Fragment Assembly in der Praxis

Am Beispiel des Systems CAP2

Vortrag baut auf dem zuletzt gehörten Vortrag auf. Nun: Fragment Assembly in der Praxis.

Vortrag ist top-down:

1. Sequence Assembly allgemein: Drei Phasen Methode der Sequence Assembly: Allgemein verwendete Methode ALLER Programme.
2. Sequence Assembly in CAP2
3. Detaillierte Beschreibung ausgewählter Methoden von CAP2.

Fragment Assembly: Zusammensetzung von DNA-Fragmenten zu Originalsequenzen. Methode Shotgun Assembly: nur Fragmente der Länge >= 400 Basen zufällig auswählen. Grund: DNA-Fragmente länger als 500-600 Basen lassen sich nicht auf einmal auslesen. Deshalb: DNA-Sequenz 5-10 clonen (Coverage); Clone in kürzere Fragmente aufspalten. Von Fragmenten der Länge >400 die ersten 400 Basen auslesen.

CAP (Contig Assembly Program) von 1991 von Xiaoqiu (sprich: Schaschu) Huang. Contig=zusammengesetzte DNA-Sequenz

CAP2 (Verbesserung) von 1995

Es gibt noch andere Programme. Test von ´94 auf Genauigkeit und Reproduzierbarkeit, mit der die vollständige DNA-Sequenz erzeugt wird. Testsieger: GCG, FAB, AutoAssembler (basiert auf CAP).

CAP2 ist gut. CAP2 eines der drei-vier Programme weltweit, die wirklich eingesetzt werden (Vingron).

Probleme bei der Fragment Assembly

- Trennung echter / zufälliger Überlappungen
- Leserichtung der Fragmente unklar
- Fragmente enthalten Lesefehler
- wiederholte Teilsequenzen
- hoher Rechenaufwand
- Superstring enthält Fehleranteil
- manuelle Nachbearbeitung nötig

Probleme: oft direkte Folgen der Techniken zur Gewinnung von DNA-Daten, die nur das Auslesen sehr kurzer Fragmente erlaubt.

•Grundproblem: Überlappungen von Fragmenten müssen berechnet werden. Welche sind zufällig, welche echt?

•Leserichtung der Fragmente unklar --> Frage, ob bei der Rekonstruktion ein gelesenes Fragment oder sein reverses Komplement verwendet werden soll.

•Fragmente enthalten Lesefehler (1%, wächst gegen Leseende auf bis zu 5% an).
Fehlerarten: falsche Base lesen, keine Base lesen, Auslassungen, Einschübe
Ursache: Fehlerbehaftete Technik des Auslesens

•Teilsequenzen wiederholen sich häufig bei höheren Lebewesen --> erhöht Fehleranfälligkeit der Assembly-Verfahren. Vor allem bei |Teilsequenz| > |Fragment|. Schlecht, weil kein Fragment mehr existieren kann, dass repetit. Sequenz überspannt und somit eindeutigen Ort zuweist.

•hoher Rechenaufwand (quadratischer Aufwand für alle Fragmentpaare und jedes einzelne Fragmentpaaar.
1000^2 Fragmentpaare mal 400^2 Vergleiche= 16 Mrd Vergleiche
---> lässt sich beschleunigen durch die Beschränkung auf „gut“ überlappende Fragmente, d.h. gewisse Mindestlänge eines gemeinsamen Teilfragments

•Ziel der Sequence Assembly: kürzesten Superstring finden. auch Superstring enthält immer gewissen Fehleranteil. Den gilt es zu minimieren. Keine Fehlerfunktion angebbar --> heuristische Methoden: auf gleichmäßige Überdeckung des Superstrings mit Fragmenten achten, Länge des Superstrings beachten.

•--> manuelle Nachbearbeitung von Problemstellen wichtig

Drei-Phasen-Methode der Sequence Assembly

1. **Overlap Detection**
   von DNA-Fragmenten

→Fragmentpaare entstehen

2. **Layout**
   von Substrings

3. **Deciding the Consensus**

→ Superstring entsteht

3-Phasen-Methode erklären:

1. **overlap detection**: Fragmentpaare werden gebildet. Hauptrechenaufwand in dieser Phase.

2. **substring layout:** Fragmentpaare zu Strings (Ketten) zusammenfassen.

   Vingrons Erklärung: beginnen mit beliebigem Fragment. Mit Greedy-Algorithmus das Fragment suchen, das am besten ‚passt', am weitesten überlappt. Wiederholen, solange Überlappungen gefunden werden können.

3. **deciding the consensus**: Teilsequenzen werden zu einem einzigen Superstring zusammengefasst. Problem bei Widersprüchen.

Methoden-Überblick von CAP2

1. Overlap Detection
   - Eliminierung schwacher Overlaps durch Filter
   - Dynamische Programmierung
2. Substring Layout
   - Teilordnung von ‚Enthalten'-Bäumen
   - Greedy-Algorithmus erstellt Contigs
   - Identifikation repetitiver Sequenzen
3. Deciding the Consensus

1. overlap detection:

   hoher Rechenaufwand: 95%

   Filter eliminiert unwahrscheinliche Overlaps schnell: Benutzt Heuristik. Fragmentpaare müssen sehr ähnliches Wort der Länge 20 besitzen.

   Filter passieren ist notwendige (nicht hinreichend) Bedingung für Fragmentpaare

   danach: Dynamische Programmierung identifiziert maximal scorende Overlaps.

2. substring layout

   Identifikation vollständiger Overlaps. Teilordnung von ‚Enthalten'-Bäumen entsteht. Nur Wurzeln gehen ins Layout ein, die enthaltenen Contigs passen sowieso

   einfache modifikation von greedy. Jeweils das global maximal scorende Fragmentpaar wird dazugenommen. Versuch, es anzulegen, sonst neuer Contig.

   Identifikation repetitiver Sequenzen durch Häufungen im Layout

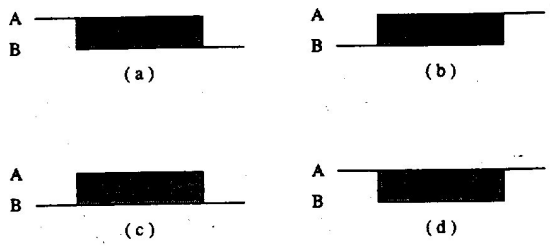
3. deciding the consensus

   Spalte für Spalte SubstingLayout untersuchen, häufigste Base geht in Superstring ein.
   Update von Huang: gibt es keine häufigste Base (z.B. 2xA,2xC,2xG), wird N gewählt.

Overlaps

partiell

vollständig



Abwandlung des Verfahrens aus erstem Vortrag.

1. Entscheidung: Welche Paare werden betrachtet?

   Anzahl der Fragmente wird verdoppelt durch Bildung sämtlicher Reverse Complements

   --> Beide möglichken Lesrichtungen werden berücksichtigt: Durch zusätzliche Prüfung mit dem Reverse Complement eines Fragmentes.
   --> (es genügt, nur noch geordnete f_i,f_j, $i<j$ näher zu betrachten.)
   --> zwangsläufig mehr falsche Paare

   Nur EINE der Overlapmöglichkeiten wird pro Fragmentpaar weitergereicht

2. Enscheidung: Welche Fragmentpaare werden weiterverwendet, welche nicht. Aufwendige dynamische Programmierung lohnt nur bei guten Overlaps.

   Idee: Fehlerrate bis fünf Prozent --> Überlappungen dürfen keine Fehlerrate größer als 10% haben.

   neuer Filter: Fragmentpaar passiert Filter nur, wenn **lückenfreie** gemeinsame Wörter der Länge 20 bestehen, innerhalb derer höchstens 2 Fehler auftreten dürfen und mindestens neun aufeinanderfolgende Basen/Nukloetide identisch sein müssen. Alles willkürliche Werte. Beispiel bringen. Erst dann wird die maximal-scorende Überlappung mit dynamischer Prog. (Aufwand O(n.m)) berechnet.

Gleichstände unwahrscheinlich wegen Technik des dynamischen Programmierens. Dennoch: Ungeklärt, was bei Gleichständen passiert. Beispiel: Suffix von A und Präfix von B überlappt genauso gut wie Präfix von A und Suffix von B.

Dynamische Programmierung

- Fragmentpaare basenweise aneinander vorbeiziehen
- maximaler Score:

$$H(i,j) = \begin{cases} \max\{H(i-1,j-1)+S(a_j,bj), \\ \quad H(i-1,j)-q, \\ \quad H(i,j-1)-q \} & \text{falls } i>0, j>0 \\ 0 & \text{sonst} \end{cases}$$

  $\max\{H(i,N),H(M,j): 1<=i<=M,1<=j<=N\}$
- minimale Overlap-Länge: 15-20 Prozent
- minimale Overlap-Ähnlichkeit: 85 Prozent

Folie fakultativ

Fragmente basenweise aneinander vorbeiziehen. Leserichtungsproblem durch Reverse Complements gelöst.

Maximaler Score: Abwandlung der Gleichung aus erstem Vortrag.

Aufwand der Originalgleichung: 2MN x MN

Aufwand der neuen Gleichung: MN (durch geschicktes Setzen der Randbedingungen.

Penalties (Strafen):

1. Hoch für Mutationen im Inneren der Fragmente: -6

2. Leicht für Lesefehler an den Fragmenträndern: -3

Matches bekommen: +2

Begründung: Mutationen deuten auf repet. Sequenzen hin --> Fragment hier falsch

Danach:

minimal overlap length: ca. 15-20 der Fragmentlänge

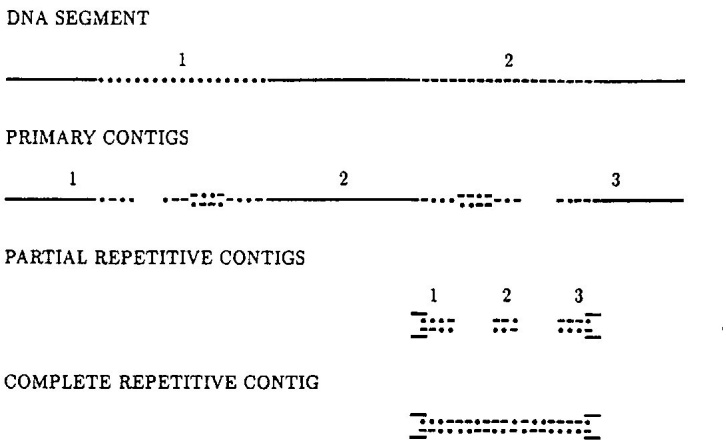
Begründung: kurze Überlappungen können Zufall sein

minimal overlap similarity: 85%.

Begründung: 2x5% Lesefehler + 5% Toleranz

Update von Huang: Fragmentpaare mit geringer Ähnlichkeit können hier wieder verworfen werden, auch wenn sie den Filter erfolgreich passiert haben.

Identifikation repetitiver Sequenzen

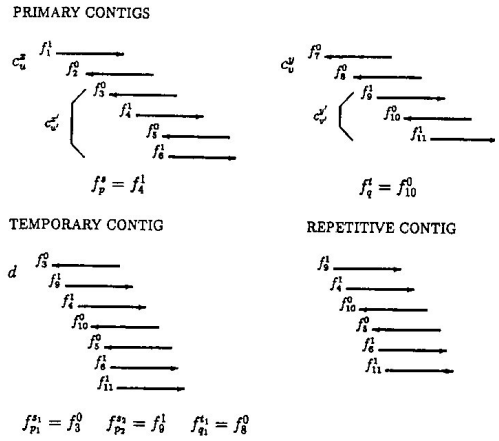


Identifikation repetitiver Sequenzen = die eigentliche Neuheit in CAP2 im Vergleich zu ursprünglichem CAP und anderen Systemen.

Allerdings: geht von folgenden Annahmen aus:

> Existieren nur 2 Kopien einer repetitiven Sequenz, so wird diese richtig erkannt, auch wenn sei mehrere tausend Basenpaare lang ist.
>
> Komplexere Repeats (3 oder mehr Kopien) werden „wahrscheinlich“ (sic!) nicht korrekt erkannt.
> Begründung: Einsichtig, dass bei dreifacher oder öfterer Wiederholung die dazwischenliegenden Stücke nicht mehr eindeutig plaziert werden können. **Zeichnung** (wenn genug Zeit).
> Anfang und Ende der Zwischenstücke identisch, beides Teilstücke der repetit. Sequenz.

Konstruktion von Contigs



Folie fakultativ: evtl weglassen

Probleme:

1. Leserichtung
2. Reduktion der Menge von in Phase 1 gebildeten Fragmentpaare nötig
3. Inkonsistenzen

2 Arten von Contigs werden gebildet: Primäre und repetitive Contigs.]

Consensus Sequence: Standardverfahren, wie beschrieben

Zusammenfassung

- komplexe Wiederholungen können zu Fehlern führen
- zahlreiche willkürliche Werte
- hoher Rechenaufwand
- Schwächen bei der Dokumentation
+ Maximale Overlaps werden ermittelt
+ Hohe Genauigkeit der Sequence Assembly
+ Professioneller Einsatz von CAP2

Folie Fakultativ!!!:
Kann keine komplexen Wiederholungen mit mehreren extrem ähnlichen Kopien erkennen

Willkürlich gesetzte Werte:

- Overlap-Vertrauenswert (Identisches Wort der Länge >=20): Muss heuristsch ermittelt werden, beeinflusst die Güte der Programmergebnisse stark.

- Mindest-Ähnlichkeitsmaß zwischen Overlaps: 70%/80%. Verschiedene Programmversionen.

- Strafpunkte für Indels 4 bzw. 3 --> beeinflusst, welche Fragmentpaare am besten überlappen

- Dokumentation oft unklar: wird zur Detektivarbeit

Flexibilität gut, aber bekommt Anschein von Stochern im Nebel.

Im Gegensatz zu Konkurrenzsystemen werden maximale Overlaps zwischen Basenpaaren immer gefunden (durch dyn. Programmieren).

CAP: Betonung der Geschwindigkeit: 1015 Fragmente werden in 4h zusammengesetzt

CAP2:

- Betonung der Genauigkeit: 1467 Fragmente --> Sequenz mit Länge 73328bp: Nur acht Fehler!
- zusätzlich 15% schneller als CAP. (durch schnellere Overlap-Detection-Phase).

--> professioneller Einsatz