Hochschule für angewandte Wissenschaften Coburg
Fakultät Elektrotechnik und Informatik

Studiengang: Informatik

Projektarbeit in der bioinformatischen Sequenzanalyse

Implementierung eines Branch-and-Bound-Algorithmus für das Global-Alignment-Problem

Schuster, Rinaldo

Abgabe des Berichts: 03.02.2025

Betreut durch:

Prof. Dr. Michael Sammeth, Hochschule Coburg

Inhaltsverzeichnis

1 Einleitung

Sequenzalignment ist ein grundlegendes Problem in der Bioinformatik, das entscheidend ist, um funktionelle, strukturelle und evolutionäre Beziehungen zwischen biologischen Sequenzen identifizieren zu können. Traditionelle Algorithmen zur Bestimmung des globalen Alignments, wie etwa der Needleman-Wunsch-Algorithmus (NW), garantieren zwar optimale Lösungen, leiden jedoch meist unter hoher Rechnenkomplexität. Im Falle des NW-Algorithmus, beträgt die zeitliche und räumliche Komplexität etwa $O(n \cdot m)$, wodurch das Alignieren langer Sequenzen mit einem hohen zeitlichen Aufwand einhergeht. Das Finden einer *optimalen* Lösung in diesem Kontext beschreibt jenes Ergebnis, das für eine gegebene Problemstellung mit konkretem Bewertungsmaß die beste Lösung ist. Dies bedeutet, es kann noch weitere Lösungen mit einer gleich guten Bewertung geben, jedoch existiert kein weiteres Resultat, das eine bessere Bewertung für das Problem besitzt. Im Gegensatz zu optimalen Algorithmen existieren ebenfalls *heuristische* Methoden, also Schätzungen und Abwägungen über die zu diesem Zeitpunkt wahrscheinlichsten Lösungen. Diese benötigen meist erheblich weniger Laufzeit, garantieren jedoch keine Optimalität des Ergebnisses, indem sie zwar eine gültige, aber nicht zwingend die beste Lösung bestimmen. Das in diesem Bericht beschriebene Projekt adressiert diese Herausforderungen durch die Entwicklung und Implementierung eines Branch-and-Bound-Algorithmus basierend auf dynamischer Programmierung für das paarweise Sequenzalignment. Es soll versucht werden, eine möglichst einfache Strategie zu entwickeln, den Suchraum des globalen Alignment-Problems möglichst stark einzuschränken und dabei stets eine optimale Lösung zu gewährleisten.

Für den vorgeschlagenen Ansatz wird eine Kostenmatrix mit spezifischen Strafpunkten für Matches (0), Mismatches (+1) und Gaps (+1) verwendet. Eine zentrale Komponente des Algorithmus ist die Verwendung einer oberen geschätzten Schranke (auch Threshold genannt), um jene Teilbereiche des Suchraums abzuschneiden, die keine optimale Lösung mehr liefern können, da diese über der gewählten Abschätzung liegen. Unsere Abschätzung soll genau jene Kosten repräsentieren, die auf jeden Fall noch ein gültiges Ergebnis liefern können. Um diesen Threshold zu bestimmen, wird zunächst eine möglichst einfache Heuristik verwendet, wie beispielsweise die maximale Länge der beiden Eingabesequenzen. Darüber hinaus soll versucht werden, Strategien zu entwickeln, um verbesserte Schranken vorzuschlagen, sodass die Effizienz weiter gesteigert werden kann. Der Algorithmus wird hinsichtlich seiner theoretischen Laufzeit und der Anzahl der berechneten Schritte in Abhängigkeit der Sequenzähnlichkeit analysiert. Dabei werden sowohl Best-Case- (hohe Sequenzähnlichkeit) als auch Worst-Case-Szenarien (geringe Sequenzähnlichkeit) diskutiert, um das Verhalten des Algorithmus in typischen Anwendungsfällen zu charakterisieren.

2 Das globale Sequenzalignment

Wie bereits in der Einleitung angedeutet, dient das Sequenzalignment dazu, zwei (oder mehr) Sequenzen so anzuordnen, dass ihre Ähnlichkeit maximiert wird. Hierbei werden Ergebnissequenzen generiert, die eine möglichst geringe paarweise Differenz, also bestenfalls eine sehr hohe Übereinstimmung in den einzelnen Symbolen aufweisen. Um dieses Resultat zu ermitteln, werden beide Sequenzen paarweise miteinander verglichen, wobei die Symbole entweder übereinstimmen (*Matches*) oder abweichen (*Mismatches*) können. Um nun die Ähnlichkeit zu maximieren, arbeitet der Algorithmus im Grunde nach folgendem Schema: Wenn beide Symbole am aktuellen Index der Zeichenketten gleich sind, wird das übereinstimmende Zeichen in beide Ergebnissequenzen übertragen. Stimmen die Zeichen am aktuellen Index nicht miteinander überein, gibt es mehrere Optionen, die gewählt werden können. Einerseits kann man die unterschiedlichen Symbole jeweils in die Ergebnissequenz übernehmen und damit ein Mismatch alignieren, was jedoch zu zusätzlichen in der Kostenmatrix definierten Strafpunkten führt. Andererseits gibt es auch die Möglichkeit, sogenannte Lücken (*Gaps*) in eine der Ergebnissequenzen einzuführen und das andere Symbol in die andere Sequenz zu übernehmen. Dadurch wird der Index nur an einer der Eingabesequenzen erhöht, während die andere Sequenz das gleiche Zeichen für den nächsten paarweisen Vergleich benutzt, da dieses noch nicht in das Resultat übertragen wurde. Wenn eine Lücke eingeführt wird, dann führt dies sowohl zu spezifizierten Kosten als auch zu dem Anwachsen der Ergebnissequenzen. Lücken zu nutzen kann manchmal zu einem besseren Alignment-Ergebnis führen, jedoch sollte die Anzahl der Gaps möglichst gering gehalten werden, um ein immer noch aussagekräftiges Ergebnis zu erhalten. Das globale Alignment, wie es in dieser Arbeit betrachtet wird, bezieht sich auf die vollständige Alignierung der Sequenzen von Anfang bis Ende.

Für die Bestimmung des globalen Alignments haben sich bestimmte Lösungsansätze durchgesetzt. So ist etwa das dem Needleman-Wunsch-Algorithmus zugrundeliegende Paradigma der dynamischen Programmierung ein weit verbreiteter Ansatz zur Lösung von allgmeinen Alignment-Problemen. Die Strategie basiert hierbei auf der Zerlegung des Problems in kleinere Teilprobleme, deren Lösungen zwischengespeichert werden. Das Gesamtproblem kann dann durch die Kombination dieser Teillösungen ermittelt werden. Im Falle unseres Problems werden Teillösungen, also Kosten für die Alignierung bestimmter Subsequenzen, in einer Matrix gespeichert und so schrittweise zu einer Gesamtlösung kombiniert. Die Matrix Der Needleman-Wunsch-Algorithmus ist einer der am weit verbreitetsten Algorithmen für die Lösung des Alignmentproblems, weist jedoch eine zeitliche und räumliche Komplexität von $O(n \cdot m)$ auf, wobei n und m die Längen der Sequenzen sind. Wenn wir davon ausgehen, dass m und n grob in einer Größenordnung liegen, was bei praktischen Problemen meistens der Fall ist, so ordnet sich der Algorithmus geschätzt in der *quadratischen Komplexität* ($O(n \cdot m)$) ein. Dies bedeutet, dass die Rechenzeit quadratisch mit der Eingabegröße n anwächst. Der hohe Rechenaufwand

ist dadurch begründet, dass für alle möglichen Alignierungen die Kosten ermittelt werden und darauf basierend das optimale Ergebnis mit den geringsten Kosten durch *Backtracking* ermittelt wird.

3 Entwicklung des Branch-and-Bound-Algorithmus

Während verschiedene Ansätze existieren, die Komplexität des Needleman-Wunsch-Algorithmus zu verringern, soll in dieser Arbeit das *Branch-and-Bound-Konzept* näher erläutert werden. Dieses stellt eine Optimierungstechnik dar, die den Lösungsraum des Problems durch Abschneiden von Teilpfaden reduziert, die keine bessere Lösung als eine vorgegebene Schranke liefern können. Diese Methode eignet sich besonders für Probleme, bei denen die Anzahl der möglichen Lösungen sehr groß ist, wie es beim globalen Sequenzalignment meistens der Fall ist.

Der entwickelte Algorithmus kombiniert die dynamische Programmierung des Needleman-Wunsch-Algorithmus mit einer Branch-and-Bound-Strategie, um das paarweise Sequenzalignment in möglichst wenigen Schritten zu lösen. Wie auch beim NW-Algorithmus üblich, wird zunächst eine Kostenmatrix mit den Kosten aller möglichen Lösungen erstellt und erst im Anschluss das tatsächliche Alignment mithilfe von Backtracking gefunden. Die Kosten sind hierbei der vorgegebenen Kostenmatrix zu entnehmen, die in der Einleitung erläutert wurde. Um die originale Laufzeitkomplexität des NW-Algorithmus zu verringern, suchen wir nun einen Weg, nicht alle Zellen der Matrix besuchen und berechnen zu müssen. Wir können eine Zelle genau dann ignorieren, wenn wir bereits wissen, dass sowieso schon eine bessere Lösung existiert und dieser Pfad nicht mehr zu einer optimalen Lösung führen kann. Da wir dies aber zu Beginn des Algorithmus nicht wissen können, müssen wir mit sogenannten Heuristiken arbeiten, die also eine Schätzung auf Basis des aktuellen Zustandes abgeben, was die maximalen Kosten zur Lösung sind. Dadurch erhalten wir eine obere Grenze, die aussagt, ab welchen Kosten ein Pfad und damit ein gesamter Teilbaum der Lösung ignoriert werden kann.

Die schwierige Aufgabe des Branch-and-Bound-Algorithmus liegt nun darin, geeignete Methoden zu finden, um gute Heuristiken aufzustellen, die den Suchraum maximal verkleinern, jedoch immer noch eine optimale Lösung garantieren. Hingegen der vorgeschlagenen trivialen Schranke von dem Maximum der Sequenzlängen, scheint der Hamming-Abstand eine bessere Methode zu sein, um Schätzungen abzugeben. Der Hamming-Abstand, oder auch unter Hamming-Distanz bekannt, gibt ein Maß für die Unterschiedlichkeit zweier Zeichenketten an, welches durch den paarweisen Vergleich der einzelnen Symbole ermittelt werden kann. Im Vergleich zu der zuerst genannten Schranke bietet dieses Maß den Vorteil, dass die Schranke um die Anzahl der Matches bereits verringert wird. Dies ist insofern sinnvoll, da der Algorithmus Matches in nahezu allen Fällen auch tatsächlich aligniert. Dadurch ergeben sich die tatsächlich maximalen Kosten, die sich durch die Anzahl der Mismatches oder Gaps ergeben. Da beide in unserem Fall die gleichen Kosten von +1 haben, entsprechen unsere maximal möglichen Kosten also genau der Hamming-Distanz der beiden Zeichenketten. Dies können wir also nutzen, um unsere initiale obere Schranke zu ermitteln.

Da die Heuristik zu Beginn nur sehr grob abschätzen kann, ist ein weiterer Schritt zur Verkleinerung des Suchraumes die dynamische Anpassung der oberen Schranke während der Laufzeit des Algorithmus. Dadurch können wir zu verschiedenen Zeitpunkten unsere Abschätzung neu berechnen und unsere Schranke nach unten anpassen. Dies ist genau dann der Fall, wenn wir die Heuristik zu einem besseren Zeitpunkt berechnen. Die Berechnung erfolgt hierbei exakt wie bei der initialen Bestimmung, jedoch schätzen wir nun lediglich die verbleibenden Kosten von der aktuellen Zelle unserer Kostenmatrix. Wenn wir eine niedrigere obere Schranke gefunden haben, resultiert dies nun in mehreren Möglichkeiten, zukünftige Teilbäume abzuschneiden.

Da dieses Vorgehen theoretisch den maximalen Gewinn genau dann bringt, wenn wir in jeder Iteration die Schranke neu berechnen, wirft sich die Frage auf, wieso dies nicht unbedingt die beste Entscheidung für die Laufzeit des Algorithmus ist. Bei einem Branch-and-Bound ist immer zu beachten, dass der Aufwand für diesen nicht den Aufwand des eigentlichen Basisalgorithmus übersteigt. Wäre dies der Fall, dann würde dieser Ansatz keinen Nutzen mehr bringen. Aus diesem Grund ist es sinnvoll, die Schranke zu bestimmten Zeitpunkten zu aktualisieren. Hierbei ist eine Überlegung, wann eine Aktualisierung vergleichsweise einen höheren Nutzen bringt. Hier greift die Tatsache, dass ein frühes Markieren von Zellen als redundant dazu führt, dass mehr Teillösungen ignoriert werden können. Eine Zelle kann genau dann ignoriert werden, wenn alle eingehenden Kanten bereits als redundant markiert sind, da diese zu keiner optimalen Lösung mehr führen können. Folglich können die zukünftigen Teillösungen auch nicht mehr zu dem optimalen Alignierungsergebnis führen. Aus dieser Beobachtung können wir nun schließen, dass eine frühere Kostenaktualisierung zu einer stärkeren Verkleinerung des Suchraumes führt. Deshalb wurde in diesem Algorithmus die Abtastrate für die Aktualisierung der Schranke abhängig von der aktuellen Iterationsnummer zu der Gesamtiterationsanzahl, welche sich aus dem Produkt der Sequenzlängen n und m ergibt, bestimmt. Hierdurch werden zu Beginn des Algorithmus öfter die verbleibenden Kosten berechnet als später.

4 Ergebnisse

Um die Auswirkungen der Branch-and-Bound-Strategie beispielhaft darzustellen, wurden verschiedene Testszenarien entworfen. Interessant zu untersuchen sind hier folgende Fälle: Einerseits der *Best-Case*, also jener Fall, in dem wir bereits zwei gleiche Eingabesequenzen erhalten. Hier besteht die Lösung aus der Diagonalen unserer Matrix, da wir jedes Symbol miteinander alignieren und damit insgesamt auf Kosten von 0 kommen. Hier sollten also alle Zellen außerhalb der Hauptdiagonalen abgeschnitten werden, was dieser Algorithmus auch erfolgreich unternimmt. Für die Eingabesequenzen *ACGTACGTACGTACGT* und *ACGTACGTACGTACGT* mit jeweils einer Länge von 16 Zeichen ergeben sich also insgesamt 256 Iterationen, um die Kostenmatrix zu befüllen. Mit dem implementierten Algorithmus werden 240 Branch-and-Bounds vollzogen, was genau der Differenz der Gesamtiterationen und der Länge der Hauptdiagonalen entspricht. Im Best-Case werden also genau nur so viele Schritte berechnet, wie nötig und insgesamt 93.75% des Suchraumes abgeschnitten.

Der *Worst-Case* liegt genau dann vor, wenn sich alle Symbole der Eingabesequenzen unterscheiden. Der tatsächliche schlechteste Fall für ein Problem liegt genau dann vor, wenn beide Eingabesequenzen keine Symbole miteinander teilen. So könnten beispielsweise die konstruierten Sequenzen *ABCDEFGHJKL* und *MNOPQRSTUVW*, welche eine Länge von 11 Zeichen aufweisen, als Beispiel dafür dienen, dass Branch-and-Bound keinen Einfluss auf die Laufzeit hat. Während der 121 Iterationen kann kein Teilbaum abgeschnitten werden, da das beste Ergebnis eben genau unserer initial geschätzten Kosten gleicht. Da nun aber besonders in der Biologie ein beschränktes Alphabet vorliegt, aus welchem die Zeichen einer Sequenz bestehen können, und zudem die Sequenzen in der Praxis bedeutend länger sind, tritt dieser Fall nahezu nie ein. In realistischeren Fällen ist es so, dass unsere initial geschätzte Schranke nur in sehr wenigen Fällen gut genug ist, um Teilbäume abzuschneiden. Für Sequenzen *ACGTACGTACGTACGT* und *TGCATGCATGCATGCA*, welche ebenso eine maximale Hamming-Distanz aufweisen, jedoch trotzdem gleiche Symbole besitzen, haben wir einen Lösungsraum von 256 und können davon 38 Teilbäume abschneiden. Dies entspricht einer Einsparung von etwa 14.8% des gesamten Suchraumes. Obwohl bei weitem nicht so effektiv wie in besseren Fällen, so ist dennoch eine Einsparung zu erkennen. Dies liegt darin, dass beide Sequenzen die gleichen Subsequenzen (*ACGT*) besitzen, nur in umgekehrter Reihenfolge. Da das Alphabet hier nur eine Mächtigkeit von 4 besitzt, können 2 Zeichen in jeder Subsequenz gematcht werden.

Um neben den konstruierten Beispielen nun einen realitätsnäheren Testfall zu haben, wurden nun die Sequenzen aus dem Übungsblatt 9 (Globales Alignment) verwendet, lediglich unter der Veränderung unserer neuen und vereinfachten Kostenmatrix. Zu vermerken ist, dass der Vergleich zu den vorherigen Beispielen nicht trivial ist, da sich sowohl die Sequenzlänge als auch die Mächtigkeit des Alphabets unterscheiden. In dem besagten durchschnittlichen Fall

haben wir Sequenzen mit Initiallängen von 745 und 734, was eine Matrix mit 546.830 Zellen und deren jeweiligen Teillösungen ergibt. Unser Branch-and-Bound-Algorithmus kann in diesem Beispiel 70.917 Teillösungen abschneiden, spart also etwa 13% ein. Dies ist weniger als bei unserem konstruierten Worst-Case von 14.8%, was jedoch, wie erwähnt, durch die schwierige Vergleichbarkeit nicht direkt ins Verhältnis gesetzt werden kann.

Um zu überprüfen, wie man die Ergebnisse weiterhin beeinflussen kann, wurde nun versucht, die Frequenz der Schrankenänderung anzupassen. Da diese Abtastrate aktuell dynamisch durch die Modulo-Berechnung in Abhängigkeit von unserem Fortschritt des Algorithmus ermittelt wird, können wir durch die Veränderung des Basis-Modulo's Einfluss nehmen. Aktuell wird diese in Abhängigkeit der Sequenzlängen bestimmt, indem der Basismodulo die Anzahl der Zellen in unserer Matrix durch die größte Sequenzlänge teilt. Wenn wir nun jedoch statt der maximalen Sequenzlänge eine möglichst hohe statische Zahl benutzen, sodass der Basismodulo und damit auch der dynamische Modulo möglichst gering werden, dann lässt sich eine Änderung in den Ergebnissen unseres Average-Cases beobachten. Da der Modulo geringer wird, wodurch die Anpassung der Schranke öfter (im Extremfall in jedem Durchlauf unserer Schleife) geschieht, so können wir bestmöglich 17.1% der Teilpfade abschneiden. Natürlich steigt mit dieser abgeänderten Methode der Aufwand für den gesamten Branch-and-Bound deutlich, wodurch zu hinterfragen ist, ob dieser dann überhaupt noch von Nutzen ist. Parameter wie diese müssen dann eben empirisch so ermittelt werden, sodass eine bestmögliche Balance zwischen Effizienz und Gewinnbringung erreicht werden kann.

5 Diskussion und Ausblick

Die vorgestellten Ergebnisse zeigen, dass der entwickelte Branch-and-Bound-Algorithmus in nahezu allen konstruierten Fällen eine laufzeiteffizientere Alternative zu Methoden wie dem klassischen Needleman-Wunsch-Algorithmus darstellt. Die Kombination von dynamischer Programmierung und Branch-and-Bound-Strategie ermöglicht eine bemerkbare Reduktion des Rechenaufwands von durchschnittlich etwa 15 bis 20%, bei ähnlichen Sequenzen von bis über 50%. Eine wichtige Anmerkung im Zuge dieser Ausarbeitung ist jedoch, dass obwohl eine theoretische Verbesserung der Laufzeiteffizienz erreicht wird, diese in dem vorgestellten Algorithmus nicht tatsächlich realisiert werden kann. Dies ist darin begründet, dass trotz abgeschnittener Teilpfade immer noch alle Zellen besucht und damit keine Schleifendurchgänge gespart werden, obwohl diese keine optimalen Lösungen mehr bieten können. Das Problem liegt jedoch nicht an der Branch-and-Bound-Methode selbst, sondern an der Implementierungslogik. Um dieses Problem zu lösen, müsste die zugrundeliegende Strategie geändert werden, zum Beispiel durch die Verwendung von Graph-basierten Algorithmen, sodass die Verbindung zu nicht benötigten Teillösungen gar nicht erst in Erwägung gezogen werden (tatsächlich abgeschnitten werden können). So kann tatsächlich eine praktische Verbesserung der Laufzeit erreicht werden (Laufzeit in Sekunden), statt nur eine theoretische Verbesserung zu beobachten. Weiterhin ist zu untersuchen, inwiefern der Aufwand zur Bestimmung des Branch-and-Boundings den der regulären Rekursion übertrifft. In dem gezeigten Fall ist es so, dass sich der zusätzliche Aufwand aufteilt. Einserseits kommen in jedem Schleifendurchlauf, also für jede Zelle unserer Matrix, die Aufwände für die Berechnung der dynamischen Aktualisierungsfrequenz der Schranke hinzu. Die tatsächliche Aktualisierung der Schranke wird jedoch seltener durchgeführt. Nur im schlechtesten Fall wird diese ebenfalls eine Komplexität von $O(n^2)$ besitzen, da der dynamische Modulo den Fallback-Wert von 1 erhält, falls der Modulo unter 1 fällt. In diesen Szenarien wird die Verwendung eines Branch-and-Bounds selten große Vorteile bringen, da der Mehraufwand zu stark ansteigt. Wann es jedoch sinnvoll ist, diese Optimierungsmethodik anzuwenden, hängt stark von der vorliegenden Problemstellung sowie der konkreten Implementierung des Branch-and-Bounds ab. Der vorgestellte Algorithmus bietet jedoch eine Basis, um eine effizientere Alignierung zweier Sequenzen im durchschnittlichen Fall zu ermöglichen. Zukünftige Arbeiten könnten sich auf die Entwicklung von verbesserten Schranken konzentrieren, die den Suchraum noch besser einschränken können. Darüber hinaus könnte der Algorithmus auf lokale Alignments oder multiple Sequenzalignments erweitert werden.