Treffen 07.11.2014

Dorle Osterode, Stefan Dang & Lukas Götz

18. November 2014

1 Implementationsmöglichkeiten der benötigten Datentypen in Genometools

Erste Ideen zur Implementierung der Datentypen fuer Scaffold-Graphen in C unter Verwendung bestehender Datentypen aus Genome-Tools.

```
/* Scaffold-Graph */

typedef enum { VC_UNIQUE, VC_REPEAT,VC_POLYMORPHIC, VC_UNKNOWN }
    VertexClass;

/* Vertex */

struct GtScaffoldGraphVertex
{
  /* eindeutige ID fuer den Knoten */
  GtUword id;
  /* Laenge der Sequenz, die der Contig darstellt */
  GtUword seqlen;
  /* Wert der A-Statistik, um Contigs als REPEAT oder UNIQUE
     klassifizieren zu koennen;
     in Genom-Tools vom Typ float */
  float astat;
  /* abgeschaetzte Anzahl an Vorkommen des Contigs im Genom */
  float copynum;
  /* zur Klassifikation des Knotens: REPEAT, UNIQUE, ... */
  VertexClass vertexclass;
  bool hasconflictinglink;
  GtUword nofedges;
  /* Sammlung von Kanten, die von dem Contig abgehen */
  struct GtScaffoldGraphEdge   **edges;
  /* Markierung fuer Algorithmen; aus Genome-Tools entnommen siehe
  match/rdj-contigs-graph.c */
  GtContigsGraphMarks color;
};

/* Edge */
```

```
struct GtScaffoldGraphEdge
{
  /*  Knoten, zu dem die Kante fuehrt */
  struct GtScaffoldGraphVertex *pend;
  /* Kante, die genau in die andere Richtung fuehrt */
  struct GtScaffoldGraphEdge *ptwin;
  /* Markierung fuer Algorithmen */
  GtContigsGraphMarks color;
  /* Abschaetzung der Entfernung der verbundenen Contigs */
  GtWord dist;
  /* Standardabweichung von der abgeschaetzten Entfernung */
  float stddev;
  /* Anzahl der Distanzinformationen, die ein Anzeichen fuer die
  Verbindung der Contigs geben */
  GtUword numpairs;
  /* enthaelt die Richtung (Sense, Antisense) und welche
     Straenge die paired-Information enthalten (die gleiche
     Richtung oder das Reverse) */
  bool antisense;
  bool reverse;
};

/* Graph */
struct GtScaffoldGraph
{
  struct GtScaffoldGraphVertex **vertices;
  GtUword nofvertices;
  struct GtScaffoldGraphEdge **edges;
  GtUword nofedges;
};
```

2 Verfeinerung des Algorithmus zur Filterung der Knoten

- graph.visit() Funktion durchläuft alle Knoten (HashMap) und ruft auf jedem Knoten die übergebene Vistor-Funktion auf. Vorher wird previsit() und danach postvisit() aufgerufen. (Quelle ScaffoldGraph.h)

- Es dürfte keine Probleme beim zusammenlegen der Filterfunktionen geben, da die Bedingungen nacheinander für jeden Knoten lokal geprüft werden können.

- Es sollte beachtet werden, dass es gegebenfalls andere Ergebnisse geben könnte, wenn in einem späteren Schritt die vorher schon herausgefilterten Knoten noch beachtet werden.

- Die repetitiven Knoten sollten schon bei der Konstruktion des Graphen heraus-

gefiltert werden.

- Die markierten Knoten und Kanten können nicht sofort gelöscht werden, da die restlichen Knoten noch nicht klassifiziert sind. Da die Klassifikation eines Knoten aber anhand aller Nachbarn durchgeführt wird, können die Knoten und Kanten noch nicht früher gelöscht werden.

- Bei SGA wird gezählt, wie viele Knoten anhand welches Kriteriums gelöscht werden. Dies könnte auch noch eingebaut werden.

Algorithm 1: Zusammengefasste Filterfunktion (Schritt 4a und 4b vereinigt)

```
1  foreach Knoten k_0 im Graph G do
2      foreach Kantenrichtung dir in [ANTISENSE, SENSE] do
3          foreach Kantenpaar (A, B) in Richtung dir do
4              k_1 = A.pend;
5              k_2 = B.pend;
6              if AmbiguousOrdering(A, B, p_cutoff) and
               k_1.estCopy + k_2.estCopy < cn_cutoff then
7                  if k_1.estCopy < k_2.estCopy then
8                      markiere k_1 als polymorph und alle aus- und eingehenden SENSE
                       und ANTISENSE Kanten von k_1 schwarz, so dass sie im nächsten
                       Schritt nicht mitbeachtet werden.;
9                  end
10                 else
11                     markiere k_2 als polymorph und alle aus- und eingehenden SENSE
                       und ANTISENSE Kanten von k_2 schwarz, so dass sie im nächsten
                       Schritt nicht mitbeachtet werden.;
12                 end
                   // bei polymorphen Knoten wird nur das erste polymorphe
                      Kantenpaar markiert
13                 if Knoten k_0 ist polymorph markiert then
14                     break;
15                 end
16             end
17         end
           // polymorphe Knoten müssen nicht mehr auf inkonsistente Kanten
              überprüft werden
18         if Knoten k_0 ist polymorph then
19             break;
20         end
21         foreach Kantenpaar (A, B) in Richtung dir do
22             if A ist nicht schwarz und B ist nicht schwarz then
23                 Berechne Overlap von A und B und speichere längsten Overlap.;
24             end
25         end
26         if längster Overlap > 400 then
27             Markiere alle ausgehenden Sense-/Antisensekanten von k_0 rot;
28         end
29     end
30 end
31 Lösche alle markierten Knoten und Kanten;
```

Algorithm 2: Funktion AmbiguousOrdering(A, B, p_cutoff)

Data: Kante A und Kante B, die auf eindeutige Ordnung geprüft werden sollen. Wahrscheinlichkeitsschwellenwert p_cutoff

Result: Ob die Kanten A und B nicht eindeutig geordnet werden können

$\mu = A.dist - B.dist$;

$\sigma^2 = A.\sigma^2 + B.\sigma^2$;

$t = \frac{-\mu}{\sigma \cdot \sqrt{2}}$;

$P_{AB} = \frac{1}{2} \cdot \left(1 + \frac{2}{\sqrt{\pi}} \int_0^t \exp -x^2 \mathrm{d}x\right)$;

$P_{BA} = 1 - P_{AB}$;

return $\max\{P_{AB}, P_{BA}\} \leq p\_cutoff$

3 Notizen zu dem Layout-Algorithmus

- terminale Knoten: Knoten, die nur SENSE oder ANTISENES Kanten haben (Quelle: ScaffoldAlgorithms)
- Zusammenhangskomponenten-Funktion wird in StringGraph/GraphSearchTree.h definiert
- beim Walk für den Scaffold wird zuerst für jede Zusammenhangskomponente jeder Pfad zwischen terminalen Knoten mit einer Breitensuche (ohne heuristische Auswahl der Reihenfolge der Kindsknoten) berechnet. Dabei wird die Pfadlänge (Gap-Größe) minimiert. Als Layout für eine Zusammenhangskomponente wird dann der Pfad mit der längsten Sequenz (ohne Gaps) gewählt. (Quelle: ScaffoldGraph und ScaffoldWalk)

3.1 Layout-Algorithmus

Algorithm 3: Berechnung der Scaffolds (Schritt 6)

Data: Graph G
Result: Graph G ohne Knoten, die nicht zum bestem Walk gehören

```
1  Markiere alle Kanten aus G schwarz;
2  Berechnung der Menge C aller Connected Components von G;
3  foreach Connected Component c_0 aus der Menge C do
4  |  Berechne Menge der terminalen Knoten T (mit ausschließlich SENSE oder
   |  ANTISENSE Kanten) für die Connected Component c_0;
5  |  foreach Terminaler Knoten t_0 aus der Menge T do
6  |  |  Berechne die Menge W aller Walks durch die Connected Component c_0 von t_0
   |  |  aus;
7  |  |  foreach Walk w_0 aus W do
8  |  |  |  if Contig-Gesamtlänge > bislang beste Contig-Gesamtlänge then
9  |  |  |  |  Setze aktuellen Walk w_0 als bestWalk;
10 |  |  |  end
11 |  |  end
12 |  end
13 end
14 Setze alle Kanten des bestWalk weiß;
15 Lösche alle schwarzen Kanten;
```

Algorithm 4: Berechnung der Walks (Schritt 6.1)

Data: terminaler Startknoten t_0
Result: Alle von diesem Knoten möglichen Walks

```
1  Konstruktionsrichtung = Richtung der vom terminalen Knoten t_0 ausgehenden
   Kanten;
2  foreach Kante A vom Knoten t_0 ausgehend (in Konstruktionsrichtung) do
3      k_0 = A.pend;
4      Speichere Startkante A und Distanz A.dist in Map an Position k_0 (für spätere
       Traversierung);
5      Schiebe Startkante A und Distanz A.dist in Queue;
6  end
7  while BFS über Queue nicht beendet do
8      Poppe Kante A und Distanz A.dist aus der Queue;
9      k_0 = A.pend;
10     foreach Kante B in Konstruktionsrichtung von k_0 aus do
11         k_1 = B.pend;
12         if Distanz zu aktuell betrachtetem Knoten k_1 < bisher ermittelte Distanz zu
           k_1 OR Knoten k_1 noch unbetrachtet then
13             Speichere Kante B und Distanz B.dist in Map an Position k_1;
14             Schiebe Kante B und Distanz B.dist in Queue;
15         end
16     end
17     if k_0 hat keine Kanten in Konstruktionsrichtung then
18         Schiebe Knoten k_0 in terminalSet;
19     end
20 end
21 foreach Knoten k_0 in terminalSet do
22     Erzeuge Walk mithilfe einer Traversierung über die Map
23 end
```