Vollständige Aufzählung der optimalen Strukturen von Gitterproteinen durch dynamische Zerlegung des assoziierten Constraint Satisfaction Problems

Diplomarbeit

zur Erlangung des akademischen Grades
Diplom-Bioinformatiker

Friedrich-Schiller-Universität Jena
Fakultät für Mathematik und Informatik

Verfasser: Martin Mann
Betreuer: Sebastian Will - Rolf Backofen - Ina Koch
Abgabe: 30. Januar 2006

Zusammenfassung

Vollständige Aufzählung der optimalen Strukturen von Gitterproteinen durch dynamische Zerlegung des assoziierten Constraint Satisfaction Problems

Die Strukturvorhersage von Gitterproteinen im HP-Modell ist ein NP-vollständiges Problem der Bioinformatik. Um alle optimalen Strukturen einer HP-Sequenz vorherzusagen, formulierten Backofen/Will das Problem als ein Constraint Satisfaction Problem (CSP). Dieser Ansatz ist sehr schnell und seine Exaktheit konnte bewiesen werden.

Die in dieser Arbeit vorgestellte *Zerlegungssuche* stellt ein neues generelles Suchverfahren dar, um alle Lösungen eines CSPs aufzuzählen. Ihre Arbeitsweise, bei der CSPs in disjunkte Teilprobleme zergliedert werden, wird am Beispiel des Strukturvorhersage CSPs demonstriert und erläutert. Die Zerlegung erfolgt rekursiv und dynamisch während der Lösungssuche und steht somit gegen viel diskutierte, statische Zerlegungsansätze. Die Lösungen werden aus den Teillösungen durch die eingeführte kartesische Vereinigung gebildet. Es werden verschiedene Strategien zur Performanzsteigerung dargelegt und deren Implementierung in C++ unter Verwendung der Programmierbibliothek ILOG Solver 6.1™ vorgestellt. Der Geschwindigkeitsgewinn durch Anwendung der Zerlegungssuche gegenüber einem weit verbreiteten Verfahren, dem Maintaining-Arc-Consistency-Ansatz (MAC), wird untersucht und diskutiert.

Aus den Laufzeitanalysen ist der Performanzgewinn unter Verwendung der Zerlegungssuche gegenüber MAC deutlich erkennbar. Weitere Untersuchungen finden die optimalen Suchstrategien und Heuristiken zur Lösung des Strukturvorhersage-CSPs. Die vorliegende Diplomarbeit liefert somit einen exakten und schnellen Ansatz, um alle optimalen Strukturen eines Gitterproteins im HP-Modell vollständig aufzuzählen. Die vorgestellte Zerlegungssuche erreicht dies durch dynamische Zerlegung des assoziierten CSPs und vermeidet redundante Arbeitsweise bisheriger Suchverfahren wie dem MAC.

Abstract

Exhaustive enumeration of optimal structures of lattice proteins via dynamic decomposition of the associated constraint satisfaction problem

The prediction of tertiary structures of lattice proteins is an NP-complete problem in bioinformatics. The approach of Backofen/Will to enumerate lattice protein structures via constraint programming is one of the fastest and proven to be exact. This thesis describes a new and general search strategy for exhaustive solution enumeration of a constraint satisfaction problem used for this approach. The speed up by the so called Clustersearch is based on a recursive and dynamic decomposition of the constraint problem in disjunctive partial problems. The overall solution is generated out of the partial solutions by a so called cartesian union. This decomposition avoids redundant work during the search and is a general strategy for constraint programming. An implementation in C++ using the constraint programming library ILOG Solver 6.1^{TM} is presented. The performance improvement of Clustersearch relative to the widely used search method Maintaining-Arc-Consistency is tested and discussed.

Inhaltsverzeichnis

Abbildungsverzeichnis

Algorithmenverzeichnis

Quellcodeverzeichnis

Abkürzungsverzeichnis

CSP	-	Constraint Satisfaction Problem
SAW	-	Self-Avoiding-Walk
CPSP	-	Constraint-based Protein Structure Prediction
MAC	-	Maintaining Arc Consistency
SP-CSP	-	Strukturvorhersage Constraint Satisfaction Problem
$\equiv \quad \not\equiv$	-	stehen für Gleichheits- oder Ungleichheitsbedingungen
$x * y$	-	steht für das Produkt von x und y
$A \times B$	-	steht für das Kreuzprodukt der Mengen A und B
$A \otimes B$	-	steht für die kartesische Vereinigung der Mengen A und B
API	-	Application Programming Interface
v.l.n.r.	-	von links nach rechts

Danksagung

Danken möchte ich zuallererst Professor Rolf Backofen, Dr. Sebastian Will und Frau Professor Ina Koch für die Übergabe dieses interessanten Themas und meiner guten Betreuung. Besonderen Dank an Sebastian, der mir stets mit offenem Ohr und einer helfenden Hand zur Stelle war, wenn ich Fragen oder Probleme hatte.

Doch ich wäre nicht wo ich bin ohne die wunderbare Unterstützung durch meine Familie, die mir ein sorgenfreies Studieren und Leben ermöglicht hat und Hilfe bot, wenn ich sie brauchte.

Meinen Freunden und Kommilitonen Hannes Kochniß, Janice Kielbassa, Ralf-Peter Weiß und Matthias Brosemann danke ich für die tolle Studienzeit und freue mich auf unsere Zukunft!

Mein größter Dank jedoch gilt meiner lieben Anke, die mich mit großer Kraft, Wissen und Freude schon lange begleitet und ohne die die hier vorliegende Arbeit nicht ihre Qualität erlangt hätten.

Kapitel 1

Einleitung

Proteine und ihre Interaktionen mit anderen Molekülen können als der „Motor des Lebens“ angesehen werden. Sie kommen in fast allen Stoffwechselwegen vor und haben das weitreichenste Wirkungsspektrum in lebenden Zellen. Diese Diversität basiert auf einer immensen dreidimensionalen Strukturvielfalt, welche unter anderem die unterschiedlichsten Funktionalitäten bedingt [Cre90]. Leider sind derzeit für die meisten bekannten Proteinsequenzen keine Strukturen bekannt. Trotz grosser Fortschritte in der biochemischen Strukturanalyse, bleibt diese aufwändig und teuer. Der Vorgang der Strukturbildung von einer eindimensionalen Aminosäurekette zur dreidimensionalen Struktur, der als *Proteinfaltung* bezeichnet wird, muss verstanden werden um diese Problematik zu lösen. Computergestütze *Strukturvorhersage* der Proteinfaltung auf Basis einer gegebenen Proteinsequenz wäre eine gute Alternative zur biochemischen Analyse, ist jedoch derzeit noch nicht hinreichend möglich. Ein Grund liegt in der Komplexität des Problems und dem geringen Wissen um grundlegende Prinzipien der Proteinfaltung.

Um allgemeine Gesetzmäßigkeiten zu erforschen definierten K. Lau und K. Dill ein stark vereinfachtes Proteinmodell, das *HP-Modell* [LD89]. Dieses reduziert die Vielfalt der Aminosäuren auf ihre hydrophoben (H) oder polaren (P) Eigenschaften und wird in Abschnitt 1.2 näher erläutert. Die einzelnen Aminosäuren können hierbei nicht frei im Raum, sondern nur auf definierten Gitterpunkten platziert werden. Zudem wird ihre räumliche Ausdehnung auf diese Punkte beschränkt und eine einfache Energiefunktion definiert. Sie modelliert die hydrophoben Kräfte, die in realen Proteinstrukturen auftreten.

Hierauf basierend kann das *Gitter-Strukturvorhersageproblem* vereinfacht wie folgt formuliert werden: Finde für eine gegebene Gitterproteinsequenz eine Struktur im Gitter mit minimaler Energie. Es beschreibt also die Proteinfaltung von der sequentiellen, sogenannten Primärstruktur zur dreidimensio-

nalen funktionalen Struktur. Hierbei wird angenommen, dass die Struktur mit minimaler Energie des Gitterproteins der in der Natur vorgefundenen, funktionalen Struktur realer Proteine entspricht.

Diese sogenannten *Gitterproteine* stellen eine grobe Vereinfachung von natürlichen Proteinstrukturen dar. Dennoch ist die Vorhersage von Strukturen mit optimaler Energie selbst im HP-Modell NP-vollständig [BL98, CGP+98] (NP-Vollständigkeit für reale Proteinfaltung [UM93]). Es sei angemerkt, dass dieses ein Modell aus einer großen Palette von Proteinstrukturmodellierungen ist [GKS93, KS04].

Trotz dieser simplifizierten Eigenschaften gibt es verschiedenste Untersuchungen basierend auf dem HP-Modell. So betrachtet zum Beispiel M. Wolfinger die *Energielandschaften* von Gitterproteinen, also die Energieniveaus und deren Verteilung über die möglichen Strukturen einer Sequenz [Wol04]. Hierzu werden Daten über alle globalen und wenig schlechteren lokalen Energieoptima (siehe Abschnitt 1.2) sowie deren Umgebung benötigt. Ähnliche Analysen der Faltungskinetiken werden von verschiedenen Gruppen durchgeführt [ARDK96, DC97, RBB97, FHSW02, WBBC05]. Neben dem Energielevel ist auch die Anzahl verschiedener Strukturen auf einem Energieniveau, die sogenannte Degeneriertheit [YFT+95], und die Proteinevolution [CBB02] im Fokus der Forschung. Zudem gibt es Studien, wie man die Ergebnisse aus der Gitterproteinfaltung auf die Faltung realer Proteinsequenzen übertragen kann [MS01].

Für all diese Analysen werden schnelle und exakte Algorithmen benötigt. Je nach Fragestellung und Motivation sind verschiedene Herangehensweisen an die Strukturvorhersage für Gitterproteine möglich. Heuristische Verfahren finden mit neuen oder klassischen Optimierungsmethoden, wie zum Beispiel dem Monte-Carlo-Verfahren und entsprechenden Erweiterungen, schnell eine möglichst gute Lösung [LW01]. Sie lassen jedoch kaum Aussagen über die Beschaffenheit der Energielandschaften oder die Qualität der gefundenen Lösungen zu. Für solche Fragestellungen sind Verfahren zur vollständigen Aufzählung aller optimalen Strukturen wie der von R. Backofen und S. Will vorgestellte Ansatz des CPSP-Verfahrens (Constraint-based protein structure prediction) via Constraintprogrammierung notwendig [BW05]. Ob der NP-Vollständigkeit des Problems und einer Vielzahl möglicher optimaler Strukturen im HP-Modell, ist die vollständige Aufzählung jedoch sehr aufwändig. Die verschiedenen Herangehensweisen werden in Abschnitt 1.3 näher vorgestellt.

1.1 Überblick

In dieser Arbeit wird die vollständige Aufzählung aller optimalen Strukturen nach [BW05] diskutiert und eine neue Art der Suche, die „Zerlegungssuche“, für die Lösung des Problems verwendet. Hierfür werden im Folgenden Gitterproteine und mögliche Verfahren zur Vorhersage optimaler Strukturen vorgestellt.

Im 2. Kapitel wird die Möglichkeit der Vorhersage optimaler Strukturen und ihrer vollständigen Aufzählung via Constraintprogrammierung betrachtet. Es werden die Stärken der Formulierung der Strukturvorhersage als Constraint Satisfaction Problem (CSP) aufgezeigt. Ein Standardlösungsverfahren, Maintaining Arc Consistency, für CSPs wird näher beleuchtet und seine Schwäche aufgrund redundanter Arbeitsweise während der Lösungssuche verdeutlicht.

Die Vermeidung dieser Redundanz und die Optimierung der vorgestellten Standardsuche bilden den Kern der Arbeit. In Kapitel 3 wird eine neue Art der Suche, die *Zerlegungssuche*, für die Lösung des nach R. Backofen und S. Will definierten CSPs vorgestellt. Hierbei wird die wiederholte und redundante Lösung von Teilproblemen durch dynamische und rekursive Zerlegung des CSPs vermieden. Diese Strategie stellt einen generischen Ansatz dar, der am Beispiel der Proteinfaltung im HP-Modell verdeutlicht werden soll. Es wird auf Heuristiken zur Performanzsteigerung der Zerlegungs- und Standardsuche eingegangen und diese diskutiert.

Ein Vergleich der „klassischen“ Standardsuche mit der eingeführten Zerlegungssuche, sowie ein Überblick über die Implementierung in ILOG Solver 6.1™, wird im Kapitel 4 gegeben.

Daran schließt sich eine zusammenfassende Betrachtung der Arbeit und die Diskussion zusätzlicher Potentiale und möglicher Erweiterungen der Zerlegungssuche an.

1.2 Gitterproteine

Proteine sind kettenförmige Polypeptide unterschiedlichster Länge. Die Kettenstruktur entsteht durch das Verknüpfen von Aminosäuren durch Peptidbindungen, wobei in natürlichen Proteinen im Allgemeinen 20 sogenannte proteinogene Aminosäuren vorkommen. Proteinogene Aminosäuren weisen eine gemeinsame Grundstruktur auf, die sich lediglich in ihren Seitenketten unterscheidet. In Abbildung 1.1 werden diese durch $R_1, R_2, ...$ dargestellt. Die Peptidbindungen formen das in der Grafik veranschaulichte Peptidrückgrat.

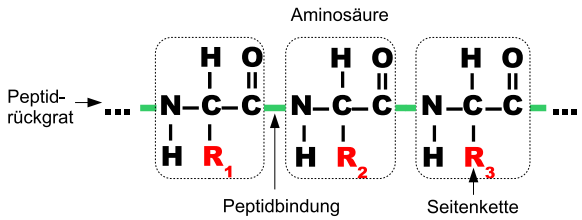

Abbildung 1.1: Vereinfachte Polypeptidstruktur von natürlichen Proteinen.

Diese eindimensionale kettenförmige Grundstruktur (auch als *Primärstruktur* bezeichnet) ordnet sich in der Regel selbstständig zu einer dreidimensionalen sogenannten *Tertiärstruktur*. Die darin enthaltenen wiederkehrenden Strukturelemente werden als Sekundärstrukturelemente bezeichnet, der Vorgang an sich als *Proteinfaltung*. Er findet aufgrund von nichtkovalenten Wechselwirkungen zwischen den Atomen des Proteins statt. Solche Wechselwirkungen sind keine festen Bindungen sondern anziehende bzw. abstoßende Kräfte innerhalb des Polypeptids und seiner Umgebung. Da die dreidimensionale Struktur von Proteinen ihre funktionalen Eigenschaften bestimmt, sich die biochemische Tertiärstrukturbestimmung aber als schwierig und aufwändig erweist, ist man an der Vorhersage der Faltungsergebnisse interessiert. Doch schon grundlegende Aussagen sind schwierig. Es konnte jedoch gezeigt werden, dass wassergelöste globuläre Proteine einen sogenannten *hydrophoben Kern* und eine hydrophile Hülle aufweisen. Als hydrophober Kern wird die dichte Packung von hydrophoben Seitenketten im Strukturinneren bezeichnet. Durch eine solche Anordnung minimiert sich der Kontakt der hydrophoben Aminosäuren mit den umgebenden Wassermolekülen, was energetisch günstiger ist. Hydrophile Aminosäuren sind hingegen meist im äußeren Bereich der Proteinstruktur zu finden und bilden die besagte Hülle. Diese hydrophoben Kräfte scheinen einen großen Beitrag zur Proteinfaltung beizutragen [BT98]. Je nach Umfeld und äußeren Einflüssen kann ein Protein aus der Primärstruktur (im Folgenden als Proteinsequenz bezeichnet) in verschiedene Tertiärstrukturen falten. Um diese zu bewerten kann die *innere Energie* zu Rate gezogen werden, die den Energiegehalt einer Materiemenge beschreibt. Je dichter eine Struktur ist und um so mehr günstige Wechselwirkungen auftreten, um so geringer ist ihre innere Energie. Die natürliche Struktur eines Proteins hat zumeist eine sehr geringe, jedoch nicht immer die minimale (global optimale) innere Energie. Die Menge aller möglichen Strukturen einer Proteinsequenz wird hierbei als *Konformationsraum* bezeichnet.

Um den Konformationsraum zu untersuchen, führten K. Lau und K. Dill das *HP-Modell* ein [LD89]. Hierbei handelt es sich um ein sehr vereinfachtes Proteinmodell, in dem Aminosäuren auf ihre hydrophoben/hydrophilen Eigenschaften reduziert werden. Ein Protein wird als lineare Kette von Monomeren (Aminosäuren) modelliert, wobei diese in hydrophob bzw. nicht polar (H) und polar bzw. hydrophil (P) gegliedert werden. Das 20-elementige Alphabet der proteinogenen Aminosäuren wird somit auf H und P reduziert.

Definition 1.2.1 (HP-Sequenz)
Eine Folge von H- und P-Monomeren, wird als HP-Sequenz *S bezeichnet.*

Die HP-Sequenz eines Gitterproteins stellt somit das Äquivalent zur Primärstruktur natürlicher Proteine dar und wird im Verlauf der Arbeit als solche behandelt.

Um den Konformationsraum zu diskretisieren und die Strukturmodellierung der Monomerketten zu vereinfachen, wird der Raum auf Gitterpunkte reduziert, auf denen jeweils ein Monomer positioniert werden kann.

Definition 1.2.2 (Gitter)
Ein Gitter[1] *ist eine Menge von Ortsvektoren L (auch* Gitterpunkte *genannt), sodass*

$$\vec{0} \in L \tag{1.1}$$

$$\vec{u}, \vec{v} \in L \text{ impliziert } (\vec{u} + \vec{v}), (\vec{u} - \vec{v}) \in L \tag{1.2}$$

wobei $+$ und $-$ Vektoraddition und -subtraktion darstellen und $\vec{0}$ der Nullvektor ist.

Zu jedem Gitter L gibt es eine Menge B von n Vektoren $\{\vec{b}_1, ..., \vec{b}_n\}$, sodass jeder Gitterpunkt aus L als ganzzahlige Linearkombination dieser dargestellt werden kann. Folglich gilt:

$$L = \{ \sum_{1 \leq i \leq n} k_i \vec{b}_i \mid k_i \in \mathbb{N} \wedge b_i \in B \}. \tag{1.3}$$

Wenn B minimal ist und die Gleichung 1.3 erfüllt, wird die Menge B als *Basis* und ihre Elemente als *Basisvektoren* bezeichnet. Ihre Kardinalität n entspricht der Dimension des Gitters.

[1] Definition aus [Wil05]

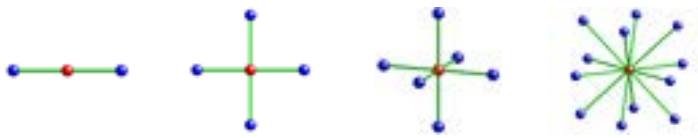

Abbildung 1.2: Nachbarschaften in verschiedenen Gittern gemäß Tabelle 1.1 (v.l.n.r.: linear, quadratisch, kubisch, flächenzentriert kubisch).

Definition 1.2.3 (Nachbarschaft)
Eine Nachbarschaft *ist eine Beziehung zwischen Gitterpunkten, welche durch eine Menge von Nachbarschaftsvektoren NV definiert ist. Zwei Gitterpunkte $\vec{u}, \vec{v}$ sind benachbart, wenn gilt*

$$(\vec{u} - \vec{v}) \in NV \tag{1.4}$$

Ohne Beschränkung der Allgemeinheit soll zudem gelten

$$\vec{0} \notin NV \tag{1.5}$$

Eine naheliegende und in dieser Arbeit verwendete Definition der Nachbarschaft NV für ein Gitter L ist die Menge der Basisvektoren B und ihrer Inversen. In Tabelle 1.1 und Abbildung 1.2 sind einige Gitterbeispiele mit den entsprechenden Nachbarzahlen und den Positionen der Nachbarn im Raum dargestellt. Vektoren, deren Komponenten sich lediglich im Vorzeichen unterscheiden, wurden in einer $\pm$-Darstellung zusammengefasst (z.B. $(1, 0)$ und $(-1, 0)$ zu $(\pm 1, 0)$). Die Nachbarschaft des flächenzentriert kubischen Gitters beinhaltet zusätzlich zur Basis $B = \{(1, 1, 0), (1, 0, 1), (0, 1, 1)\}$ und ihrer Inversen noch die Menge $\{b_i - b_j \mid b_i, b_j \in B \wedge b_i \neq b_j\}$.

Gittertyp	Nachbarzahl	NV
linear	2	$\{(\pm 1)\}$
quadratisch	4	$\{(\pm 1, 0), (0, \pm 1)\}$
kubisch	6	$\{(\pm 1, 0, 0), (0, \pm 1, 0), (0, 0, \pm 1)\}$
flächenzentriert kubisch	12	$\{(\pm 1, \pm 1, 0), (\pm 1, 0, \pm 1), (0, \pm 1, \pm 1)\}$

Tabelle 1.1: Mögliche Nachbarschaften verschiedener Gitter

Definition 1.2.4 (Self-avoiding-walk)
Ein Self-Avoiding-Walk *(SAW) ist ein mathematisches Modell eines Gitterpfades im Gitter L mit Nachbarschaft NV, der keinen Gitterpunkt zweimal betritt. Ein SAW der Länge k ist eine Folge von k Gitterpunkten $(f_1, ..., f_k)$, für die gilt:*

$$f_i \in L \qquad : 1 \leq i \leq n \tag{1.6}$$
$$(f_i - f_{i+1}) \in NV \quad : 1 \leq i < n \tag{1.7}$$
$$f_i \neq f_j \qquad : 1 \leq i < j \leq n \tag{1.8}$$

Somit ist ein „Schritt“ im Gitter durch die Nachbarschaft NV bestimmt.

Definition 1.2.5 (Konformationsraum einer HP-Sequenz)
Die Menge aller möglichen Strukturen einer HP-Sequenz S wird als Konformationsraum $conf(S)$ bezeichnet.

Definition 1.2.6 (Struktur einer HP-Sequenz)
Eine Struktur *oder* Kettenkonformation *$K \in conf(S)$ einer HP-Sequenz S der Länge k für ein Gitter L mit Nachbarschaft NV, ist ein Self-Avoiding-Walk der Länge k im Gitter L. Hierbei wird jedem Sequenzmonomer $S_i \in \{H, P\}$ die SAW-Gitterposition $f_i \in L$ zugewiesen $(1 \leq i \leq k)$.*

Wie man durch die Abbildung 1.2 erkennt, ist die Güte der Proteinstrukturmodellierung sehr stark von der Art des Gitters und der zugehörigen Nachbarschaft abhängig. So ist zum Beispiel ein lineares oder quadratisches Gitter offensichtlich nicht geeignet, eine reale dreidimensionale Proteinstruktur zu modellieren. Auch lässt die Nachbarschaft des normalen kubischen Gitters nur eine grobe Nachbildung der Strukturen realer Proteine zu, was im flächenzentriert kubischen Gitter besser möglich ist.

Um die innere Energie einer HP-Struktur $K \in conf(S)$ für eine Sequenz S ermitteln zu können, definierten K. Lau und K. Dill die in Gleichung 1.9 gegebene Energiefunktion für zwei Strukturpositionen K_i und K_j. Durch sie werden die hydrophoben Kräfte in einer realen Proteinstruktur modelliert.

$$Energie(K_i, K_j) = \begin{cases} -1 & \text{wenn } S_i, S_j = H \text{ und } K_i, K_j \text{ benachbart} \\ 0 & \text{sonst} \end{cases} \tag{1.9}$$

Die *Gesamtenergie* $E(K)$ einer Struktur K ergibt sich aus der Summe der Energiebeiträge aller benachbarten Moleküle, wobei nur *HH-Kontakte* einen Energiebeitrag liefern. Als *Kontakt* wird hierbei die Nachbarschaft zweier Positionen, als HH-Kontakt die Nachbarschaft zweier H-Monomere bezeichnet. Die in Abbildung 1.3 dargestellte Beispielstruktur, bei der H-Monomere

schwarz, P-Monomere weiß und das verbindende Sequenzrückgrat durchgezogen grau dargestellt sind, hat somit eine Gesamtenergie von -2. Die HH-Kontakte sind gestrichelt schwarz eingetragen.

a)

	H	P
H	-1	0
P	0	0

b)

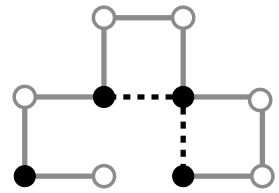

Abbildung 1.3: a) Energiefunktion des HP-Modells und b) Beispiel einer HP-Struktur im quadratischen Gitter [Wil05].

Die Strukturvorhersage im HP-Modell folgt der Annahme, dass eine Struktur mit minimaler Energie im Gitter einer funktionalen Struktur realer Proteine entspricht. Somit geht man vereinfachend davon aus, dass reale Proteine in die energetisch günstigste Struktur falten. Jedoch ist die reale Proteinfaltung und die resultierende Struktur stark von äußeren Einflüssen, wie z.B. vom pH-Wert [BT98], abhängig, was im HP-Modell vernachlässigt wird.

Die strukturellen und diskretisierenden Vereinfachungen ermöglichen eine vollständige Aufzählung und Untersuchung des Konformationsraumes von HP-Sequenzen. Allerdings wurde die *NP-Vollständigkeit des Strukturvorhersageproblems* im HP-Modell nachgewiesen, wodurch kein in der Laufzeit polynomieller Algorithmus erwartet werden kann [BL98, CGP+98].

Häufige Zielsetzungen bei der Untersuchung des Konformationsraumes sind eine gute, die beste oder alle Strukturen mit einer bestimmten Energie vorherzusagen. Dies ermöglicht den Einsatz verschiedener Optimierungsverfahren und anderer Methoden, über die im Abschnitt 1.3 ein Überblick gegeben wird.

Aufgrund der Energiefunktion weisen Strukturen mit maximaler HH-Kontaktzahl die geringste innere Energie auf. Diese wird durch die Bildung eines „hydrophoben Kerns“ möglich, was auch dem genannten Faltungsverhaltung von natürlichen wassergelösten, globulären Proteinen entspricht. Somit ist das HP-Modell zur Untersuchung von allgemeinen Faltungseigenschaften dieser Proteine geeignet.

Definition 1.2.7 (Optimalität von Strukturen)
Eine Struktur $K_o \in conf(S)$ einer Sequenz S ist optimal, wenn gilt:

$$\forall_{K \in conf(S)} : E(K_o) \leq E(K)$$

Im HP-Modell ist dies eine Struktur mit maximaler Anzahl HH-Kontakte, da HP- oder PP-Kontakte keinen Energiebeitrag liefern (siehe Gleichung

1.9). Aufgrund dieser Besonderheit genügt die Betrachtung der H-Monomer-Positionen, um die Energie einer Struktur zu ermitteln.

Definition 1.2.8 (H-Kern einer Struktur)
Der H-Kern $core(K)$ *einer Struktur* $K \in conf(S)$ *einer HP-Sequenz* S *ist die Menge der Gitterpositionen* K_i *aller H-Monomere und entspricht*

$$core(K) = \{K_i \mid K_i \in K \wedge S_i = H\}$$

Die Elemente des H-Kerns müssen sich nicht zwangsläufig im Inneren der Struktur befinden. Die Energiebeiträge ihrer HH-Kontakte bestimmen seine Energie.

1.3 Strukturvorhersage im HP-Modell

Je nach Zielsetzung der Analyse des Konformationsraumes im HP-Modell sind verschiedene Herangehensweisen möglich. Wenn es nur eine repräsentative und möglichst optimale Struktur zu finden gilt, bieten sich heuristische Verfahren an, wie zum Beispiel das von T. Beutler und K. Dill für das quadratische Gitter vorgestellte CG-Verfahren (Core-directed chain Growth). Es baut zuerst einen möglichst kompakten H-Kern auf, um diesen anschließend an die Sequenz anzupassen [BD96]. Hierbei wird der anfangs kompakte H-Kern immer weiter ausgebreitet, um P-Monomere einzufügen und die Ketteneigenschaft der Struktur zu gewährleisten. 2005 stellten T. Bui und G. Sundarraj einen genetischen Algorithmus vor, der „Sekundärstrukturelemente“ evolviert [BS05]. Auf dem Gebiet der heuristischen Strukturvorhersage-Algorithmen für das HP-Modell wurde und wird viel geforscht und die verschiedensten Methodiken angewandt [DFC93, LW01, YE02, LMW03].

All diese Verfahren finden lokal optimale Strukturen. Sie lassen keine Aussagen über die Güte einer Struktur zu. Für solche Untersuchungen wird die vollständige Aufzählung aller optimalen Strukturen bzw. aller Strukturen einer gegebenen Energie benötigt. Hierfür existieren im kubischen Gitter bisher nur zwei Algorithmen, die keine „brute force“-Aufzählung vornehmen: das von K. Yue und K. Dill eingeführte „constrained hydrophobic core construction (CHCC)“- und das „constraint-based protein structure prediction (CPSP)“-Verfahren von R. Backofen und S. Will [YD95, BW05]. Für beide Ansätze kann bewiesen werden, dass sie global optimale Strukturen vorhersagen. Allerdings wurde die Unvollständigkeit der vollständigen Aufzählung aller optimalen Strukturen mit der CHCC-Methode gezeigt [BW05]. Für das flächenzentriert-kubischen Gitter existiert lediglich das Verfahren von R. Backofen und S. Will [BW05].

Das CPSP-Verfahren ist auf beliebige Gitter anwendbar und in 3 wesentliche Schritte gegliedert:

1. Berechnung einer unteren Energieschranke für eine gegebene Sequenz
2. Konstruktion optimaler und suboptimaler H-Kerne
3. Strukturvorhersage mit Hilfe der H-Kerne

Sowohl die Berechnung der Energieschranken, als auch die rechenaufwändige Konstruktion der H-Kerne, erfordert lediglich die Anzahl von H-Monomeren in der Sequenz. Sie ist von der genauen Beschaffenheit und Länge der HP-Sequenz unabhängig. Dies ermöglicht eine Präprozessierung, bei der optimale und suboptimale H-Kerne verschiedener Größen und ihre Energien vorberechnet werden. Diese einmal generierte Datenbank von H-Kernen kann danach für die Strukturvorhersage verschiedener Sequenzen (Schritt 3) genutzt werden. Schritt 1 und 2 finden hierdurch keine weitere Betrachtung in der Arbeit und die H-Kern-Datenbank wird als gegeben vorausgesetzt. Sie ermöglicht einen direkten Einstieg in Schritt 3, wobei dieser allein im Folgenden vereinfachend als Strukturvorhersage bezeichnet wird.

Definition 1.3.1 (Optimalität von H-Kernen)
Ein optimaler H-Kern *der Größe n ist hierbei eine möglichst kompakte Menge von n H-Monomeren im Gitter, deren Energie gemäß Gleichung 1.9 minimal ist (siehe Abschnitt 1.2).*

Ein suboptimaler H-Kern *hat eine entsprechend höhere Energie und ist weniger kompakt.*

Die HP-Struktur aus Abbildung 1.3 enthält beispielsweise einen suboptimalen H-Kern der Größe 4 (alle schwarzen Gitterpositionen) mit der Energie -2, der für die Strukturvorhersage von Sequenzen mit 4 H's verwendet werden kann. (Der optimale H-Kern der Größe 4 hat eine Energie von -4 und entspricht den Ecken eines Quadrates.)

Da lediglich H-Monomere einen Energiebeitrag liefern, führt die Optimalität des H-Kerns direkt zur optimalen Struktur. Das ist möglich, wenn alle H-Monomere der Sequenz auf Kernpositionen gelegt werden und die Struktur ein SAW ist. Jedoch ist nicht jede Sequenz mit jedem H-Kern kompatibel, da die Ketteneigenschaft der Struktur nicht immer erfüllt werden kann. So ist zum Beispiel die Sequenz „HHHH“ nicht mit dem H-Kern aus Abbildung 1.3 vereinbar, obwohl er die richtige Größe hat (4 H's $\leftrightarrow$ 4 Kernpositionen).

Zur vollständigen Aufzählung aller optimalen Strukturen einer Sequenz ist es also nötig, zunächst sämtliche H-Kerne der entsprechenden Größe in Betracht zu ziehen. Hierbei werden die H-Kerne aufsteigend nach ihrer Energie

geordnet und nacheinander in die Strukturvorhersage einbezogen. Ein Aktivitätendiagramm ist in Abbildung 1.4 dargestellt. Hierdurch wird ersichtlich, dass für die Aufzählung aller optimalen Strukturen einer Sequenz zuerst die optimalen und anschließend immer suboptimalere H-Kerne berücksichtigt werden. Sobald eine Sequenz mit einem H-Kern kompatibel und somit die vorhergesagte Anzahl aller Strukturen mit diesem H-Kern größer Null ist, werden lediglich H-Kerne mit der gleichen Energie weiter betrachtet, um weitere Strukturen mit dieser Energie zu finden. Die Optimalität der generierten Strukturen ist sichergestellt, da die H-Kerne sortiert und auf H-Kernen mit niedrigeren Energien bis dahin keine Lösungen gefunden wurden. Durch eine Initialisierung des Parameters `Max` kann eine obere Schranke für die Energie der Lösungen eingeführt werden. Die Strukturvorhersage nach [BW05] wird im Detail in Kapitel 2 besprochen.

1.4 Zerlegungsstrategien für Constraint Satisfaction Probleme

Das Problem der Strukturvorhersage kann, wie durch R. Backofen und S. Will gezeigt, als Constraint Satisfaction Problem (CSP) formuliert werden. Bei dieser Art der Formulierung wird das Problem durch Variablen mit ihren möglichen Wertebereichen modelliert. Zudem wird eine Menge von Bedingungen zwischen den Variablen aufgestellt, welche durch eine Lösung der CSPs erfüllt sein müssen. Eine formale Definition der Begriffe wird in Kapitel 2 gegeben.

Um CSPs zu lösen, kann als erster Ansatz eine Tiefensuche verwendet werden. Bei dieser wird in jeder Suchverzweigung eine Variable ausgewählt und deren Wertebereich aufgespalten. Das resultierende CSP mit gleicher Variablenanzahl, bei der mindestens ein Wertebereich eingeschränkt ist, wird als *Unterproblem* bezeichnet. Diese Zerlegung in Unterprobleme kann auf vielfältige Art und Weise geschehen. So kann ein Wert ausgewählt (siehe Kapitel 2) oder der Wertebereich in zwei oder mehr Teilbereiche aufgespalten werden [SSHF99]. J. Larrosa führt zum Beispiel eine Zerlegung der Wertebereiche ein, bei der eine Zusammenfassung von Werten in Teilmengen erfolgt, die ähnliche Unterprobleme erzeugen. Die so entstandenen Gruppierungen werden später zum Finden der Lösung wieder aufgehoben. Hierdurch wird die Vermeidung redundanter Arbeit während der Suche ermöglicht, was jedoch zu Lasten der Reduzierbarkeit der Wertebereiche geschieht [Lar97].

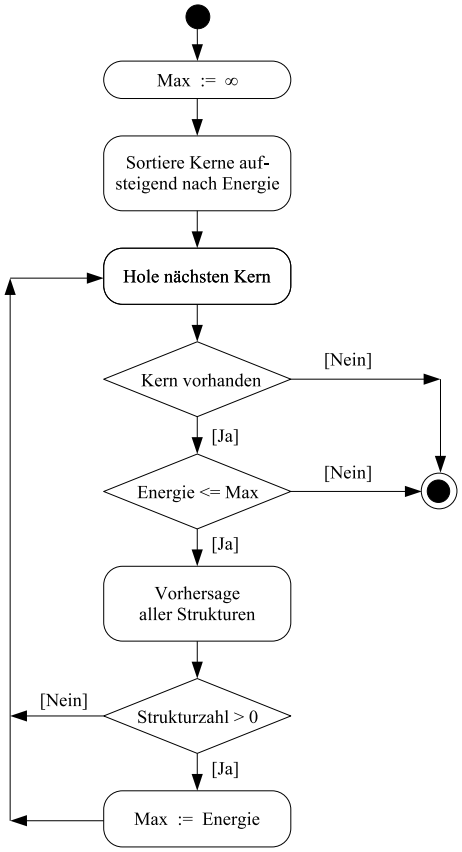

Abbildung 1.4: Aktivitätendiagramm der vollständigen Aufzählung aller optimalen Strukturen nach [BW05].

Verschiedene Abwandlungen der Tiefensuche wie *Maintaining Arc Consistency* (in Kapitel 2 als *Standardsuche* vorgestellt) können somit zur Lösung von CSPs eingesetzt und zum Teil kombiniert werden [Bar99, FH95, RN03].

Andere Herangehensweisen ziehen oftmals die Struktur des CSPs und des damit assoziierten Constraintgraphen zu Rate. Hierauf basierend ist die Einführung einer statischen Zerlegungsreihenfolge in Unterprobleme möglich [JT02]. Desweiteren können Konzepte aus der Datenbanktheorie (z.B. „hypertree decomposition“) oder anderer mit Graphen arbeitenden Gebieten angewendet werden, um strukturelle Eigenschaften von CSPs zu deren Lösung auszunutzen [GLF00, Sam05, GLS02].

Die verschiedenen Ansätze haben gemeinsam, dass sie meist auf statischen

Zerlegungen des CSPs basieren und dieses in Unterprobleme aufspalten. Bei der vollständigen Aufzählung aller Lösungen führt das jedoch in den meisten Fällen zur redundanten Lösung von *Teilproblemen*, die sich dynamisch während der Suche herausbilden (siehe Abschnitt 2.3). Ein Teilproblem umfasst nur eine Teilmenge der Variablen des Ausgangsproblems und die mit ihnen verknüpften Bedingungen. Im Gegensatz dazu beinhaltet ein Unterproblem alle Variablen des initialen Problems.

Die in Kapitel 3 vorgestellte Zerlegungssuche geht bei der Aufspaltung des CSPs andere Wege. Sie stellt einen generellen Ansatz zur vollständigen Aufzählung dar und kann als Erweiterung zu bestehenden Verfahren, wie dem „Maintaining Arc Consistency“ (MAC), verwendet werden. Hierbei wird, wenn möglich, ein CSP während der Suche in disjunkte Teilprobleme aufgespalten. Diese werden dann solange unabhängig voneinander mit einem Verfahren wie dem MAC weiter behandelt, bis sie gelöst oder weitere Zerlegungen in Teilprobleme möglich sind. Die Zerlegbarkeit wird vor jeder Verzweigung in Unterprobleme geprüft und die Teillösungen am Ende zur Gesamtlösung kombiniert. Es findet also eine dynamische Zerlegung auf Basis des aktuellen CSPs statt und keine statische Voreinschätzung. Hierdurch ergibt sich ein neuer Anspruch an die Variablenauswahl und Wertebereichsspaltung, da nur eine frühzeitige Zerlegung in möglichst viele Teilprobleme den vollen Leistungsumfang der Zerlegungssuche ausnutzt. Die Lösungsmengen der Teilprobleme werden während des Backtrackings der Suche durch die Bildung der in Kapitel 3 eingeführten kartesischen Vereinigung zur Gesamtlösungsmenge kombiniert.

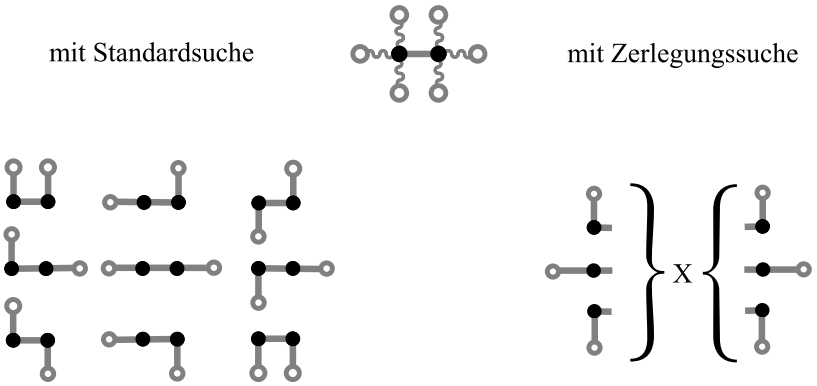

Abbildung 1.5: Erzeugte Lösungsmengen der Standard- und Zerlegungssuche für die Sequenz „PHHP“.

Um den Vorteil und das Ziel der Zerlegungssuche zu verdeutlichen sind in Abbildung 1.5 die gebildeten Lösungsmengen für Standard- und Zerlegungssuche dargestellt. Hierbei sind für die Enden der Strukturen der Sequenz „PHHP“ jeweils drei Gitterpositionen möglich. Im Verlauf der Standardsuche werden nun alle Kombinationen der beiden Enden erzeugt. Die Zerlegungssuche hingegen löst das Problem für jedes Ende unabhängig vom anderen und kombiniert im Anschluss deren mögliche Teillösungen.

Die Verwendung der Zerlegungssuche verhindert die redundante Lösung von disjunkten Teilproblemen, welche bei Verwendung von weitverbreiteten Standard-Suchverfahren wie dem MAC auftritt, die lediglich in Unterprobleme zerlegen. Sie stellt somit ein übergeordnetes Suchverfahren dar.

Kapitel 2

Strukturvorhersage als Constraint Satisfaction Problem

Um optimale Strukturen einer gegebenen HP-Sequenz im Gitter vorherzusagen, formulierten R. Backofen und S. Will das Problem als ein Constraint Satisfaction Problem [BW05].

2.1 Finite Constraint Satisfaction Problem und Constraint-Programmierung

Definition 2.1.1 (Finite Constraint Satisfaction Problem)
Ein Finite Constraint Satisfaction Problem (CSP) *ist ein Tripel* $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ *bestehend aus*

- *einer Menge von Variablen* $\mathcal{X} = \{X_1, \ldots, X_n\}$,
- *einer Menge von endlichen Wertebereichen* $\mathcal{D} = \{D_1, \ldots, D_n\}$, *wobei* $D_i = D(X_i, \mathcal{D})$ *den zu* $X_i \in \mathcal{X}$ *gehörenden Wertebereich aus* $\mathcal{D}$ *liefert,*
- *einer Menge von Bedingungen* $\mathcal{C}$, *welche die Abhängigkeiten der Variablen untereinander beschreiben.*
  Die Teilmenge der Variablen $\{X_j, \ldots, X_k\} \subseteq \mathcal{X}$, *die durch eine Bedingung* $C \in \mathcal{C}$ *beschränkt werden, wird verkürzt als* $X(C)$ *beschrieben.*

Eine Bedingung C, für die $|X(C)| = 2$ gilt, wird als binäre Bedingung bezeichnet. Enthält ein CSP nur binäre Bedingungen in $\mathcal{C}$, wird es auch als *binäres CSP* bezeichnet.

Um CSPs zu visualisieren bedient man sich im Allgemeinen eines Constraintgraphen. Viele Techniken zur Lösung von CSPs basieren auf dieser Graphendarstellung und dessen Struktur.

Definition 2.1.2 (Constraintgraph)
Ein Constraintgraph $G = (V, E, \mu)$ *für ein binäres CSP* $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ *ist ein ungerichteter Graph mit folgenden Eigenschaften:*

- $V = \{v_1, \ldots, v_n\}$ *ist eine Menge von Knoten, wobei jeder Knoten* v_i *einer Variablen* X_i *aus* $\mathcal{X} = \{X_1, \ldots, X_n\}$ *zugeordnet ist.*
- $E \subseteq \{\{v_j, v_k\} \mid v_j, v_k \in V \wedge v_j \neq v_k\}$ *ist eine Menge von Kanten für die gilt:* $\{v_j, v_k\} \in E \iff \exists_{C \in \mathcal{C}} : X(C) = \{X_j, X_k\}$.
- $\mu : V \cup E \to T$ *markiert Knoten und Kanten mit den Beschriftungen* $\mu(v_i) = (X_i, D_i)$ *und* $\mu(\{v_j, v_k\}) = \{C_l \mid C_l \in \mathcal{C} \wedge X(C_l) = \{X_j, X_k\}\}$

Um diese Definition zu veranschaulichen, ist in Abbildung 2.1 der Constraintgraph für ein CSP mit den Variablen $A \in \{3, 5\}, B \in \{3, 4\}, C \in \{1, 2\}$ und $D \in \{1, 2\}$ sowie den 4 Bedingungen $A \not\equiv B, B \not\equiv C, C \not\equiv D$ und $D \not\equiv A$ gezeigt. Auf der rechten Seite dieser Grafik sieht man eine vereinfachte Form der Darstellung, bei der die Wertebereiche D_i in die Knoten eingetragen wurden. Zudem ist die Markierung der Kanten verkürzt und zu einer Markierung zusammengefasst worden, da nur Ungleichheitsbedingungen vorhanden sind.

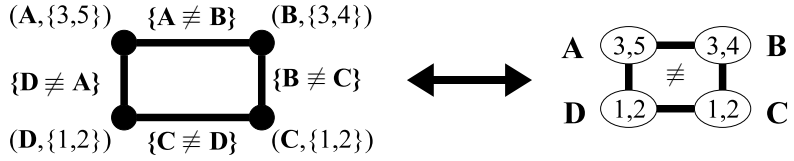

Abbildung 2.1: links: Beispiel-Constraintgraph; rechts: vereinfachte Darstellung.

Diese Art der Darstellung ist auf binäre CSPs beschränkt. Es existieren jedoch verschiedene Möglichkeiten der Binärisierung, um n-näre[1] Bedingungen auf Paare von darin enthaltenen Variablen abzubilden. So zum Beispiel die Methode des *Dual encodings* oder des *Hidden variable encodings* [Bar01]. Dabei muss jedoch auf die Lösungsäquivalenz des neu entstandenen CSPs

[1]Für eine *n-näre* Bedingung C gilt $|X(C)| > 2$.

geachtet werden [RD90]. Aufgrund dieser Binärisierungsmöglichkeiten wird im Folgenden ohne Beschränkung der Allgemeinheit von einem binären CSP ausgegangen.

Definition 2.1.3 (Binden einer Variablen)
Eine Variable $X_i \in \mathcal{X}$ des CSPs $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ mit dem zugehörigen Wertebereich $D_i = D(X_i, \mathcal{D})$ wird als gebunden *bezeichnet, wenn gilt $|D_i| = 1$. Dies wird verkürzt durch das Tupel (X_i, d_i) mit $d_i \in D_i$ und beschrieben.*

Die Bindung aller Variablen eines CSPs wird als *Variablenbelegung* bezeichnet.

Definition 2.1.4 (Variablenbelegung eines CSPs)
Eine Variablenbelegung *des CSPs $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ mit $X_i \in \mathcal{X}$ und $D_i = D(X_i, \mathcal{D})$ ist eine Menge A von Tupeln (X_i, d_i) für die gilt*

1. *A enthält genau ein Tupel (X_i, d_i) für jede Variable $X_i \in \mathcal{X}$.*
2. *Ein Tupel $(X_i, d_i) \in A$ entspricht der Bindung von X_i auf den Wert $d_i \in D_i$.*

Anmerkung: Für das CSP $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ mit $\mathcal{X} = \{X_1, \ldots, X_n\}$ und den zugeordneten $d_i \in D_i = D(X_i, \mathcal{D})$ ist

$$A = \{(X_1, d_1), \ldots, (X_n, d_n)\}. \tag{2.1}$$

Hieraus folgt für A :

$$\forall_{X_i \in \mathcal{X}} \quad : \quad \exists (X_i, d_i) \in A \tag{2.2}$$

$$\forall_{(X_j, d_j), (X_k, d_k) \in A} \quad : \quad X_j \neq X_k \tag{2.3}$$

Die Menge aller Variablenbelegungen bilden den *Suchraum* eines CSPs.

Definition 2.1.5 (Suchraum eines CSPs)
Der Suchraum $SP^{(\mathcal{X}, \mathcal{D}, \mathcal{C})}$ *eines CSPs $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ ist definiert als die Menge*

$$SP^{(\mathcal{X}, \mathcal{D}, \mathcal{C})} = \{A \mid A \text{ ist Variablenbelegung von } (\mathcal{X}, \mathcal{D}, \mathcal{C})\} \tag{2.4}$$

Da Bedingungen meist nur einen Teil der Variablen aus $\mathcal{X}$ beschränken, muss nur ein *Ausschnitt* einer Variablenbelegung betrachtet werden, um festzustellen ob diese eine *Bedingung erfüllt*.

Definition 2.1.6 (Ausschnitt einer Variablenbelegung)
Der Ausschnitt $E^A_{\mathcal{X}'}$ *einer Variablenbelegung A eines CSPs $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ der auf die Menge von Variablen $(\mathcal{X}' \cap \mathcal{X})$ beschränkt ist, entspricht*

$$E^A_{\mathcal{X}'} = \{(X_j, d_j) \mid (X_j, d_j) \in A \wedge X_j \in \mathcal{X}'\} \tag{2.5}$$

Definition 2.1.7 (Bedingung durch Variablenbelegung erfüllt)
Eine Bedingung $C \in \mathcal{C}$ eines CSPs $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ mit $X(C) \subseteq \mathcal{X}$ wird durch die Variablenbelegung A erfüllt, *wenn die Variablenbindungen des Ausschnitts $E^A_{X(C)}$ mit C vereinbar sind. Dies wird verkürzt als $(A \models C)$ beschrieben.*

Im Fall $E^A_{X(C)} = \emptyset$ gilt $(A \models C)$.

Wenn alle Bedingungen aus $\mathcal{C}$ durch eine Variablenbelegung $A \in SP^{(\mathcal{X},\mathcal{D},\mathcal{C})}$ erfüllt sind, wird diese *Lösung des CSPs* genannt.

Definition 2.1.8 (Lösung eines CSPs)
Eine Variablenbelegung $L \in SP^{(\mathcal{X},\mathcal{D},\mathcal{C})}$ für die gilt

$$\forall_{C \in \mathcal{C}} : L \models C, \tag{2.6}$$

ist Lösung *für ein CSP $(\mathcal{X}, \mathcal{D}, \mathcal{C})$. Die Menge aller Lösungen eines CSPs wird als* Lösungsmenge $\mathcal{L}$ *bezeichnet und es folgt $\mathcal{L} \subseteq SP^{(\mathcal{X},\mathcal{D},\mathcal{C})}$.*

Zwei CSPs P^1 und P^2 sind *lösungsäquivalent*, wenn für ihre Lösungsmengen $\mathcal{L}^1$ und $\mathcal{L}^2$ gilt: $\mathcal{L}^1 = \mathcal{L}^2$.
Falls der Suchraum eines CSPs nur noch Variablenbelegungen enthält welche eine Bedingung $C \in \mathcal{C}$ erfüllen, so wird C als *domain-entailed* bezeichnet.

Definition 2.1.9 (Domain-Entailment einer Bedingung)
Wenn für eine Bedingung $C \in \mathcal{C}$ des CSPs $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ mit dem Suchraum $SP^{(\mathcal{X},\mathcal{D},\mathcal{C})}$ gilt

$$\forall_{A \in SP^{(\mathcal{X},\mathcal{D},\mathcal{C})}} : A \models C, \tag{2.7}$$

so ist diese domain-entailed.

Wenn eine Bedingung $C \in \mathcal{C}$ aus dem CSP $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ domain-entailed ist, kann sie somit keinen der an ihr beteiligten Wertebereiche weiter einschränken. Hierdurch kann C aus $\mathcal{C}$ entfernt werden und man erhält ein neues, lösungsäquivalentes CSP $(\mathcal{X}, \mathcal{D}, \mathcal{C}')$ mit $\mathcal{C}' \subset \mathcal{C}$. Die Menge der Kanten E des zugehörigen Constraintgraphen verringert sich unter Umständen um die entsprechende Kante (siehe Constraintgraphen Definition 2.1.2). Die Entfernung domain-entailter Bedingung aus $\mathcal{C}$ wird im Folgenden als gegeben vorausgesetzt.

Um alle Lösungen zu finden, ist normalerweise die Betrachtung des gesamten Suchraum nötig, wobei auch Teilräume ohne Lösungen durchsucht werden. Das Ziel der *Constraint-Programmierung* ist es, diese unnötigen Teilsuchen zu vermeiden und Lösungen möglichst effizient und schnell zu finden. Dieser Vorgang wird auch als *Enumeration* oder Aufzählung bezeichnet. Er

entspricht in einem einfachen Fall dem im Algorithmus 2.1 angegebenen Pseudocode. Bei diesem Ansatz wird das CSP durch Binden einer Variable an einen Wert bzw. Entfernung des Wertes aus dem Wertebereich in *Unterprobleme* (subproblems) aufgespalten. Danach werden Werte, die Bedingungen verletzen, aus den Wertebereichen entfernt. Wenn keine Lösung gefunden und keine Inkonsistenz mit einer Bedingung festgestellt wurde, wiederholt sich dies rekursiv.

Dieses als *Maintaining Arc Consistency* [Bar99, Bar01] bekannte Verfahren wird im Folgenden als *Standardsuche* bezeichnet, da es die Grundlage der von R. Backofen und S. Will eingeführten Strukturvorhersagemethode darstellt. Sie ist ein weit verbreiteter Ansatz zur CSP-Lösung und dient der in Kapitel 3 vorgestellten Zerlegungssuche als Basis und Vergleich.

Definition 2.1.10 (Unterproblem eines CSPs)
Ein Unterproblem $(\mathcal{X}', \mathcal{D}', \mathcal{C}')$ *des CSPs* $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ *beschreibt einen Teil des Suchraums* $SP^{(\mathcal{X},\mathcal{D},\mathcal{C})}$. *Es gilt* $\mathcal{X}' = \mathcal{X}, \mathcal{D}' \subset \mathcal{D}$ *und* $\mathcal{C}' \subseteq \mathcal{C}$ *(*$\mathcal{D}' \subset \mathcal{D}$ *steht für* $\exists_{X_i \in \mathcal{X}} : D(X_i, \mathcal{D}') \subset D(X_i, \mathcal{D})$*).*

Algorithmus 2.1 Enumeration durch Standardsuche

```
 1: procedure ENUMERATION(X, D, C)
                                          ▷ verkleinere Wertebereiche
 2:     (D', C') ← WERTEBEREICHSREDUKTION(X, D, C)
                                          ▷ prüfe, ob Blatt im Rek.-Baum
 3:     if BEDINGUNGVERLETZT(X, D', C') then
 4:         return
 5:     else if ALLEVARIABLENGEBUNDEN(X, D') then
 6:         BEHANDLELÖSUNG(X, D')
 7:         return
 8:     else
                                          ▷ Suche aufspalten
 9:         (X_i, d_i) ←WÄHLEVARIABLEUNDWERT(X, D')
10:         ENUMERATION( X, D', C' ∪ (X_i ≡ d_i))
11:         ENUMERATION( X, D', C' ∪ (X_i ≢ d_i))
12:     end if
13: end procedure
```

Die Kommentare in Algorithmus 2.1 verdeutlichen die Einteilung der Enumeration durch Standardsuche in 3 Schritte:

1. **Reduktion der Wertebereiche:** Hierbei werden die die Bedingungen verletzenden Werte aus den Wertebereichen entfernt. Dieses wird auch als *Constraint-Propagation* bezeichnet und die Effizienz der Suche hängt vom Umfang dieser Reduktion ab. Ist eine Bedingung domain-entailed und ermöglicht somit keine weiteren Einschränkungen der Wertebereiche, wird diese aus $\mathcal{C}$ entfernt.

2. **Blattprüfung:** Ist eine Bedingung verletzt, kann in diesem Suchzweig keine Lösung mehr gefunden und die Suche abgebrochen werden. Beim Finden einer Lösung muss diese behandelt werden. In beiden Fällen ist ein Blatt im Rekursions-Suchbaum erreicht.

3. **Suchverzweigung:** Wenn keine Bedingung verletzt, aber auch keine Lösung gefunden wurde, wird die Suche in zwei disjunkte Suchen aufgespalten. Hierzu wird eine Variable X_i aus $\mathcal{X}$ und ein Wert d_i des entsprechenden Wertebereichs D_i aus $\mathcal{D}'$ ausgewählt und die Menge $\mathcal{C}'$ der rekursiven Suchen um $(X_i \equiv d_i)$ bzw. $(X_i \not\equiv d_i)$ ergänzt. Das CSP wird hierdurch in zwei Unterprobleme aufgespalten, die weiter nach Lösungen durchsucht werden.

Die in Schritt 3 eingeführten unären[2] Bedingungen können direkt durch Wertebereichsreduktion domain-entailed und danach wieder aus $\mathcal{C}$ entfernt werden. Hierbei wird der Wertebreich D_i bei $(X_i \equiv d_i)$ auf $\{d_i\}$ und für $(X_i \not\equiv d_i)$ um d_i reduziert.

Die effiziente Propagation aller Wertebereiche stellt ein zentrales Problem der Constraintprogrammierung dar. Ziel ist es hierbei, eine sogenannte *Kantenkonsistenz (arc-consistency)* für alle Bedingungen zu erreichen. Hierfür existieren verschiedenste Ansätze (z.B. AC-3-Algorithmus) [Bar01]. Im Folgenden wird von einer effizienten und vollständigen Wahrung der globalen Kantenkonsistenz des CSPs ausgegangen.

Definition 2.1.11 (Kantenkonsistenz)
Eine Bedingung $C \in \mathcal{C}$ des CSPs $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ ist kantenkonsistent*, wenn gilt*

$$\forall_{X_i \in X(C)} \forall_{d_i \in D(X_i, \mathcal{D})} \exists_{A \in SP^{(\mathcal{X}, \mathcal{D}, \mathcal{C})}} : (X_i, d_i) \in A \wedge A \models C \tag{2.8}$$

Ein CSP ist hierdurch lokal kantenkonsistent. Es wird global kantenkonsistent, wenn gilt

$$\forall_{C \in \mathcal{C}} : C \text{ ist kantenkonsistent} \tag{2.9}$$

[2]Für eine *unäre* Bedingung C gilt $|X(C)| = 1$.

Zur Visualisierung der Enumeration eignet sich ein *Suchbaum* der Rekursion. Das *Ausgangsproblem* der Suche wird im Folgenden auch als *initiales CSP* bezeichnet. Der Suchbaum weist folgende Besonderheiten auf:

- Die Wurzel des Baumes entspricht dem initialen CSP.
- Innere Knoten zeigen Unterprobleme des initialen CSPs.
- Kanten werden mit den neu eingefügten Bedingungen markiert. Sie symbolisieren die Wertebereichsreduktion während der Suche.
- Blätter stehen für Lösungen oder Bedingungsverletzungen und somit den Suchabbruch in diesem Rekursionszweig.
- Jedes CSP (Knoten) wird in Form eines Constraintgraphen dargestellt.

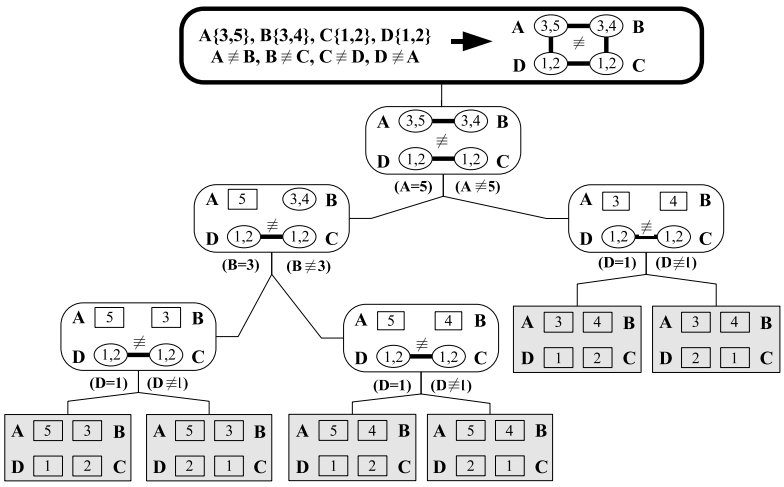

Abbildung 2.2: Suchbaum einer Standardsuche.

Ein Suchbaum für das schon zuvor eingeführte CSP mit den Variablen $A \in \{3,5\}, B \in \{3,4\}, C \in \{1,2\}$ und $D \in \{1,2\}$ sowie den vier Bedingungen $A \not\equiv B, B \not\equiv C, C \not\equiv D$ und $D \not\equiv A$ ist in Abbildung 2.2 zu sehen. Das jeweilige Unterproblem (Knoten) ist mit den eingeschränkten Wertebereichen $\mathcal{D}'$ nach dem Reduktionsschritt als Constraintgraph dargestellt. Lösungen und gebundene Variable sind rechteckig hervorgehoben.

Definition 2.1.12 (Suchstrategie)
Im Folgenden wird die Strategie, nach der die Variablen- und Werte-Auswahl während einer Suche stattfindet, als Suchstrategie *bezeichnet.*

Die Suchstrategie ist maßgebend für die Effizienz der Enumeration. In der hier vorgestellten Form ist die Standardsuche unabhängig davon, ob diese dynamisch oder durch eine statische Reihenfolge festgelegt wird. Im Folgenden wird jedoch von einer dynamischen Auswahl ausgegangen. Obwohl die genaue Suchstrategie problemspezifisch ist, existieren problemunabhängige Herangehensweisen. Da zum Beispiel die Wertebereichsgröße eine obere Schranke für die nötige Anzahl Suchverzweigungen mit der entsprechenden Variable ist, wird zumeist die Variable mit dem kleinsten Wertebereich gewählt. Für die Wertauswahl lassen sich jedoch keine generellen Aussagen treffen.

2.2 Strukturvorhersage-CSP-Formulierung

Das in Kapitel 1 vorgestellte Gitter-Strukturvorhersageproblem kann als CSP $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ formuliert werden. Hierfür umfasst die endliche Menge von Gitterpunkten F den Raumausschnitt, in dem die Strukturfaltung stattfindet. $HD \subset F$ entspricht den ins Zentrum von F verschobenen H-Kern-Positionen ($PD = F \backslash HD$). Das CSP für die Strukturvorhersage *(SP-CSP)* nach [BW05] für eine HP-Sequenz S der Länge n (mit gegebenem optimalen H-Kern) definiert sich wie folgt

- Für jede Sequenzposition S_i wird eine Variable X_i eingeführt.
- Der Wertebereich $D_i = \begin{cases} HD & \text{wenn } S_i = H \\ PD & \text{sonst} \end{cases}$ für alle $1 \leq i \leq n$
- $\mathcal{C} = \{Neighbor(X_i, X_{i+1}) : 1 \leq i < n\} \cup \{(X_k \not\equiv X_l) : k < l\}$
  (Nachbarschaftsbedingungen für Ketteneigenschaft sowie Ungleichheits-Bedingungen für Selbstvermeidung der Struktur)

Hierbei wurde die n-näre Self-Avoiding-Walk Bedingung durch eine Reihe von binären Ungleichheits- und Nachbarschaftsbedingungen ersetzt. Diese liefern eine lösungsäquivalente, binäre Formulierung und sind einfacher zu berechnen und zu prüfen [Bar99, BW05].

Im Folgenden werden die Variablen für H-Monomere als *H-Variablen* und für P-Monomere entsprechend als *P-Variablen* bezeichnet. Aufgrund ihrer Initialisierung sind die Wertebereiche von H- und P-Variablen disjunkt, wodurch die zwischen ihnen eingeführten Ungleichheitsbedingungen keine Reduktion ermöglichen. Da die Nachbarschaftsbedingung die Ungleichheit bein-

haltet (keine Gitterposition ist ihr eigener Nachbar), müssen auch für aufeinanderfolgende Sequenzpositionen keine zusätzlichen Ungleichheitsbedingungen eingefügt werden. Man kann diese bei der Initialisierung von $\mathcal{C}$ durch die zusätzliche Prämisse $(l - k \neq 1)$ vermeiden und zudem Ungleichheitsbedingungen nur zwischen Variablen gleichen Typs einführen $(S_k = S_l)$. Hieraus resultiert die folgende erweiterte Definition von $\mathcal{C}$:

$$\begin{aligned} \mathcal{C} &= \{Neighbor(X_i, X_{i+1}) : 1 \leq i < n\} \\ &\quad \cup \{(X_k \not\equiv X_l) : (k < l) \wedge (l - k \neq 1) \wedge (S_k = S_l)\} \end{aligned}$$

Diese Formulierung vorausgesetzt ergibt sich für die Sequenz PHPHHP der in Abbildung 2.3 dargestellte Constraintgraph. Variablen für H-Positionen wurden grau unterlegt.

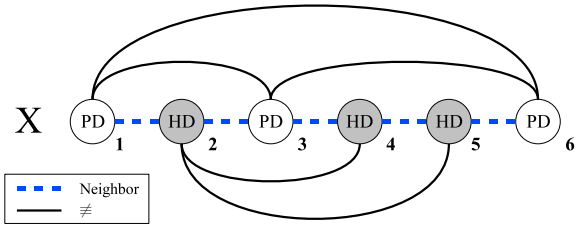

Abbildung 2.3: Constraintgraph des SP-CSPs für die Sequenz PHPHHP.

Eine Lösung dieses SP-CSPs entspricht einer Struktur gemäß Kapitel 1.2, da jeder Sequenzposition eine Gitterposition zugewiesen wird und die Self-Avoiding-Walk Bedingung erfüllt ist. Ihre Energie ist aufgrund der Optimalität des H-Kerns global optimal.

2.3 Redundanz der Suche

Die standardmäßige Aufzählung aller Lösungen eines CSPs findet diese mit einer optimalen Suchstrategie zwar möglichst direkt, arbeitet aber redundant. Das geschieht immer dann, wenn ein CSP *unabhängige Teilprobleme* enthält und wird schon am kleinen Beispiel aus Kapitel 2.1 deutlich. Wie man in Abbildung 2.2 sieht, wird das Teilproblem ($\{1,2\} \ni C \not\equiv D \in \{1,2\}$) im Verlauf der Suche drei mal gelöst, nachdem zuvor das erste Teilproblem ($\{3,5\} \ni A \not\equiv B \in \{3,4\}$) gelöst wurde.

Definition 2.3.1 (Teilproblem eines CSPs)
Ein Teilproblem *des CSPs* $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ *ist hierbei das CSP* $(\mathcal{X}', \mathcal{D}', \mathcal{C}')$, *mit* $\mathcal{X}' \subset \mathcal{X}$ $(\mathcal{X}' \neq \emptyset)$ *und der entsprechenden Menge* $\mathcal{D}' = \{D(X_i', \mathcal{D}) \mid X_i' \in \mathcal{X}'\}$, *sowie* $\mathcal{C}' = \{C \mid X(C) \subseteq \mathcal{X}'\}$.

Definition 2.3.2 (Teillösung eines CSPs)
Die Lösung $L' \in SP^{(\mathcal{X}', \mathcal{D}', \mathcal{C}')}$ *des Teilproblems* $(\mathcal{X}', \mathcal{D}', \mathcal{C}')$ *eines CSPs* $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ *mit*

$$\forall_{C' \in \mathcal{C}'} : L' \models C' \tag{2.10}$$

und wird als Teillösung *von* $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ *bezeichnet. Die Menge aller dieser Teillösungen wird entsprechend als* Teillösungsmenge $\mathcal{L}'$ *geführt.*

Zwei Teilprobleme eines CSPs, die keine gemeinsamen Variablen aus $\mathcal{X}$ beschreiben und für die es keine Bedingung aus $\mathcal{C}$ gibt, die Variablen aus beiden Teilproblemen beschränkt, werden als unabhängig oder disjunkt bezeichnet.

Definition 2.3.3 (Unabhängigkeit von Teilproblemen)
Zwei Teilprobleme P^1 *und* P^2 *mit* $(\mathcal{X}^k, \mathcal{D}^k, \mathcal{C}^k)$ $(k \in \{1,2\})$ *des CSPs* P *mit* $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ *sind hierbei* unabhängig/disjunkt, *wenn gilt:*

- $\mathcal{X}^1 \cap \mathcal{X}^2 = \emptyset$
- *für alle* $C \in \mathcal{C}$ *mit* $(X(C) \cap \mathcal{X}^1 \neq \emptyset) \wedge (X(C) \cap \mathcal{X}^2 \neq \emptyset)$ *gilt* C *ist domain-entailed nach Definition 2.1.9.*

Eine bessere Aufzählungsstrategie wäre es, die unabhängigen Teilprobleme getrennt zu lösen und die Gesamtlösungen aus den Teillösungen zu generieren.

Diese Herangehensweise wird im folgenden als *Zerlegungssuche* bezeichnet und in Kapitel 3 eingehend beschrieben.

Kapitel 3

Zerlegungssuche am Beispiel des Strukturvorhersage-CSPs

Wie im Abschnitt 2.3 beschrieben, werden bei der Standardsuche unabhängige Teilprobleme mehrfach gelöst. Diese Redundanz kann durch Anwendung der *Zerlegungssuche* verhindert werden.

Bei der Zerlegungssuche wird vor einer Suchaufspaltung geprüft, ob sich das CSP in disjunkte Teilprobleme zerlegen lässt. Ist dies der Fall, werden die Teilprobleme unabhängig voneinander betrachtet und ihre Teillösungen im Anschluss zu Gesamtlösungen des CSPs kombiniert. Diese Nachprozessierung kann durch eine *kartesische Vereinigung* der Lösungsmengen der Teilprobleme erreicht werden, wie im Folgenden gezeigt wird.

Definition 3.1.1 (Disjunkte Partialzerlegung)
Die Aufspaltung eines CSPs P mit $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ in k disjunkte Teilprobleme P^k mit $(\mathcal{X}^k, \mathcal{D}^k, \mathcal{C}^k)$ wird disjunkte Partialzerlegung *genannt, wenn gilt*

$$1) \quad \mathcal{X} = \bigcup \mathcal{X}^k \tag{3.1}$$

$$2) \quad \mathcal{C} = \bigcup \mathcal{C}^k \cup \{C \mid C \in \mathcal{C} \wedge C \text{ ist domain-entailed}\} \tag{3.2}$$

$$3) \quad \text{alle } P^k \text{ sind paarweise unabhängig} \tag{3.3}$$

Definition 3.1.2 (Kartesische Vereinigung)
Die kartesische Vereinigung *zweier Mengen über Mengen A und B (geschrieben $A \otimes B$) liefert eine Menge über Mengen:*

$$(A \otimes B) = \{a \cup b \mid a \in A \wedge b \in B\} = (B \otimes A) \tag{3.4}$$

$$(A \otimes \emptyset) = \emptyset = (\emptyset \otimes A) \tag{3.5}$$

Die kartesische Vereinigung der Mengen $A_1, \ldots, A_n$ wird verkürzt geschrieben als

$$\bigotimes_{1 \leq i \leq n} A_i = A_1 \otimes \ldots \otimes A_n \tag{3.6}$$

Satz 1
Die Menge aller Lösungen $\mathcal{L}$ eines CSPs $P = (\mathcal{X}, \mathcal{D}, \mathcal{C})$, das sich durch disjunkte Partialzerlegung in k Teilprobleme $P^k = (\mathcal{X}^k, \mathcal{D}^k, \mathcal{C}^k)$ aufspalten lässt, entspricht der kartesischen Vereinigung seiner Teillösungsmengen $\mathcal{L}^k$.

Beweis 1
*Um Satz 1 zu beweisen, muss gezeigt werden, dass **I)** die kartesische Vereinigung ausschliesslich Elemente des Suchraums von P erzeugt und **II)** alle Lösungen von P in der kartesischen Vereinigung enthalten sind. Es genügt, dies für den Fall einer disjunkten Partialzerlegung des CSPs P in zwei Teilprobleme P^1 und P^2 zu beweisen, da sich die kartesische Vereinigung aller $\mathcal{L}^k$ rekursiv zusammensetzt.*

I) Die kartesische Vereinigung der Teillösungen erzeugt nur Lösungen von P

Hierfür ist zu zeigen, dass die kartesische Vereinigung der Teillösungsmengen nur Elemente des Suchraums $SP^{(\mathcal{X},\mathcal{D},\mathcal{C})}$ von P erzeugt und dass diese alle Bedingungen $C \in \mathcal{C}$ erfüllen.

Zuerst soll bewiesen werden, dass die kartesische Vereinigung $\mathcal{L}^1 \otimes \mathcal{L}^2$ mit $\mathcal{L}^k \subseteq SP^{(\mathcal{X}^k,\mathcal{D}^k,\mathcal{C}^k)}$ des Teilproblems P^k nur Variablenbelegungen aus $SP^{(\mathcal{X},\mathcal{D},\mathcal{C})}$ von P erzeugt.
Eine Lösung $L^k \in \mathcal{L}^k \subseteq SP^{(\mathcal{X}^k,\mathcal{D}^k,\mathcal{C}^k)}$ des Teilproblems $P^k = (\mathcal{X}^k, \mathcal{D}^k, \mathcal{C}^k)$ ist eine Variablenbelegung für P^k, die alle Bedingungen $C \in \mathcal{C}$ erfüllt. Letzteres wird im Folgenden vorerst vernachlässigt und L^k lediglich als Variablenbelegung behandelt, sodass mit $\mathcal{X}^k = \{X_1^k, \ldots, X_m^k\}$ und $d_i^k \in D_i = D(X_i^k, \mathcal{D}^k)$ nach Definition 2.1.4 gilt:

$$L^k = \{(X_1^k, d_1^k), \ldots, (X_m^k, d_m^k)\} \tag{3.7}$$

Somit gilt für L^k

$$\begin{aligned} \forall_{X_i^k \in \mathcal{X}^k} &: \exists (X_i^k, d_i^k) \in L^k \\ \forall_{(X_i^k, d_i^k), (X_j^k, d_j^k) \in L^k} &: X_i^k \neq X_j^k \end{aligned}$$

Daraus, aus der Unabhängigkeit der Teilprobleme ($\mathcal{X}^1 \cap \mathcal{X}^2 = \emptyset$) *und Gleichung 3.1* ($\mathcal{X} = \bigcup \mathcal{X}^k$) *folgt für die Elemente* ($L = L^1 \cup L^2$) $\in (\mathcal{L}^1 \otimes \mathcal{L}^2)$

$$\begin{aligned} \forall_{X_i \in \mathcal{X}} &: \exists (X_i, d_i) \in L \\ \forall_{(X_i,d_i),(X_j,d_j) \in L} &: X_i \neq X_j \end{aligned}$$

Hieraus folgt, $L \in (\mathcal{L}^1 \otimes \mathcal{L}^2)$ *ist eine Variablenbelegung für* P*. Also erzeugt die kartesische Vereinigung von Variablenbelegungen für Teilprobleme, die durch disjunkte Partialzerlegung entstanden sind, nur Variablenbelegungen für das Gesamtproblem. Es gilt*

$$\forall_{L \in (\mathcal{L}^1 \otimes \mathcal{L}^2)} : L \in SP^{(\mathcal{X},\mathcal{D},\mathcal{C})} \tag{3.8}$$

Nun muss noch gezeigt werden, dass die Elemente der kartesischen Vereinigung der Teillösungsmengen $\mathcal{L}^k$ *der Teilprobleme* P^1 *und* P^2 *alle Bedingungen aus* $\mathcal{C}$ *erfüllen und dadurch Lösungen von* P *sind.*

Aus den Definitionen 2.3.1 bis 2.3.3 folgt für alle Teillösungen $L^k \in \mathcal{L}^k$ *des Teilproblems* $(\mathcal{X}^k, \mathcal{D}^k, \mathcal{C}^k)$ *aus* $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ *dass*

1) $\mathcal{C}^k = \{C \mid X(C) \subseteq \mathcal{X}^k\}$
2) $\forall_{C^k \in \mathcal{C}^k} : L^k \models C^k$
3) $\mathcal{X}^k \cap \mathcal{X}^l = \emptyset$ für $k \neq l$
4) $\forall_{C \in \mathcal{C}}$ mit $(X(C) \cap \mathcal{X}^k \neq \emptyset) \wedge (X(C) \cap \mathcal{X}^l \neq \emptyset)$
   gilt C is domain-entailed ($L^k \models C$)

Hieraus und aus Definition 2.1.7 folgt

$$\forall_{C \in \mathcal{C}} : L^k \models C \tag{3.9}$$

Für ($L = L^1 \cup L^2$) *gilt folglich durch Gleichung 3.2*

$$\forall_{C \in \mathcal{C}} : L_j \models C \tag{3.10}$$

Da L *Element von* $(\mathcal{L}^1 \otimes \mathcal{L}^2)$ *ist und alle Bedingungen aus* $\mathcal{C}$ *erfüllt folgt* $L \in \mathcal{L}$*, weil* L *eine Lösung des Problems* P *ist.*

Die kartesische Vereinigung $\bigotimes_{1 \leq a \leq k} \mathcal{L}^a$ *erzeugt somit ausschließlich Lösungen von* P*.*

II) Alle Lösungen von P sind in der kartesischen Vereinigung seiner Teillösungen enthalten

Es gilt zu zeigen, dass jede Lösung $L \in \mathcal{L}$ von P auch in $(\mathcal{L}^1 \otimes \mathcal{L}^2)$ vorhanden ist, wenn sich P durch disjunkte Partialzerlegung in P^1 und P^2 aufteilen lässt.

Aufgrund der disjunkten Partialzerlegung gilt

$$\forall_{L \in \mathcal{L}} \exists_{L^k \in \mathcal{L}^k} : L^k = E^L_{\mathcal{X}^k}$$

Somit gilt durch $(\mathcal{X} = \bigcup \mathcal{X}^k)$ und $(\mathcal{X}^1 \cap \mathcal{X}^2 = \emptyset)$

$$\forall_{L \in \mathcal{L}} \exists_{L^1 \in \mathcal{L}^1} \exists_{L^2 \in \mathcal{L}^2} : L = (L^1 \cup L^2) \in \mathcal{L}^1 \otimes \mathcal{L}^2$$

Also sind alle Lösungen aus $L \in \mathcal{L}$ in der kartesischen Vereinigung $\bigotimes_{1 \leq a \leq k} \mathcal{L}^a$ vorhanden.

□

Eine disjunkte Partialzerlegung entspricht der Aufteilung des assoziierten Constraintgraphen in disjunkte Teilgraphen, seine Zusammenhangskomponenten. Somit lässt sich umgekehrt aus der Zerlegbarkeit des Constraintgraphen die Möglichkeit einer disjunkten Partialzerlegung ableiten. Für die Identifikation von Zusammenhangskomponenten kann auf einen linearen Algorithmus aus der Graphentheorie zurückgegriffen werden. Auch die später erwähnte Suche nach Artikulationspunkten basiert auf dem Constraintgraphen und seiner strukturellen Äquivalenz zum dargestellten CSP.

Lemma 1
Seien $\mathcal{A}$ und $\mathcal{B}$ Mengen von Mengen mit $\forall_{A \in \mathcal{A}} \forall_{B \in \mathcal{B}} : A \cap B = \emptyset$.
Dann gilt $|\mathcal{A} \times \mathcal{B}| = |\mathcal{A} \otimes \mathcal{B}|$, wobei $\mathcal{A} \times \mathcal{B}$ für das kartesische Produkt der Mengen steht.

Beweis 2
Gegeben sind $\mathcal{A}$ und $\mathcal{B}$ aus Lemma 1. $|\mathcal{A} \times \mathcal{B}| = |\mathcal{A} \otimes \mathcal{B}|$ folgt aus der Bijektivität der Abbildung

$$f : \mathcal{A} \times \mathcal{B} \to \mathcal{A} \otimes \mathcal{B}, \quad (A, B) \to A \cup B \tag{3.11}$$

Somit muss für die Funktion f gezeigt werden, dass sie injektiv und surjektiv ist.

Für die Injektivität von f muss gezeigt werden, dass jedes Element der Zielmenge $A \cup B \in \mathcal{A} \otimes \mathcal{B}$ nur einmal als Funktionswert von f angenommen wird. Es muss also gelten

$$(A \cup B = A' \cup B') \to (A = A' \wedge B = B') \tag{3.12}$$

Damit Gleichung 3.12 nicht *erfüllt ist, muss die Annahme gelten*

$$\begin{aligned} & (A \cup B = A' \cup B') \wedge (A \neq A' \vee B \neq B') \\ = \quad & (A \cup B = A' \cup B' \wedge A \neq A') \vee (A \cup B = A' \cup B' \wedge B \neq B') \end{aligned} \tag{3.13}$$

Für den Fall $(A \cup B = A' \cup B' \wedge A \neq A')$ *muss also gelten*

$$(A \cup A') \cap B \neq \emptyset$$

und somit

$$A \cap B \neq \emptyset \vee A' \cap B \neq \emptyset. \tag{3.14}$$

Für den Fall $(A \cup B = A' \cup B' \wedge B \neq B')$ *folgt entsprechend*

$$B \cap A \neq \emptyset \vee B' \cap A \neq \emptyset. \tag{3.15}$$

Dies entspricht aber in beiden Fällen nicht der Voraussetzung des Lemmas 1 $\forall_{A \in \mathcal{A}} \forall_{B \in \mathcal{B}} : A \cap B = \emptyset$*, wodurch die Annahme aus Gleichung 3.13 falsch ist. Deshalb gilt Gleichung 3.12 und die Injektivität von* f*.*

Für die Surjektivität von f *muss gezeigt werden, dass jedes Element der Zielmenge* $A \cup B \in \mathcal{A} \otimes \mathcal{B}$ *als Funktionswert von* f *vorkommt. Es muss somit gelten*

$$\forall_{e \in \mathcal{A} \otimes \mathcal{B}} \exists_{(A,B) \in \mathcal{A} \times \mathcal{B}} : f((A, B)) = A \cup B = e$$

Dies folgt direkt aus der Definition 3.1.2 der kartesischen Vereinigung. Hieraus ergibt sich, dass f *auch surjektiv ist und daraus die Bijektivität von* f*.*

□

Hieraus kann folgender Satz abgeleitet werden

Satz 2
Die Kardinalität der Lösungsmenge $\mathcal{L}$ *eines CSPs* P*, dass sich durch disjunkte Partialzerlegung in die Teilprobleme* P^k *aufteilen lässt, kann durch* $\prod_{1 \leq i \leq k} |\mathcal{L}^i|$ *berechnet werden.*

Beweis 3
Da die Teillösungsmengen $\mathcal{L}^1, \ldots, \mathcal{L}^k$ *der Teilprobleme* $P^1, \ldots, P^k$ *aus der disjunkten Partialzerlegung von* P *die Voraussetzung aus Lemma 1 erfüllen, gilt somit auch für sie*

$$|\mathcal{L}^1 \times \ldots \times \mathcal{L}^k| = |\mathcal{L}^1 \otimes \ldots \otimes \mathcal{L}^k|$$

Hieraus folgt, dass die sich die Kardinalität von $\mathcal{L} = \mathcal{L}^1 \otimes \ldots \otimes \mathcal{L}^k$ *als Produkt der Kardinalitäten seiner* $\mathcal{L}^i$ *berechnen lässt.*

$$|\mathcal{L}| = |\mathcal{L}^1 \otimes \ldots \otimes \mathcal{L}^k| = |\mathcal{L}^1 \times \ldots \times \mathcal{L}^k| = \prod_{1 \leq i \leq k} |\mathcal{L}^i| \tag{3.16}$$

□

Ein weiterer Vorteil dieser Zerlegungsstrategie ist, dass sich bei einer disjunkten Partialzerlegung in Teilprobleme mit jeweils nur einer ungebundenen Variablen direkt die Teillösungsmengen ohne weitere Suchverzweigungen ableiten lassen. Mit maximal einem Reduktionsschritt (wenn noch Bedingungen vorhanden sind) enspricht der resultierende Wertebereich der ungebundenen Variable den Lösungen des Teilproblems. Diese können dann direkt zu Lösungen des zerlegten CSPs über eine kartesische Vereinigung kombiniert werden. Das ermöglicht es, die Lösungen des folgenden Beispiel-CSPs, wie in Abbildung 3.1 dargestellt, aufzuzählen. Gegeben sei das CSP P aus Abbildung 2.2, welches sich in die zwei disjunkten Teilprobleme $P^1 \Leftrightarrow (\{3,5\} \ni A \not\equiv B \in \{3,4\})$ und $P^2 \Leftrightarrow (\ \{1,2\} \ni C \not\equiv D \in \{1,2\})$ zerlegen lässt. Nach einer disjunkten Partialzerlegung wurde die Suche für jedes Teilproblem in je einem eigenen Suchbaum dargestellt. Kartesische Vereinigungen der Lösungsmengen werden durch „X“ repräsentiert.

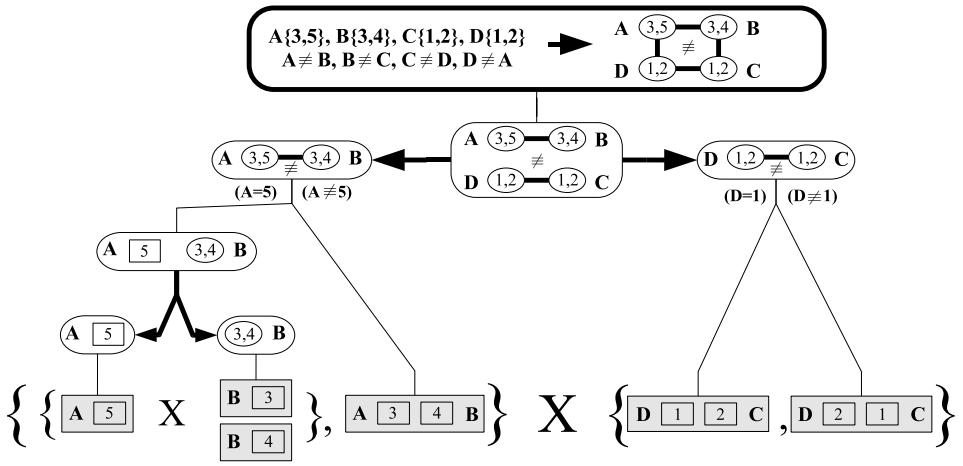

Abbildung 3.1: Suchbäume einer Zerlegungssuche.

Der Suchbaum wird also dynamisch in einen Wald zerlegt, dessen Teillösungen im Anschluss durch zwei kartesische Vereinigungen zu den sechs Lösungen des Ausgangsproblems kombiniert werden. So ist es möglich mit zwei disjunkten Partialzerlegungen und nur zwei Suchverzweigungen (anstatt fünf) alle Lösungen zu ermitteln. Die Einsparungen für das in Abschnitt 2.2 vorgestellte SP-CSP werden im Kapitel 4.2 für verschiedene Suchstrategien näher beleuchtet.

Algorithmus 3.1 Lösungszählung durch Zerlegungssuche

```
1:  function LÖSUNGSZAHL(𝒳, 𝒟, 𝒞)
                                                  ▷ verkleinere Wertebereiche
2:      (𝒟', 𝒞') ← WERTEBEREICHSREDUKTION(𝒳, 𝒟, 𝒞)
                                          ▷ prüfe, ob Blatt im Rek.-Baum
3:      if BEDINGUNGVERLETZT(𝒳, 𝒟', 𝒞') then
4:          return 0
5:      else if ALLEVARIABLENGEBUNDEN(𝒳, 𝒟') then
6:          return 1
7:      else if NUREINEUNGEBUNDEN(𝒳, 𝒟') then          ▷ Optimierung
8:          return MAXGRÖSSE(𝒟')
9:      else
10:         L ← 1                                  ▷ disjunkte Partialzerlegung
11:         for all Teilprobleme (𝒳^k, 𝒟^k, 𝒞^k) von (𝒳, 𝒟', 𝒞') do
                                                  ▷ Suche aufspalten
12:             (X_i, d_i) ←WÄHLEVARIABLEUNDWERT(𝒳^k, 𝒟^k)
13:             L = L * LÖSUNGSZAHL( 𝒳^k, 𝒟^k, 𝒞^k ∪ (X_i ≡ d_i))
14:             L = L * LÖSUNGSZAHL( 𝒳^k, 𝒟^k, 𝒞^k ∪ (X_i ≢ d_i))
15:         end for
16:         return L                                ▷ Gesamtlösungszahl
17:     end if
18: end function
```

Um die Zerlegungssuche zu verdeutlichen, ist im Algorithmus 3.1 die Rekursion für die Zählung aller Lösungen eines CSPs mit disjunkten Partialzerlegung angegeben. Um diese explizit zu generieren, wäre lediglich die Verwaltung und Behandlung der Teillösungen nötig, der Ablauf der Suche an sich bleibt gleich. Zu Zeile 11 sei angemerkt, dass ein nichtzerlegbares CSP als Pseudoteilproblem behandelt wird. Hierbei können unter Umständen zur Performanzsteigerung die gebundenen Variablen als selbständiges Teilproblem herausgelöst werden (wie in Abbildung 3.1 geschehen). Der zuvor beschriebene Spezialfall, dass ein Teilproblem nur eine ungebundene Variable enthält, wird in Zeile 7 und 8 abgefangen und ausgenutzt. Die Zerlegungssu-

che funktioniert jedoch auch ohne diese Optimierung.

Die für die Standardsuche beschriebenen Schritte 1) Reduktion der Wertebereiche, 2) Blattprüfung und 3) Suchverzweigung sind immer noch vorhanden und identisch. Jedoch wurde die Suchverzweigung um die disjunkte Partialzerlegung erweitert, sodass sich die Verzweigung und Rekursion lediglich auf die Teilprobleme bezieht und auswirkt. Hier wird deutlich, dass die Zerlegungssuche nicht eigenständig zum Lösen von CSPs in der Lage ist, sondern als zusätzliche Strategie bestehende Suchverfahren erweitert (in diesem Fall Maintaining Arc Consistency).

Der Nutzen aus der disjunkten Partialzerlegung eines CSPs stellt einen neuen Anspruch an die Suchstrategie. Nun ist neben geringem Aufwand und möglichst wenig Irrläufen auch die Teilbarkeit des CSPs in unabhängige Teilprobleme zu berücksichtigen. Die Umsetzung dessen wird in Abschnitt 3.1 näher beschrieben. Hierbei fließen sowohl problemspezifische statische und dynamische Heuristiken als auch allgemeine Zerlegungsstrategien ein.

Das Verfahren der Zerlegungssuche stellt einen iterativen, generellen Ansatz zur vollständigen Aufzählung aller Lösungen von Constraint Satisfaction Problemen dar. Es ist nicht auf das Strukturvorhersage CSP (SP-CSP) beschränkt, wird jedoch an diesem ausgeführt und veranschaulicht.

3.1 Gruppierungsheuristik der Variablenauswahl

Die besondere Beschaffenheit des SP-CSP-Constraintgraphen ermöglicht eine initiale, statische Heuristik für die Variablenauswahl.

Die Menge der Bedingungen $\mathcal{C}$ besteht lediglich aus binären Ungleichheits- und Nachbarschaftsbedingungen. Die Bindung einer Variablen X_i auf den Wert d_i ermöglicht es, dass alle sie betreffenden Bedingungen des Unterproblems durch einmalige Reduktion domain-entailed werden und entfallen (siehe Definition 2.1.9). Dies ist dadurch begründet, dass in einem einzigen Reduktionsschritt d_i aus den Wertebereichen der von X_i als ungleich geforderten Variablen entfernt wird, wodurch alle betroffenen Ungleichheitsbedingungen erfüllt sind. Ein Beispiel ist die Bindung von D auf den Wert 1 ($D \equiv 1$) in Abbildung 2.2 und 3.1. Für die Nachbarschaftsbedingung gilt gleiches, jedoch mit einer umfangreicheren Reduktion der betreffenden Wertebereiche.

Des Weiteren sind die Wertebereiche von H- und P-Variablen per Initialisierung ungleich und benötigen dafür keine Bedingungen. Hierdurch bestehen zu Beginn zwei vollständige Teilgraphen, einer für alle H-Variablen

und einer für alle P-Variablen, mit Kanten für Ungleichheits-Bedingungen. Die zwei Teilgraphen werden durch die Kanten für Nachbarschaftsbedingungen verknüpft, was in der Abbildung 3.2 dargestellt ist. Hierbei wurden nur Nachbarschaftsbedingungen als Kanten eingetragen und die Kanten der Ungleichheitsbedingungen in den Teilgraphen durch Gruppierung der Knoten dargestellt.

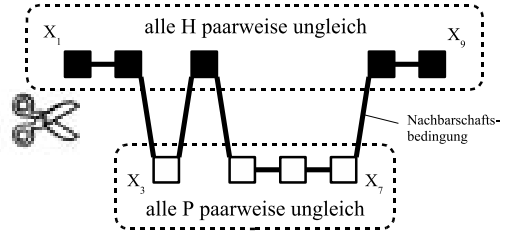

Abbildung 3.2: Zielzerlegung der Gruppierungsheuristik.

Dies bedeutet, dass sich das initiale CSP durch disjunkte Partialzerlegung nach Domain-Entailment der verbindenden Nachbarschaftsbedingungen aufspalten lässt, was durch die folgende Gruppierungsheuristik erreicht wird. Die Schere in Abbildung 3.2 deutet die nötigen Domain-Entailments an. Die Anzahl der H und P verbindenden Nachbarschaftsbedingungen einer Sequenz der Länge n ist nach oben durch $(n-1)$ beschränkt und entspricht dabei einer Folge „..HPHP...“. Die Zahl der Kanten eines vollständigen Graphen für die Ungleichheitsbedingungen der Größe $\frac{n}{2}$ beträgt jedoch $(\frac{n}{2} * (\frac{n}{2} + 1))/2$, und wächst somit quadratisch in n.

Somit ist es sinnvoll, zuerst die H-P-verbindenden Kanten zu entfernen um den Constraintgraphen in die zwei Teilprobleme mit Ungleichheitsbedingungen zu zerlegen. Hierfür kann die folgende Gruppierungsheuristik eingeführt werden, um eine effiziente Zerlegung zu ermöglichen.

Die Variablen werden in drei Gruppen gegliedert:

1. Singlets
2. Randvariablen
3. Rest

Als *Singlets* werden hierbei Variablen bezeichnet, die genau 2 Sequenznachbarn eines anderen Monomertyps (H oder P) aufweisen (z.B. das H in

„..PHP..“). *Randvariablen* besitzen genau einen verschiedenartigen Nachbarn in der Sequenz, der *Rest* nur gleichartige. Somit entfallen bei Bindung der Singlets zwei und bei Randvariablen eine Nachbarschaftsbedingung.

Die Variablen $X_1, \ldots, X_9$ aus Abbildung 3.2 können wie folgt gruppiert werden:

$$\begin{aligned} \text{Singlets} &= X_3, X_4 \\ \text{Randvariablen} &= X_2, X_5, X_7, X_8, X_9 \\ \text{Rest} &= X_6 \end{aligned}$$

Für diese Gruppen kann nun eine statische Bearbeitungsreihenfolge festgelegt werden. Zum Beispiel könnte man erst Singlets, dann Randvariablen und zum Schluss den Rest binden, um schnellstmöglich zumindest eine disjunkte Partialzerlegung in die zwei Ungleichheitsteilprobleme zu ermöglichen. Zudem reduziert die Nachbarschaftsbedingung die Wertebereiche am stärksten (siehe Abschnitt 3.3.2). In Kapitel 4.2 wird die Performanz verschiedener solcher Gruppierungsheuristiken zur Lösung des SP-CSPs mit Zerlegungssuche untersucht.

Zusätzlich zu dieser Gruppierungsheuristik können zur Reihenfolgebestimmung innerhalb der Gruppen weitere Kriterien zu Rate gezogen werden. So zum Beispiel die schon vorgestellte minimale Wertebereichsgröße zur Abschätzung des Aufzählungsaufwandes einer Variablen. Weitere Kriterien werden im Abschnitt 3.3 weitergehend vorgestellt.

Die Heuristik stellt somit eine statische Vorsortierung bzw. Gruppierung und keine direkte Auswahl dar. Da diese nur greift, solange das Problem sowohl H- als auch P-Variablen enthält, muss bei Homogenität des Variablentyps unter Umständen die Strategie gewechselt werden. Wenn das Problem lediglich Elemente der Gruppe 3 enthält, sollten diese aufgrund der nun möglichen Zerlegung in Teilprobleme gleichen Typs sein (nur H oder P). Zudem sollte der Constraintgraph durch die fortschreitende Reduktion der Wertebereiche soweit gelockert sein, dass generelle Strategien zur Graphenzerlegung anwendbar werden.

Eine solche ist die Identifikation von *Artikulationspunkten*.

Definition 3.1.1 (Artikulationspunkt)
Ein Artikulationspunkt $v_a \in V$ *(cut vertex) ist ein Knoten im Graph* $G = (V, E)$*, bei dessen Entfernung und all seiner mit ihm verbundenen Kanten der Graph in zwei oder mehr disjunkte Teilgraphen (Zusammenhangskomponenten) zerfällt.*

Für einen Constraintgraphen eines SP-CSPs bedeutet dies, dass das Unterproblem in zwei oder mehr disjunkte Teilprobleme zerfällt. Hierfür genügt jedoch, dass alle Bedingungen C mit $X_a \in X(C)$ domain-entailed werden

(X_a ist die vom Artikulationspunkt v_a repräsentierte Variable). Diese und ihre Kanten können danach aus $\mathcal{C}$ bzw. E entfernt und das CSP bzw. der Graph können zerlegt werden. Domain-Entailment all dieser Bedingungen wird durch Bindung der Variable X_a erreicht (siehe oben). Da die Suche nach Artikulationspunkten aufwändig ist (sie bedarf zweimaliger Traversierung des kompletten Graphen durch Breitensuche [Wes96]), sollte diese Strategie erst bei relativ lichten Graphen zum Tragen kommen. Auch die Artikulationspunktidentifikation kann als Gruppierung mit den oben genannten Vorteilen betrachtet werden. Sie ist eine generelle und dynamische Erweiterung der vorgestellten, problemspezifisch statischen Gruppierungsheuristik.

Eine weitere günstige Strategie kann die Identifikation von *Brücken* im Constraintgraphen sein.

Definition 3.1.2 (Brücke im Graph)
Eine Kante $e_b \in E$ des Graphen $G = (V, E)$ wird Brücke *genannt, wenn bei ihrer Entfernung $(E \setminus \{e_b\})$ der Graph in zwei Zusammenhangskomponenten zerfällt.*

In einem Constraintgraphen kann dies erreicht werden, indem die Wertebereiche D_i der an der Bedingung C_b beteiligten Variablen $X_i \in X(C_b)$ aufgespalten werden (C_b durch e_b repräsentiert). Jedoch ist sowohl die Identifikation einer Brücke, als auch eine solche Wertebereichsaufspaltung aufwändig. Es gelten die gleichen Überlegungen wie für Artikulationspunkte.

Die Brücken- und Artikulationspunktidentifikation wird hier lediglich theoretisch als Erweiterung diskutiert.

3.2 Bearbeitungsreihenfolge der Teilprobleme

Ein weiterer wichtiger Punkt zur Effizienzsteigerung der Zerlegungssuche ist die Bearbeitungsreihenfolge der Teilprobleme. Wenn ein Teilproblem P^x einer disjunkten Partialzerlegung des CSPs P keine Lösung hat, besitzt auch P keine Lösung (siehe Definition 3.1.2 und Satz 2).

Somit sollten Teilprobleme, die mit hoher Wahrscheinlichkeit keine Lösung besitzen, zuerst betrachtet werden. Wenn ein solches tatsächlich keine Lösung aufweist, kann die Behandlung der restlichen Teilprobleme abgebrochen und deren unnötige Enumeration vermieden werden.

Die Bearbeitungsreihenfolge der Teilprobleme stellt eine Optimierung dar und ist für die Zerlegungssuche nicht essentiell. Ihre Festlegung findet dynamisch während der Suche statt und kann nicht vom initialen Problem abgeleitet werden.

3.3 Variablenauswahl

Die richtige Variablenauswahl für die Suchverzweigung ist ausschlaggebend für die Effizienz eines Suchverfahrens. Sie sorgt für eine frühzeitige Erkennung von Bedingungsverletzungen und ein möglichst direktes Auffinden von Lösungen. Durch eine frühe Identifikation von Bedingungsverletzungen kann die unnötige Aufzählung ganzer Teilbäume des Suchbaums vermieden werden.

3.3.1 Kombinationsansätze

Um eine möglichst gute Variablenauswahl zu ermöglichen, ist oft eine Kombination aus generellen und problemspezifischen Wichtungsfunktionen nötig. Die auf das Strukturvorhersage CSP (SP-CSP) angewandten Kriterien werden in Abschnitt 3.3.2 vorgestellt. Die Kombination von mehreren Wichtungsfunktionen lässt zwei verschiedene Ansätze zu.

Zum einen ist ein *hierarchischer Ansatz* möglich, bei dem die Kriterien entsprechend einer vorgegebenen Reihenfolge nacheinander zu Rate gezogen werden. Wenn ein Kriterium keine Aussage ermöglicht (also zwei Variablen die gleiche Wichtung aufweisen), wird die Entscheidung anhand des nächsten Kriteriums gemäß der Reihenfolge gefällt.

Zum anderen steht ein *summierender Ansatz* zur Wahl, bei dem die Ergebnisse der Wichtungsfunktionen (selbst über Parameter gewichtet) aufsummiert werden. Die Variable mit der optimalen Bewertung wird ausgewählt. Dieser Ansatz ermöglicht eine feine Gewichtung der Kriterien. Er bedarf aber für die Summierung der Wertungen einer sehr guten und problemspezifischen Anpassung der Wichtungsparameter.

Ein Vergleich beider Ansätze wird in Kapitel 4.2 diskutiert.

3.3.2 Wichtungsfunktionen

Die im Folgenden vorgestellten Wichtungsfunktionen dienen der Bewertung von Variablen für die Lösung des SP-CSPs. Um für einen summierenden Ansatz sinnvolle und vergleichbare Ergebnisse zu erhalten, sollten die Funktionen Werte im Intervall $(0, 1]$ liefern. Die Normierung kann dabei mit Hilfe des jeweiligen Maximums erreicht werden, welches jedoch dynamisch ermittelt werden muss.

Zudem müssen alle Wichtungsfunktionen die gleiche Optimierungsstrategie verfolgen, d.h. das Optimum hat eine mini- oder maximale Bewertung. Andernfalls ist der in Abschnitt 3.3.1 vorgestellte, summierende Ansatz nicht möglich.

Eine Analyse über die Performanz und Praxisrelevanz der hier vorgestellten Kriterien wird in Kapitel 4.2 vorgestellt.

Wertebereichsgröße

Die *Wertebereichsgröße* $m = |D_i = D(X_i, \mathcal{D})|$ einer Variablen X_i stellt eine obere Schranke für den Aufzählungsaufwand derselben dar. Für eine Wertebereichsgröße m sind maximal $(m-1)$ Suchverzweigungen für diese Variable nötig, um alle Bindungen der Variablen (X_i, d_i) mit $\exists_{L_k \in L} : (X_i, d_i) \in L_k$ zu finden. Somit ist es eine weitverbreitete Herangehensweise, die Variable mit minimaler Wertebereichsgröße für die Suchverzweigung auszuwählen. Dadurch soll der Suchbaum aufgrund der folgenden Reduktion der Wertebereiche so klein wie möglich gehalten werden und Bedingungsverletzungen so früh wie möglich erkannt werden. Diese Strategie wird auch als „Fail-first“ bezeichnet.

Anzahl ungebundener Sequenznachbarn

Da die Nachbarschaftsbedingung die stärkste Bedingung im SP-CSP darstellt und ihre Reduktionsleistung am größten ist, kann sie sehr gut für ein Kriterium genutzt werden.

Die *Zahl der ungebundenen Sequenznachbarn* $neigh(X_i)$ einer Variable $X_i \in \mathcal{X}$ des SP-CSPs $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ entspricht:

$$\begin{aligned} neigh(X_i) \;=\; & |\{X_j \mid (|D_j = D(X_j, \mathcal{D})| > 1) \wedge \\ & \exists_{C \in \mathcal{C}} : \{X_i, X_j\} = X(C) \wedge C \text{ ist Nachbarschaftsbedingung}\}| \end{aligned}$$

Wenn eine Variable gebunden wird, sorgt die Reduktion aufgrund der Nachbarschaftsbedingung dafür, dass die Wertebereiche ihrer beiden Sequenznachbarn lediglich Nachbarpositionen enthalten. Deren Anzahl ist scharf durch die dem SP-CSP zugeordnete Gitternachbarschaft nach oben begrenzt (siehe Tabelle 1.1). Diese Reduktion ist weitaus höher als die Reduktion durch Ungleichheit, da bei letzterer meist nur ein Wert aus dem als ungleich geforderten Wertebereich entfernt wird.

Die Zahl ungebundener Sequenznachbarn liefert somit die Zahl der wegfallenden Nachbarschaftsbedingungen bei Bindung der Variablen und die dadurch folgende umfassende Reduktion der Wertebereiche ihrer Nachbarn. Dieses Kriterium bewertet den Umfang der zu erwartenden Reduktion.

Knotengrad im Constraintgraphen

Der *Knotengrad* $d(v_i)$ eines Variablenknotens $v_i \in V$ im Constraintgraphen $G = (V, E, \mu)$ des CSPs $(\mathcal{X}, \mathcal{D}, \mathcal{C})$, also die Anzahl der mit ihm durch Kanten verbundenen Knoten, kann ebenfalls als Maß für die Variablenauswahl dienen. Wenn v_i für die Variable $X_i \in \mathcal{X}$ steht, entspricht sein Knotengrad im SP-CSP :

$$d(v_i) = |\{C \mid X_i \in X(C)\}|$$

Da bei Bindung einer Variablen sämtliche sie betreffenden Bedingungen durch Reduktion domain-entailed werden, reduziert sich danach die Anzahl der Kanten dieses Unterproblems um den Knotengrad der gebundenen Variablen (siehe Abschnitt 3.1). Dies kann eine gute Strategie sein, um den Graphen möglichst schnell in viele kleine unabhängige Teilprobleme zu zerlegen. Zudem folgt aus einem hohen Knotengrad eine Vielzahl von Wertebereichsreduktionen der an den wegfallenden Bedingungen beteiligten Variablen. Der Knotengrad gibt eine Abschätzung der folgenden Reduktionsleistung.

Anzahl resultierender Teilprobleme

Weil die Zerlegungssuche aus der Zerteilung in viele disjunkte Teilprobleme profitiert, bietet die *Anzahl der resultierenden Teilprobleme* eine gute Bewertung für die Gruppe der Artikulationspunkte. Da diese bei Wegfall ihrer Kanten den Graphen in mindestens zwei disjunkte Teilgraphen zerlegen, sollte man den Knoten mit der größten Zerlegungsleistung wählen. Doch die Identifikation der Artikulationspunkte ist aufwändig, durch die nötige zweimalige Traversierung des Graphen (siehe Abschnitt 3.1).

Für normale Knoten/Variablen ist die Anzahl der resultierenden Teilprobleme jedoch immer 1 und liefert somit keine zusätzliche Information.

Die Anzahl der resultierenden Teilprobleme sollte also lediglich für die Gruppe der Artikulationspunkte als Kriterium in Betracht gezogen werden.

Kapitel 4

Umsetzung und Ergebnisse

4.1 Implementierung in *ILOG Solver 6.1™*

> ILOG Solver is ILOG's constraint-based optimization engine.
>
> A core engine of the ILOG Optimization Suite, this software component provides cutting-edge optimization technology for powering scheduling, sequencing, timetabling, configuration, dispatching and resource-allocation applications with logical constraints. [1]

ILOG Solver 6.1™ ist eine Programmier-Bibliothek für die Erweiterung einer objektorientierten Programmiersprache (wie Java oder C++) um die Funktionalitäten der Constraintprogrammierung. Hierbei wird dem Anwender eine umfangreiche Programmierschnittstelle (API) zur Verfügung gestellt, um CSPs zu formulieren und zu lösen. Es steht eine breite Auswahl von generischen und performanten Algorithmen zur Wahl, mit denen viele Probleme ohne umfassende Neuimplementierung gelöst werden können. Jedoch ist die Formulierung der Wertebereiche von Variablen auf wenige Basistypen wie `int` oder `float` beschränkt.

Um das Strukturvorhersage-CSP (SP-CSP) in einem mehrdimensionalen Gitter mit ILOG Solver 6.1™ lösen zu können, ist somit eine Kodierung der Gitterpunkte durch solche Basistypen nötig. Da der Raumausschnitt F, in dem die Strukturvorhersage stattfindet, endlich ist (siehe Kapitel 2.2), können dessen Gitterpunkte indiziert werden. Hierdurch erhält man eine eineindeutige Abbildung von Gitterpositionen auf `int`-Werte und kann die effiziente und umfangreiche ILOG Solver 6.1™ Bibliothek zur Lösung des SP-CSPs nutzen.

[1] (28.12.2005) http://www.ilog.com/products/solver/

Die derzeitige Implementierung des SP-CSPs beinhaltet die Nachbarschaften für das kubische und flächenzentriert kubische Gitter, die in Abschnitt 1.2 vorgestellt wurden. Diese sind modular eingebettet, sodass sich das Programm ohne weiteres auf andere Gitter erweitern lässt.

Im Folgenden wird lediglich die Zählung von Lösungen durch Zerlegungssuche vorgestellt und nicht ihre konkrete Generierung. Die Zählung verdeutlicht jedoch einfacher die Arbeitsweise und die Vorteile der Zerlegungssuche. Die explizite Generierung der Lösungen folgt dem gleichen Schema und bedarf zusätzlich nur einer Verwaltung der Teillösungen.

Implementierungsstruktur

Bei der vorliegenden Implementierung in C++ wurde auf eine strenge Objektorientierung geachtet. Die Abhängigkeiten der Klassen untereinander sind vereinfacht durch das Klassendiagramm in Abbildung 4.1 dargestellt. Die einzelnen Teile wie Kerndatenbank, Nachbarschaftsbedingung oder die Suchen wurden weitestgehend eigenständig implementiert, um eine hohe Wiederverwertbarkeit und Modularisierung zu erreichen. Um die Performanz und den Speicherbedarf zu optimieren, sind generell relevante Informationen in der statischen Klasse `CGlobal` zusammengefasst. Die zentrale Hauptklasse `Threading` initialisiert und leitet die Lösung des SP-CSPs. Der hierfür nötige Gittertyp und die zugehörige Nachbarschaft mit entsprechenden Methoden zur Indizierung wurden in der Klasse `CLattice` gesammelt. Ein Objekt von `CCore` verwaltet die Punktmenge eines H-Kerns (HD) und erzeugt die notwendigen *Kernhüllen* (siehe Abschnitt 4.1.1).

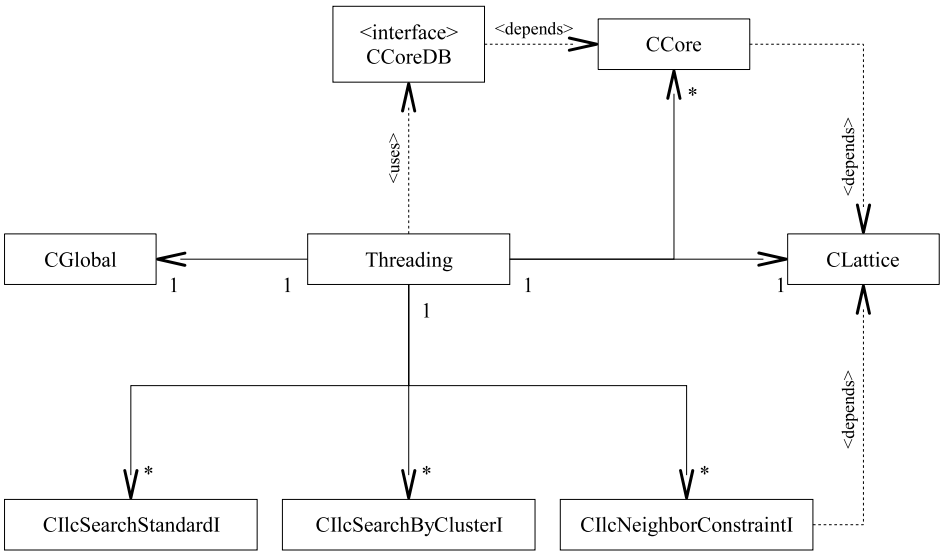

Abbildung 4.1: Vereinfachtes Klassendiagramm der Implementierung.

4.1.1 Strukturvorhersage-CSP-Modellierung

Variablen und Wertebereiche

Die HP-Sequenz ist intern als `string`-Objekt `CGlobal::sSequence` repräsentiert. Für jede Position wird ein `IloIntVar`-Objekt erzeugt und in einem entsprechenden Array `domains` abgelegt. Der Wertebereich wird, wie in Abschnitt 2.2 vorgestellt, zugewiesen, jedoch mit folgender Erweiterung: P-Variablen werden nicht mit $PD = F \setminus HD$ sondern mit den entsprechenden *Kernhüllen* initialisiert.

Definition 4.1.1 (Kernhülle)
Eine Kernhülle CH_i $(i \geq 1)$ *ist hierbei eine Teilmenge von* PD *und definiert sich wie folgt:*

$$CH_0 = HD \tag{4.1}$$
$$CH_1 = \{x \mid x \in PD \wedge \exists_{y \in HD} : x, y \text{ benachbart}\} \tag{4.2}$$
$$CH_{i>1} = CH_{i-1} \cup \{x \mid x \in PD \wedge \exists_{y \in CH_{i-1}} : x, y \text{ benachbart}\} \tag{4.3}$$

Die erste Kernhülle CH_1 umfasst alle Gitterpunkte um den H-Kern, CH_2 die um CH_1 und diese selbst und so fort. Die nullte Kernhülle ist keine Hülle

sondern direkt der H-Kern, was die Implementierung erleichtert. Der Hüllenindex i einer Variablen X entspricht dem minimalen Abstand in der Sequenz zum nächsten H. Für die Sequenz „HPPHPPPP“ ist er $[0, 1, 1, 0, 1, 2, 3, 4]$.

Diese Initialisierung ist eine Optimierung, um die Wertebereiche und damit den Suchraum so klein wie möglich zu halten, ohne Lösungen zu verlieren. Sie könnte auch durch die Einführung entsprechender unärer Bedingungen vom System durch Reduktion erfüllt werden. Für die Vielzahl der Variablen ist jedoch die einmalige Berechnung der Kernhüllen performanter als die Reduktion jeder einzelnen Variablen.

Bedingungen

Für das SP-CSP umfasst die Menge $\mathcal{C}$ nur drei Gruppen von Bedingungen.

1. Nachbarschaftsbedingungen entsprechend der Sequenzpositionen
2. Ungleichheitsbedingungen für alle H-Variablen untereinander
3. Ungleichheitsbedingungen für alle P-Variablen untereinander

Nachbarschaftsbedingungen

Um die Nachbarschaftsbedingung mit dem ILOG Solver 6.1™-System zu implementieren, war die Definition einer von `IlcConstraintI` abgeleiteten Klasse `CIlcNeighborConstraintI` und ihrer Modellierungsklassen nötig. Diese übernimmt die Reduktion der an der Bedingung beteiligten Wertebereiche. Man spricht in diesem Zusammenhang auch von *Propagation*, welche in der Elementfunktion `propagate()` ausgeführt wird. Durch sie wird für eine Nachbarschaftsbedingung C geprüft, ob die Wertebereiche $D(X, \mathcal{D})$ der Variablen $X \in X(C)$ die Nachbarschaft verletzende Elemente enthalten und entfernt diese. Das Programm ruft die Funktion auf, wenn sich die Wertebereiche der beteiligten Variablen ändern.

Da diese Reduktion der Überprüfung aller Elemente der Wertebereiche bedarf, ist sie sehr zeitaufwändig ($O(d)$, wobei d der maximalen beteiligten Wertebereichsgröße entspricht). Um Kosten und Nutzen aufeinander anpassen zu können, wurden drei Propagationsstufen mit aufsteigender Reduktionsleistung implementiert.

`IlcBasic:` Auf dieser Propagationsstufe werden die Wertebereiche nur geprüft und eingeschränkt, wenn eine der beiden Variablen gebunden ist. Sie stellt die schwächste Reduktionsstufe dar.

`IlcMedium:` Um den Propagationsaufwand in einem festgelegten Rahmen zu halten, wurde für die Stufe `IlcMedium` ein Maximum für die zu untersuchende Wertebereichsgröße eingeführt (`MAXDOMSIZE`). Wenn eine von beiden Variablen dieses erreicht oder unterschreitet, werden beide Wertebereiche reduziert. Die Leistungsfähigkeit hängt hierbei direkt von der gewählten `MAXDOMSIZE` ab. Für `MAXDOMSIZE` $= 1$ entspricht sie `IlcBasic`, für `MAXDOMSIZE` $= \infty$ der von `IlcExtended`.

`IlcExtended:` Dies ist die stärkste und aufwändigste Stufe der Propagation. Hierbei wird versucht, beide Variablen-Wertebereiche zu reduzieren, sobald einer von ihnen eingeschränkt wurde.

Ungleichheitsbedingungen

Für die Bedingung, dass n Variablen verschieden sein sollen, stellt die ILOG Solver 6.1$^{\text{TM}}$-Bibliothek die generische Bedingungsklasse `IloAllDiff` zur Verfügung. Diese erzeugt eine n-näre Bedingung, deren Implementierung effizienter als $(n * (n-1)/2)$ binäre Bedingungen die Wertebereiche reduzieren kann. Zudem benötigt ein n-näres Bedingungs-Objekt weniger Speicherplatz. Die genaue Arbeitsweise von `IloAllDiff` wird in der ILOG Solver 6.1$^{\text{TM}}$-API nicht näher beschrieben, lediglich auf ihre Vorteile hingewiesen.

Um die Reduktionsleistung via `IloAllDiff` zu steuern, bietet ILOG Solver 6.1$^{\text{TM}}$ verschiedene Filterstufen an. Zum Beispiel werden auf der Standardstufe `IlcBasic` die Wertebereiche wie durch die erwähnte Menge von binären Ungleichheitsbedingungen behandelt. Auf der Stufe `IlcExtended` erfolgt eine erweiterte Propagation, bei der auch indirekte Zusammenhänge zwischen den Wertebereichen für deren Reduktion genutzt werden. Auf dieser Stufe ist die Propagation umfangreicher aber im Allgemeinen langsamer [ilo05c].

Da alle H- und alle P-Variablen jeweils ungleich sein sollen, wurde für jede dieser zwei Gruppen je ein `IloAllDiff`-Objekt erzeugt.

Für die Betrachtung des Constraingraphen wurden jedoch zu seiner Vereinfachung binäre Ungleichheitsbedingungen angenommen. Dies wird in Abschnitt 4.1.4 näher beschrieben.

4.1.2 Variablen- und Werteauswahl

Wie in schon Abschnitt 3.3.2 dargelegt, hat die Variablen- und Werteauswahl einen großen Einfluss auf die Effizienz einer Suche. Deswegen wurde bei der Implementierung hierauf besonderen Wert gelegt.

Variablenauswahl

Sowohl der hierarchische als auch der summierende Ansatz zur Kombination von Wichtungsfunktionen fanden ihre Umsetzung. Als Wichtungsfunktionen stehen dem Nutzer die Ansätze

- Wertebereichsgröße
- Knotengrad im Constraintgraphen
- Anzahl ungebundener Sequenznachbarn

zur Verfügung, deren jeweilige Funktionalität in Abschnitt 3.3.2 vorgestellt wurde. Alle Wichtungsfunktionen liefern Werte im Intervall $(0, 1]$, wobei die „beste“ Variable mit dem kleinsten Rückgabewert bewertet wird.

Standardmäßig sind die Ansätze „Knotengrad im Constraintgraphen“ und „Anzahl ungebundener Sequenznachbarn“ mit dem der „Wertebereichsgröße“ kombiniert. Dies geschieht, um die Anzahl nötiger Suchverzweigungen für die gewählte Variable nicht zu vernachlässigen. Im hierarchischen Ansatz erfolgt die Bewertung nach Wertebereichsgröße an zweiter Stelle. Somit liegt der Schwerpunkt auf der ersten Wichtung, welche aufgrund von Knotengrad oder ungebundener Nachbarn bestimmt wird. Beim summierenden Ansatz folgt die Kombination der Rückgabewerte dem Schema:

$$\text{Wichtung} = (W^1 * \text{ Erste\_Wichtung}) + (W^2 * \text{ Zweite\_Wichtung}).$$

Die Parameter W^i entsprechen in der vorliegenden Implementierung $W^1 = 2$ und $W^2 = 1$.

Die Performanz beider Kombinationsansätze mit den vorgestellten Wichtungsarten und Parametern wird in Abschnitt 4.2 ausführlich untersucht.

Werteauswahl

Für die Werteauswahl wurden zwei Varianten implementiert.

Zum einen die Wahl des kleinsten Wertes eines Wertebereiches, der effizient aufgrund der Datenhaltung zu bestimmen ist. Diese Art der Werteauswahl nutzt allerdings keinerlei problemspezifische Information.

Zum anderen kann die Auswahl eines Wertes aufgrund der durch ihn indizierten Gitterposition erfolgen. Hierzu wird der Mittelpunkt des H-Kerns bestimmt. Der Abstand des indizierten Gitterpunktes zu diesem dient als Maß für die Werteauswahl. Es wird der Gitterpunkt mit dem geringsten Abstand gewählt. Hintergrund hierfür ist, dass zuerst Werte aufgezählt werden, die mit hoher Wahrscheinlichkeit mit den Bedingungen konform sind. Dies ist die Standardauswahl für Werte eines Wertebereichs in der hier vorgestellten Implementierung.

4.1.3 Standardsuche

Um das Standardsuchverfahren mit der Zerlegungssuche vergleichen zu können, wurde nicht auf die in der ILOG Solver 6.1^{TM}-Bibliothek verfügbare Standardsuche zurückgegriffen. Eine eigene Implementierung ermöglichte auch eine einheitliche Variablen- und Werte-Auswahl für beide Verfahren. Da das Constraintprogrammierungssystem die Schritte 1 und 2 aus dem in Abschnitt 2.1 vorgestellten Algorithmus 2.1 übernimmt, muss lediglich die Suchverzweigung definiert werden.

Deren Implementierung erfolgt über die Ableitung der Klasse `IlcGoalI` (`CIlcSearchStandardI` aus Klassendiagramm 4.1) und den zugehörigen Modellklassen. Die eigentliche, rekursive Verzweigung geschieht in der überschriebenen Elementfunktion `execute()`. Sie wird aufgerufen, wenn keine Wertebereichsreduktionen mehr möglich, keine Bedingungen verletzt und noch nicht alle Variablen gebunden sind.

Der zugehörige Quelltext ist unter Quellcode 4.1 gezeigt. Die Elementfunktion `getHandler(..)` erzeugt hierbei ein neues Objekt der Standardsuchverzweigung. Die Variablen sind im Array `dom` zusammengefasst.

Hierbei handelt es sich nicht um eine direkte Rekursion, sondern um die logische Verknüpfung von Suchen in Unterproblemen. Diese von `execute()` zurückgegebenen `IlcGoal`-Objekte werden vom ILOG Solver 6.1^{TM}-System in einem Stack verwaltet und schrittweise abgearbeitet. Somit werden diesem bei jedem Aufruf zwei Unterprobleme hinzugefügt, sodass ihre Abarbeitung dem vorgestellten rekursiven Schema entspricht.

Wenn das initiale CSP n Variablen und keine Bedingungen enthielte, stellten alle n Variablen jeweils disjunkte Teilprobleme dar. Die Lösung die-

Quellcode 4.1 Suchverzweigung der Standardsuche

```
IlcGoal CIlcSearchStandardI::execute() {

    // variable waehlen
    int next = Util_Search::chooseDomain(dom);
    if (next == -1)    return 0;    //alles gebunden => rekursionsabbruch

    // wert waehlen
    int nextVal = Util_Search::chooseValue(dom[next]);

    // verzweigen der suche
    return IlcOr( IlcAnd( dom[next] == nextVal, CIlcSearchStandardI::getHandler(dom)),
                  IlcAnd( dom[next] != nextVal, CIlcSearchStandardI::getHandler(dom)));
}
```

ses CSPs mit der Standardsuche würde die erste Lösung sehr schnell in einer Baumtiefe von n finden. Allerdings bedarf die Aufzählung aller Lösungen der vollständigen Kombination jedes Wertes aller Wertebereiche mit jedem. Somit ergibt sich die Tiefe der letzten Lösung aus $\sum_{1 \leq i \leq n}(|D_i| - 1)$. Wenn man eine durchschnittliche Wertebereichsgröße von d annimmt, ergibt sich die maximale Baumtiefe von $n * (d - 1)$. Dies entspricht der Anzahl Suchverzweigungen um die letzte Lösung zu finden. Für die vollständige Enumeration aller d^n Kombinationen werden $d^n - 1$ Suchverzweigungen benötigt.

Dieses CSP stellt das ungünstigste Szenario für diese Art von Suche dar.

4.1.4 Zerlegungssuche

Die Suchverzweigung der Zerlegungssuche wurde, wie bei der Standardsuche, in einer eigenen Klasse `CIlcSearchByClusterI` implementiert. Da die ILOG Solver 6.1™-Bibliothek keine Initiierungen weiterer Suchen während einer laufenden Suche unterstützt, musste für die Implementierung der Zerlegungssuche auf die Verwendung einer Hilfsvariable `trigger` zurückgegriffen werden. Diese steuert die Erkennung von Teilsuchen sowie die Vereinigung der Teillösungen zur Gesamtlösung und wird im folgenden Abschnitt 4.1.5 näher erläutert. Die Suchverzweigung erfolgt in der Elementfunktion `execute()` der Klasse `CIlcSearchByClusterI`. Auch die Identifikation und Durchführung einer möglichen disjunkten Partialzerlegung geschieht hier und wird mit Hilfe der `trigger`-Variablen markiert.

Da der Constraintgraph aufgrund sich reduzierender Wertebereiche und Zerlegung des Graphen an Größe und Zusammenhang verliert, ist es nicht nötig, diesen immer neu zu berechnen. Hierzu wird eine entsprechende Adjazenzlisten-Repräsentation des Graphen in der Rekursion weitergereicht und bei Domain-Entailment oder Wegfall von Bedingungen entsprechend aktua-

lisiert. Teilprobleme können hierbei durch einmalige Traversierung des Constraintgraphen identifiziert werden. Bei einer Aufspaltung in Teilsuchen werden sowohl die Variablen als auch die Adjazenzlisten entsprechend aufgeteilt, um effizient zu arbeiten. Die Adjazenzlisten verwalten binäre Nachbarschafts- und Ungleichheitsbedingungen und nicht die initiale n-näre `IloAllDiff`, da diese, wie in Abschnitt 4.1.1 dargelegt, lediglich eine interne Optimierung darstellt.

Quellcode 4.2 zeigt die Suchverzweigung via Zerlegungssuche. Die Umsortierung der Teilproblemreihenfolge, wie in Abschnitt 3.2 vorgeschlagen, wurde hierbei aus Übersichtlichkeitsgründen weggelassen, da sie lediglich eine Optimierung darstellt.

Quellcode 4.2 Aufspaltung und Suchverzweigung bei Zerlegungssuche

```
IlcGoal CIlcSearchByClusterI::execute() {
    if (dom.getSize() == 1)  // dann nur eine variable vorhanden
        return getChoicePoint(new vecInt(1,0));

    // suche nach teilproblemen von ungebundenen variablen des arrays <dom>
    // newClusters[0] enthaelt alle indizes von <dom> die gebunden sind
    vecInt newClusters = checkClustering(false);

    int i=0;                          // laufvariable
    IlcGoal retGoal = 0;              // rueckgabewert (0 == rekursionsabbruch)

    if (newClusters.size() > 2) {     // zerfallen in teilprobleme

        // zerlegung eroeffnen mit ersten beiden teilproblemen
        IlcGoal clustGoal = IlcOr( trigger == OPEN,
                                   getChoicePoint((vecInt*)newClusters[1]),
                                   trigger == NEXTSTART,
                                   getChoicePoint((vecInt*)newClusters[2]));

        // restliche teilprobleme anfuegen
        for (i=3; i<newClusters.size(); i++){
            clustGoal = IlcOr( clustGoal,
                               trigger == NEXT,
                               getChoicePoint((vecInt*)newClusters[i]));
        }

        // zerlegung schliessen
        retGoal = IlcOr(clustGoal, trigger == CLOSE);

    } else if (newClusters.size() == 2)     // nur ein zusammenhaengendes cluster
        retGoal = getChoicePoint((vecInt*)newClusters[1]);

    for (i=0; i< newClusters.size(); i++)   // speicher aufraeumen
        delete ((vecInt*)newClusters[i]);

    return retGoal;
}
```

Die Funktion `checkClustering(bool)` aktualisiert die Adjazenzlisten und prüft die Zerlegbarkeit des CSPs anhand des Constraintgraphen. Schon gebundene Variablen werden in einem eigenen Teilproblem zusammengefasst, um die Teilprobleme so klein wie möglich zu halten.

Die eigentliche Suchverzweigung der Zerlegungssuche findet in der Elementfunktion `getChoicePoint(vecInt* clustInd)` statt. Die genaue Definition ist unter Quellcode 4.3 abgebildet. Die dort angegebene Suchverzweigung entspricht vollständig der in Quellcode 4.1 vorgestellten Suchverzweigung der Standardsuche. Lediglich die optionale Behandlung von Teilproblemen mit nur einer ungebundenen Variablen wurde ergänzt. Dabei wird die Variable `clustElems` mit der entsprechenden Wertebereichsgröße als Pseudolösung zurück gegeben.

Quellcode 4.3 Suchverzweigung von Teilproblemen bei Zerlegungssuche

```
IlcGoal CIlcSearchByClusterI::getChoicePoint(vecInt* clustInd) {

   // sammle teilproblem daten
   IlcIntVarArray nDom = getCluster(dom, clustInd);
   vecSetInt nAdjList = getCluster(adjList, clustInd);

   // dann einelementiges cluster => muss nicht aufgezaehlt werden
   if (nDom.getSize() == 1) {
       return (clustElems == nDom[0].getSize());
   }

   // waehle variable
   int nextDom = Util_Search::chooseDomain(nDom, &nAdjList);

   // naechster wert
   IlcInt nextVal = Util_Search::chooseValue(nDom[nextDom]);

   // suchverzweigung
   return IlcOr( IlcAnd( nDom[nextDom]==nextVal,
                    CIlcSearchByClusterI::getHandler( nDom,nAdjList, trigger, clustElems)),
                 IlcAnd( nDom[nextDom]!=nextVal,
                    CIlcSearchByClusterI::getHandler( nDom,nAdjList, trigger, clustElems)));
}
```

Gegeben das ungünstige CSP für die Standardsuche aus Abschnitt 4.1.3, kann für die maximale Rekursionstiefe für die Zerlegungssuche folgende Abschätzung gegeben werden. Da die n Variablen ohne verknüpfende Bedingungen sind, können diese direkt in disjunkte Teilprobleme aufgespalten werden. Entsprechend wird jede Variable für sich aufgezählt und erst am Ende die Gesamtlösungsmenge generiert. Bei einer durchschnittlichen Wertebereichsgröße d bedarf jede vollständige Enumeration eines solchen Teilproblems der Suchtiefe $(d-1)$ bei der vorgestellten Verwendung der Standardsuche. Die maximale Baumtiefe der Suche ist somit für alle Teilprobleme $(d-1)$ und die

Gesamtzahl von Suchverzweigungen sinkt auf $n * (d - 1)$ gegenüber $d^n - 1$. Die exponentielle Zahl der Aufspaltungen in Unterprobleme durch die Standardsuche kann somit drastisch aufgrund der disjunkten Partialzerlegung reduziert werden. Dieses Szenario ist das Günstigste für die Anwendung von Zerlegungssuche.

Bei Verwendung der Hilfsvariable `clustElems` für Teilprobleme mit nur einer ungebundenen Variablen kann das beschriebene CSP mit nur einer disjunkten Partialzerlegung und ohne weitere Aufzählung der Variablen gelöst werden. Die Teillösungsmenge wird dabei direkt vom Wertebereich abgeleitet. Dies zeigt deutlich den Performanzgewinn aus der Zerlegbarkeit in viele kleine disjunkte Teilprobleme, der in Abschnitt 3.3 erwartet wurde.

Die hier vorgestellte Implementierung der Zerlegungssuche liefert den in Abbildung 3.1 vorgestellten Suchbaum für das dargestellte CSP bei entsprechender Variablen- und Wertauswahl.

4.1.5 Lösungszählung

Für einige Untersuchungen im HP-Modell sind im ersten Schritt nicht die optimalen Strukturen einer HP-Sequenz an sich relevant, sondern allein deren Anzahl ist von Interesse. Hierfür muss somit die Kardinalität der Lösungsmenge $\mathcal{L}$ eines CSPs P bestimmt werden.

Die mit der ILOG Solver 6.1$^{\mathrm{TM}}$-Bibliothek implementierte Standardsuche liefert nacheinander alle Lösungen eines CSPs, sodass die Kardinalität von $\mathcal{L}$ nur durch Zählen aller gefundenen Lösungen erfolgen kann.

Wie in Kapitel 3 bewiesen wurde, kann die Kardinalität der Lösungsmenge $\mathcal{L}$ bei disjunkter Partialzerlegung durch Produkt der Kardinalitäten der Teillösungsmengen $\mathcal{L}^k$ berechnet werden (siehe Satz 2). Wie schon erwähnt, unterstützt die ILOG Solver 6.1$^{\mathrm{TM}}$-Bibliothek keine Initiierung neuer Suchen aus einer laufenden Suche heraus. Damit man auf diese verzichten kann, wurde die Hilfsvariable `trigger` eingeführt. Diese wird als Pseudolösung mit definierten Signalen zurückgegeben, um die Lösungsverarbeitung zu steuern. Die Suchen in den Teilproblemen P^k werden dabei von solchen Signalen eingerahmt, wie in Quellcode 4.2 zu sehen ist. Das Schema einer solchen Zerlegung entspricht

$$
\begin{aligned}
(\texttt{trigger=OPEN}) \quad & \vee \mathit{durchsuche}(P^1) \vee (\texttt{trigger=NEXTSTART}) \\
& \vee \mathit{durchsuche}(P^2) \vee (\texttt{trigger=NEXT}) \\
& \vee \ldots \\
& \vee \mathit{durchsuche}(P^{k-1}) \vee (\texttt{trigger=NEXT}) \\
& \vee \mathit{durchsuche}(P^k) \vee (\texttt{trigger=CLOSE})
\end{aligned}
$$

Lösungen werden hierbei durch `ODER`-Verknüpfungen ($\vee$) getrennt. Die Suchen für die Teilprobleme P^k ($durchsuche(P^k)$) liefern nacheinander die Teillösungen $L^k \in \mathcal{L}^k$.

Diese Signalstruktur ermöglicht es, dass die gefundenen Lösungen zwischen zwei `trigger`-Signalen zu den Kardinalitäten der entsprechenden $\mathcal{L}^k$ aufsummiert werden. Aus diesen kann dann gemäß Satz 2 die Kardinalität von $|\mathcal{L}| = \prod_{1 \leq i \leq k} |\mathcal{L}^i|$ berechnet werden.

4.2 Ergebnisse

Im Folgenden soll das Laufzeitverhalten der vollständigen Aufzählung aller Lösungen von SP-CSPs unter Verwendung von Standard- und Zerlegungssuche miteinander verglichen werden. Alle Aufrufe arbeiten auf einem kubischen Gitter mit der Nachbarschaft $NV = \{(\pm 1, 0, 0), (0, \pm 1, 0), (0, 0, \pm 1)\}$, die in Abschnitt 1.2 vorgestellt wurde. Da eine große Reihe verschiedener Parametersätze getestet werden soll, muss die Laufzeit pro Aufruf in überschaubaren Maßen bleiben. Jedoch weisen HP-Sequenzen ob der einfachen Energiefunktion eine immens hohe Zahl optimaler Strukturen auf. Aus diesem Grund wurden für die Testsätze maximal erlaubte Anzahlen von Strukturen pro Sequenz festgelegt. HP-Sequenzen, deren SP-CSPs mehr Lösungen aufwiesen, wurden aus dem Testsatz entfernt.

Die Betrachtungen basieren auf den folgenden Testsätzen.

Testsatz 1 (T33)

Der Testsatz T33 umfasst 254 HP-Sequenzen der Länge 33, welche weniger als 500.000 optimalen Strukturen aufweisen.

Hierfür wurden 442 zufällige Sequenzen generiert mit gleichverteilten H's und P's. Von diesen wiesen 188 mehr als 500.000 optimale Strukturen auf und wurden aus dem Testsatz entfernt.

Testsatz 2 (T54)

Der Testsatz T54 wird durch 218 HP-Sequenzen der Länge 54 gebildet, die maximal 1.000.000 optimale Strukturen ausbilden können.

Von den 1352 zufällig und gleichverteilt generierten HP-Sequenzen konnten 1134 nicht berücksichtigt werden, da sie mehr Strukturen aufwiesen.

4.2.1 Standard- versus Zerlegungssuche

Voruntersuchungen haben gezeigt, dass sowohl die Standard- als auch die Zerlegungssuche von einer starken Propagation durch Ungleichheitsbedingungen profitieren. Aufgrund dessen wurden alle hier vorgestellten Suchen mit `IloAllDiff`-Arbeitsstufe `IlcExtended` durchgeführt (siehe Abschnitt 4.1.1).

Um Performanzunterschiede zu untersuchen, wurde die Standardsuche unter Verwendung verschiedener Suchstrategien mit der Zerlegungssuche verglichen. Hierbei gilt bei allen Aufrufen eine Gruppierungsheuristik mit der Reihenfolge „Singlets-Randvariablen-Rest“ (siehe Abschnitt 3.1).

Die verschiedenen Suchstrategien werden wie folgt abgekürzt:

MINSIZE	=	minimale Wertebereichsgröße
MINNEIGH	=	minimale Anzahl gebundener Nachbarn
MAXCONSTR	=	maximaler Knotengrad im Constraintgraphen

MINNEIGH und MAXCONSTR sind jeweils mit MINSIZE kombiniert (siehe Abschnitt 4.1.2). Die Verwendung des hierarchischen oder summierenden Kombinationsansatzes wurde entsprechend durch „hier.“ oder „sum.“ gekennzeichnet.

Im Folgenden wird auf die Darstellung der absoluten Laufzeiten des Programms verzichtet. Für den Testsatz T33 lagen die absoluten Zeiten zwischen 3 und 100 und in T54 zwischen 3 und 180 Sekunden bei Verwendung der Standardsuche. Diese Ergebnisse wurden auf einem AMD Opteron^{TM} 875 mit 2.2 GHz erzielt. Diese Laufzeiten bieten weniger informative Aussagen als ihre Relationen zueinander, da die Performanz der Suchen miteinander verglichen werden soll. Somit werden die Geschwindigkeitsänderungen relativ zueinander als Maß für die Performanz der Standard- und Zerlegungssuche zu Rate gezogen.

Aus Gründen der Übersichtlichkeit werden die folgenden Abkürzungen eingeführt:

STD	=	Standardsuche
ZLG	=	Zerlegungssuche mit disjunkter Partialzerlegung
ZLGSTD	=	Zerlegungssuche ohne disjunkte Partialzerlegung

$$time(X/Y) = \frac{\text{absolute Laufzeit mit Suche X}}{\text{absolute Laufzeit mit Suche Y}}$$

Da die Zerlegungssuche auf der Standardsuche aufbaut und ihre Suchstrategie verwendet, liefert ZLGSTD denselben Suchablauf wie STD. Allerdings beinhaltet ZLGSTD den Mehraufwand für die Constraintgraphenverwaltung und die Prüfung auf mögliche disjunkte Partialzerlegungen.

Geschwindigkeitszuwachs

In Tabelle 4.1 können die relativen Geschwindigkeiten der Zerlegungssuche im Vergleich zur Standardsuche abgelesen werden. Die dargestellten Werte entsprechen $time(STD/ZLG)$ und wurden über alle Sequenzen der jeweiligen Testsätze gemittelt.

	Suchstrategie	$time(STD/ZLG)$ in Testsatz T33	$time(STD/ZLG)$ in Testsatz T54
	MINSIZE	1,0	0,9
hier.	MINNEIGH	1,5	1,7
sum.	MINNEIGH	1,5	1,7

Tabelle 4.1: Durchschnittliche Geschwindigkeit der Zerlegungssuche mit disjunkter Partialzerlegung relativ zur Standardsuche

Die Tabelle 4.1 zeigt, dass die Verwendung der Zerlegungssuche in der Regel keine durchschnittlich schlechteren sondern meist bessere Laufzeiten liefert.

Um zu untersuchen wie stark die zusätzliche Verwaltung des Constraintgraphen und die Überprüfung auf mögliche disjunkte Partialzerlegungen die Aufzählung verlangsamen, wurde die Zerlegungssuche ohne disjunkte Partialzerlegung gestartet (ZLGSTD). Sie entspricht somit in ihrem Ablauf der Standardsuche, jedoch mit dem zusätzlichen, beschriebenen Aufwand. In Tabelle 4.2 wird deren Laufzeit relativ zur Standardsuche angegeben. Es wird deutlich, dass der zu leistende Mehraufwand in seiner derzeitigen Implementierung so enorm ist, dass sich die durchschnittliche Laufzeit verdoppelt.

	Suchstrategie	$time(STD/ZLGSTD)$ in Testsatz T33	$time(STD/ZLGSTD)$ in Testsatz T54
	MINSIZE	0,4	0,5
hier.	MINNEIGH	0,4	0,5
sum.	MINNEIGH	0,4	0,5

Tabelle 4.2: Durchschnittliche Geschwindigkeit der Zerlegungssuche ohne disjunkte Partialzerlegung relativ zur Standardsuche

Trotz der zusätzlichen Verwaltungsarbeit ist die Zerlegungssuche mit disjunkter Partialzerlegung schneller. Jedoch wird schon in Tabelle 4.1 deutlich, dass die Wahl der Suchstrategie großen Einfluss auf die Laufzeiten hat. Um die beste der drei verfügbaren Suchstrategien herauszufinden, wurden die Laufzeiten der Zerlegungssuche mit und ohne disjunkter Partialzerlegung in Relation gesetzt. Hierdurch wird allein der Geschwindigkeitsgewinn durch

die Verwendung disjunkter Partialzerlegung messbar und die Suchstrategien werden besser vergleichbar. Die Ergebnisse sind in Tabelle 4.3 zusammengefasst.

	Suchstrategie	$time(ZLGSTD/ZLG)$ in Testsatz T33	$time(ZLGSTD/ZLG)$ in Testsatz T54
	MINSIZE	2,8	2,5
hier.	MINNEIGH	4,7	5,2
hier.	MAXCONSTR	4,4	4,4
sum.	MINNEIGH	4,7	5,2
sum.	MAXCONSTR	4,4	4,4

Tabelle 4.3: Durchschnittliche Geschwindigkeit der Zerlegungssuche mit disjunkter Partialzerlegung relativ zur Zerlegungssuche ohne disjunkte Partialzerlegung

Durch die Daten ist erkennbar, dass die Suchstrategie MINNEIGH die besten Resultate erzielt. Dies geschieht unabhängig vom gewählten Testsatz. Die Strategie MAXCONSTR liefert zwar immer bessere Laufzeiten als MINSIZE, ist jedoch im Schnitt langsamer als MINNEIGH. Aufgrund dessen kann die Suchstrategie MINNEIGH, die die Variable mit den wenigsten gebundenen Nachbarn wählt, als die günstigste für die Lösung des SP-CSPs durch Zerlegungssuche betrachtet werden. Dies entspricht auch Untersuchungen im SP-CSP unter Verwendung der Standardsuche beim Vergleich der Suchstrategien MINSIZE und MINNEIGH. Die umfangreiche Reduktion durch Nachbarschaftsbedingungen, auf denen der Fokus von MINNEIGH liegt, sorgt für die guten Laufzeiten mit dieser Suchstrategie.

Aus den Tabellen 4.1 und 4.3 ist ebenfalls zu erkennen, dass sich der Geschwindigkeitsgewinn durch die Verwendung der Zerlegungssuche noch steigern ließe, wenn der Geschwindigkeitsverlust durch den nötigen Mehraufwand sinken würde. Die Implementierung hierfür muss also überprüft und wenn möglich optimiert werden.

Außerdem kann man aus beiden Tabellen ablesen, dass die Verwendung der zwei zur Verfügung stehenden Kombinationsansätze für die Wichtungsfunktionen (hierarchisch und summierend) im Durchschnitt keine signifikanten Laufzeitunterschiede bewirken.

Insgesamt betrachtet kann ein deutlicher Geschwindigkeitszuwachs durch die Verwendung der Zerlegungssuche beobachtet werden. Dieser ist, wie beschrieben, noch ausbaufähig.

Suchbaumbeschaffenheit

Der Grund für den Geschwindigkeitszuwachs liegt in der veränderten Beschaffenheit des Suchbaumes. Wie schon in Kapitel 3 beschrieben wurde, sorgt disjunkte Partialzerlegung dafür, dass ein Suchbaum in einen Wald aus kleineren Suchbäumen aufgespalten wird. Dadurch sinkt die durchschnittliche Anzahl der nötigen Suchverzweigungen fast um das 8-fache was aus Tabelle 4.4 abzulesen ist. Hierzu ist im Vergleich die durchschnittliche Suchbaumbeschaffenheit der Zerlegungssuche ohne disjunkte Partialzerlegung (ZLGSTD) als Referenz angegeben. Diese entspricht der Standardsuche und ist mit allen Suchstrategien kompatibel.

		Testsatz T33		Testsatz T54	
	Suchstrategie	ZLGSTD	ZLG	ZLGSTD	ZLG
	MINSIZE	2232/125/0	281/152/29	2600/541/0	771/503/41
hier.	MINNEIGH	2222/115/0	285/163/42	2357/298/0	306/326/26
hier.	MAXCONSTR	2265/158/0	343/196/34	2541/482/0	480/502/19
sum.	MINNEIGH	2222/115/0	285/163/42	2357/298/0	306/326/26
sum.	MAXCONSTR	2265/158/0	343/196/34	2541/482/0	490/502/19

Tabelle 4.4: Durchschnittliche Knotenzahl unter Verwendung der Zerlegungssuche mit (ZLG) und ohne (ZLGSTD) disjunkte Partialzerlegung (X/Y/Z : X = Anzahl Suchverzweigungen, Y = Anzahl Rekursionsabbrüche, Z = Anzahl disjunkte Partialzerlegungen)

Die Anzahl der Rekursionsabbrüche ohne Lösung bleibt durchschnittlich im Rahmen der Werte, die bei Verwendung der Standardsuche auftreten. Meist ist sie jedoch leicht höher. Dies liegt an einer späten Erkennung von Teilproblemen ohne Lösung. Wenn diese früher erkannt und die Aufzählung der anderen Teilprobleme daraufhin abgebrochen würde, kann die gesamte Knotenzahl weiter verringert werden. Dies zeigt, dass die Reihenfolge der Teilprobleme eine große Rolle spielt und hier weitere Überlegungen die Performanz der Zerlegungssuche noch steigern können.

Zudem wird aus Tabelle 4.4 ersichtlich, dass die durchschnittliche Anzahl von disjunkten Partialzerlegungen allein kaum Aussagen über die Performanz der Zerlegungssuche ermöglicht. So hat zum Beispiel die Zerlegungssuche mit MINSIZE im Testsatz T54 eine höhere durchschnittliche Anzahl Partialzerlegungen als mit der Suchstrategie MINNEIGH, aber trotzdem viel schlechtere Laufzeiten (siehe Tabelle 4.3). Dies liegt an der mehr als doppelten Anzahl Suchverzweigungen gegenüber MINNEIGH.

Zusammenfassend kann man sagen, dass weder die Anzahl von Suchverzweigungen noch die Anzahl von Rekursionsabbrüchen oder disjunkten

Partialzerlegungen allein die Geschwindigkeit der Zerlegungssuche bestimmen, sondern ihre Zusammensetzung. Die Reduktion der Suchverzweigungen bei ähnlicher Anzahl von Rekursionsabbrüchen begründet den Performanzgewinn der Zerlegungssuche. Jedoch ist zum derzeitigen Stand noch kein fester Zusammenhang zwischen den Größen erkennbar. Hierzu müssen weitere und umfangreichere Testsätze zu Rate gezogen werden. Die Verwendung eines hierarchischen oder summierenden Ansatzes für die Kombination der Wertebereiche hat keinerlei Auswirkungen auf die durchschnittliche Suchbaumbeschaffenheit, was direkt aus Tabelle 4.4 abzulesen ist.

4.2.2 Gruppierungsheuristik und Wichtungskombination für die Zerlegungssuche

Gruppierungsheuristik

Im Abschnitt 3.1 wurde eine problemspezifische Gruppierungsheuristik für die Variablenauswahl diskutiert. Die Gruppierungen sind hierbei entsprechend der implementierten Parameter wie folgt benannt:

A : Es findet keinerlei Gruppierung statt.

AH : Zuerst werden alle H-Variablen gebunden, bevor P-Variablen bewertet werden. Diese Strategie folgt dem Wissen, dass die Wertebereiche von H-Variablen (also der H-Kern) in der Regel kleiner sind als die der restlichen Variablen (der umgebende Raum).

SA : Bei dieser Strategie werden zuerst Singlets gebunden, da diese, durch ihre zwei Nachbarschaftsbedingungen zu Variablen ungleichen Typs, die größte erwartete Reduktion der Wertebereiche ermöglichen.

SBb : In diesem Fall wird die in Abschnitt 3.1 vorgeschlagene Reihenfolge „Singlets-Randvariablen-Rest“ verwendet.

Um einen möglichen Geschwindigkeitszuwachs durch die Einführung dieser problemspezifischen Gruppierungen zu veranschaulichen, wurden deren Laufzeiten in Relation zur Laufzeit der ungruppierten Suche (A) gesetzt. Die Ergebnisse sind in Tabelle 4.5 zusammengefasst.

		Testsatz T33			Testsatz T54		
	Suchstrategie	AH	SA	SBb	AH	SA	SBb
	MINSIZE	1	1	1	1	1	1
hier.	MINNEIGH	0,9	1	1	0,8	1,3	1,3
hier.	MAXCONSTR	3,1	4,2	4,4	3,7	5	5,3
sum.	MINNEIGH	0,9	1	1	0,8	1,3	1,4
sum.	MAXCONSTR	1,3	1,5	1,5	1,6	2,2	2,2

Tabelle 4.5: Durchschnittliche Geschwindigkeit der Zerlegungssuche mit verschiedenen Gruppierungsheuristiken relativ zur ungruppierten Variante A (die relativen Geschwindigkeiten entsprechen $time(A/\ldots)$)

Aus Tabelle 4.5 ist gut zu erkennen, dass alle Gruppierungsheuristiken unter Verwendung der Suchstrategie MINSIZE keinen Geschwindigkeitsgewinn erzielen. Der Grund dafür liegt in den Gruppierungen selbst, da Variablen mit kleinen Wertebereichen zumeist der jeweilig ersten Gruppe angehören (zum Beispiel H-Variablen oder Singlets). Somit ändert sich die auf dieser Suchstrategie basierende Reihenfolge kaum.

Für die Suchstrategien MINNEIGH und MAXCONSTR verhält sich dies anders, und im Allgemeinen sind klare Geschwindigkeitsgewinne zu verzeichnen. Es wird deutlich, dass die Gruppierung SBb die besten Laufzeiten aufweist. Etwas längere Laufzeiten werden mit der Gruppierung SA erzielt und unter Verwendung von AH ist die Performanz stark von der gewählten Suchstrategie abhängig.

Da SBb im Vergleich zu SA nur geringe Performanzunterschiede aufweist, liegt der Schluss nahe, dass ein Großteil des Geschwindigkeitszuwaches beider Gruppierungen aus der vorrangigen Behandlung von Singlet-Variablen resultiert. Wie schon im Abschnitt 3.1 dargelegt, haben diese Variablen die größten Aussichten benachbarte Wertebereiche stark einzuschränken. Diese umfangreiche Reduktion und primäre Wahrung der Nachbarschaftsbedingungen führt zu den großen Geschwindigkeitsgewinnen der beiden Gruppierungsheuristiken. Der erweiterte Fokus auf Nachbarschaftsbedingungen durch die Gruppierung SBb wirkt sich sichtbar unter Verwendung der hierarchischen Suchstrategie MAXCONSTR aus.

Die schlechten Resultate der Heuristik AH im Zusammenhang mit der Suchstrategie MINNEIGH liegen an den konträren Ideen der beiden Ansätze. Die Gruppierung AH folgt dem Gedanken, dass H-Variablen die kleinsten Wertebereiche aufweisen, die Suchstrategie hingegen fokusiert auf die Erfüllung von Nachbarschaftsbedingungen. Wenn eine HP-Sequenz nun eine hohe Anzahl Ps mit zwei H-Nachbarn in der Sequenz (P-Singlets) besitzt, wird deren Aufzählung aufgrund der Gruppierung solange verzögert, bis alle

H-Variablen gebunden sind. Die umfangreiche Reduktion, welche durch die Bindung der P-Singlets möglich wäre, wird durch die Gruppierungsheuristik unterbunden. Auch wird die meist geringe Wertebereichsgröße von P-Singlets nicht ausgenutzt. Dies führt zu den in Tabelle 4.5 schlechtesten Laufzeiten.

Die sehr viel schnelleren Laufzeiten der Gruppierungen AH, SA und SBb unter Verwendung der hierarchischen Suchstrategie MAXCONSTR basieren auf den schlechten Laufzeiten ohne Gruppierung (A) für diese Strategie. Hier wird deutlich, dass der Knotengrad im Constraintgraphen allein ohne eine Gruppierung nicht immer aussagekräftig genug ist, um eine gute Variablenauswahl zu treffen. Es ist zu erwarten, dass sich diese Tendenz bei größeren Problemen fortsetzt und noch stärker manifestiert. Der Grund für dieses Laufzeitverhalten liegt vermutlich darin, dass viele Variablen mit hohem Knotengrad große Wertebereiche aufweisen. Hierdurch wird deren Aufzählung aufwändig und der Performanzgewinn der Zerlegungssuche durch weniger Suchverzweigungen sinkt. Die genannten Gruppierungsheuristiken wirken diesem offensichtlich entgegen, da sie, wie schon erwähnt, Variablen mit erwartungsgemäß kleinen Wertebereichen bevorzugen.

Zusammenfassend ist die Gruppierungsheuristik SBb, bei der erst Singlets, dann Randvariablen und danach die Restlichen gebunden werden, die beste Wahl für die vollständige Aufzählung aller Lösungen im SP-CSP.

Hierarchische versus summierende Wichtungskombination für die Variablenauswahl

In Abschnitt 3.3.1 wurden zwei mögliche Kombinationsansätze für Wichtungsfunktionen vorgestellt. Zum einen der hierarchische Ansatz und zum anderen eine summierende Variante. Um die Performanz beider Herangehensweisen zu diskutieren, wurden ihre Laufzeiten in Relation gesetzt. Tabelle 4.6 zeigt die Resultate dieser Studie.

		Testsatz T33				Testsatz T54			
	Suchstrategie	A	AH	SA	SBb	A	AH	SA	SBb
sum.	MINNEIGH	1	1	1	1	1	1	1	1
sum.	MAXCONSTR	1,8	1	1	1	2,1	1	1	1

Tabelle 4.6: Durchschnittliche Geschwindigkeit der Zerlegungssuche mit summierender Kombination der Wichtungsfunktionen und verschiedenen Gruppierungen relativ zum hierarchischen Ansatz ($time(hier./sum.)$)

Es ist eindeutig, dass beide Ansätze in fast allen Fällen keinerlei messbare Laufzeitunterschiede aufweisen. Der Grund für dieses Verhalten wird schon

in Abschnitt 4.2.1 bei der Betrachtung der Suchbaumeigenschaften angesprochen. So liefern beide Varianten für die in Tabelle 4.6 angegebenen Parametersätze durchschnittlich identische Suchbaumeigenschaften. Es bleibt zu untersuchen, ob andere Parametersätze für den summierenden Ansatz längere oder kürzere Laufzeiten liefern. Die derzeitige Gewichtung der Hauptwichtungsfunktion mit 2 und der zusätzlichen Wichtung nach Wertebereichsgröße mit 1 (siehe Abschnitt 4.1.2), liefern offensichtlich äquivalente Resultate zum hierarchischen Ansatz.

Einzig für die Suchstrategie MAXCONSTR ist eine Laufzeitdifferenz für die Kombinationsansätze erkennbar, wenn keine Gruppierung der Variablen vorgenommen wird. Die Gründe hierfür wurden schon im vorangehenden Abschnitt erläutert. Jedoch wird deutlich, dass dieses schlechte Laufzeitverhalten durch die Einführung einer Gruppierungsheuristik vollständig behoben werden kann.

Zusammenfassend kann somit die hierarchische Kombination der Wichtungsfunktionen als günstiger betrachtet werden, da für sie keinerlei Normierung der Einzelwichtungen notwendig ist. Der hierfür anfallende Mehraufwand, wenn auch gering, entfällt bei ihrer Verwendung.

Kapitel 5

Zusammenfassung und Ausblick

Die in dieser Arbeit neu vorgestellte Zerlegungssuche zur Lösung von Constraint Satisfaction Problemen (CSPs) bietet einen schnellen und exakten Ansatz zur vollständigen Aufzählung aller optimalen Strukturen einer HP-Sequenz durch Constraintprogrammierung. Durch die Anwendung disjunkter Partialzerlegung, bei der ein CSP in unabhängige Teilprobleme aufgespalten wird, kann der Suchaufwand gegenüber dem normalen Ablauf eines Standardsuchverfahrens wie Maintaining-Arc-Consistency (MAC) deutlich verringert werden. Dies geschieht unter Vermeidung der sonst auftretenden Redundanz und unter Verwendung der gleichen Suchstrategie.

Die Zerlegungssuche stellt einen allgemeinen Ansatz dar, um andere Suchverfahren zur Lösung von CSPs zu erweitern und zu beschleunigen. Dies wird erreicht, indem die redundante Lösung von Teilproblemen verhindert wird. Sie kann somit als Meta-Suche betrachtet werden. Die in Kapitel 4 vorgestellte Implementierung und deren Laufzeitbetrachtungen unterstützen die theoretischen Aussagen aus Kapitel 3. Die vorliegende Umsetzung der Zerlegungssuche ist so konzipiert, dass sie problemlos auf andere Gittermodelle und Nachbarschaften angewendet werden kann. Hierzu müssen lediglich entsprechende H-Kerne vorberechnet werden.

Während der Diskussion über die Ergebnisse der Laufzeitanalysen sind Optimierungsmöglichkeiten der derzeitigen Implementierung sichtbar geworden. Diese werden demnächst weiter untersucht. Außerdem soll getestet werden ob aus der Identifikation von Artikulationspunkten ein Nutzen entsteht.

Da die Anwendung eines Symmetrieausschlusses im Allgemeinen erheblich bessere Laufzeiten ermöglicht [BW98], sollte über eine Kombination mit der Zerlegungssuche nachgedacht werden. Auch eine Einbindung der Wertebereichsaufspaltung à la J. Larrosa [Lar97] (siehe Abschnitt 1.4) wäre im Zusammenhang mit Brückenidentifikation lohnenswert. Die Anwendung von Zerlegungssuche auf andere Suchverfahren als das MAC und die Performanz

einer solchen Kombination könnte zudem interessant sein.

Eine andere Erweiterungsmöglichkeit der Zerlegungssuche wäre die Verwendung einer komplexeren Energiefunktion. Zum Beispiel unterteilt das sogenannte HPNX-Modell, eine Erweiterung des HP-Modells, die modellierten Aminosäuren in vier statt in zwei Gruppen [BB97, BWBB99].

H	für	hydrophobe,
P	für	positiv geladene,
N	für	negativ geladene und
X	für	neutrale Aminosäuren.

Aufbauend auf dieser Einteilung kann die Energiefunktion für zwei benachbarte Sequenzpositionen durch die folgende Tabelle dargestellt werden.

	H	P	N	X
H	-4	0	0	0
P	0	1	-1	0
N	0	-1	1	0
X	0	0	0	0

Die Unterscheidung der polaren Aminosäuren und die Einbindung ihrer Anziehungskräfte liefert somit eine bessere Energiefunktion. Die Auswirkungen der Zerlegungssuche auf dieses Gitterproteinmodell sollten untersucht werden.

Zudem wäre es interessant, die Zerlegungssuche auf andere CSPs anzuwenden und ihre Performanz für diese zu testen. In einem ersten Schritt könnten Analysen mit zufällig generierten CSPs allgemeine Abschätzungen über die Leistungsfähigkeit der Zerlegungssuche liefern.

Zusammenfassend kann man sagen, das mit der neu eingeführten Zerlegungssuche die vollständige Aufzählung aller Lösungen eines CSPs im Vergleich zu herkömmlichen Verfahren stark beschleunigt werden kann. Ihre Anwendung auf das Strukturvorhersage-CSP (SP-CSP) hat gezeigt, dass für ihre erhöhte, von disjunkten Partialzerlegungen abhängige Performanz problemspezifische Heuristiken und die richtige Suchstrategie entscheidend sind. Es wurde im Verlauf der Arbeit die günstigste Suchstrategie (MINNEIGH) und die hierfür beste Gruppierungsheuristik (SBb) identifiziert. Somit ist die vollständige Aufzählung aller optimalen Strukturen beliebiger Gitterproteine des HP-Modells durch dynamisch angewandte disjunkte Partialzerlegung des assoziierten SP-CSPs in exakter und schneller Weise möglich.

Literaturverzeichnis

[ARDK96] A. Sali A. R. Dinner and M. Karplus. The folding mechanism of larger model proteins: Role of native structure. *Proc. Natl. Acad. Sci. USA*, 93:8356–8361, 1996. 14

[Bar99] Roman Bartak. Constraint programming - what is behind? In J. Figwer, editor, *Proceedings of the Workshop on Constraint Programming in Decision and Control*, Poland, June 1999. 24, 31, 34

[Bar01] Roman Barták. Theory and practise of constraint programming. In *3rd Workshop on Constraint Programming for Decision and Control (CPDC2001)*, pages 7–14. Wydavnictvo Pracovni Komputerowej, Gliwice, Poland, 2001. 28, 31, 32

[BB97] Erich Bornberg-Bauer. Chain growth algorithms for HP-type lattice proteins. In *Proc. of the* 1^{st} *Annual International Conference on Computational Molecular Biology (RECOMB)*, pages 47 – 55. ACM Press, 1997. 74

[BD96] Thomas C. Beutler and Ken A. Dill. A fast conformational search strategy for finding low energy structures of model proteins. *Protein Science*, 5:2037–2043, 1996. 21

[BL98] B. Berger and T. Leighton. Protein folding in the hydrophobic-hydrophilic (HP) model is NP-complete. *Journal of Computational Biologie*, 5(1):27–40, 1998. 14, 20

[BS05] Thang N. Bui and Gnanasekaran Sundarraj. An efficient genetic algorithm for predicting protein tertiary structures in the 2d hp model. In *GECCO '05: Proceedings of the 2005 conference on Genetic and evolutionary computation*, pages 385–392, New York, NY, USA, 2005. ACM Press. 21

[BT98] Carl-Ivar Branden and John Tooze. *Introduction to protein structure*. Garland Publishing, Inc., 2nd edition, 1998. 16, 20

[BW98] Rolf Backofen and Sebastian Will. Excluding symmetries in concurrent constraint programming. In *Workshop on Modeling and Computing with Concurrent Constraint Programming*, 1998. 73

[BW05] Rolf Backofen and Sebastian Will. A constraint-based approach to fast and exact structure prediction in three-dimensional protein models. *Journal of Constraints*, 2005. accepted for publication. 14, 15, 21, 23, 24, 27, 34

[BWBB99] Rolf Backofen, Sebastian Will, and Erich Bornberg-Bauer. Application of constraint programming techniques for structure prediction of lattice proteins with extended alphabets. *Bioinformatics*, 15(3):234–242, 1999. 74

[CBB02] H.S. Chan and E. Bornberg-Bauer. Perspectives on protein evolution from simple exact models. *Applied Bioinformatics*, 1:121–144, 2002. 14

[CGP+98] P. Crescenzi, D. Goldman, C. Papadimitriou, A. Piccolboni, and M. Yannakakis. On the complexity of protein folding. *JCB*, 5(3):423–65, 1998. 14, 20

[Cre90] Thomas E. Creighton. Protein folding. *Biochemical Journal*, 270:1–16, 1990. 13

[DC97] Ken A. Dill and Hue S. Chan. From levinthal to pathways to funnels. *Nature Structural Biology*, 4(1):10–19, 1997. 14

[DFC93] Ken A. Dill, Klaus M. Fiebig, and Hue Sun Chan. Cooperativity in protein-folding kinetics. *Proc. Natl. Acad. Sci. USA*, 90:1942–1946, 1993. 21

[FH95] E. C. Freuder and P. D. Hubbe. A disjunctive decomposition control schema for constraint satisfaction. In V. Saraswat and P. Van Hentenryck, editors, *Principles and Practice of Constraint Programming*, pages 319–335. MIT Press, London, 1995. 24

[FHSW02] Christoph Flamm, Ivo L. Hofacker, Peter F. Stadler, and Michael T. Wolfinger. Barrier trees of degenerate landscapes. *Z.Phys.Chem*, 216:155–173, 2002. 14

[GKS93] A. Godzik, A. Kolinski, and J. Skolnick. Lattice representations of globular proteins: How good are they? *J. Comput. Chemistry*, 14:1194–1202, 1993. 14

[GLF00] G. Gottlob, N. Leone, and F.Scarcello. A comparison of structural csp decomposition methods. *Artif. Intell.*, 124(2):243–282, 2000. 24

[GLS02] G. Gottlob, N. Leone, and F. Scarcello. Hypertree decomposition and tractable queries. *J. of Computer and System Sciences*, 64(3):579–627, 2002. 24

[ilo05a] *ILOG Concert Technology Reference Manual*. ILOG, Inc, April 2005.

[ilo05b] *ILOG Solver 6.1 Reference Manual*. ILOG, Inc, April 2005.

[ilo05c] *ILOG Solver 6.1 User's Manual*. ILOG, Inc, April 2005. 55

[JT02] Philippe Jégou and Cyril Terrioux. Hybrid backtracking bounded by tree-decomposition of constraint networks. *LSIS*, 2002. 24

[KS04] A. Kolinskia and J. Skolnick. Reduced models of proteins and their applications. *Polymer*, 45:511–524, 2004. 14

[Lar97] Javier Larrosa. Merging constraint satisfaction subproblems to avoid redundant search. In *IJCAI'97*, 1997. 23, 73

[LD89] Kit Fun Lau and Ken A. Dill. A lattice statistical mechanics model of the conformational and sequence spaces of proteins. *Macromolecules*, 22:3986–3997, 1989. 13, 17

[LMW03] Neal Lesh, Michael Mitzenmacher, and Sue Whitesides. A complete and effective move set for simplified protein folding. In *RECOMB '03: Proceedings of the seventh annual international conference on Research in computational molecular biology*, pages 188–195, New York, NY, USA, 2003. ACM Press. 21

[LW01] Faming Liang and Wing Hung Wong. Evolutionary monte carlo for protein folding simulations. *Journal of Chemical Physics*, 115:7:3374–3380, 2001. 14, 21

[MS01] Leonid Mirny and Eugene Shakhnovich. Protein folding theory: from lattice to all-atom models. *Annual Review of Biophysics and Biomolecular Structure*, 30:361–396, 2001. 14

[RBB97] A. Renner and E. Bornberg-Bauer. Exploring the fitness landscapes of lattice proteins. In *2nd. Pacif. Symp. Biocomp.* Singapore, 1997. 14

[RD90] F. ROssi and V. Dhar. On the equivalenve of constraint satisfaction problems. In *ECAI90*, pages 550–556. Stockholm, Sweden, 1990. 29

[RN03] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Englewood Cliffs, NJ, 2nd edition edition, 2003. 24

[Sam05] Marko Samer. Hypertree-decomposition via branch-decomposition. In *IJCAI'05*, pages 1535–1536, 2005. 24

[SSHF99] M.-C. Silaghi, D. Sam-Haroud, and B. Faltings. Intelligent domain splitting for csps with ordered domains. In *???*, 1999. 23

[UM93] R. Unger and J. Moult. Finding lowest free energy conformation of a protein is an np-hard problem: Proof and implications. *Bull. Math. Biol.*, 55:1183–1198, 1993. 14

[WBBC05] Richard Wroe, Erich Bornberg-Bauer, and Hue Sun Chan. Comparing folding codes in simple heteropolymer models of protein evolutionary landscape: robustness of the superfunnel paradigm. *Biophys J*, 88(1):118–31, 2005. 14

[Wes96] D. B. West. *Introduction to Graph Theory*. Prentice Hall, 1996. 47

[Wil05] Sebastian Will. *Exact, Constraint-Based Structure Prediction in Simple Protein Models*. PhD thesis, Friedrich-Schiller-University Jena, 2005. 17, 20

[Wol04] Michael T. Wolfinger. *Energy Landscapes of Biopolymers*. PhD thesis, University Vienna, 2004. 14

[YD95] K Yue and KA Dill. Forces of tertiary structural organization in globular proteins. *PNAS*, 92(1):146–50, 1995. 21

[YE02] Berrin Yanikoglu and Burak Erman. Minimum energy configurations of the 2-dimensional hp-model of proteins by self-organizing networks. *Journal of Computational Biology*, 9:4:613–620, 2002. 21

[YFT+95] K. Yue, K. M. Fiebig, P. D. Thomas, H. S. Chan, E. I. Shakhnovich, and K. A. Dill. A test of lattice protein folding algorithms. *PNAS*, 92(1):325–9, 1995. 14

Selbstständigkeitserklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe.

Jena,