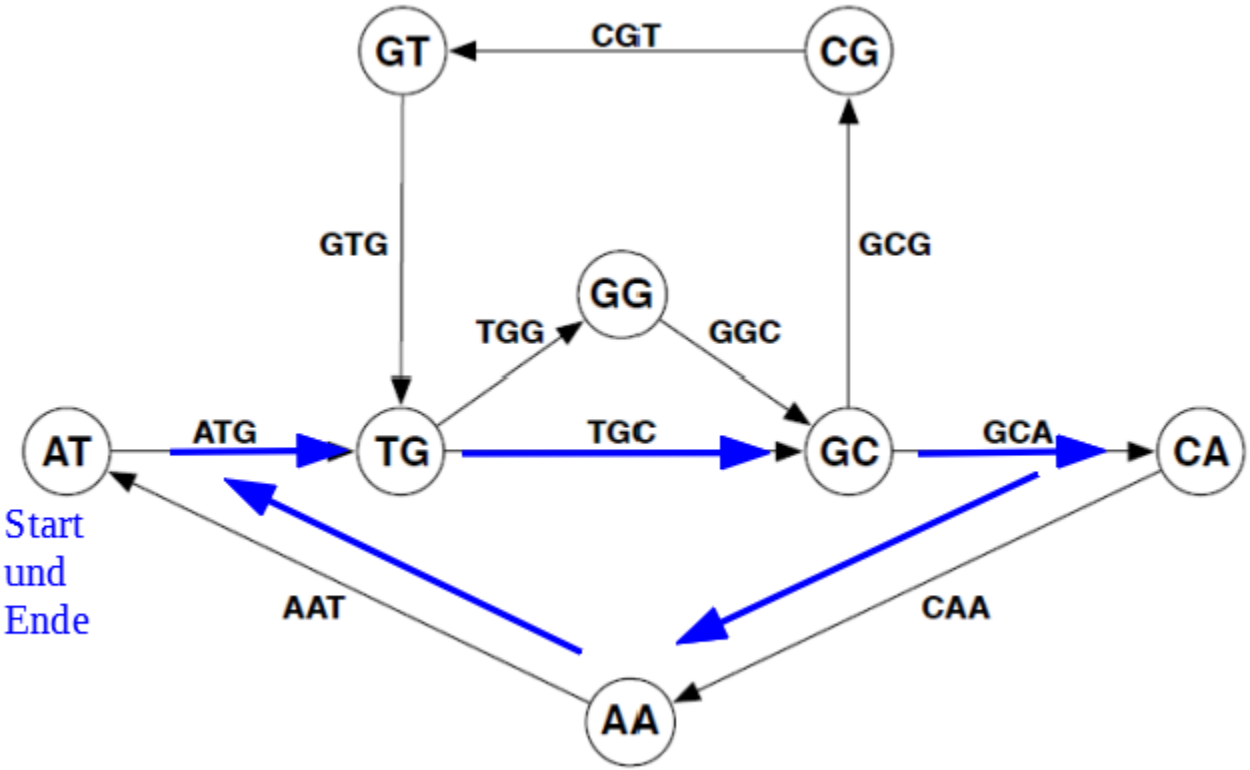
A 1753 portrait by Emanuel Handmann. This portrayal suggests problems of the right eyelid, and possible strabismus. The left eye, which here appears healthy ..

http://en.wikipedia.org/wiki/Leonhard_Euler

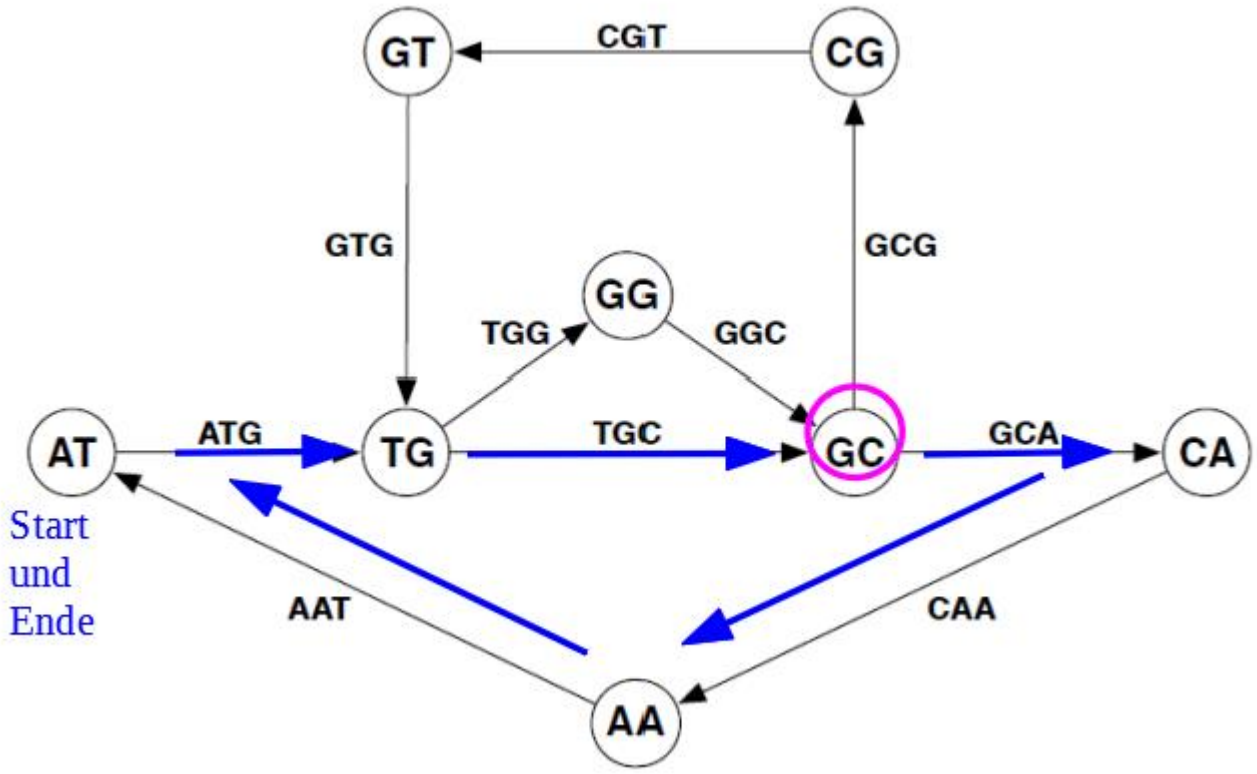
Konstruktion eines Eulerschen Pfades I



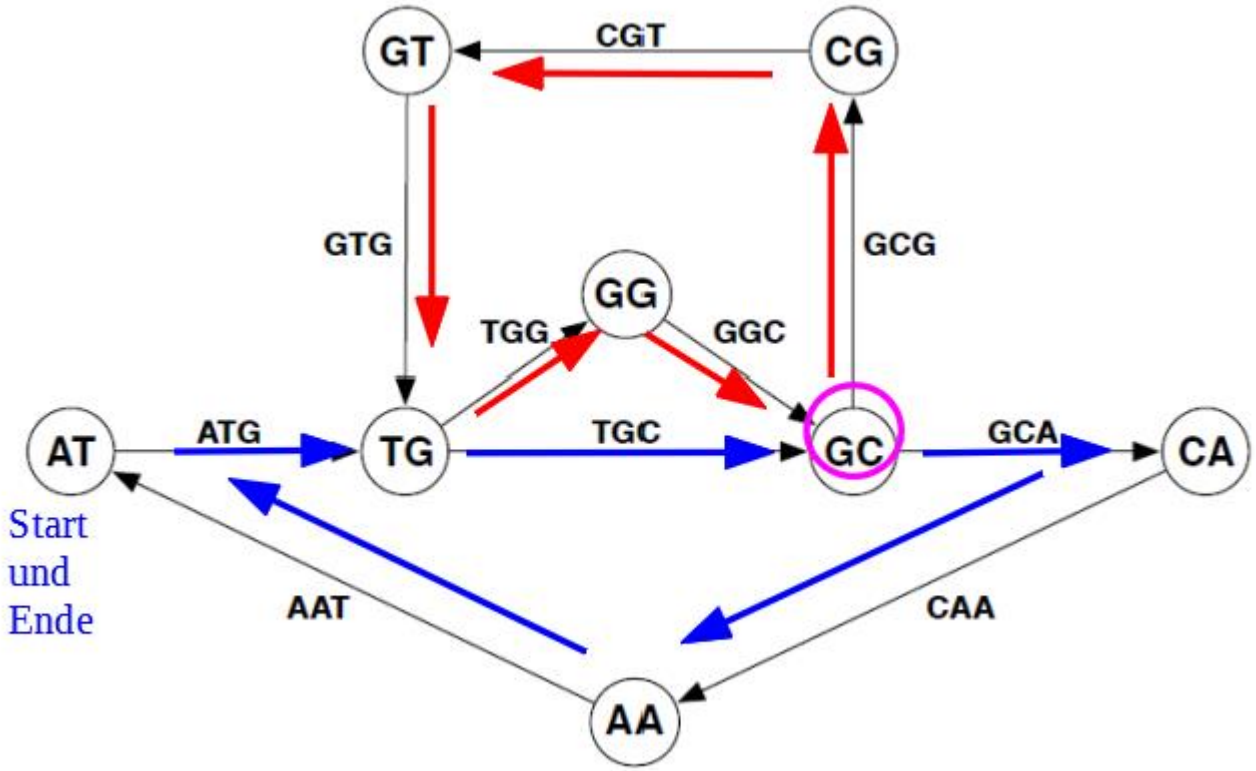
Konstruktion eines Eulerschen Pfades II



Konstruktion eines Eulerschen Pfades III



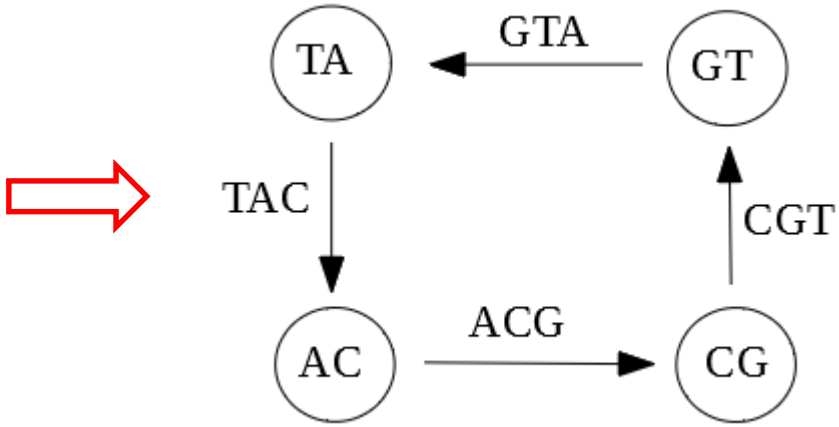
Konstruktion eines Eulerschen Pfades IV



Assemblierung eines Genoms mit Repeats

ACGTACGT Genom
ACGTAC Read1
TACGT Read2

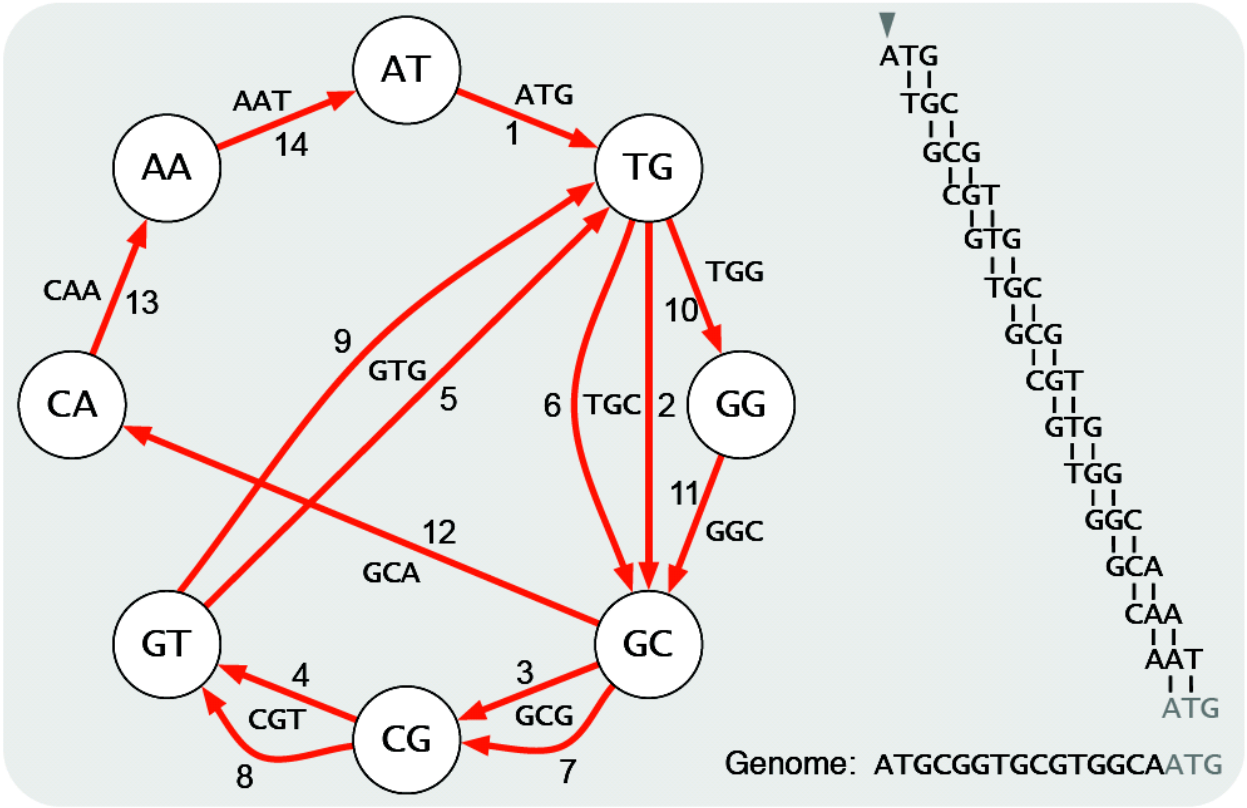
Spektrum (3-mers): ACG, CGT, GTA, TAC
$(k-1)$-mers: AC, CG, GT, TA



Sequenz: ACGT nur eine Instanz des Repeats

Assemblierung eines Genoms mit Repeats

Abhilfe: die Multiplizität eines jeden k-mers in Betracht ziehen!



Compeau, Pevzner, Tesler, N**at Biotechnol.** 2011 Nov 8;29 (11):987-91.

Fragmentlängen bei verschiedenen NGS-Sequenzierungsmethoden

Stand 2014

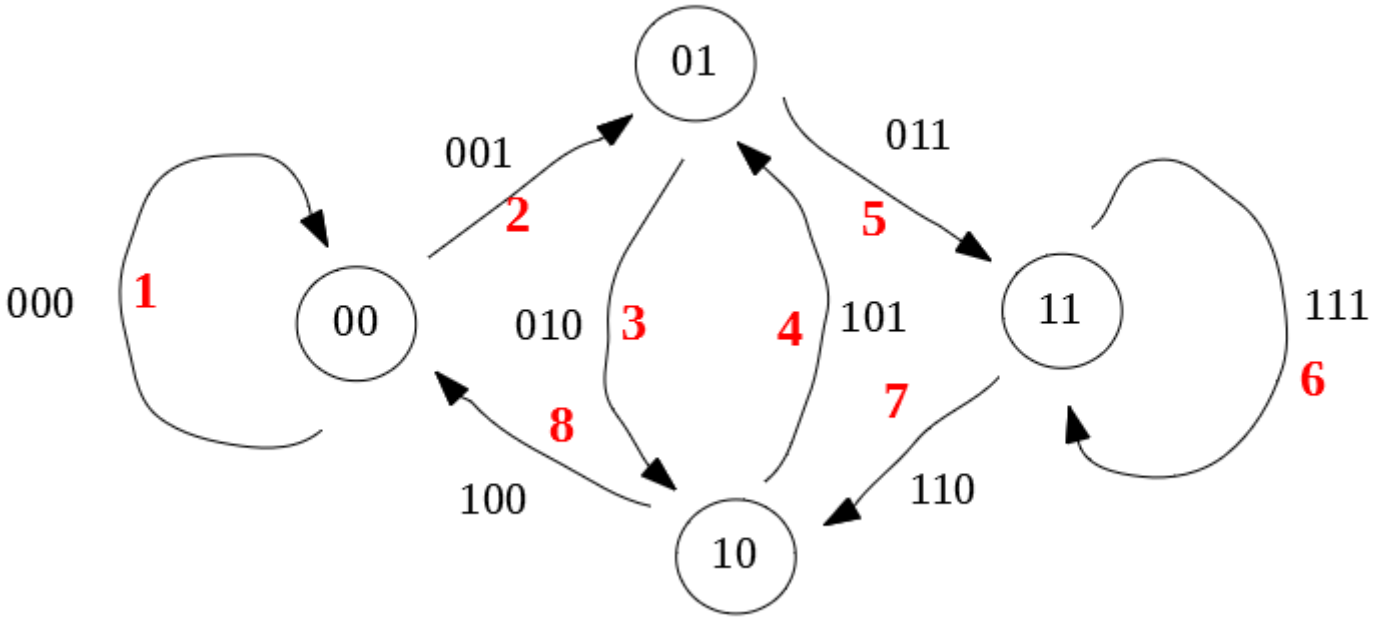
Methode	**Fragmentlänge** ("Read length")	**Genauigkeit** ("Accuracy")	**Fragmente je Lauf** ("Reads per run")
Roche 454	700 bp	99.9%	1 Million
Illumina HiSeq	50 bis 300 bp	98%	3 Milliarden
SOLiD	50+50 bp	99.9%	bis 1.5 Milliarden

Ursprüngliches Problem von de Bruijn I

- Gegeben sei ein “Alphabet”, z.B. (0,1) oder (A,B,C)
- **Aufgabe:** Finde den kleinsten (zirkulären) Superstring auf diesem Alphabet, der **alle** möglichen Strings einer festgesetzten Länge k als Unterstrings enthält
- **Beispiel:** Alphabet: (0,1) und $k = 3$
- $2^3 = 8$ mögliche Substrings: 000, 001, 010, 011, 100, 101, 110, 111
- **alle** möglichen $(k-1)$-tupel: 00, 01, 10, 11 (beim Assemblieren werden nicht alle, sondern nur die Präfixe und Suffixe der k-tupel einbezogen)

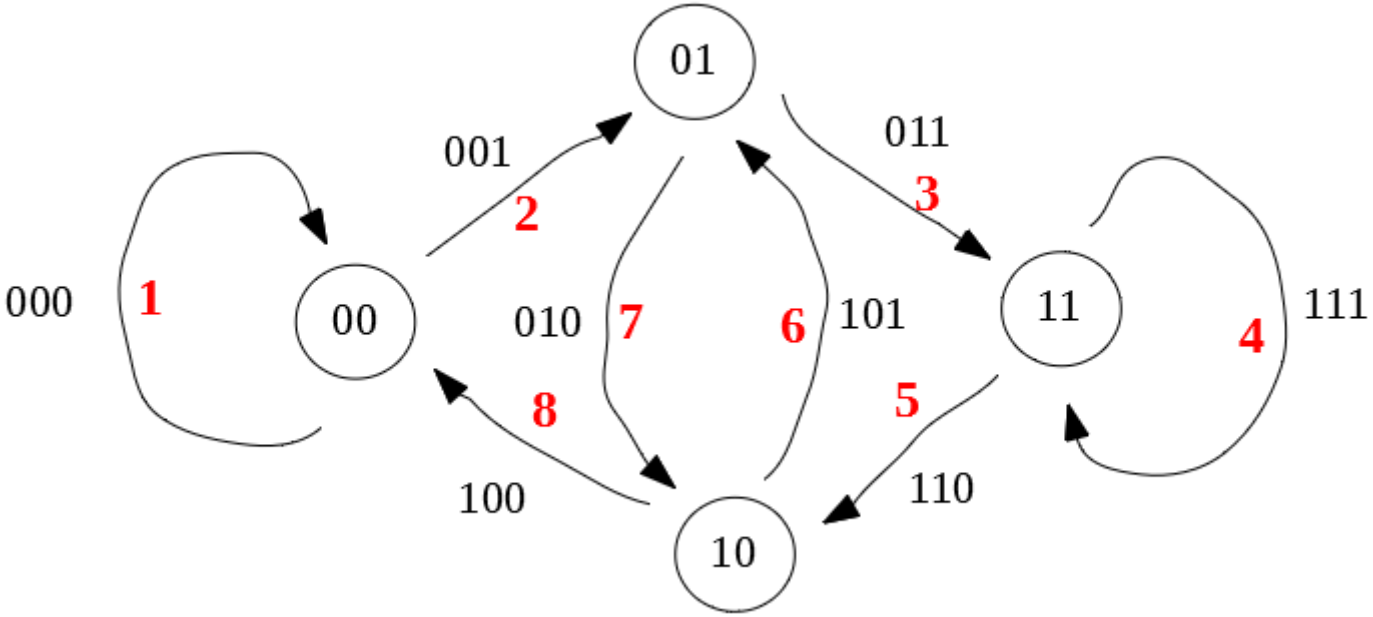
Ursprüngliches Problem von de Bruijn II

- mögliche Substrings: 000, 001, 010, 011, 100, 101, 110, 111
- mögliche $(k-1)$-tupel: 00, 01, 10, 11



Eulerscher Pfad ergibt die Lösung: **0001011100**

Ursprüngliches Problem von de Bruijn III



Eulerscher Pfad ergibt die Lösung: **0001110100**

Ein Beispiel auf (0,1) mit $k = 4$ in:

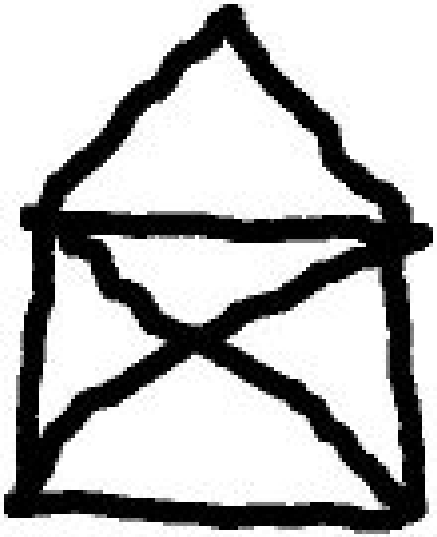
Compeau et al., **Nat Biotechnol.** 29(11):987-91 (2011).

Wann sind de Bruijn Graphen sinnvoll zur Assemblierung? (und wann eher nicht ...)

http://www.homolog.us/Tutorials/

The readers are possibly wondering why de Bruijn graphs were not widely used with Sanger sequencing. It was because the de Bruijn graphs did not preserve long-range positional information. Suppose a de Bruijn graph is constructed from a long Sanger read. If the Sanger read has repetitive structure, de Bruijn graph for the read itself will contain loops. That means one cannot go back from the de Bruijn graph to the read. However, the read itself is a true representation of part of the genome. By converting it into de Bruijn graph, we lost what was already known about that part of the genome.

Prominentes Beispiel für einen (ungerichteten) Graphen, in dem ein Eulerpfad gesucht wird



Euler Path, Euler Circuit

An **Euler path** is a path that uses every edge of a graph exactly once.

An **Euler circuit** is a circuit that uses every edge of a graph exactly once.

- An Euler path starts and ends at different vertices.
- An Euler circuit starts and ends at the same vertex.

From: **Jeremy L. Martin**, University of Kansas

Undirected Graph

The inescapable conclusion ("based on reason alone"):

If a graph G has an Euler circuit, then all of its vertices must be even vertices.

Or, to put it another way,

If the number of odd vertices in G is anything other than 0, then G cannot have an Euler circuit.

From: **Jeremy L. Martin**, University of Kansas

Handshaking Theorem

The Handshaking Theorem says that

In every graph, the sum of the degrees of all vertices equals twice the number of edges.

If there are n vertices $V_1, \ldots, V_n$, with degrees $d_1, \ldots, d_n$, and there are e edges, then

$$d_1 + d_2 + \cdots + d_{n-1} + d_n = 2e$$

Or, equivalently,

$$e = \frac{d_1 + d_2 + \cdots + d_{n-1} + d_n}{2}$$

From: **Jeremy L. Martin**, University of Kansas

- The number of edges in a graph is

$$\frac{d_1 + d_2 + \cdots + d_n}{2}$$

  which must be an **integer**.

- Therefore, $d_1 + d_2 + \cdots + d_n$ must be an **even number**.

- Therefore, the numbers $d_1, d_2, \cdots, d_n$ must include an **even number of odd numbers**.

- **Every graph has an even number of odd vertices!**

From: **Jeremy L. Martin**, University of Kansas

Connection between # of odd vertices and Eulerian path

Here is the answer Euler gave:

# odd vertices	Euler path?	Euler circuit?
0	No	**Yes***
2	**Yes***	No
4, 6, 8, . . .	No	No
1, 3, 5,	No such graphs exist	

* *Provided the graph is connected.*

Next question: If an Euler path or circuit exists, how do you find it?

From: **Jeremy L. Martin**, University of Kansas

Don't burn your bridges

To find an Euler path or an Euler circuit:

1. Make sure the graph has either 0 or 2 odd vertices.
2. If there are 0 odd vertices, start anywhere. If there are 2 odd vertices, start at one of them.
3. Follow edges one at a time. If you have a choice between a bridge and a non-bridge, always choose the non-bridge.
4. Stop when you run out of edges.

This is called **Fleury's algorithm**, and it always works!

From: **Jeremy L. Martin**, University of Kansas