Aufgabe 1

Suche nach Polymorphismen

Alle Algorithmen, die zur Polymorphismensuche eingesetzt werden gehen auf den Algorithmus von **Needleman-Wunsch**[1] zurück. Der Algorithmus erstellt eine $n \times m$ Matrix aus der, wenn sie einmal erstellt ist, das beste **globale** Alignment ausgelesen werden kann. Die Matrix wird wie folgt berechnet:

Zunächst wird eine sogenannte „Similarity-Matrix“ benötigt, die angibt wie „ahnlich“ sich zwei Zeichen der beiden Sequenzen sind. Diese Matrizen können so kompliziert sein wie das folgende Beispiel, die Einheitsmatrix funktioniert aber prinzipiell auch.

$$\left(\begin{array}{c|cccc} - & A & G & C & T \\ \hline A & 10 & -1 & -3 & -4 \\ G & -1 & 7 & -5 & 0 \\ C & -3 & -5 & 9 & 0 \\ T & -4 & -3 & 0 & 8 \end{array}\right) \tag{1}$$

Nun wird für jede Zelle der maximale Alignment-Wert nach der folgenden Formel berechnet. Der Wert für die aktuelle Zelle ist entweder der Wert des Vorgängers ($a_{n-1,m-1}$) addiert mit dem entsprechenden Wert aus der Similarity-Matrix oder, falls dort ein größerer Wert existiert, ein Wert aus der nächstkleineren Zeile/Spalte. In diesem Fall wird eine „Gap-Penalty“ d auf den Wert addiert. Diese sollte sinnvollerweise ≤ 0 sein.

$$a_{n,m} = \max(a_{n-1,m-1} + S(\mathrm{pos}_a(n), \mathrm{pos}_b(n)), a_{n-1,m-j} + d, a_{n-i,m-1} + d) : 0 < i \leq n, 0 < j \leq m$$

Um das **globale** Alignment abzulesen wird jeweils der größte Wert in der höchstwertigen Spalte bzw. Zeile gesucht. Anschließend werden alle Zeilen und Spalten die größer oder gleich der aktuelle Zelle sind verworfen und die nächsthöchste Zahl gesucht.

Beispiel

Musste aus Schlafmangel leider ausfallen..

Varianten bzw. Weiterentwicklungen

Alle diese Algorithmen basieren auf dem beschriebenen Needleman-Wunsch-Algorithmus. Ansatz für Veränderungen und Verbesserungen sind unter Anderem die Similarity-Matrix und die Gap-Penalty. So können Unterschiede („Missmatches“, Mutationen) und Lücken („Gaps“) unterschiedlich bewertet werden.

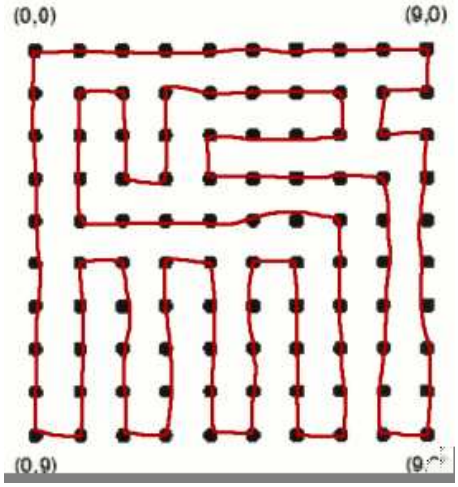
- Sellers; verwendet negative Werte für Fehler („Missmatches“) und eine Gap-Penalty. Häufig wird auch der Needleman-Wunsch-Algorithmus selbst mit diesen Eigenschaften beschrieben.
- Smith-Waterman; Sehr ähnlich zum „Sellers-Algorithmus“, jedoch werden Zellen, die einen negativen Wert erhalten würden auf 0 gesetzt. So ist es möglich auch **lokale** Alignments zu finden.
- BLAST Verkürzt die Laufzeit des vorgestelten Algorithmus indem es Heuristiken verwendet. Wie der Algorithmus genau funktioniert weiß ich leider nicht.

Aufgabe 2

Der minimale Abstand zwischen einem Punkt und einem beliebigen Nachbarn ist in diesem Graph immer 1. Eine Lösung des TSP mit 100 Knoten hat genau 100 Kanten, daher hat die kürzeste Rundreise eine Länge von $100 \cdot 1 = 100$.

[1] Nachdem eine Suche im Web in etwa so viele Varianten dieses Algorithmus wie Treffer liefert, werde ich eine Variante beschreiben die mir logisch erscheint – die Orginalfassung des Algorithmus war vermutlich etwas anders formuliert.

Eine der möglichen minimalen Lösungen ist diese:



TSP beim Chip-Design

Ein Problem, dass beim Chip-Design häufig auftritt, ist das sogenannte „Quadratic Assignment Problem“. Dabei geht es darum Einheiten, die miteinander kommunizieren müssen, möglichst nahe zueinander zu plazieren. Nachdem aber eine Einheit oft mit mehreren anderen Einheiten kommuniziert, die selbst wieder mit mehreren Einheiten kommunizieren, wird die Sache schnell kompliziert.

Häufig kann die selbe Funktionalität schnell (aber groß) oder platzsparend (aber langsam) implementiert werden. Somit könnte unter Umständen auch das Rucksack-Problem eine Rolle beim Chip-Design spielen.

Zu guter letzt ist das Traveling-Salesman-Problem beim Platinen-Design wichtig: Da sich ein Bohrkopf, der Löcher in eine Platine bohrt, aus vielerlei Gründen so wenig wie möglich bewegen soll, ist es naheliegent die Route, die dieser Bohrkopf zurücklegt, zu minimieren. Dabei entspricht jedes Bohrloch einer Stadt.