Protein Threading

im Rahmen des Seminars „Optimierungsprobleme in der Bioinformatik“
unter Leitung von Prof. Dr. Gerhard Reinelt, Dr. Marcus Oswald und Hanna Seitz,
Sommersemester 2005, Universität Heidelberg

Hans Joachim Ferreau
ferreau@urz.uni-heidelberg.de

20.04.2005

Inhaltsverzeichnis

1 Biologischer Hintergrund

1.1 Proteine

Proteine sind die wichtigsten Substanzen in allen lebenden Organismen. Sie dienen als Gerüstmaterial zum Aufbau von Geweben und Organen, sind Träger der Immunabwehr, steuern Nervenimpulse und vieles mehr. Entsprechend weitreichend sind die medizinischen Anwendungen, falls ihre genauen Strukturen und Funktionsweisen ermittelt werden können. Die vollständige Sequenzierung des menschlichen Erbguts im Rahmen des Human-Genom-Projekts bedeutete einen wichtigen Schritt hin zur Verwirklichung dieses ergeizigen Ziels. Die dabei gewonnen Daten müssen nun ausgewertet werden; das in diesem Vortrag behandelte *Protein Threading* leistet dazu einen Beitrag.

Ein Protein ist eine unverzweigte Kette von *Aminosäuren*. Jede Aminosäure ist ein ca. 10-30 Atome umfassendes Molekül, das zwei weitere (beliebige) Aminosäuren an sich binden kann. Ordnet man den an der Proteinbildung beteiligten 20 verschiedenen Aminosäuren Buchstaben zu, so lässt sich jedes Protein als Wort dieses Amino-Alphabets schreiben – die sog. Primärstruktur (Abb. 1). Die Zahl der Aminosäuren pro Proteien schwankt zwischen knapp 100 und 5 000, wobei 300 ein typischer Zwischenwert ist (vgl. [11]).

```
KDIQLLNVSYDPTRELYEQYNKAFSAHWKQETGDNVVIDQSHGGSGKQATSVINGIEADTVTLALAYDVNAIAERGRI
DKNWIKRLPDDSAPYTSTIVFLVRKGNPKQIHDWNDLIKPGVSVITPNPKSSGGARWNYLAAWGYALHHNNNDQAKAE
DFVKALFKNVEVLDSGARGSTNTFVERGIGDVLIAWENEALLATNELGKDKFEIVTPSESILAEPTVSVVDKVVEKKD
TKAVAEAYLKYLYSPEGQEIAAKNFYRPRDADVAKKYDDAFPKLKLFTIDEVFGGWAKAQKDHFADGGTFDQISKR
```

Abbildung 1: Die 310 Aminosäuren umfassende Primärstruktur des Proteins 1SBP (Quelle: [10])

Anzahl, Art und Reihenfolge der Aminosäuren sind für jedes Protein durch ein Gen in der DNA kodiert. Mit der Identifizierung eines Gens ist somit die eindimensionale Primärstruktur eines Proteins bekannt – in [10] sind derzeit rund 30 000 aufgeführt. Entscheidend für die biologische Funktion eines Proteins ist jedoch dessen *dreidimensionale Struktur*. Sie entsteht durch Wechselwirkungen der Atome innerhalb des Proteins – hauptsächlich in Form von Wasserstoffbrückenbindungen und elektrostatischer Kräfte – und ist durch die Primärstruktur bestimmt. Das Ermitteln dieser dreidimensionalen Struktur anhand einer bekannten Sequenz von Aminosäuren wird als *Proteinfaltung* bezeichnet.

1.2 Protein Threading

Zur Bestimmung der dreidimensionalen Struktur eines Proteins stehen zwei physikalische Methoden zur Verfügung: Die Röntgen-Strukturanalyse und die NMR-Spektroskopie. Da beide jedoch sehr zeitaufwendig sind (laut [12] im Bereich von Monaten pro Struktur), konnten sie erst auf einen Bruchteil der bekannten Proteine angewendet werden. Daher liegt es nahe, Computer zur Lösung des Problems einzusetzen. Auf Grund der Vielzahl der beteiligten Atome (zusammen mit umgebenden Wassermolekülen ergeben sich Zahlen von bis zu über einer Million), sind direkte Methoden, die die Gesetze der (Quanten-)Mechanik simulieren, auch für Supercomputer wie IBMs Blue Gene bisher nur für kleine Proteine praktikabel.

Daher geht man beim *Protein Threading* einen anderen Weg: Im Laufe der Evolution haben sich innerhalb der Amino-Sequenzen zwei Arten von besonders stabilen *Kernregionen*, auch Sekundärstrukturen genannt, herausgebildet (für weitere Details siehe z. B. [5]):

1. α-*Helices*: 5 bis 40 benachbarte Aminosäuren verbinden sich (neben ihrer linearen Anordnung) zusätzlich so untereinander, dass sie eine rechtsläufigen Spiralstruktur bilden.

2. β-*Blätter*: Mehrere entfernte Abschnitte der Amino-Sequenz, sog. – etwa 5-10 Aminosäuren lange – β-*Stränge*, verknüpfen sich nebeneinander zu einer Faltblattstruktur.

α-Helices und β-Stränge werden durch *Schleifen* verbunden. Dies sind kurze Abschnitte der Amino-Kette, die keine signifikanten Wechselwirkungen mit anderen Bereichen zeigen. Die Gesamtheit aller Kernregionen eines Proteins bezeichnet man als *Kern* oder auch als Tertiärstruktur des Proteins.
Aus den physikalisch bestimmten Strukturen lassen sich *statistische Daten* über das Vorkommen der 20 verschiedenen Aminosäuren in den beiden Kernregionen ableiten. Ferner weiß man, dass sich hydrophobe (also Wasser meidende) Aminosäuren bevorzugt in den vorwiegend im Innern gelegenen Kernregionen aufhalten, während hydrophile die meist nach außen gewundenen Schleifen präferieren. Zu einem gegebenen Kern

(d. h. einer Abfolge und räumlichen Anordnung von Kernregionen) kann man so nach einer *Ausrichtung* der Amino-Sequenz suchen, die diese Vorgaben *am wahrscheinlichsten* erfüllt. Da die Sequenz dabei natürlich nicht zerrissen werden darf, spricht man davon, dass sie in den Kern *eingefädelt* wird und daher vom *Protein Threading* (Abb. 2).

Diese Methode setzt natürlich voraus, dass man sämtliche Kernregionen des Proteins bereits kennt. Dies erscheint zunächst paradox, da ja gerade die Bestimmung dieser Struktur der wesentliche Teil des Ziel ist. Zur Erklärung brauchen wir eine weitere biologische Beobachtung: Im Laufe der Evolution haben sich nicht nur Kernregionen herausgebildet, sondern auch ganze Kerne blieben bei verschiedenen Proteinen erhalten. Schätzungen gehen davon aus, dass jedes der vielen Millionen Proteine, die in allen Lebewesen vermutet werden (davon ca. 100 000 beim Menschen), genau eine von nur etwa 1 000 bis 8 000 verschiedenen Kernstrukturen annimmt (vgl. [8], [9]). Kennt man alle diese *Kernschablonen* – wovon wir ab sofort ausgehen –, so kann man wie folgt vorgehen:

1. Bestimme zu jeder Schablone die wahrscheinlichste Ausrichtung der Amino-Sequenz.
2. Wähle aus diesen wahrscheinlichsten Ausrichtungen die „beste“ aus und erhalte damit die gesuchte dreidimensionale Struktur des Proteins (oder zumindest eine gute Näherung).

Dies wirft natürlich die Frage auf, wie eine geeignete *Zielfunktion* konstruiert werden kann, mit der die Ausrichtungen bewertet und verglichen werden können. Im Rahmen dieses Vortrags können wir dieses Problem nur kurz streifen (siehe Abschnitt 2.3). Vielmehr werden wir verschiedene Möglichkeiten vorstellen, wie man zu einer gegebenen Zielfunktion und *einer* gegebenen Kernschablone die optimale Ausrichtung der Amino-Sequenz bestimmt. Dabei werden wir uns hauptsächlich den in [2] beschriebenen Formulierungen aus dem Gebiet der gemischt-ganzzahligen Programmierung zuwenden.

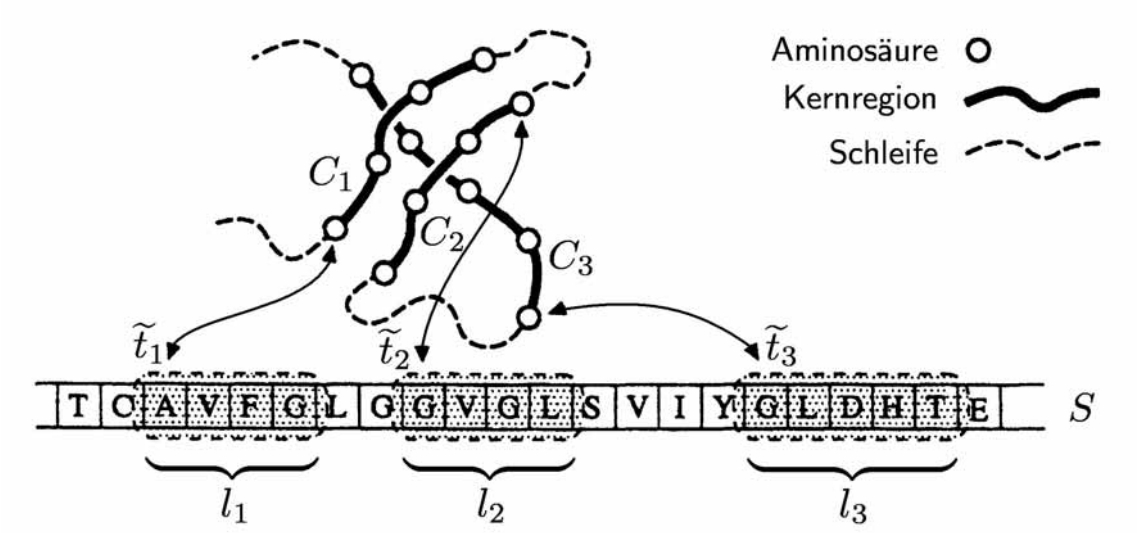

Abbildung 2: Die Amino-Sequenz wird in eine Kernschablone eingefädelt (aus [1], nachbearbeitet).

2 Modellierung

2.1 Formale Problembeschreibung

Wir führen zunächst einige Bezeichnungen ein (siehe auch Abb. 2):

- C sei die zu überprüfende Kernschablone. $C := (C_1, C_2, \ldots, C_m)$ ist eine geordnete Menge von $m \in \mathbb{N}$ Kernregionen (α-Helix oder β-Strang) der Länge $l_i \in \mathbb{N}$, $i = 1, \ldots, m$;
- Σ bezeichne das 20 Zeichen umfassende Amino-Alphabet, also $|\Sigma| = 20$;
- $S \in \Sigma^{\widetilde{n}}$ stehe für die einzufädelnde, $\widetilde{n} \in \mathbb{N}$ Säuren lange Amino-Sequenz;
- $\widetilde{t}_i \in \mathbb{N}$, $\widetilde{t}_i \leq \widetilde{n}$, für $i = 1, \ldots, m$ sei die absolute Position innerhalb der Amino-Sequenz, an der für eine bestimmte Ausrichtung die Kernregion C_i beginnt.

Wenn wir davon sprechen, dass die Kernregion C_i an der Stelle $\widetilde{t}_i$ innerhalb von S beginnt, so soll dies implizieren, dass auch die Positionen $\widetilde{t}_{i+1}, \ldots, \widetilde{t}_{i+l_i-1}$ von dieser Kernregion belegt werden. Insbesondere sind also Lücken oder Einfügungen innerhalb der Kernregionen ausgeschlossen.

Man spricht von einer *gültigen Ausrichtung* der Amino-Sequenz S an einer Kernschablone C, falls

1. $1 \leq t_1$, $t_m \leq \widetilde{n} + 1 - l_m$ und
2. $t_i + l_i \leq t_{i+1}$ für $i = 1, \ldots, m-1$ gilt.

Für eine gültige Ausrichtung fordern wir also, dass sich die Kernregionen nicht überlappen und ihre Reihenfolge eingehalten wird. Manchmal ist es sinnvoll, für die Länge einer – zwei Kernregionen verbindenden – Schleife eine Mindestlänge vorzugeben, z. B. wenn die Enden der Kernregionen einen gewissen räumlichen Abstand haben, der überbrückt werden muss. Wir verzichten jedoch auf eine explizite Einführung dieser Größen, da es in unserem Modell genügt, die Längen l_i um diese Mindestlängen zu erhöhen.

Des Weiteren führen wir die *relativen* Positionen t_i und die Anzahl der Freiheitsgrade (wenigstens für die erste Kernregion) n ein:

$$t_i := \widetilde{t}_i - \sum_{j=1}^{i-1} l_i \quad \text{und} \quad n := \widetilde{n} + 1 - \sum_{i=1}^{m} l_i \,. \tag{2.1}$$

Man sieht dann leicht, dass eine Ausrichtung genau dann gültig ist, falls für die relativen Positionen

$$1 \leq t_1 \leq t_2 \leq \ldots \leq t_m \leq n \tag{2.2}$$

gilt (ein Gleichheitszeichen bedeutet, dass die betreffenden Kernregionen ohne Schleife verbunden sind).

Als letztes benötigen wir noch die bereits erwähnte *Zielfunktion*, welche für die vorgegebene Sequenz festgelegt wird. Sie soll zwei Arten von Informationen liefern:

1. $c_{ik} \in \mathbb{R}$ für $i = 1, \ldots, m$ und $k = 1, \ldots, n$ sei eine Bewertung (je kleiner umso besser) dafür, dass Kernregion C_i an die k. relative Position platziert wird. Bedeutet diese Platzierung etwa, dass die Aminosäuren dieser Kernregion eine α-Helix bilden, obwohl sie bevorzugt in Schleifen oder β-Strängen vorkommen, so würde man c_{ik} einen hohen Wert geben.
2. c_{ikjl} für $(i,j) \in L$ und $1 \leq k \leq l \leq n$ bewerte die *paarweisen Interaktionen* zweier Kernregionen. Wird also Kernregion C_i an der (relativen) Stelle k positioniert und Kernregion C_j an der (relativen) Stelle l, so gibt c_{ikjl} eine Bewertung dafür, ob es wahrscheinlich ist, dass sich die betreffenden Aminosäuren in dieser räumlichen Konstellation befinden. $L \subseteq \{(i,j) \mid 1 \leq i < j \leq m\}$ bezeichne dabei die Menge aller Kernregion-Paare, zwischen denen – etwa auf Grund räumlicher Nähe – eine Interaktion besteht.

Das *Protein Threading Problem (PTP)* kann nun wie folgt formuliert werden:

$$\min_{t_1, t_2, \ldots, t_m} \left\{ \sum_{i=1}^{m} c_{it_i} + \sum_{(i,j) \in L} c_{it_ijt_j} \;\middle|\; 1 \leq t_1 \leq t_2 \leq \ldots \leq t_m \leq n \right\} . \tag{2.3}$$

2.2 Netzwerk-Fluss-Modell

Als Grundlage für die angestrebten MIP-Formulierungen führen wir nun das in [2] vorgestellte Netzwerk-Fluss-Modell für das PTP ein (Abb. 3). Dazu konstruieren wir folgenden gerichteten Graphen $G = (V, E)$:

$$\begin{aligned}
V &:= \big\{(i,k) \;\big|\; i = 1, \ldots, m \;\wedge\; k = 1, \ldots, n\big\} \cup \{s, t\} \,, \\
E_L &:= \big\{\big((i,k),(j,l)\big) \;\big|\; (i,j) \in L \;\wedge\; 1 \leq k \leq l \leq n\big\} \,, \\
E_x &:= \bigcup_{1 \leq k \leq l \leq n} \big\{\big((i,k),(i+1,l)\big) \;\big|\; i = 1, \ldots, m-1\big\} \\
&\quad \cup \big\{\big(s,(1,k)\big) \;\big|\; k = 1, \ldots, n\big\} \cup \big\{\big((m,k),t\big) \;\big|\; k = 1, \ldots, n\big\} \,, \\
E &:= E_L \cup E_x \,.
\end{aligned}$$

Ein Knoten (i, k) repräsentiert das Positionieren der Kernregion C_i an die k. relative Position. Damit entsprechen die (s, t)-Wege im Graphen $G_x := (V, E_x)$ genau den gültigen Ausrichtungen: Dem Weg $\big(s, (1, t_1), (2, t_2), \ldots, (m, t_m), t\big)$ entspricht die Ausrichtung der Kernregionen an den relativen Positionen $t_1, t_2, \ldots, t_m$ und umgekehrt. Die Kanten E_L repräsentieren die paarweisen Interaktionen zwischen den Kernregionen, wobei wir mit $E_z := E_L \setminus E_x$ die Interaktionen *nichtbenachbarter* Kernregionen bezeichnen.

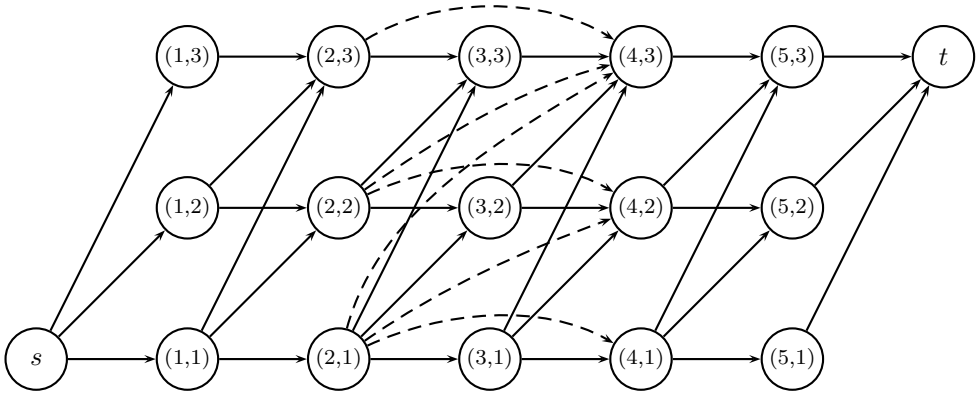

Abbildung 3: Beispiel für das Netzwerk-Fluss-Modell des PTP: Die Kanten E_x sind durchgehend, die Kanten E_z gestrichelt dargestellt (nach [2]).

Den Kanten des Graphens G ordnen wir nun entsprechend unserer Zielfunktion Kosten zu, und zwar

- den Kanten $\big(s, (1, k)\big) \in E_x$ die Kosten 0,
- den Kanten $\big((m, k), t\big) \in E_x$ die Bewertung c_{mk},
- den Kanten $\big((i, k), (j, l)\big) \in E_x$ die Summe
  - der Bewertung c_{ik} und
  - der Bewertung c_{ikjl} (0 falls $(i, j) \notin L$),
- den Kanten $\big((i, k), (j, l)\big) \in E_z$ die Bewertung c_{ijkl}.

Wir sprechen davon, dass ein (s, t)-Weg die Kante $\big((i, k), (j, l)\big) \in E_z$ *aktiviert*, falls sowohl (i, k) als auch (j, l) auf diesem Weg liegen. Ein (s, t)-Weg zusammen mit den von ihm aktivierten Kanten wird *augmentierter Weg* genannt. Das PTP besteht also darin, in unserem Netzwerk einen kürzesten (d. h. mit minimalen Gesamtkosten) augmentierten Weg zu finden.

2.3 Bemerkungen zur Zielfunktion

Wie bereits in der Einleitung erwähnt, werden zur Ermittlung der Zielfunktion statistische Verfahren auf die bereits bekannten Proteinstrukturen angewendet. Dabei werden Informationen über einzelne Aminosäuren und deren Interaktion benutzt. Für eine *feste* Sequenz erhält man die Zielfunktionskoeffizienten, indem man für jede Kernregion die entsprechenden Werte der beteiligten Aminosäuren aufsummiert. Dies wird *einmal* am Anfang erledigt und die Daten für alle möglichen Positionen der Kernregionen in einem großen Array gespeichert, damit auf sie in konstanter Zeit zugegriffen werden kann (vgl. [7]).

Des weiteren soll in dieser Stelle noch einmal betont werden, dass die in diesem Vortrag behandelte Verfahren nur eine optimale Ausrichtung einer Amino-Sequenz an *einer* Kernschablone liefern. Um die tatsächliche Struktur des Proteins zu erhalten, werden die Verfahren auf *jede* bekannte Kernschablone angewendet. Die Zielfunktionswerte werden dann verglichen und die Kernstruktur mit der niedrigsten Bewertung ist die wahrscheinlichste. Dies ist jedoch nur möglich wenn die Bewertungen vorher *normalisiert* werden. So verändert etwa allein die Anzahl der Kernregionen einer Kernschablone den Zielfunktionswert. Ein Verfahren um derartige Effekte auszugleichen wird in [9] beschrieben.

Ein weiterer Aspekt ist das sog. *Selbsteinfädeln (self-threading)*. Dafür nimmt man den Kern einer bekannten Proteinstruktur als Schablone und richtet die zugehörige Amino-Sequenz an ihm aus. Dabei sollte sich natürlich (normalisiert) ein niedrigerer Zielfunktionswert als bei allen anderen Schablonen ergeben. Auf diese Art kann die Zielfunktion und ihre Normalisierung geeicht werden.

2.4 Komplexitätsbetrachtungen

Mit Hilfe des Netzwerk-Fluss-Modells erkennt man sofort, dass das Problem effizient gelöst werden kann, falls man die paarweisen Interaktionen nichtbenachbarter Kernregionen ignoriert. In diesem Falle besteht das Problem einfach im Auffinden eines kürzesten Weges in G_x. Da das Netzwerk ein azyklischer Digraph ist, kann dies mit Hilfe einer topologischen Sortierung der Knoten in $\mathcal{O}(E_x) = \mathcal{O}(mn^2)$ realisiert werden (vgl. [3]). Ebenfalls polynomiale Laufzeit erhält man, wenn man für die Schleifen eine feste Länge vorgibt. Diese Vereinfachungen sind jedoch biologisch meist nicht gerechtfertigt.

Das PTP in der von uns betrachteten Form (also mit beliebigen paarweisen Interaktionen) ist $\mathcal{NP}$-schwer – und das selbst bei Außerachtlassung der Codierungslänge der verwendeten Zahlen. Es wurde nämlich gezeigt, dass das zugehörige Entscheidungsproblem zum Lösen einer Variante des Erfüllbarkeitsproblems (namentlich ONE-IN-THREE 3SAT) benutzt werden kann und somit $\mathcal{NP}$-vollständig ist (siehe [6] und [7]). Überdies wurde in [1] gezeigt, dass das PTP sogar $\mathcal{MAX}$-$\mathcal{SNP}$-schwer ist. Dies bedeutet, dass sich die optimale Lösung in polynomialer Zeit nicht beliebig genau approximieren lässt, falls $\mathcal{P} \neq \mathcal{NP}$ gilt. Dennoch ist die Lage nicht hoffnungslos, da wir bei den praktischen Tests (Abschnitt 3.6) sehen werden, dass sich die meisten Proteine in *polynomialer* Zeit einfädeln lassen (vgl. auch [14]). Dieses Phänomen ist zum einem theoretisch noch nicht verstanden, zum anderen hängt es entscheidend von einer geschickten MIP-Formulierung des Problems ab.

3 MIP-Formulierungen

3.1 Kürzeste-Wege-Formulierungen

Zunächst formulieren wir die Nebenbedingungen für ein Polytop, dessen Ecken genau zu (s,t)-Wegen korrespondieren. Dabei drücken die x-Variablen (x_e, $e \in E_x$) den Fluss im Netzwerk über die entsprechenden Kanten aus. Weiter bezeichne $\Gamma(v)$ die Menge aller Kanten mit Anfangsknoten v und $\Gamma^{-1}(v)$ die Menge aller Kanten mit Endknoten v.

Polytop X:

$$\sum_{e \in \Gamma(s)} x_e = 1 \tag{3.1}$$

$$\sum_{e \in \Gamma^{-1}(t)} x_e = 1 \tag{3.2}$$

$$\sum_{e \in \Gamma(v)} x_e - \sum_{e \in \Gamma^{-1}(v)} x_e = 0 \quad \forall\, v \in V \setminus \{s,t\} \tag{3.3}$$

$$x_e \geq 0 \quad \forall\, x_e \in E_x \tag{3.4}$$

Die Bedingungen (3.1) und (3.2) lassen genau eine Einheit im Netzwerk vom Knoten s zum Knoten t fließen. Die Flusserhaltungsbedingungen (3.3) verhindern Quellen bzw. Senken innerhalb des Netzwerkes. Da das Polytop nur ganzzahlige Ecken besitzt (vgl. [3]), können die Ganzzahligkeitsbedingungen $x_e \in \{0,1\}$ zu der Ungleichung (3.4) relaxiert werden.

Als zweites beschreiben wir eine Menge ganzzahliger Punkte, die genau zu den (s,t)-Wegen korrespondieren. Dazu führen wir Binärvariablen y_{ik} ein, die genau dann 1 sind, falls die Kernregion C_i der relativen Position k zugewiesen wird.

Menge ganzzahliger Punkte Y:

$$\sum_{k=1}^{n} y_{ik} = 1 \quad \forall\, i = 1, \ldots, m \tag{3.5}$$

$$\sum_{l=1}^{k} y_{il} - \sum_{l=1}^{k} y_{i+1,l} \geq 0 \quad \forall\, i = 1, \ldots, m-1 \wedge k = 1, \ldots, n-1 \tag{3.6}$$

$$y_{ik} \in \{0,1\} \quad \forall\, i = 1, \ldots, m \wedge k = 1, \ldots, n \tag{3.7}$$

Die Gleichungen (3.5) stellen sicher, dass jede Kernregion genau eine Position zugewiesen bekommt. Über die Einhaltung der Reihenfolge wachen die Ungleichungen (3.6): Wird C_{i+1} der Position k zugewiesen, so muss auch C_i bis zur k. Position platziert werden.

Ausgehend von einer der beiden Mengen müssen nun die Interaktionskanten nichtbenachbarter Kernregionen (in Abb. 3 gestrichelt dargestellt) mit Hilfe von Variablen $z_e = z_{ikjl}$, $e \in E_z$, aktiviert werden.

3.2 Nichtlineare Formulierung

Eine naheliegende PTP-Formulierung ist

$$\begin{aligned} \min_{y} \quad & \sum_{i=1}^{m}\sum_{k=1}^{n} c_{ik}y_{ik} + \sum_{(i,j)\in L}\sum_{k=1}^{n}\sum_{l=k}^{n} c_{ik}y_{ik}y_{jl} \\ s.t. \quad & y \in Y\,. \end{aligned}$$

Wir linearisieren sie, indem wir das Produkt $y_{ik}y_{jl} = \min\{y_{ik}, y_{jl}\}$ für alle $(i,k) \in L$ und $1 \le k \le l \le n$ durch Variablen z_{ikjl} ersetzen und an diese die zusätzlichen Bedingungen

$$z_{ikjl} \le y_{ik}\,, \quad z_{ikjl} \le y_{jl}\,, \quad z_{ikjl} \ge y_{ik} + y_{jl} - 1\,, \quad 0 \le z_{ikjl} \le 1 \tag{3.8}$$

stellen. Eine einfache Fallunterscheidung zeigt, dass die Gleichungen das Gewünschte leisten und auch die Ganzzahligkeit von z aus der von y folgt. Praktische Tests in [14] zeigen jedoch, dass diese MIP-Formulierung ineffizient ist, da ihre LP-Relaxierung zu schwache Schranken liefert.

3.3 MXZ-Formulierung

Wir führen zunächst noch ein paar Bezeichnungen ein:

- $A := \{(i,j) \in L \mid j = i+1\}$ sei die Menge von *benachbarten* Kernregionen, zwischen denen eine paarweise Interaktion besteht;
- $R := L \setminus A$ bezeichne entsprechend die Menge aller *nichtbenachbarten* Kernregionen, zwischen denen eine paarweise Interaktion besteht;
- $R_C := \{i \mid (i,j) \in R\} \cup \{j \mid (i,j) \in R\}$ stehe für die Kernregionen, die an *nichtbenachbarten*, paarweisen Interaktionen beteiligt sind.

Nun kommen wir zu einer direkten Umsetzung des Netzwerk-Fluss-Models:

$$\begin{aligned} \min_{x,\,z} \quad & \sum_{e\in E_x} c_e x_e + \sum_{e\in E_z} c_e z_e \\ s.t. \quad & \sum_{1\le k\le l\le n} z_{ikjl} = 1 && \forall\,(i,j) \in R && (3.9)\\ & z_{ikjl} \le \sum_{e\in\Gamma(i,k)} x_e && \forall\,\big((i,k),(j,l)\big) \in E_z && (3.10)\\ & z_{ikjl} \le \sum_{e\in\Gamma^{-1}(j,l)} x_e && \forall\,\big((i,k),(j,l)\big) \in E_z && (3.11)\\ & x \in X && && (3.12)\\ & z_e \in \{0,1\} && \forall\, e \in E_z\,. && (3.13) \end{aligned}$$

Ausgehend von nichtaugmentierten kürzesten Wegen werden die Kanten aus E_z folgendermaßen aktiviert: Für jedes Kernregionen-Paar $(i,j) \in R$ wird genau eine z-Kante berücksichtigt (Gleichungen (3.9)). Ferner wird eine Kante z_{ikjl} höchstens dann aktiviert wird, falls sowohl durch ihren Anfangsknoten (i,k) als auch ihren Endknoten (j,l) ein Fluss über x-Kanten fließt (Ungleichungen (3.10) und (3.11)). In diesem Falle muss dann sogar der gesamte von s losgeschickte Einheitsfluss (Gleichung (3.1)) über (i,k) und (j,l) fließen. Die Eigenschaften des Polytops X garantieren daher auch die Ganzzahligkeit von x.

Man sieht leicht ein, dass auch umgekehrt die Ganzzahligkeit der z-Variablen aus der der x-Variablen folgen würde. Bezeichnet man die Anzahl der möglichen Festlegungen von x bzw. z mit N_x bzw. N_z, so zeigt die – mit elementarer Kombinatorik gewonnenen – Ungleichung

$$N_z = \binom{|R_C| + n - 1}{|R_C|} \leq \binom{m + n - 1}{m} = N_x \,, \tag{3.14}$$

dass der Suchraum der z-Variablen nicht größer als der der x-Variablen ist. Daher macht es Sinn, die Ganzzahligkeit für die z-Variablen zu fordern.

3.4 MXYZ-Formulierung

Wir fügen der MXZ-Formulierung nun y-Variablen für die Knoten hinzu:

$$\begin{aligned}
\min_{x,\,y,\,z} \quad & \sum_{e \in E_x} c_e x_e + \sum_{e \in E_z} c_e z_e \\
s.\,t. \quad & \sum_{k=1}^{n} y_{ik} = 1 && \forall\, i \in R_C && (3.15)\\
& y_{ik} = \sum_{e \in \Gamma(i,k)} x_e && \forall\, i \in R_C \wedge k = 1, \ldots, n && (3.16)\\
& y_{ik} = \sum_{l=k}^{n} z_{ikjl} && \forall\, (i,j) \in R \wedge k = 1, \ldots, n && (3.17)\\
& y_{jl} = \sum_{k=1}^{l} z_{ikjl} && \forall\, (i,j) \in R \wedge l = 1, \ldots, n && (3.18)\\
& y_{ik} \in \{0,1\} && \forall\, i \in R_C \wedge k = 1, \ldots, n && (3.19)\\
& x \in X && && (3.20)\\
& z_e \geq 0 && \forall\, e \in E_z \,. && (3.21)
\end{aligned}$$

Wieder gehen wir von einem nichtaugmentierten kürzesten Weg aus. Für jede Kernregion aus R_C wird genau eine Position aktiviert (Gleichungen (3.15)) und zwar die, durch die der Einheitsfluss fließt (Gleichungen (3.16)). z-Kanten werden nur berücksichtigt, falls sowohl Anfangs- als auch Endknoten aktiviert wurden (Gleichungen (3.17) und (3.18)). Auch hier lässt sich die Ganzzahligkeit von x und z leicht aus der von y folgern.

Die Verbesserungen dieser Formulierung sind signifikant:

1. Man kann zeigen, dass die LP-Relaxierung von MXYZ näher an der konvexen Hülle aller ganzzahligen Lösungen liegt als die von MXZ.
2. Bei gleicher Größe des Suchraums wird die Zahl der Binärvariablen von $|R|\, n(n+1)$ (in der MXZ-Formulierung) auf $|R_C|\, n$ reduziert; man beachte dabei die Ungleichung $|R_C| \leq 2\,|R|$.

3.5 MYZ-Formulierung

Falls wir die y-Variablen nicht nur für die Positionen von Kernregionen aus R_C einführen, sondern für alle Knoten unseres Netzwerkes, so können wir auf die x-Variablen vollständig verzichten:

$$\begin{aligned}
\min_{y,\,z} \quad & \sum_{i=1}^{m} \sum_{k=1}^{n} c_{ik} y_{ik} + \sum_{e \in E_L} c_e z_e \\
s.\,t. \quad & y_{ik} = \sum_{l=k}^{n} z_{ikjl} && \forall\, (i,j) \in L \wedge k = 1, \ldots, n && (3.22)\\
& y_{jl} = \sum_{k=1}^{l} z_{ikjl} && \forall\, (i,j) \in L \wedge l = 1, \ldots, n && (3.23)\\
& y \in Y && && (3.24)\\
& z_e \geq 0 && \forall\, e \in E_L \,. && (3.25)
\end{aligned}$$

Die y-Variablen stammen aus der Menge Y und definieren somit einen (s,t)-Fluss. Die z-Variablen werden analog zur MXYZ-Formulierung aktiviert (Gleichungen (3.22) und (3.23)).

Die Veränderungen in der Zahl der Variablen und der Gleichungen gegenüber der MXYZ-Formulierung hängen entscheidend vom realen Problem ab. Und zwar davon, welcher Anteil der paarweisen Interaktionen auf benachbarte Kernregionen entfällt. [2] spricht von etwa 10% Reduzierung der Variablenzahl und etwa 10% Steigerung der Anzahl der Nebenbedingungen.

3.6 Praktischer Vergleich

Nun sollen die vier vorgestellten MIP-Formulierungen anhand realer Beispiele verglichen werden. In [14] wird dargelegt, dass die nichtlineare und die MXZ-Formulierung bereits bei kleinen Testproblemen teilweise um drei bis vier Größenordnungen langsamer sind als die MXYZ-Formulierung. Daher wollen wir uns darauf beschränken, die MXYZ- mit der MYZ-Formulierung zu vegleichen. Die vorgestellte, repräsentative Testauswahl ist aus [2] entnommen – genauso wie sämtliche MIP-Formulierungen aus Abschnitt 3. Sie wurde auf einem Pentium-PC mit 2,4 GHz und 4 GByte RAM mittels des LP-Lösers CPLEX 7.1 ausgeführt.

Angesichts der erwähnten Komplexitätsresultate ergibt sich ein überraschendes und zugleich praktisch sehr bedeutsames Ergebnis: Die LP-Relaxierung beider Modelle liefert in ca. 95% der Fälle eine *ganzzahlige Lösung*, welche somit direkt zu einer gültigen also *optimalen Ausrichtung* korrespondiert. Kurioserweise zeigt sich dieser Effekt nur bei Zielfunktionen mit realem molekularbiologischen Hintergrund. Werden die Koeffizienten der Zielfunktionen dagegen zufällig gewählt, macht sich die $\mathcal{NP}$-Härte des PTP voll bemerkbar. Gleiches geschieht, wenn in den Formulierungen Gleichungen zusammengefasst werden. Abbildung 4 zeigt die Laufzeiten einiger Probleme, bei denen die LP-Relaxierung eine ganzzahlige Lösung liefert. Dabei ist jeweils die Sequenzlänge $\widetilde{n}$ (oben) und die Größe des Suchraums N_x (unten) gemäß Gleichung (3.14) angegeben.

In Fällen, in denen die LP-Relaxierung eine *gebrochene* Lösung lieferte, wurde die optimale Ausrichtung mit einem Branch&Bound-Ansatz ermittelt: Neben den Werten 0 und 1 nahmen wenige Variablen den Wert $0{,}5$ an, welche dann zum Verzweigen innerhalb des Branch&Bound-Baumes benutzt wurden. Dieser umfasste jedoch nie mehr als elf Knoten, meist sogar nur zwei. Außerdem lag der Zielfunktionswert der LP-Relaxierung stets nur geringfügig unterhalb des Wertes einer optimalen Ausrichtung. Die Abweichungen waren sogar so gering, dass oft ein Vergleich der optimalen Ausrichtungen an verschiedenen Kernschablonen auf Grundlage der LP-Relaxierung möglich ist. Abbildung 5 zeigt Beispiele mit gebrochener Lösung der LP-Relaxierung (Bezeichnung wie oben beschrieben). Ihr entnimmt man auch, dass die meiste Zeit für das Lösen der anfänglichen LP-Relaxierung benötigt wurde.

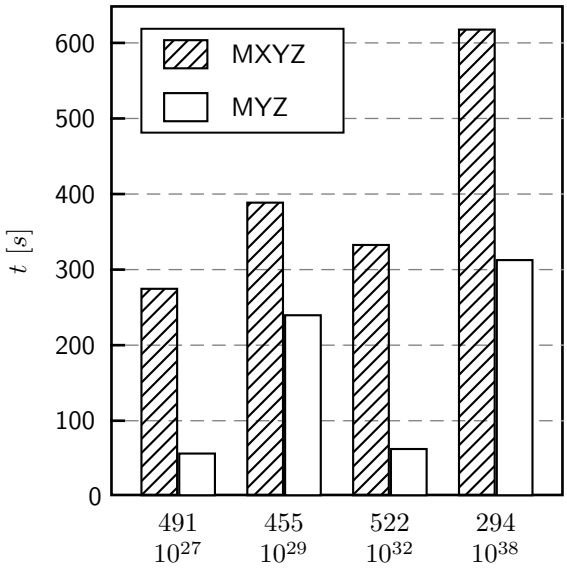

Abbildung 4: Laufzeitvergleich im Falle einer *ganzzahligen* Lösung der LP-Relaxierung (Daten aus [2]).

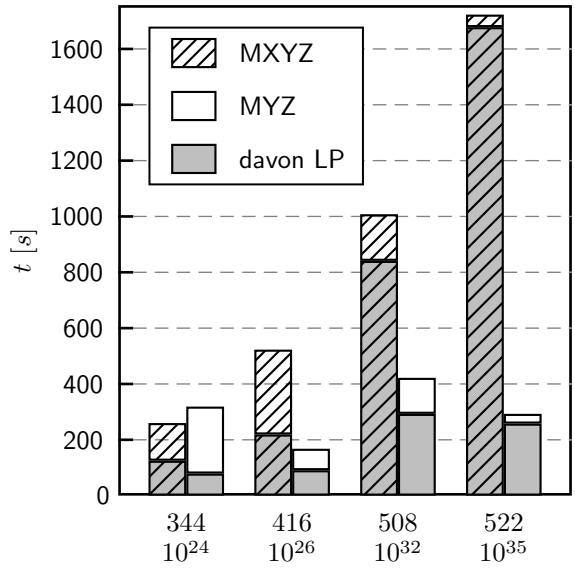

Abbildung 5: Laufzeitvergleich im Falle einer *gebrochenen* Lösung der LP-Relaxierung (Daten aus [2]).

Diese Resultate motivieren die Aussage, dass die Menge der real vorkommenden PTP in *polynomialer* Zeit lösbar ist (so wie das unterliegende LP). Neben der Ähnlichkeit der Zielfunktion ist dies ist eine weitere Parallele zum unkapazitierten Standortproblem (UFLP), dessen sog. starke LP-Relaxierung ebenfalls sehr häufig ganzzahlige Lösungen liefert (vgl. [4]).

4 Zusätzliche Lösungsstrategien

4.1 Divide et Impera

Wie wir gesehen haben, liefert bereits die LP-Relaxierung fast immer optimale Ergebnisse. Sie zu lösen kann wegen der enormen Größe der Probleme jedoch sehr schwierig sein. In [14] wird dazu ein Beispiel (mit $\widetilde{n} = 508$, $n = 81$, $m = 36$) angegeben, dessen Formulierung 741 264 Zeilen, 360 945 Spalten und 54 145 231 Nichtnull-Einträge umfasst. Um das Lösen solcher Probleme zu beschleunigen oder überhaupt erst zu ermöglichen, stellen wir nun ein Verfahren (aus [2]) zur Erzeugung von Teilproblemen vor.

Die Aufteilung erfolgt, indem man für die möglichen Positionen *einer* Kernregion C_i obere und untere Schranken

$$1 \leq L_i \leq t_i \leq U_i \leq n \tag{4.1}$$

vorgibt. Beispielsweise in der MYZ-Fomulierung werden dazu einfach gewisse Variablen auf 0 fixiert:

$$y_{ik} := 0 \;\; \forall k \notin \{L_i, \ldots, U_i\} \quad \text{und} \quad z_{ikjl} := 0 \;\; \forall (i,j) \in L \wedge k \notin \{L_i, \ldots, U_i\} \; . \tag{4.2}$$

Macht man dies mit q verschiedenen Schranken L_i^α, U_i^α für $\alpha = 1, \ldots, q$ und gilt

$$\{1, \ldots, n\} = \bigcup_{1 \leq \alpha \leq q} \{L_i^\alpha, \ldots, U_i^\alpha\} \; , \tag{4.3}$$

so hat man das Problem sinnvoll in Teilprobleme aufgespalten. Denn der kleinste Wert aller Zielfunktionsminima der Teilprobleme entspricht dann offensichtlich dem minimalen Zielfunktionswert des Originalproblems. Eine Verallgemeinerung auf mehreren Kernregionen $C_i, C_j, \ldots$ ist leicht möglich.

Diese Vorgehensweise bietet neben der Verkleinerung der zu behandelnden Probleme noch einen weiteren Vorteil: Hat man bereits eine gewisse Anzahl von Problemen gelöst und dabei einen vorläufigen Minimalwert der Zielfunktion erhalten, so kann man die anderen Teilprobleme beim Lösen mit der dualen Simplex-Methode sofort abbrechen, sobald dieser Minimalwert während der Iteration überschritten wird. Einige Teilprobleme müssen so gar nicht zuende gelöst werden, was die Laufzeit positiv beeinflusst.

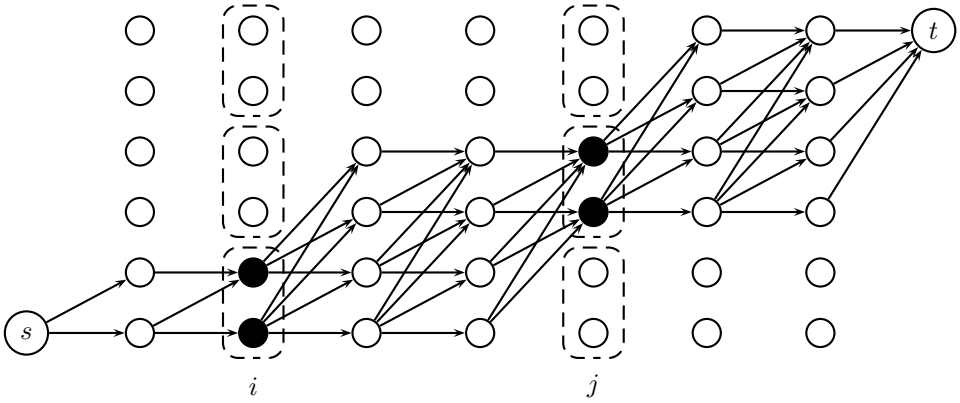

Abbildung 6: Visualisierung des doppelten Aufsplittens an den Kernregionen C_i und C_j (nach [2]).

Es stellen sich sofort mehrere Fragen:

1. Wieviele Teilprobleme sollen erzeugt werden?
2. Wie werden die „Intervalle“ $\{L_i^\alpha, \ldots, U_i^\alpha\}$ gewählt?
3. An wievielen Kernregionen wird aufgespalten?
4. An welchen Kernregionen wird aufgespalten?
5. In welcher Reihenfolge werden die Teilprobleme gelöst?

Bei der Frage nach der Zahl der Teilprobleme müssen zwei Gesichtspunkte berücksichtigt werden: Je mehr Teilprobleme, desto einfacher sind die einzelnen zu lösen. Je weniger Teilprobleme, desto wahrscheinlicher ist es, dass die gesuchte Optimallösung bereits nach wenigen Problemen gefunden wird und die restlichen früh abgebrochen werden können. Wir nehmen einen Moment an, dass uns eine gute Teilproblemanzahl bekannt sei und teilen die Intervalle der Einfachheit halber disjunkt und äquidistant auf (soweit dies im Rahmen der Ganzzahligkeit möglich ist). Wollen wir nur an einer Kernregion splitten, wählen wir sie so, dass das schwierigste entstehende Teilproblem möglichst leicht wird. Bezeichnet $\nu_{j\alpha}$ die Anzahl der Variablen – welche wir als Maß für die Schwierigkeit nehmen – des α. Teilproblems beim Splitten an der Kernregion C_j, so wählen wir C_i mit

$$i = \operatorname*{argmin}_{1 \le j \le m} \left\{ \max_{1 \le \alpha \le q} \nu_{j\alpha} \right\} . \tag{4.4}$$

Als „Ausgleich“ sortieren wir die entstehenden Teilprobleme absteigend und beginnen das Lösen mit dem größten. Dabei ist die Anzahl der (s,t)-Wege durch den Knoten (i,k) analog zu Gleichung (3.14) durch

$$\binom{i+k-2}{i-1}\binom{m-i+n-k}{m-i} \tag{4.5}$$

gegeben. In [2] wurde dieses Vorgehen, SPLIT1 genannt, zusammen mit der Variante SPLIT2 getestet. Bei letzterer wird analog an zwei Kernregionen aufgeteilt (Abb. 6), wodurch bei q Teilintervallen $\frac{q}{2}(q+1)$ Teilprobleme entstehen (da das Netzwerk „aufsteigend“ durchlaufen wird, sind nicht alle q^2 Kombinationen möglich). Diese Tests liefern in Ermangelung theoretischer Resultate eine heuristische Vorgabe für die Wahl von q. Ein qualitativ repräsentativer Verlauf der Abhängigkeit der Laufzeit von der Anzahl der Teilprobleme ist in Abbildung 7 dargestellt.

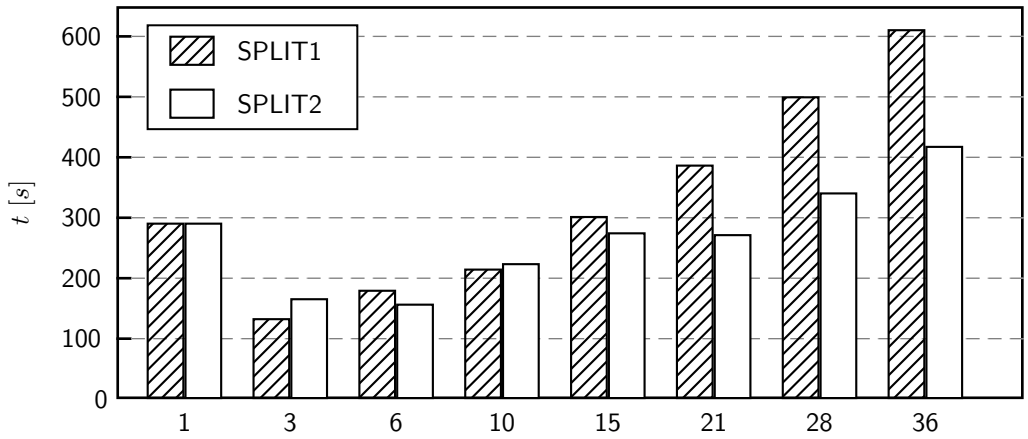

Abbildung 7: Laufzeitvergleich der beiden SPLIT-Varianten für verschiedene Teilproblemanzahlen; anhand eines Beispiels mit $N_x \approx 10^{30}$ (Daten aus [2]).

Das Ergebnis zeigt, dass sich bei geschickter Wahl von q bzw. $\frac{q}{2}(q+1)$ die Laufzeit um den Faktor 2 verringert. Wird die Anzahl der Teilprobleme jedoch zu groß gewählt, lassen sowohl die verringerte Wahrscheinlichkeit für das frühe Auffinden des Optimalwertes als auch wachsender Initialisierungsaufwand die Gesamtlaufzeit wieder deutlich steigen. Die Variante SPLIT2 zeigt sich dabei robuster gegen eine ungeeignete Wahl der Teilproblemanzahl, weshalb sie vorzuziehen ist.

4.2 Parallelisierung

Die soeben vorgestellte Methode zur Zerlegung in Teilprobleme führt beinahe zwangsläufig zu der Frage, ob man diese Teilaufgaben effizient auf $p > 1$ verschiedenen Prozessoren ausführen kann. Zur Beantwortung beschränken wir uns darauf, die in [2] und [13] dargestellten Resultate qualitativ zu beschreiben.

Da die Ausführungszeit der Teilprobleme stark variieren kann (es sei noch einmal erwähnt, dass es sich um ein $\mathcal{NP}$-hartes Problem handelt) und kaum vorhersagbar ist, kommt nur eine *dynamische Lastverteilung* in Frage. Diese lässt sich am einfachsten mit einem *Master/Slave-Prinzip* realisieren, bei dem ein Prozessor die Teilprobleme verwaltet und an unbeschäftigte Prozessoren verteilt. Außerdem lassen sich so mit geringem Kommunikationsaufwand die *lokale* (d. h. für einen einzelnen Prozessor gültige) und *globale* (d. h. für das Gesamtproblem gültige) *obere Schranke* zwischen den Prozessoren austauschen: Sobald ein Slave seine lokale obere Schranke verbessert, teilt er sie dem Master mit. Dieser ersetzt die bisherige globale obere Schranke durch die lokale obere Schranke des Slaves, falls letztere niedriger ist. Deshalb fragen alle Slaves in regelmäßigen Abständen (z. B. alle 500 bis 1 000 Simplex-Iterationen) beim Master die aktuelle globale obere Schranke ab und benutzen sie als vorzeitiges Abbruchkriterium. Dieses zwischenzeitliche Abfragen erhöht zwar den Kommunaktionsaufwand, ist aber zwingend erforderlich, da sonst ein einziger Prozessor mit einem sehr aufwendigen Teilproblem die Rechenzeit dominieren könnte.

Zu klären bleibt, wieviele Teilprobleme man erzeugen und auf wieviele Prozessoren man diese verteilen sollte. Die praktischen Tests lassen sich wie folgt zusammenfassen:

- Genaue Vorhersagen der optimalen Kombination von Teilproblem- und Prozessorzahl sind schwierig, da sie problemabhängig sind.
- Bei günstiger Wahl von Teilproblem- und Prozessorzahl liegt die Effizienz teilweise über 1 (bis zu 1,7)*.
- Steigende Prozessorzahlen verringern deutlich den Einfluss der Teilproblemanzahlen auf die Laufzeiten.
- Variante SPLIT2 ist noch überlegener als im sequentiellen Fall und erreicht selbst bei zwölf Prozessoren noch Effizienzwerte zwischen 0,5 und 1.

Je mehr Teilprobleme gleichzeitig gelöst werden, desto wahrscheinlicher ist es, dass die Optimallösung schnell gefunden wird. Insbesondere verringert sich gegenüber dem sequentiellen Falle die Gefahr, dass ein Prozessor zunächst einige „uninteressante“ Teilprobleme löst, bevor er auf solche mit guten (optimalen) oberen Schranken trifft. Wir haben gesehen, dass dieses Risiko mit der Teilproblemanzahl steigt und genau hier wirkt die Parallelisierung entgegen. Dadurch dass die aktuell beste obere Schranke regelmäßig an alle Prozessoren weitergeleitet wird, können große Teilprobleme ggf. frühzeitig abgebrochen werden, was die Rechenzeit deutlich verringert. Ab einem gewissen Parallelisierungsgrad reduziert die nötige Kommunikation die Effizienz jedoch spürbar. Stehen mehr Prozessoren zur Verfügung als sinnvollerweise eingesetzt werden sollten, so kann man sie derart organisieren, dass sie die Amino-Sequenz gleichzeitig mit mehreren Kernschablonen vergleichen.

5 Zusammenfassung

Dieser Vortrag konnte nur einen kurzen Einblick in die komplexen Fragestellungen rund um das Protein Threading Problem geben. Ausgehend vom Netzwerk-Fluss-Modell wurden im Rahmen der gemischt-ganzzahligen Programmierung Formulierungen vorgestellt, die eine Amino-Sequenz mit de facto *polynomialem* Zeitaufwand in eine Kernschablone einfädeln können. Angesichts der riesigen Anzahl von kombinatorischen Möglichkeiten – 10^{40} und mehr – sind Laufzeiten von wenigen Minuten bereits ein sehr beachtenswertes Resultat. Man sollte sich aber vor Augen halten, dass der Vergleich eines mittelgroßen Proteins mit einigen tausend Kernschablonen dennoch etwa eine Woche benötigt.

Neben den unmittelbaren biologischen Anwendungen ist das Problem auch theoretisch interessant. Eine wichtige Frage bleibt, weshalb es trotz nachgewiesener $\mathcal{NP}$- und $\mathcal{MAX}$-$\mathcal{SNP}$-Härte in den allermeisten Fällen „leicht“ lösbar ist. Eine Antwort darauf könnte helfen, eine Vielzahl anderer Algorithmen effizienter zu gestalten.

*Diese Aussage ist in der Tat sinnvoll, da auf allen Prozessoren mit der MYZ-Formulierung der schnellste bekannte sequentielle Algorithmus verwendet wird. Das Ergebnis zeigt, dass man – zumindest theoretisch – diesen sequentiellen Algorithmus beschleunigen kann, indem man ihn seine eigene parallele Ausführung simulieren lässt.

Literatur

[1] Akutsu, T.; Satoru, M.: *On the approximation of protein threading*. Theoretical Computer Science, Vol. 210, S. 261-275 (1999)

[2] Andonov, R.; Balev, S.; Yanev, N.: *Protein Threading: From Mathematical Models to Parallel Implementations*. INFORMS Journal on Computing, Vol. 16, S. 393-405 (2004)

[3] Cook, W. J.; Cunningham, W. H., Pulleyblank, W. R.; Schrijver, A.: *Combinatorial Optimization*. Wiley, New York (1998)

[4] Cornuéjols, G.; Nemhauser, G. L.; Wolsey, L. A.: *The Uncapacitated Facility Location Problem*. Erschienen in (Kapitel 3): Mirchandani, P.; Francis, R. (Eds.): *Discrete Location Theory*. Wiley, New York (1990)

[5] Knippers, R.: *Molekulare Genetik*. Georg Thieme Verlag, Stuttgart (1997)

[6] Lathrop, R. H.: *The Protein Threading Problem With Sequence Amino Acid Interaction Preferences Is NP-Complete*. Protein Engineering, Vol. 7, S. 1059-1068 (1994)

[7] Lathrop, R. H.; Rogers Jr., R. G.; Bienkowska, J.; Bryant, B. K. M.; Buturović, L. J.; Gaitatzes, C.; Nambudripad, R.; White, J. V.; Smith, T. F: *Analysis and algorithms for protein sequence-structure alignment*. Erschienen in (Kapitel 12): Salzberg, S. L.; Searls, D. B.; Kasif, S. (Eds.): *Computational Methods in Molecular Biology*. Elsevier, Amsterdam (1998)

[8] Lengauer,T.: *Computational Biology at the Beginning of the Post-genomic Era*. Erschienen in: Wilhelm, R. (Ed.): *Informatics—10 Years Back, 10 Years Ahead*. Lecture Notes in Computer Science, No. 2000, Springer, Heidelberg (2001)

[9] Marin, A.; Pothier, J.; Zimmermann, K.; Gibrat, J.-F.: *FROST: A Filter-Based Fold Recognition Method*. PROTEINS: Structure, Function and Genetics, Vol. 49, S. 493-509 (2002)

[10] RCSB Protein Data Bank: `http://www.rcsb.org/pdb/`.

[11] Setubal, J. C.; Meidanis, J.: *Introduction to Computational Molecular Biology*. Brooks/Cole Publishing Company, Pacific Grove, CA (1997)

[12] Xu, J.; Li, M.; Lin, G.; Kim, D.; Xu, Y.: *Protein Threading Problem by Linear Programming*. Proceedings of the 17th Pacific Symposium on Biocomputing (PSB), S. 264-275 (2003)

[13] Yanev, N.; Andonov, R.: *Solving the Protein Threading Problem in Parallel*. Workshop on HiCOMB'03, 17th IPDPS (2003)

[14] Yanev, N.; Andonov, R.: *The Protein Threading Problem is in P?*. Institut National de Recherche en Informatique et en Automatique, Research Report 4577 (2002)

Das Handout und die Präsentationsfolien stehen unter
`http://www.mathi.uni-heidelberg.de/~ferreau/proteinThreading/`
zum Download bereit.