communications in mathematical
match
and in computer chemistry

no. 39, March 1999

ISSN 0340-6253

MATCDY (39) 39-126 (1999)

Strategien zur Konstruktion diskreter Strukturen und ihre Anwendung auf molekulare Graphen

Thomas Grüner[1]
Lehrstuhl II für Mathematik, Universität Bayreuth
D-95440 Bayreuth

Überblick

Wir möchten uns in dieser Arbeit mit der Theorie der Konstruktion diskreter Strukturen beschäftigen. Die dabei auftretenden Schwierigkeiten sind

– die Konstruktion der numerierten Strukturen, sowie
– das Isomorphieproblem.

Bisherige Arbeiten aus diesem Problemkreis befaßten sich zumeist ausschließlich (oder zumindest schwerpunktsmäßig) mit der Lösung des Isomorphieproblems.
Nachdem im Kapitel I die grundlegenden Algorithmen zum Rechnen mit Permutationsgruppen im Computer sowie zur Berechnung einer kanonischen Numerierung vorgestellt wurden, beschäftigt sich Kapitel II mit *Strategien* zur Konstruktion diskreter Strukturen. Zunächst werden allgemeine Konstruktionsstrategien vorgestellt sowie ihre jeweiligen Vor- und Nachteile diskutiert. Anschließend befassen wir uns mit einigen interessanten *problemspezifischen Konstruktionsalgorithmen*, die natürlich im Vergleich zu den allgemeinen Konstruktionsstrategien spezieller, aber zur Lösung des jeweiligen Problems auch effizienter sind.
Der Schwerpunkt dieser Arbeit schließlich (Kapitel III) liegt auf der Konstruktion diskreter Strukturen (insbesondere von molekularen Graphen) zu gegebenen *Nebenbedingungen*, die den Lösungsraum sehr stark einschränken können, so daß die Lösung des Isomorphieproblems in diesen Fällen *nicht* spielentscheidend ist. Vielmehr ist es wichtig, eine leistungsfähige Strategie zur Konstruktion der numerierten Strukturen einzusetzen, die

[1] Gefördert durch das **BMBF** Projekt „Mathematisch-algorithmische Aspekte der Kombinatorischen Chemie" (03KE7BA14)

möglichst unabhängig von den speziellen Nebenbedingungen ist, so daß leicht zusätzliche Restriktionstypen hinzugefügt werden können.
Im folgenden werden drei Konstruktionsstrategien vorgestellt, die quasi das Handwerkszeug zur Konstruktion diskreter Strukturen darstellen:

1.) Homomorphieprinzip
Mit dem Homomorphieprinzip ist es möglich, durch Anwendung geeigneter Homomorphismen komplizierte mathematische Probleme auf einfachere zurückzuführen. Will man Bahnenrepräsentanten und zugehörige Stabilisatoren einer Gruppenoperation berechnen, so bildet man das Problem mittels einer homomorphen Abbildung auf eine einfacher zu lösende Gruppenoperation ab, berechnet dort Repräsentanten samt Stabilisator und muß dann nur noch mit dem Urbild des gefundenen Stabilisators auf der Urbildmenge des Repräsentanten operieren. Hat man eine Folge solcher Homomorphismen, so wird im allgemeinen die operierende Gruppe immer kleiner, weil man jeweils nur mit dem Stabilisator weiteroperieren muß.
Der Nachteil des Homomorphieprinzips besteht offensichtlich darin, daß man geeignete Homomorphismen benötigt, um das jeweilige Konstruktionsproblem zu vereinfachen, was mit (möglicherweise vielen) zusätzlichen Restriktionen keineswegs trivial ist.
Das Homomorphieprinzip wurde beispielsweise zur Konstruktion von schlichten Graphen mit vorgegebener Gradpartition verwendet. Bei dieser Anwendung stellt sich die Situation wie folgt dar:

- Die Konstruktion der numerierten Strukturen ist sehr einfach.
- Eine Gruppe G, dargestellt als labeled branching, operiert auf einer Menge von 0/1-Matrizen mit vorgegebenen Zeilen- und Spaltensummen (an dieser Stelle kann ordnungstreue Erzeugung eingesetzt werden).
- Ist die operierende Gruppe G trivial (d.h. $G = \{id\}$), so ist es möglich, sogenannte implizite Lösungen zu finden.
- Mit steigender Knotenzahl steigt i.a. auch die Generierungsgeschwindigkeit: so können etwa bei ca. 30-50 Knoten bis zu 10^{30} paarweise nicht isomorphe Graphen mit vorgegebener Gradpartition pro Sekunde **konstruiert** werden.

2.) Ordnungstreue Erzeugung
Zur Lösung von Problemen, die sich durch Homomorphismen nicht weiter vereinfachen lassen, bietet sich die ordnungstreue Erzeugung an. Ordnungstreue Erzeugung beruht auf einer Totalordnung $\leq$ auf einer Menge Z. Für eine Gruppe G, die auf Z operiert, ist die Menge

$$rep_{<}(G\backslash\backslash Z) := \{z \in Z \mid \forall g \in G \ : \ z \leq z^g\}$$

der minimalen Bahnrepräsentanten eine kanonische Transversale der Bahnen.

Die Menge $(Z, \leq)$ sei total geordnet und die Gruppe G operiere auf Z. Wie bereits oben erwähnt, ist dann $rep_{<}(G\backslash\backslash Z)$ eine kanonische Transversale der Bahnen von G auf Z.
Sei $M = \{m_1 < ... < m_i\}$ und $Z = 2^M$. Definiere $M_j := \{m_1, ..., m_j\}$. Die numerierten Strukturen werden in lexikographisch aufsteigender Reihenfolge konstruiert und für jeden Kandidaten $K \in 2^{M_j}$ mit $1 \leq j \leq i$ wird ein sogenannter *Minimalitätstest* durchgeführt, d.h. es wird geprüft, ob in der Bahn von K bzgl. der Operation von G auf Z ein lexikographisch kleineres Element $z \in Z$ existiert. Ist kein solches Element z vorhanden,

dann werden im Fall $j < i$ die Fortsetzungen von K (dies sind dann Elemente von M_{j+1}) betrachtet. Falls $j = i$, dann gehört K zur gesuchten Lösungsmenge. Diese Erzeugungsstrategie geht zurück auf Read ([43]).

Alternativ besteht die Möglichkeit, den Minimalitätstest nur für Kandidaten $K \in 2^M$ durchzuführen. Falls K dann nicht minimal in seiner Bahn ist, so wird versucht, diese Information zu nutzen, um einige der folgenden Kandidaten zu überspringen. Diese Vorgehensweise wird *Lerneffekt* ([16])genannt. Bei der ordnungstreuen Erzeugung können zusätzliche Restriktionen i.a. leichter berücksichtigt werden als bei Anwendung des Homomorphieprinzips, natürlich erreicht man mit ordnungstreuer Erzeugung aber keine so großen Lösungsanzahlen, wie es mit Hilfe des Homomorphieprinzips manchmal möglich ist. Ordnungstreue Erzeugung wird z.B. eingesetzt in

- **MOLGEN** (bis Version 3.5).
- **Genreg**, einem Programm zur Generierung regulärer Graphen [37].

3.) Zielgerichtete Erzeugung
Die zielgerichtete Erzeugung, auf die wir uns in dieser Arbeit konzentrieren möchten, ist eine sehr variable Konstruktionsstrategie, die gegenüber der ordnungstreuen Erzeugung den Vorteil hat, daß sie keine feste Konstruktionsreihenfolge benutzt, sondern sich dynamisch dem Konstruktionsproblem anpassen kann. Restriktionen werden in der Regel während der Konstruktion geprüft und steuern die Konstruktionsreihenfolge. Falls man also nur relativ wenige Strukturen mit ganz speziellen Eigenschaften konstruieren möchte, so wird man sich in vielen Fällen für diese Konstruktionsstrategie entscheiden.

Eben erwähnte Konstruktionsstrategie wurde für die Implementation der neuesten **MOLGEN**-Version verwendet. Unser Algorithmus mußte die folgenden Anforderungen erfüllen:

- Modularer Aufbau,
- Erweiterbarkeit,
- Testen der Nebenbedingungen **während** der Generierung.

In wenigen Worten ausgedrückt, füllen wir die Adjazenzmatrix Zeile für Zeile und entscheiden nach jeder Zeile dynamisch während der Generierung, in Abhängigkeit von den gegebenen Restriktionen und dem aktuellen Füllzustand der Adjazenzmatrix, welche Zeile als nächstes gefüllt wird. Natürlich werden zur Berechnung der Füllreihenfolge „nur" Heuristiken benutzt, die jedoch in vielen Fällen zu Geschwindigkeitssteigerungen führen können und durchaus geeignet sind, größere Konstruktionsprobleme erst lösbar zu machen. Vor jedem Einfügen einer zusätzlichen Kante wird eine Testfunktion gerufen, die anhand der gegebenen Restriktionen und des aktuellen Molekülanfangsstücks entscheidet, ob man nach dem Einfügen dieser Kante noch zu einer gültigen Gesamtlösung gelangen kann. Durch das Rufen dieser Testfunktion entsteht natürlich ein gewisser Overhead, der bei „einfacheren" Konstruktionsproblemen eher zu einem Geschwindigkeitsverlust führt. Dies kann jedoch in Kauf genommen werden, weil man in der Praxis sowieso meist nur an einer überschaubaren Zahl von Lösungen interessiert ist; d.h. wird die Lösungsanzahl zu hoch, so wird man wahrscheinlich die Konstruktion abbrechen, und versuchen, durch Angabe weiterer Nebenbedingungen die Lösungsmenge zu verkleinern.

Weil wir uns bei dieser Strategie auf die Konstruktion der numerierten Strukturen konzentrieren, können wir den Isomorphietest einfach mit kanonischer Numerierung und einer Hashverwaltung durchführen. Selbstverständlich wird aber die sog. „Semikanonizität" bereits während der Generierung berücksichtigt.
Mit Hilfe dieser neuen Konstruktionsstrategie wurde der **MOLGEN4.0**-Generator implementiert, der u.a. die folgenden Eingaben verarbeiten kann:

- (weiche) Bruttoformel,
- Vorgabe von Atomtypen (Atomname, Valenz, Ladung, Radikalstelle),
- Vorgabe nichtüberlappender Fragmente,
- Goodlisteinträge (können überlappen),
- verbotene Fragmente,
- sehr variable Beschreibung der Umgebung von Fragmenten,
- Wasserstoffverteilung,
- Hybridisierungszustände,
- Anzahl von Kreisen bestimmter Länge,
- Anzahl von Bindungen bestimmter Vielfachheit,
- Anzahl von ^{13}C-NMR Signalen.

Inhaltsverzeichnis

I. Grundlegende Algorithmen

1 Diskrete Strukturen und Backtracking

1.1 Partitionen, Operationen und Nebenklassen

1.1.1 Definition:
Aus technischen Gründen definieren wir für $n \in \mathbb{N}$

$$\bar{n} := \{0, ..., n-1\} \text{ und } \underline{n} := \{1, ..., n\}.$$

1.1.2 Definition:

- Zu $n \in \mathbb{N}, n > 0$ heißt jede endliche Folge natürlicher Zahlen

$$\lambda := (\lambda_1, \lambda_2, ..., \lambda_m) \text{ mit } \sum_i \lambda_i = n$$

  Partition von n (kurz $\lambda \models n$). Gilt darüber hinaus, daß $\lambda_1 \geq \lambda_2 \geq ...$, so nennen wir die Partition λ *kanonisch* (kurz $\lambda \vdash n$).
- Durch die Zerlegung von $\Omega = \{\omega_1, ..., \omega_n\}$ in eine Mengenpartition

$$\Omega_i^\lambda := \left\{ \omega_j \,\middle|\, \sum_{\nu=1}^{i-1} \lambda_\nu < j \leq \sum_{\nu=1}^{i} \lambda_\nu \right\},$$

  bei der die i-te Menge genau λ_i Elemente enthält, läßt sich die zu $\lambda \models n$ gehörige *Younguntergruppe* der S_Ω definieren als :

$$S_{\Omega,\lambda} := \prod_{i \in \underline{m}} S_{\Omega_i^\lambda}.$$

1.1.3 Definition:
Sei G eine endliche Gruppe und Ω eine nichtleere endliche Menge. Eine Abbildung

$$\varphi : \Omega \times G \longrightarrow \Omega, \qquad (\omega, g) \longmapsto \omega^g := \varphi(\omega, g)$$

heißt *Operation* von G auf Ω, falls φ mit der Verknüpfung in G verträglich ist, und falls id die Elemente von Ω fest läßt. Es muß also gelten:

- $\forall \omega \in \Omega \quad \forall g, g' \in G \;:\; \omega^{gg'} = (\omega^g)^{g'}$
- $\forall \omega \in \Omega \;:\; \omega^{id} = \omega$

Operiert G auf Ω, so wird diese Operation mit ${}_G\Omega$ bezeichnet. Die Operation heißt *transitiv*, falls gilt:

$$\forall \omega, \omega' \in \Omega \quad \exists g \in G \ : \ \omega^g = \omega'.$$

Mit Hilfe des Begriffs der Operation lassen sich eine ganze Reihe weiterer Gruppen und Mengen definieren:

1.1.4 Definition:
Sei G eine Gruppe, die auf einer nichtleeren Menge Ω operiert. Seien $U \subseteq G, \omega \in \Omega$ und $\Delta \subseteq \Omega$.

i) $C_G(\Delta) := \{g \in G \mid \forall \delta \in \Delta \ : \ \delta^g = \delta\}$ heißt *Zentralisator* oder *punktweiser Stabilisator* von Δ in G bzgl. ${}_G\Omega$. Ist $\Delta = \{\delta\}$ so schreibt man auch $C_G(\delta)$.
ii) $N_G(\Delta) := \{g \in G \mid \forall \delta \in \Delta \ : \ \delta^g \in \Delta\}$ heißt *Normalisator* oder *mengenweiser Stabilisator* von Δ in G bzgl. ${}_G\Omega$.
iii) $Fix_\Omega(U) := \{\omega \in \Omega \mid \forall u \in U \ : \ \omega^u = \omega\}$ heißt *Fixpunktmenge* von $U \subseteq G$. Ein Element von $Fix_\Omega(U)$ heißt *Fixpunkt* von U.
iv) $\omega^G := \{\omega^g \in \Omega \mid g \in G\}$ heißt *Bahn* von ω unter G.
v) $G\backslash\backslash\Omega := \{\omega^G \mid \omega \in \Omega\}$ ist die Menge der Bahnen der Operation ${}_G\Omega$.

1.1.5 Definition:
Sei G eine Gruppe, Ω eine Menge und ${}_G\Omega$ eine Operation. Eine Menge $T \subseteq \Omega$ heißt *vollständiges Repräsentantensystem* oder *Transversale* der Bahnen dieser Operation, falls gilt:

$$\forall B \in G\backslash\backslash\Omega \ : \ |B \cap T| = 1.$$

Eine Transversale T enthält also aus jeder Bahn genau ein Element. Die Menge aller Transversalen wird mit $\mathcal{T}(G\backslash\backslash\Omega)$ bezeichnet.
Der Begriff der Bahn aus Definition 1.1.4 läßt sich auch über eine Äquivalenzrelation definieren. Dies hat den Vorteil, daß man elementare Eigenschaften von Bahnen ohne Rechnung erkennen kann.

1.1.6 Satz:
Sei G eine Gruppe, Ω eine Menge und ${}_G\Omega$ eine Operation. Dann gilt:

i) Je zwei Bahnen von G auf Ω sind entweder gleich oder disjunkt:

$$\forall \omega, \omega' \in \Omega \ : \ \omega^G \cap \omega'^G \neq \emptyset \quad \Longleftrightarrow \quad \omega' \in \omega^G.$$

ii) $$\forall T \in \mathcal{T}(G\backslash\backslash\Omega) \ : \ \Omega = \dot{\bigcup_{\omega \in T}} \omega^G.$$

iii) Klassengleichung : $$\forall T \in \mathcal{T}(G\backslash\backslash\Omega) \ : \ |\Omega| = \sum_{\omega \in T} |\omega^G|.$$

1.1.7 Definition:
Sei G eine Gruppe, Ω eine Menge und ${}_G\Omega$ eine Operation. Weiter sei $\Gamma \in \mathcal{T}(G\backslash\backslash\Omega)$ eine Transversale der Bahnen dieser Operation. Die Abbildung $\rho : \Omega \longrightarrow G$ heißt *fusionierende Abbildung*, falls gilt:

$$\forall \omega \in \Omega \ : \ \omega^{\rho(\omega)} \in \Gamma$$

$\rho(\omega)$ bildet somit ω auf den zugehörigen Bahnrepräsentanten ab.

1.1.8 Definition:
Seien G, A, B Gruppen mit $A \leq G$ und $B \leq G$. Sei $g \in G$.

i) $Ag := \{ag \mid a \in A\}$ heißt *Rechtsnebenklasse* von A in G.
Die Menge aller Rechtsnebenklassen von A in G wird mit $A \setminus G$ bezeichnet.
ii) $gA := \{ga \mid a \in A\}$ heißt *Linksnebenklasse* von A in G.
Die Menge aller Linksnebenklassen von A in G wird mit G/A bezeichnet.
iii) $AgB := \{agb \mid a \in A, b \in B\}$ heißt *Doppelnebenklasse* von A und B in G.
Die Menge aller Doppelnebenklassen von A und B in G wird mit $A \setminus G/B$ bezeichnet.

Die in Definition 1.1.8 auftretenden Nebenklassen lassen sich auch als Bahnen der Untergruppen A und B auf G auffassen. Als Operation wird dabei die Multiplikation von rechts bzw. von links verwendet. (Bei der Multiplikation von links ist mit dem Inversen zu multiplizieren!)

1.1.9 Satz:
Sei G eine Gruppe, Ω eine Menge und ${}_G\Omega$ eine Operation. Sei weiter $\omega \in \Omega$. Dann ist die Abbildung

$$\phi : \quad \omega^G \longrightarrow C_G(\omega) \setminus G \qquad\qquad \omega^g \longmapsto C_G(\omega)g$$

eine Bijektion. Insbesondere gilt:

$$|\omega^G| = \frac{|G|}{|C_G(\omega)|}.$$

Beweis:
Es gilt:

$$\omega^{g'} = \omega^g \quad \Leftrightarrow \quad \omega^{g'g^{-1}} = \omega \quad \Leftrightarrow \quad g'g^{-1} \in C_G(\omega) \quad \Leftrightarrow \quad C_G(\omega)g = C_G(\omega)g'.$$

□

1.2 Diskrete Strukturen

1.2.1 Definition:
Eine *(numerierte) diskrete Struktur* Δ ist ein Tripel $\Delta = \{X, Y, f\}$.
Dabei sind X, Y Mengen und $f \in Y^X$ eine Abbildung von X nach Y.

1.2.2 Bemerkung:
Seien X, Y Mengen und G eine Gruppe, die auf X operiert. Dann ist mit

$$\varphi : Y^X \times G \longrightarrow Y^X, \qquad (\alpha, g) \longmapsto \alpha^g$$

auf folgende Weise eine Operation definiert: $\forall x \in X \ : \ (\alpha^g)(x) := \alpha(x^{g^{-1}})$.
Denn es gilt:

$$(\alpha^{g_1 g_2})(x) = \alpha(x^{g_2^{-1} g_1^{-1}}) = \alpha\left((x^{g_2^{-1}})^{g_1^{-1}}\right) = (\alpha^{g_1})(x^{g_2^{-1}}) = ((\alpha^{g_1})^{g_2})(x)$$

und

$$\forall x \in X \quad \forall \alpha \in Y^X \ : \ (\alpha^{id})(x) = \alpha(x^{id}) = \alpha(x)$$

1.2.3 Definition:
Seien X, Y Mengen, $f \in Y^X$, und U eine Gruppe, die auf X operiert. Die *Automorphismengruppe* von f bzgl. U ist

$$Aut_U(f) := \{u \in U \mid f^u = f\}$$

1.2.4 Graphen. 1.2.5 Definition:
Sei $p \in \mathbb{N}$. Wir definieren für $V := \{v_1, ..., v_p\}$, $p \in \mathbb{N}$

$$V^{[2]} := \{\{i, j\} \subseteq V \mid i \neq j\}$$

als die Menge der Zweierteilmengen von V. Sei $m \in \mathbb{N}, m > 2$. Dann heißen die Elemente von

– $\bar{m}^{V^{[2]}}$ numerierte Multigraphen mit p Knoten.
– $\{0, 1\}^{V^{[2]}}$ numerierte schlichte Graphen mit p Knoten.

1.2.6 Definition:
Seien $m, p \in \mathbb{N}, m > 2$ und $V := \{v_1, ..., v_p\}$.

– Die Äquivalenzklassen der Äquivalenzrelation auf $\bar{m}^{V^{[2]}}$

$$f \sim f' :\Longleftrightarrow \exists \pi \in S_{\underline{p}} \ : \ \forall \{i, j\} \in V^{[2]} \ : \ f(\{i, j\}) = f'(\{\pi(i), \pi(j)\})$$

heißen Multigraphen mit p Knoten.
Die Äquivalenzklassen der Äquivalenzrelation auf $\{0, 1\}^{V^{[2]}}$

$$f \sim f' :\Longleftrightarrow \exists \pi \in S_{\underline{p}} \ : \ \forall \{i, j\} \in V^{[2]} \ : \ f(\{i, j\}) = f'(\{\pi(i), \pi(j)\})$$

heißen schlichte Graphen mit p Knoten.

1.2.7 Bemerkung:
Einem numerierten Graphen A mit p Knoten entspricht seine Adjazenzmatrix

$$(a_{i,j})_{1 \leq i,j \leq p} \text{ mit } a_{i,j} := \begin{cases} l, & \text{falls Knoten } i \text{ und } j \ \ l\text{-fach verbunden sind} \\ 0, & \text{sonst.} \end{cases}$$

Unter dem Grad des Knotens Nr.i verstehen wir

$$d_i(A) := \sum_{j=1}^{p} a_{i,j} = \sum_{j=1}^{p} a_{j,i} \ ,$$

also die Anzahl der von ihm ausgehenden Kanten. Die Gradpartition von A sei definiert als $\lambda(A) := (\lambda(A)_0, \lambda(A)_1, ...) \models p$, wobei $\lambda(A)_k :=$ Anzahl der Knoten vom Grad k.

1.2.8 Satz:
Eine Partition $d = (d_1, d_2, ..., d_p) \vdash 2k$ heißt graphisch, wenn es mindestens einen schlichten Graphen A gibt mit $d_i(A) = d_i$ für alle $i = 1, ..., p$. d ist genau dann graphisch, wenn gilt:

i) Für alle i mit $1 \leq i \leq p$ gilt: $d_i \leq p - 1$.

ii) Für alle r' mit $1 \leq r' \leq p - 1$ gilt: $\sum_{i=1}^{r'} d_i \leq r'(r' - 1) + \sum_{i=r'+1}^{p} min(r', d_i)$.

Beweis:
Beweis durch Induktion nach p. Beim Induktionsschritt nehmen wir einen Knoten K maximalen Grades gr und verbinden ihn mit Knoten $K_1, ..., K_{gr}$, wobei gilt: Für alle Knoten $K' \notin \{K, K_1, ..., K_{gr}\}$ gilt $grad(K') \leq max\{grad(K), grad(K_1), ..., grad(K_{gr})\}$. Die Idee ist klar, jedoch ist die genaue Durchführung ziemlich kompliziert und findet sich in [21]. ⌉

1.2.9 Molekulare Graphen. In dieser Arbeit betrachten wir keine dreidimensionalen Strukturen. Chemische Moleküle werden nur durch die Nachbarschaftsbeziehungen ihrer Atome zueinander modelliert.

1.2.10 Definition:

- Jedes Atom eines Moleküls trägt einen Namen (z.B. 'C', 'O', 'N', ...). Unter dem *Atomnamen* eines Atoms verstehen wir diesen Namen.
- Sei p ein Atom. Dann ist der *Atomtyp* von p definiert als

$$AT(p) := (AN(p), val(p), rad(p), chg(p)).$$

  Dabei gelten die folgenden Abkürzungen:
  - $AN(p)$ bezeichnet den Atomnamen von p.
  - $val(p)$ ist die *Valenz*, also die Anzahl der von p ausgehenden Bindungen.
  - $rad(p) \in \{WAHR, FALSCH\}$ gibt an, ob p eine Radikalstelle besitzt.
  - $chg(p) \in \{-3, -2, ..., +3\}$ ist die *Atomladung* von p.
- Wir unterscheiden
  - Wasserstoffatome (Atomtyp = ('H',1,FALSCH,0)) und
  - Heteroatome (alle Atome außer Wasserstoff).

1.2.11 Schreibweise:
Gegeben sei ein Atomtyp ϑ. Dann bezeichne

- $\vartheta.AN$ den zu ϑ gehörigen Atomnamen,
- $\vartheta.val$ die zu ϑ gehörige Valenz,
- $\vartheta.rad$ die boolesche Variable, die Auskunft über die Existenz einer Radikalstelle gibt,
- $\vartheta.chg$ die zu ϑ gehörige Ladung.

1.2.12 Definition:
Sei $AM := \{\vartheta_1, ..., \vartheta_n\}$ eine Menge von Atomen und sei $AM^{[2]} := \{\{i, j\} \subseteq AM\}$.

- Sei $f \in \bar{4}^{AM^{[2]}}$. Fassen wir die Atome aus AM als Knoten eines Graphen auf, so erhalten wir einen Multigraphen g. Wir nennen f *numerierten molekularen Graphen* (bzgl. AM).
- Die Äquivalenzklassen der Äquivalenzrelation

$$f \sim f' :\Longleftrightarrow \exists \pi \in S_{\underline{n}} : \begin{cases} \forall\, i \in \underline{n} \;:\; AT(\vartheta_i) = AT(\vartheta_{\pi(i)}) \text{ und} \\ \forall\, \{i, j\} \in n^{[2]} \;:\; f(\{\vartheta_i, \vartheta_j\}) = f'\left(\{\vartheta_{\pi(i)}, \vartheta_{\pi(j)}\}\right) \end{cases}$$

  auf $\bar{4}^{AM^{[2]}}$ heißen *molekulare Graphen* (bzgl. AM).
- Zu $g \in \bar{4}^{AM^{[2]}}$ bezeichne $\bar{g} \in S_n \backslash\backslash \bar{4}^{AM^{[2]}}$ die Äquivalenzklasse von g.
- Sei $f \in \bar{4}^{AM^{[2]}}$ molekularer Graph und $ANV := \{(an_1, v_1), ..., (an_k, v_k)\}$. Dabei gelte:
  - Für alle $i = 1, ..., k$ bezeichnet an_i einen Atomnamen.
  - Für alle $i = 1, ..., k$ gilt: $v_i \in \mathbb{N}$.
  - $\sum_{i=1}^{k} v_i = n$.
  - $an_i \neq an_j$ für $1 \leq i < j \leq k$.
  - Für alle $i = 1, ..., k$ existiert ein $T_i = \{\epsilon_{i,1}, ..., \epsilon_{i,v_i}\} \subseteq AM$ mit $AN(p) = an_i$ für alle $p \in T_i$.

Dann heißt ANV *Atomnamenverteilung* von f. Gelegentlich nennen wir die Atomnamenverteilung von f auch „Bruttoformel von f".

- Sei $f \in \bar{4}^{AM[2]}$ molekularer Graph und $ATV := \{(at_1, v_1), ..., (at_k, v_k)\}$. Dabei gelte:
  - Für alle $i = 1, ..., k$ bezeichnet at_i einen Atomtyp.
  - Für alle $i = 1, ..., k$ gilt: $v_i \in \mathbb{N}$.
  - $\sum_{i=1}^{k} v_i = n$.
  - $at_i \neq at_j$ für $1 \leq i < j \leq k$.
  - Für alle $i = 1, ..., k$ existiert ein $T_i = \{\epsilon_{i,1}, ..., \epsilon_{i,v_i}\} \subseteq AM$ mit $AT(p) = at_i$ für alle $p \in T_i$.

  Dann heißt ATV *Atomtypenverteilung* von f.
- Sei $f \in \bar{4}^{AM[2]}$ ein molekularer Graph. Per Definition besteht zwischen ϑ_i und ϑ_j genau dann eine Bindung, wenn $f(\{\vartheta_i, \vartheta_j\}) \neq 0$. Wir sprechen dann von einer Bindung $\{\vartheta_i, \vartheta_j\}_f$ der Vielfachheit $f(\{\vartheta_i, \vartheta_j\})$.
- Sei $f \in \bar{4}^{AM[2]}$ ein numerierter molekularer Graph. AM enthalte genau h Wasserstoffatome. Wir können annehmen, daß $\{\vartheta_1, ..., \vartheta_{n-h}\}$ Heteroatome sind. Unter der (reduzierten) *Adjazenzmatrix* von f verstehen wir die (symmetrische) $(n-h) \times (n-h)$-Matrix $Adj(f)$ mit

$$Adj(f)(i,j) := \begin{cases} 0, & \text{falls } i = j, \\ f(\{\vartheta_i, \vartheta_j\}), & \text{sonst.} \end{cases}$$

  Dabei sei Zeile (bzw. Spalte) i beschriftet mit $(AT(\vartheta_i), h_i)$, wobei h_i die Anzahl der zu ϑ_i benachbarten Wasserstoffatome bezeichnet.

Zur Konstruktion diskreter Strukturen müssen wir in vielen Fällen im Prinzip alle Möglichkeiten durchlaufen. Solche Suchen werden im allgemeinen mit Hilfe einer depth-first Prozedur, genannt *Backtracking* verwirklicht.

1.2.13 Allgemeine Backtrackprozedur. Unsere allgemeine Problemstellung läßt sich folgendermaßen formulieren:
Wir suchen alle Tupel $(x_1, x_2, ..., x_n)$, die einer Bedingung $P_n(x_1, x_2, ..., x_n)$ genügen. Beim Entwurf eines Backtrackalgorithmus müssen wir für alle $k = 0, ..., n-1$ Zwischenbedingungen $P_k(x_1, ..., x_k)$ definieren, so daß gilt:

$$P_{k+1}(x_1, ..., x_k, x_{k+1}) \text{ impliziert } P_k(x_1, ..., x_k).$$

Falls also $(x_1, ..., x_k)$ der Bedingung P_k *nicht* genügt, dann kann auch $(x_1, ..., x_k, x_{k+1})$ nicht P_{k+1} genügen. Folglich kann dann auch kein erweiterter Tupel $(x_1, ..., x_n)$ die Originalbedingung P_n erfüllen. Die allgemeine Backtrackprozedur berechnet systematisch alle Lösungen $(x_1, ..., x_n)$, die der Bedingung P_n genügen, indem alle Teillösungen $(x_1, ..., x_k)$, die P_k erfüllen, in Betracht gezogen werden:

1.2.14 Algorithmus: *Backtracking*

(1) $k := 0$.
(2) Berechnen der Nachfolger: ($P_k(x_1, ..., x_k)$ ist erfüllt.)
$S_k := \{x_{k+1} \mid P_{k+1}(x_1, ..., x_k, x_{k+1}) \text{ gilt }\}$.
(3) **Falls** $|S_k| = 0$, gehe zu (6).
(4) Wähle $x_{k+1} \in S_k$, $S_k := S_k \setminus x_{k+1}$, $k := k+1$.
(5) **Falls** $k < n$, gehe zu (2), **sonst** gebe $(x_1, ..., x_n)$ aus und gehe zu (6).
(6) $k := k-1$. **Falls** $k \geq 0$, gehe zu (3), **sonst** beende die Suche.

1.2.15 Aufwandsabschätzung. Da die Zeitkomplexität von Backtrackalgorithmen meistens nicht bekannt ist, ist es in einfachen Fällen praktikabel, die Laufzeit des Algorithmus mit Hilfe der sog. *Monte Carlo* Methode abzuschätzen ([28]). Diese Methode besteht darin, einen (oder auch mehrere) Äste des Suchbaums zu durchlaufen und die Laufzeit dann hochzurechnen. Voraussetzung ist allerdings, daß die Gestalt des Suchbaums hinreichend gut bekannt ist, so daß eine relativ genaue Vorhersage des Zeitaufwands möglich wird. Insbesondere bei der Konstruktion molekularer Graphen mit vielen zusätzlichen Nebenbedingungen kann über die Struktur des Suchbaums meistens keine Aussage gemacht werden, so daß eine sinnvolle Aufwandsabschätzung leider nicht möglich ist.

1.2.16 Definition:
Seien X, Y Mengen. Unter einer *Restriktion* $\mathcal{R}$ auf Y^X verstehen wir eine Abbildung

$$\mathcal{R} : Y^{(2^X)} \longrightarrow \{\text{WAHR}, \text{FALSCH}\}.$$

Das heißt, $\mathcal{R}(f)$=WAHR, falls $f \in Y^Z$ mit $Z \subseteq X$ die Restriktion $\mathcal{R}$ erfüllt.
Seien X, Y Mengen und $\mathcal{NB} \neq \emptyset$ eine Menge von Restriktionen auf Y^X. Gesucht seien alle diskreten Strukturen $f \in Y^X$, die den Restriktionen aus $\mathcal{NB}$ genügen.
Eine sehr einfache, aber meist auch sehr ineffektive Vorgehensweise besteht darin, zunächst alle $f \in Y^X$ zu konstruieren und dann im Nachhinein zu prüfen, ob die Restriktionen aus $\mathcal{NB}$ erfüllt sind.
Um einen Konstruktionsalgorithmus evtl. zu verbessern, müssen die Restriktionen aus $\mathcal{NB}$ näher betrachtet werden.

1.2.17 Definition:
Seien X, Y Mengen und $f \in Y^X$. Sei $Z \subset X$ und $g \in Y^Z$. g heißt *Anfangsstück* von f, falls $g(z) = f(z)$ für alle $z \in Z$. Umgekehrt heiß f dann *Fortsetzung* von g.

– Ist g Anfangsstück von f, so schreiben wir: $g \subset f$.
– Ist f Fortsetzung von g, so schreiben wir: $f \supset g$.

1.2.18 Definition:
Seien X, Y, Z Mengen mit $Z \subset X$ und $\mathcal{R}$ eine Restriktion auf Y^X. $\mathcal{R}$ heiße *konsistent*, falls für alle $f \in Y^X, g \in Y^Z$ mit $f \supset g$ gilt:

$$\mathcal{R}(g) = \text{FALSCH} \Longrightarrow \mathcal{R}(f) = \text{FALSCH}.$$

Andernfalls heiße $\mathcal{R}$ *inkonsistent*.
Die Bezeichnung „konsistent“ wurde von Colbourn, Read ([13]) eingeführt.

1.2.19 Beispiel:

i) Sei $p \in \mathbb{N}$ und $X := p^{[2]}, Y := \{0, 1\}$, $\theta \in \mathbb{N}$ mit $4 \leq \theta \leq p$ und $\mathcal{R}$ eine Restriktion auf Y^X. $\mathcal{R}$ sei definiert durch

$$\mathcal{R}(f) := \begin{cases} \text{WAHR, falls } f \text{ Taillenweite } \geq \theta \text{ hat.} \\ \text{FALSCH, sonst.} \end{cases}$$

Dann ist $\mathcal{R}$ konsistent.

ii) Sei $p \in \mathbb{N}$ und $X := \bar{p}^{[2]}, Y := \{0,1\}$ und $\mathcal{R}$ eine Restriktion auf Y^X. $\mathcal{R}$ sei definiert durch

$$\mathcal{R}(f) := \begin{cases} \text{WAHR, falls } f \text{ einen Kreis der Länge 5 enthält.} \\ \text{FALSCH, sonst.} \end{cases}$$

Dann ist $\mathcal{R}$ inkonsistent.

Häufig funktionieren Konstruktionsalgorithmen für diskrete Strukturen nach dem Schema, zu einem gefundenem Anfangsstück alle möglichen Fortsetzungen zu berechnen. Sind die Restriktionen in $\mathcal{NB}$ konsistent, so liegt der Gedanke nahe, schon **während** der Konstruktion der diskreten Strukturen in gewissen Abständen zu prüfen, ob die Restriktionen aus $\mathcal{NB}$ für das aktuelle Anfangsstück A alle erfüllt sind, da man andernfalls A verwerfen kann.

1.2.20 Bemerkung:
Möchte man eine Restriktion schon während der Generierung prüfen, so ist klar, daß man im Einzelfall abwägen muß, ob sich dieser Zusatzaufwand lohnt. Generell sollte man v.a. „schnelle" Tests während der Generierung verwenden, jedoch kann man das natürlich nicht verallgemeinern.
An dieser Stelle sei noch erwähnt, daß unter gewissen Voraussetzungen auch inkonsistente Restriktionen während der Generierung geprüft werden und u.U. zu erheblichen Geschwindigkeitssteigerungen führen können. Denn seien X, Y, Z wiederum Mengen mit $Z \subset X$ und $\mathcal{R}$ eine inkonsistente Restriktion auf Y^X. Dann folgt für $f \in Y^X$, $g \in Y^Z$ mit $f \supset g$ und $\mathcal{R}(g)$ =FALSCH nicht automatisch $\mathcal{R}(f)$ =FALSCH. Eventuell läßt sich aber mit geringem Zusatzaufwand erkennen, daß im vorliegenden Fall trotzdem $\mathcal{R}(f)$=FALSCH folgt. Man sollte sich also die Mühe machen, auch inkonsistente Restriktionen dahingehend zu untersuchen.

2 Simskette und Labeled Branching

Zur Darstellung von und zum Rechnen mit Permutationsgruppen kleinen Grades im Computer bedienen wir uns der Hilfsmittel Simskette und labeled branching. Wenn man mit Homomorphieprinzip und ordnungstreuer Erzeugung arbeiten möchte, so muß man auf diese Grundlagen zurückgreifen können. Zum besseren Verständnis der folgenden Kapitel ist es von Vorteil, wenn man mit den Begriffen Simskette und labeled branching umgehen kann. Aus diesem Grund seien die folgenden Algorithmen hier noch einmal vorgestellt.
Eine Implementation dieser Algorithmen wurde z.B. in [29] vorgenommen: vom Autor wurden die Algorithmen außerdem zwecks Verwendung im MOLGEN Generator in C++ implementiert.

2.1 Berechnen aus Erzeugern

Die Gruppe $G \leq S_n$ operiere in kanonischer Weise auf der Menge $\Omega := \{1, \ldots, n\}$, kurz $G\Omega$, und $\pi = \pi_1, \ldots, \pi_n$ sei eine Anordnung der Punkte von Ω. $G^{(i)} := C_G(\{\pi_1, \ldots, \pi_{i-1}\})$ für alle $i = 1, \ldots, n$. Unter der Simskette [46] von G bezüglich π versteht man die Untergruppenkette

$$G = G^{(1)} \geq G^{(2)} \geq \ldots \geq G^{(n-1)} \geq G^{(n)} = \{(id)\}.$$

$S \subseteq G$ heißt starkes Erzeugendensystem von G bzgl. π, falls

$$\forall\, i = 1, \ldots, n \;:\; G^{(i)} \;=\; \langle S \cap G^{(i)} \rangle.$$

- Ein branching auf Ω bzgl. π ist eine Menge von gerichteten Bäumen mit Kanten der Form (π_i, π_j), $i < j$.
- Ein branching B heißt labeled branching für G [24] bzgl. π, falls jede Kante (π_i, π_j) mit einer Permutation $\sigma_{i,j}$ markiert ist, und folgendes gilt:
  i) $\sigma_{i,j} \in G^{(i)}$ und $\pi_i^{\sigma_{i,j}} = \pi_j$.
  ii) Die Menge aller $\sigma_{i,j}$ erzeugt G. $\sigma_{i,j}$ heißt auch Kantenmarke von π_j in B.
  π_i heißt dann Vater von π_j.
- Ein labeled branching B heißt vollständig, wenn außerdem gilt:
  Falls $\pi_k \in \pi_i^{G^{(i)}}$, dann existiert ein gerichteter Pfad in B von π_i nach π_k.

Wir markieren zusätzlich jeden Knoten π_j mit einer Permutation τ_j, wobei τ_j das Produkt der Kantenmarken von π_r nach π_j ist. (π_r ist dabei die Wurzel der Zusammenhangskomponente, die π_j enthält). Ist π_j selbst Wurzel, so setzen wir $\tau_j := id$. τ_j heißt auch Eckenmarke von π_j. Weiterhin sei $parent(j) := k$, falls π_k der Vater von π_j ist. Ist π_j Wurzel, so sei $parent(j) := j$. Für die Speicherung einer beliebigen Gruppe $G < S_n$ als labeled branching benötigt man also $2n$ Permutationen der Länge n.

2.1.1 Bemerkung:
Sei B labeled branching für $G^{(i+1)}$ und $T^{(i+1)}$ sei eine Transversale von $G^{(i)} \,/\, G^{(i+1)}$ mit $1 \leq i < n$. Dann gilt:

$$t_1, t_2 \in T^{(i+1)} \Longrightarrow [t_1 \neq t_2 \iff \pi_i^{t_1} \neq \pi_i^{t_2}].$$

2.1.2 Prozedur: *SimsToLabra*
Sei B labeled branching für $G^{(i+1)}$ und $T^{(i+1)}$ sei eine Transversale von $G^{(i)} \,/\, G^{(i+1)}$ mit $1 \leq i < n$. Wir fügen in B evtl. zusätzliche Erzeuger ein, so daß B anschließend $G^{(i)}$ repräsentiert:

(1) **Für alle** $t \in T^{(i+1)}$ **tue**
(2) Wähle j so, daß $\pi_j = \pi_i^t$
(3) **Falls** $parent(j) = j$
(4) $parent(j) := i$.
(5) $\sigma_{i,j} := t$.
ende
ende
(6) Multipliziere von den Wurzeln abwärts und passe so sukzessive die Eckenmarken an.

Eine wichtige Aufgabe ist es, zu gegebenen Erzeugern $\{\varrho_1, \ldots, \varrho_\mu\}$ einer Gruppe $G < S_n$ eine Darstellung als Simskette bzw. labeled branching zu finden. Der im folgenden beschriebene Algorithmus hat Zeitkomplexität $O(n^5)$ und sollte deshalb wohlüberlegt verwendet werden. Die im Rahmen dieser Arbeit entstandenen Computerprogramme benutzen diesen Algorithmus deshalb nur, um Gruppen, die in Form von Erzeugern eingegeben werden, beim Programmstart in labeled branchings umzurechnen.
Nach 2.1.2 genügt es, eine Simskette

$$G = G^{(1)} \geq G^{(2)} \geq \ldots \geq G^{(n-1)} \geq G^{(n)} = \{(id)\}.$$

von G samt aller Transversalen U_i von $G^{(i)}$ in $G^{(i-1)}$ zu berechnen ($i = 2, \ldots, n$).

2.1.3 Prozedur: *Aufbau eines Branchings*

Für $i := 2$ **bis** n **tue**
(1) Verwende Erzeuger von $G^{(i-1)}$, um U_i zu berechnen (2.1.4).
(2) Berechne eine Menge M_i von Elementen aus $G^{(i)}$,
mit $\langle M_i \rangle = G^{(i)}$ und $|M_i| \leq n^2$ (2.1.5).
(3) Berechne aus M_i eine Menge $\tilde{M}_i$ mit $\langle M_i \rangle = \langle \tilde{M}_i \rangle$ und $|\tilde{M}_i| < n$ (2.1.6).
ende

2.1.4 Prozedur: *Berechnung von Nebenklassenrepräsentanten*
Es bezeichne Δ die Bahn von $G^{(i-1)}$, die i enthält. Eingabe sei eine Menge K von Erzeugern von $G^{(i-1)}$. Wir berechnen Δ zusammen mit einer Transversale U_i von $G^{(i)}$ in $G^{(i-1)}$.

(1) $\Delta := \{i\}$.
(2) $S := \{i\}$.
(3) $U_i[i] := id$.
(4) **Solange** $S \neq \{\}$, **tue**
(5) Wähle $j \in S$.
(6) Entferne j aus S.
(7) **Für** $\pi \in K$ **tue**
(8) $k := j^{\pi}$.
(9) **Falls** $k \notin \Delta$:
(10) $U_i[k] := U_i[j] \circ \pi$.
(11) $\Delta := \Delta \cup \{k\}$.
(12) $S := S \cup \{k\}$.
ende
ende
ende

Es wird einfach jeder Erzeuger auf jedes Element der Bahn angewendet. Die Nebenklassenrepräsentanten werden aufmultipliziert.

2.1.5 Berechnung des neuen Erzeugendensystems: *(Schreier)*
Sei K ein Erzeugendensystem von $G^{(i-1)}$, Δ die Bahn von $G^{(i-1)}$, die i enthält, und $\{\varrho_k : k \in \Delta\}$ eine Transversale von $G^{(i)}$ in $G^{(i-1)}$. Es gelte $i^{\varrho_k} = k$ für alle $k \in \Delta$ und $\varrho_i = id$. Dann ist

$$\tilde{K} := \{\varrho_j \pi \varrho_k^{-1} : \pi \in K,\ j, k \in \Delta,\ j^{\pi} = k\}$$

ein Erzeugendensystem von $G^{(i)}$. Die Elemente von $\tilde{K}$ heißen Schreiererzeuger.

Beweis:
Sei $\alpha = s_1 \cdot \ldots \cdot s_\epsilon$ ein Element aus $G^{(i)}$, mit $s_j \in K$ für alle j. Wir definieren $\tilde{s}_k := \prod_{l=1}^{k} s_l$ und $\omega_k := i^{\tilde{s}_k}$ für alle $k = 1, \ldots, \epsilon - 1$. Dann können wir schreiben:

$$\alpha = (\varrho_i s_1 \varrho_{\omega_1}^{-1})(\varrho_{\omega_1} s_2 \varrho_{\omega_2}^{-1}) \cdot \ldots \cdot (\varrho_{\omega_{\epsilon-1}} s_\epsilon \varrho_i^{-1}).$$

Also erhalten wir α als Produkt von Schreiererzeugern. $\square$

2.1.6 Prozedur: *Siften der Erzeuger*
Gegeben sei eine Menge M_i von Erzeugern, sowie ein leeres branching B. Wir wollen alle Erzeuger aus M_i nacheinander in B einfügen, so daß wir ein branching für $\langle M_i \rangle$ erhalten. Die Eckenmarken von B bilden dann die gesuchte Menge $\tilde{M}_i$.
Seien nun bereits $j-1$ Erzeuger in B eingefügt. Jetzt soll der j–te Erzeuger $\pi \in M_i$ „gesiftet" werden. Aus dem branching B entsteht dabei das modifizierte branching $\tilde{B}$. Dabei soll gelten: $\langle \tilde{B} \rangle = \langle B, \pi \rangle$. Die Einfügeprozedur $sift(\pi)$ funktioniert wie folgt:

(1) $j := max\{h \in \{1, ..., n\} : \pi \in G^{(h-1)}\}$; $k := j^{\pi}$;
(2) **Solange** in B ein Pfad von j nach k mit $j < n$ existiert, **tue**
(3) $\quad \pi := \pi \tau_k^{-1} \tau_j$.
(4) $\quad j := max\{h \in \{1, ..., n\} : \pi \in G^{(h-1)}\}$, $k := j^{\pi}$.
ende
(5) **Falls** $j = n$ return.
(6) **Solange** in B eine Kante (m, k) mit $m > j$ existiert, **tue**
(7) $\quad \pi := \pi \sigma_{m,k}^{-1}$, $k := m$.
ende
(8) **Falls** k noch nicht Endpunkt einer Kante (m, k) ist, dann
(9) $\quad$ Füge die Kante (j, k) mit der Kantenmarke π in B ein.
(10) $\quad$ Aktualisiere die Eckenmarken.
(11) **sonst**
(12) $\quad$ Füge die Kante (j, k) mit der Kantenmarke π in B ein.
(13) $\quad \pi := \sigma_{m,k} \pi^{-1}$.
(14) $\quad$ Entferne die Kante (m, k) aus B.
(15) $\quad$ Aktualisiere die Eckenmarken.
(16) $\quad sift(\pi)$.
ende

Nach Zeile (4) stabilisiert π einen Punkt mehr als zuvor, also können die Zeilen (2)-(4) höchstens $n-1$ mal durchlaufen werden und haben damit Aufwand $O(n^2)$. Ebenso haben die Zeilen (6)-(7) und Zeile (10) Aufwand $O(n^2)$. Bevor also evtl. $sift$ rekursiv aufgerufen wird, wurde Aufwand $O(n^2)$ benötigt.
In den Zeilen (12)-(15) gilt: $(k-j) < (k-m)$, also wurde die Summe der Kantenlängen von B verringert. Da die maximale Summe der Kantenlängen $n(n-1)/2$ beträgt, kann *$sift$ für alle Erzeuger aus M_i zusammen* höchstens $O(n^2)$ mal rekursiv gerufen werden. Aus $|M_i| = O(n^2)$ folgt, daß $sift$ insgesamt $O(n^2)$ mal gerufen wird. Somit benötigen wir für das Siften der Erzeuger Aufwand $O(n^4)$.

2.2 Basiswechsel

Zu einem labeled branching B für G zur Basis

$$\pi = \pi_1, ..., \pi_{r-1}, \pi_r, ..., \pi_s, \pi_{s+1}, ..., \pi_n$$

soll ein labeled branching B' für G zur Basis

$$\pi' = \pi_1, ..., \pi_{r-1}, \pi_s, \pi_r, \pi_{r+1}, ..., \pi_{s-1}, \pi_{s+1}, ..., \pi_n$$

berechnet werden, d.h. wir erhalten das labeled branching B' bzgl. π' aus B durch einen sogenannten *cyclic shift* von r bis s. Der Algorithmus, der dies leistet besteht aus den drei Teilen Cycle-node-bottom, Cycle-node-middle und Cycle-node-top (siehe [12]). Im folgenden sei für alle $i = 1, ..., n$

$$root[i] := \begin{cases} 1, & \text{falls } \pi'_i \text{ Wurzel in } B', \\ 0, & \text{sonst.} \end{cases}$$

2.2.1 Prozedur: *Cycle-node-bottom*

(1) Für alle $i := 1, ..., n$ sei $root[i] := 1$, $parent'(i) := i$ und $merke\_anz := 0$.
(2) **Für** $j := s + 1$ **bis** n **tue**
(3) $i := parent(j)$.
(4) **Falls** $s < i$ und $i \neq j$:
(5) $parent'(j) := i$.
(6) $root[j] := 0$.
(7) $merke\_anz := merke\_anz + 1$.
(8) $bottom\_merke[merke\_anz] := j$.
ende
ende

2.2.2 Algorithmus: *Cycle-node-middle*
$\triangle^{(i)} := {\pi_i}^{G^{(i)}}, G^{(i)} := C_G(\pi'_1, ..., \pi'_{i-1}), \triangle^{(i)} := {\pi'_i}^{G^{(i)}}$.
Wir gehen folgendermaßen vor:
Für $j := s$ downto r fülle die Kantenmarken von B' auf, die von π'_j ausgehen. Es gilt:

$$\text{Für alle } j \text{ mit } r < j \leq s \ : \ \triangle^{(j)} = {\pi'_j}^{G^{(j)}} = {\pi_{j-1}}^{C_G(\pi_1, ..., \pi_{j-2}, \pi_s)}$$

und

$$\triangle^{(r)} = {\pi_s}^{G^{(r)}}.$$

Also

$$\text{Für alle } j \text{ mit } r < j \leq s \ : \ \triangle^{(j)} \subseteq \triangle^{(j-1)}.$$

2.2.3 Lemma:
Sei $r < j \leq s, \pi_m = {\pi_{j-1}}^{\alpha} \in \triangle^{(j-1)}$, mit $\alpha \in G^{(j-1)}$. Dann gilt:

$$\pi_m \in \triangle^{(j)} \iff {\pi_s}^{\alpha^{-1}} \in \pi_s^{G^{(j)}}$$

Beweis:

" $\Longrightarrow$ ": Wegen $\pi_m \in \triangle^{(j)}$ ist $\pi_m = {\pi_{j-1}}^{\beta}$ mit einem $\beta \in G^{(j)} = C_G(\pi_1, ..., \pi_{j-2}, \pi_s)$. Aus ${\pi_{j-1}}^{\beta\alpha^{-1}} = {\pi_m}^{\alpha^{-1}} = \pi_{j-1}$ folgt $\beta\alpha^{-1} \in G^{(j)}$. Also ist ${\pi_s}^{\alpha^{-1}} = {\pi_s}^{\beta\alpha^{-1}} \in {\pi_s}^{G^{(j)}}$.
" $\Longleftarrow$ ": Wegen ${\pi_s}^{\alpha^{-1}} \in {\pi_s}^{G^{(j)}}$ ist ${\pi_s}^{\alpha^{-1}} = {\pi_s}^{\beta}$ mit einem $\beta \in G^{(j)}$. Daraus folgt $\beta\alpha \in G^{(j)}$ und damit $\pi_m = {\pi_{j-1}}^{\alpha} = {\pi_{j-1}}^{\beta\alpha} \in \triangle^{(j)}$.

□

2.2.4 Prozedur: *Cycle-node-middle*
Für $j = r, ..., s$ repräsentieren *orbit* und *cosetrep* ${\pi_s}^{G^{(j)}}$ mit $orbit[i] = 1 :\Longleftrightarrow \pi_i \in {\pi_s}^{G^{(j)}}$ und $cosetrep[i] \in G^{(j)} : {\pi_s}^{cosetrep[i]} = \pi_i$, falls $orbit[i] = 1$. Weiterhin sei $orbitlist := \pi_s^{G^{(s)}}$.

(1) Initialisiere *orbit*, *cosetrep* und *orbitlist* für ${\pi_s}^{G^{(s)}}$ mit Hilfe von B.
(2) **Für** $j := s$ **downto** $r+1$ **tue**
(3) $\triangle := \triangle^{(j-1)} = \pi_{j-1}^{G^{(j-1)}}$. // verwende dazu B
(4) **Für alle** $\pi_m \in \triangle$ **tue**
(5) $\pi_p := \pi_s^{\tau_m^{-1} \cdot \tau_{j-1}}$.
// $\tau_{j-1}^{-1} \cdot \tau_m$ = Pfadprodukt von π_{j-1} nach π_m
(6) **Falls** $orbit[p] = 1$:
(7) ${\pi'}_q := \pi_m$.
(8) **Falls** $q \neq j$ und $root[q] = 1$,
verbinde ${\pi'}_j$ mit ${\pi'}_q$ in B':
(9) ${\sigma'}_{j,q} := cosetrep[p] \cdot \tau_{j-1}^{-1} \cdot \tau_m$.
(10) $root[q] := 0$ und $parent'(q) := j$.
ende
ende
ende
(11) update-orbit(B, π, *orbit*, *cosetrep*, s, j-1).
// erweitere *orbit* und *cosetrep* für $\pi_s^{G^{(j-1)}}$
ende
(12) $\triangle := \pi_s^{G^{(r)}}$.
// *orbit* und *cosetrep* repräsentieren nun $\pi_s^{G^{(r)}}$
(13) **Für alle** $\pi_m \in \triangle$ **tue**
(14) ${\pi'}_q := \pi_m$.
(15) **Falls** $root[q] = 1$ und $q \neq r$,
verbinde ${\pi'}_r$ mit ${\pi'}_q$:
(16) ${\sigma'}_{r,q} := cosetrep[m]$ // nur Zeiger tauschen
Beachte: ${\pi'}_r^{cosetrep[m]} = {\pi_s}^{cosetrep[m]} = \pi_m = {\pi'}_q$
(17) $root[q] := 0$ und $parent'(q) := r$.
ende
ende
(18) **Für** $j := 1$ **bis** *merke_anz* **tue**
(19) $k := bottom\_merke[j]$ und $l := parent'(k)$.
(20) ${\sigma'}_{l,k} := \sigma_{l,k}$.
// nur Zeiger tauschen
ende

Die Zeilen (18)-(20) können nicht in Cycle-node-bottom abgearbeitet werden, weil in der Funktion update-orbit die Kantenmarken von B noch belegt sein müssen.

2.2.5 Prozedur: *update-orbit*
Eingabe : B, π, *orbit*, *cosetrep*, s, *ind*.

Aufgabe : Erweitere *orbit* und *cosetrep* von $\pi_s^{G^{(ind+1)}}$ zu $\pi_s^{G^{(ind)}}$
$newgens := \{\sigma_{i,j} \in B \mid i = ind\}$, $genlist := \{\sigma_{i,j} \in B \mid i \geq ind\}$.

```
(1)   newpoints := {}.
(2)   Für alle g ∈ newgens tue
(3)         Für alle π_p ∈ orbitlist tue
(4)               π_q := π_p^g.
(5)               Falls orbit[q] ≠ 1:
(6)                     orbit[q] := 1.
(7)                     cosetrep[q] := cosetrep[p] · g.
(8)                     newpoints := newpoints ∪ π_q.
                  ende
            ende
      ende
(9)   Solange newpoints ≠ {} tue
(10)        orbitlist := orbitlist ∪ newpoints.
(11)        pointlist := newpoints.
(12)        newpoints := {}.
(13)        Für alle g ∈ genlist tue
(14)              Für alle π_p ∈ pointlist tue
(15)                    π_q := π_p^g.
(16)                    Falls orbit[q] ≠ 1:
(17)                          orbit[q] := 1.
(18)                          cosetrep[q] := cosetrep[p] · g.
(19)                          newpoints := newpoints ∪ π_q.
                        ende
                  ende
            ende
      ende
```

2.2.6 Bemerkung:
In den Zeilen (1)-(8) wird jeder neue Erzeuger auf jedes alte Element der Bahn angewandt. Neue Elemente der Bahn merkt man sich in *newpoints*. In den Zeilen (9)-(19) werden dann alle Erzeuger von $G^{(ind)}$ auf jeden neuen Punkt angewandt, wobei weitere neue Bahnelemente in *newpoints* aufgenommen werden. Insgesamt entspricht also der Aufwand für alle Funktionsaufrufe von *update-orbit* zusammen der Berechnung von $\pi_s^{G^{(r)}}$.

2.2.7 Algorithmus: *Cycle-node-top*
Nun sind alle $\sigma'_{i,j}$ mit $i \geq r$ in B' eingefügt. Es fehlen noch:

i) $\{\ \sigma'_{i,j} \mid i < r$ und $j < r\}$, sowie $\{\ \tau'_i \mid i < r\ \}$
ii) $\{\ \sigma'_{i,j} \mid i < r$ und $j \geq r\}$, sowie $\{\ \tau'_i \mid i \geq r\ \}$

2.2.8 Prozedur: *Cycle-node-top*

(1) **Für** $i := r$ **bis** n **tue**
(2) **Falls** $root[i] = 1$
(3) Folge $\pi'_i = \pi_m$ zurück in B ,d.h.
$m \longrightarrow parent(m) \longrightarrow parent(parent(m))$ ...
bis man zum ersten π_j mit $j < r$ kommt // falls möglich
(4) **Falls** π_j existiert
(5) $parent'(i) := j$.
(6) $\tau'_i := \tau_m$, weil $\tau'_i = \tau_j \cdot (\tau_j^{-1} \cdot \tau_m)$.
// nur Zeiger tauschen
(7) **Falls** $parent(m) = j$
(8) $\sigma'_{j,i} := \sigma_{j,m}$. // nur Zeiger tauschen
ende
(9) **sonst** $\sigma'_{j,i} := \tau_j^{-1} \cdot \tau'_i$.
ende
ende
(10) **sonst** $\tau'_i := \tau'_{parent'(i)} \cdot \sigma'_{parent'(i),i}$.
ende // ii) wurde abgearbeitet
(11) **Für** $i := 1$ **bis** $r - 1$ **tue**
(12) $parent'(i) := parent(i)$.
(13) $\tau'_i := \tau_i$. // nur Zeiger tauschen
(14) $\sigma'_{parent'(i),i} := \sigma_{parent(i),i}$. // nur Zeiger tauschen
ende // i) wurde abgearbeitet

2.2.9 Bemerkung:
Sei B ein labeled branching vom Grad n und $1 \leq i \leq n$. Dann gilt:

$$\{parent(i) = i\} \Longleftrightarrow \{\sigma_{parent(i),i} = (id) \ \wedge \ \tau_i = (id)\}$$

Falls also $parent(i) = i$, so bleibt die zugehörige Eckenmarke und Kantenmarke beim Basiswechsel uninitialisiert.

3 Kanonische Numerierung

Im folgenden beschreiben wir kurz die Methode der iterierten Klassifizierung, um diskrete Strukturen in kanonischer Art und Weise zu numerieren. Diese Vorgehensweise geht zurück auf Brendan Mc.Kay ([33]). Eine ausführlichere Beschreibung der iterierten Klassifizierung findet sich z.B. in [3], [41]. Eine allgemeine, von der speziellen Struktur unabhängige Implementation zur Isomorphieerkennung wurde in [44] vorgenommen. Bei der Konstruktion diskreter Strukturen stellt sich üblicherweise das folgende Problem:
Es seien i paarweise nicht isomorphe Objekte $o_1, ..., o_i$ berechnet, sowie eine weitere Struktur S. Nun muß festgestellt werden, ob S zu einer Struktur o_j isomorph ist.
Eine Vorgehensweise wäre z.B., Invarianten $\{i_1, ..., i_k\}$ für S zu berechnen, sodann alle Objekte $\{\varrho_1, ..., \varrho_u\} \subseteq \{o_1, ..., o_i\}$ zu bestimmen, die den Invarianten $\{i_1, ..., i_k\}$ genügen, und S dann paarweise auf Isomorphie mit jedem Element aus $\{\varrho_1, ..., \varrho_u\}$ zu testen.
Wenn i zu groß wird, ist allerdings damit zu rechnen, daß auch u recht groß wird. Da u (zeitaufwendige) Isomorphietests durchgeführt werden müssen, ist diese Methode also für

viele Probleme nicht akzeptabel.
Verfügt man über einen Algorithmus zur kanonischen Numerierung der betrachteten Strukturen, so speichern wir die Objekte $o_1, ..., o_i$ jeweils zusammen mit ihrer kanonischen Numerierung und berechnen die kanonische Numerierung von S (nur **eine** Berechnung). Mit Hilfe einer Hashtabelle finden wir sodann alle in Frage kommenden Objekte $\{\sigma_1, ..., \sigma_v\} \subseteq \{o_1, ..., o_i\}$, die evtl. isomorph zu S sein könnten (wir dürfen annehmen, daß die Zahl der Kollisionen, also auch v sehr klein ist). Nun ist nur noch ein direkter Vergleich (kein Isomorphietest!) von S mit jedem σ_i nötig. Da in der Praxis ein Isomorphietest etwa so aufwendig ist wie die Berechnung einer kanonischen Numerierung, ist die kanonische Numerierung in der Regel also der Verwendung von Invarianten vorzuziehen.
Da wir uns in dieser Arbeit in erster Linie mit der Konstruktion molekularer Graphen beschäftigen werden, sei es erlaubt, den Algorithmus der iterierten Klassifizierung angewandt auf den Spezialfall der kanonischen Numerierung von molekularen Graphen darzustellen.

3.0.10 Definition:

– Sei $AM := \{\vartheta_1, ..., \vartheta_n\}$ eine Menge von Atomen.

$$f : \bar{4}^{AM^{[2]}} \longrightarrow \bar{4}^{AM^{[2]}}$$

heißt *kanonisierende Abbildung* auf $\bar{4}^{AM^{[2]}}$, falls gilt:
- Für alle $g_1, g_2 \in \bar{4}^{AM^{[2]}}$ mit $g_1 \sim g_2 : f(g_1) = f(g_2)$.
- Für alle $g \in \bar{4}^{AM^{[2]}} \ : \ f(g) \sim g$.

– Sei $f : \bar{4}^{AM^{[2]}} \longrightarrow \bar{4}^{AM^{[2]}}$ eine *kanonisierende Abbildung* und $g \in \bar{4}^{AM^{[2]}}$. $f(g)$ heißt dann *kanonische Numerierung* von $\bar{g}$.

3.0.11 Bemerkung:
Natürlich gibt es zu einer festen Menge $\bar{4}^{AM^{[2]}}$ viele verschiedene kanonisierende Abbildungen. Naheliegend wäre es z.B., das lexikographisch kleinste Element aus jeder Äquivalenzklasse zu wählen. Für die Anwendung ist es allerdings vollkommen egal, welche kanonisierende Abbildung man verwendet, wichtig ist für uns nur, daß die verwendete Abbildung sich schnell berechnen läßt.

3.1 Algorithmus

Gegeben sei ein numerierter molekularer Graph $g \in \bar{4}^{AM^{[2]}}$. Wir suchen eine kanonische Numerierung von $\bar{g}$. Der Algorithmus der *iterierten Klassifizierung* funktioniert folgendermaßen:

3.1.1 Prozedur: *CanNum*

(1) *transv_nr* $:= \infty$.
(2) Berechne die Startklassifizierung.
(3) IteriereKlassen(1).
(4) **Falls** alle Klassen einelementig sind, belege den Lösungsvektor.
(5) **sonst** CanRek(1).
(6) Berechne die Automorphismengruppe.

3.1.2 Berechnung der Startklassifizierung:

Wir können voraussetzen, daß auf der Menge der Atomtypen von g eine Ordnung „<" existiert. Wir fassen AM in Blöcke zusammen, so daß zwei Atome von g genau dann den gleichen Atomtyp besitzen, wenn sie im gleichen Block liegen. Diese Blöcke nennen wir auch Klassen (der Stufe 1). Aufgrund der Ordnung „<" auf den Atomtypen können wir die Klassen der Stufe 1 in kanonischer Art und Weise anordnen. Die einelementigen Klassen der Stufe 1 behandeln wir gesondert.
Wir haben nun also einen geordneten Tupel der Klassen $(K_{1,1}, ..., K_{1,\alpha_1})$ mit $|K_{1,j}| > 1$ und $K_{1,j} \subseteq AM$, sowie den Tupel einelementiger Klassen $e_1 = (e_{1,1}, ..., e_{1,\beta_1})$ berechnet.

3.1.3 Verfeinerung der Klasseneinteilung: *IteriereKlassen(i)*

Gegeben sei ein Tupel von Klassen $(K_{i,1}, ..., K_{i,\alpha_i})$ der Stufe i mit $i \geq 1$, sowie ein Tupel $e_1 = (e_{i,1}, ..., e_{i,\beta_i})$ der einelementigen Klassen der Stufe i. Zwei Elemente $a_1, a_2 \in K_{i,j}$ $(1 \leq j \leq \alpha_i)$ können sich bzgl. ihres Nachbarschaftsverhältnisses zu $e_{i,l}$ $(1 \leq l \leq \beta_i)$ unterscheiden, d.h. a_1 kann mit $e_{i,l}$ entweder überhaupt nicht, oder mit einer 1, 2, 3-fachen Kante verbunden sein. Dies gibt uns die Möglichkeit, $K_{i,j}$ evtl. aufzuspalten. Wir können für alle $K_{i,j}$ und für alle $e_{i,l}$ so vorgehen. Wenn wir sowohl die einelementigen Klassen als auch die mehrelementigen Klassen in aufsteigender Reihenfolge durchlaufen, und evtl. neu entstehende Klassen wiederum kanonisch anordnen, erreichen wir eine Verfeinerung der Klassenaufteilung, die kanonisch ist.

3.1.4 Berechnen einer kanonischen Numerierung durch Backtracking:

Da nach der Verfeinerung der Klassen der Stufe 1 in der Regel noch mehrelementige Klassen existieren, haben wir in diesen Fällen noch keine eindeutige Numerierung der Atome gefunden. Wir helfen uns durch Backtracking:

3.1.5 Prozedur: *CanRek(i)*

(1) Wähle aus den mehrelementigen Klassen der Tiefe i auf kanonische Art und Weise eine Klasse K_i aus.
(2) **Für** jedes Element $\xi_i \in K_i$ **tue**
(3) **Falls** ξ_i nicht das erste gewählte Element aus K_i ist, **tue**
(4) $transv\_nr := min(transv\_nr, i)$.
(5) Initialisiere die Klassen der Stufe $i+1$ mit den Klassen der Stufe i.
(6) Entferne ξ_i aus K_{i+1} und hänge ξ_i an die Liste der einelementigen Klassen der Stufe $i+1$ an.
(7) IteriereKlassen$(i+1)$.
(8) **Falls** alle Klassen der Stufe $i+1$ einelementig sind
(9) **Falls** wir einen Automorphismus ρ gefunden haben
(10) Füge ρ in die entsprechende Transversale der Automorphismengruppe ein.
(11) Falls $i > transv\_nr$, return(WAHR).
ende
(12) **Falls** wir eine lexikographisch größere Lösung gefunden haben
(13) Überschreibe die bisherige Lösung mit der neuen Numerierung.
(14) Lösche alle bisher notierten Automorphismen.
(15) $transv\_nr := \infty$.
ende
(16) **sonst**

```
(17)            Falls noch keine vorläufige Lösung gefunden ist
(18)                Rufe CanRek(i + 1).
(19)            sonst
(20)                Falls die aktuelle Anfangsnumerierung nicht kleiner als
                        die bisherige Lösung ist
(21)                    Falls CanRek(i + 1)=WAHR und i > transv_nr
(22)                        return(WAHR).
                    ende
                ende
            ende
        ende
return(FALSCH).
```

3.1.6 Bemerkungen:

- Bei der Auswahl der Klasse am Beginn von *CanRek* möchte man erreichen, daß der Backtrackingbaum nicht zu tief wird. Eine plausible Strategie ist z.B., immer die erste Klasse mit minimal vielen Elementen zu nehmen.
- Es ist klar, daß die Blätter des Baumes Numerierungen von $\tilde{g}$ repräsentieren. Weiterhin ist klar, daß die Menge der Blätter des Baumes, also die Menge der in Frage kommenden Numerierungen nicht von g, sondern nur von $\tilde{g}$ abhängig ist. Somit ist es sinnvoll, als kanonische Numerierung von $\tilde{g}$, die lexikographisch größte dieser möglichen Numerierungen zu wählen. Wir vergleichen Numerierungen indem wir die zugehörigen Adjazenzmatrizen des molekularen Graphen vergleichen. Beim Vergleich der Adjazenzmatrizen können wir uns auf die rechte obere Dreiecksmatrix beschränken. Die Plätze der Adjazenzmatrix $A = (a_{i,j})$ seien dabei wie folgt numeriert: $(a_{1,2}, a_{1,3}, a_{2,3}, a_{1,4}, a_{2,4}, a_{3,4}, a_{1,5}, ...)$. Diese Anordnung der Plätze der Adjazenzmatrix ermöglicht es, auch an den inneren Knoten des Baumes die aktuelle Anfangsnumerierung mit der bisher besten Numerierung zu vergleichen.
- Zusätzlich berechnen wir eine Transversalenkette der Automorphismengruppe von $\tilde{g}$. Dies tun wir nicht nur, weil wir uns für die Automorphismengruppe interessieren, sondern auch weil dadurch die Berechnung der kanonischen Numerierung evtl. stark beschleunigt wird.

3.1.7 Berechnung der Automorphismengruppe:

Da während des Backtrackings eine Transversalenkette der Automorphismengruppe berechnet wurde, ist es kein Problem, ein labeled branching der Automorphismengruppe zu erstellen.

3.2 Aufwand

Die Frage, ob ein polynomialer Entscheidungsalgorithmus zur Lösung des Graphenisomorphieproblems, d.h. ein Algorithmus, welcher die Isomorphie von n-punktigen Graphen in einer durch ein Polynom $p(n)$ beschränkten Anzahl von Schritten entscheidet, ist noch offen. Bekanntlich kann das allgemeine Isomorphieproblem auf das Graphenisomorphie-Problem zurückgeführt werden ([39]), d.h. falls die Isomorphie von Graphen in polynomialer Zeit entschieden werden kann, so kann es auch die Isomorphie von beliebigen endlichen

Strukturen.
Ein interessantes Ergebnis stammt von Babai, Erdös, Selkow ([1]), die zeigten, daß ein Algorithmus existiert, der für fast alle Graphen X (also alle bis auf $o\left(2^{\binom{n}{2}}\right)$ der $2^{\binom{n}{2}}$ n-punktigen Graphen) die Isomorphie aller Graphen Y zu X in der Zeit $O(n^2)$ entscheidet, wobei n die Knotenzahl von X bezeichnet.
Der eigentliche Durchbruch im Graphenisomorphie-Problem wurde von Eugene Luks ([32]) erzielt. Sein Isomorphietest beruht auf algebraischen Methoden und entscheidet die Isomorphie von d-valenten Graphen auf n Punkten in der Zeit $O(n^{t(d)})$ für eine geeignete Funktion $t : \mathbb{N} \to \mathbb{N}$. Das für uns wichtigste Ergebnis ist ein polynomialer Normalisierungs-Algorithmus für Graphen beschränkter Valenz, der von Walter Schnyder entwickelt wurde ([45]), da wir uns in dieser Arbeit v.a. mit der Konstruktion molekularer Graphen beschäftigen, die natürlich von beschränkter Valenz sind.

II. Konstruktionsstrategien

4 Homomorphieprinzip

Im letzten Kapitel haben wir eine Möglichkeit kennengelernt, bei der Konstruktion diskreter Strukturen das Isomorphieproblem zu lösen. Diese Methode setzt allerdings voraus, daß alle bisher gefundenen Lösungen abgespeichert sind, was bei bestimmten Anwendungen von Nachteil sein kann. Da die kanonische Numerierung i.a. erst nach der Konstruktion der numerierten Strukturen ins Spiel kommt, hat sie zwar den Vorteil, daß sie völlig unabhängig von der Konstruktionsmethode ist, andererseits ist es aber auch nicht möglich, die Konstruktionsmethode so zu wählen, daß das Isomorphieproblem von vornherein vereinfacht wird. Bei Konstruktionsproblemen, bei denen die Lösung des Isomorphieproblems der entscheidende Flaschenhals ist, sollte man daher versuchen, durch die Wahl geeigneter Homomorphismen (siehe auch [30]) die Konstruktion in möglichst viele kleine Einzelschritte zu zerlegen, die es erlauben, auch das Isomorphieproblem schrittweise zu lösen. Da das Isomorphieproblem i.a. exponentiellen Zeitaufwand besitzt, ist es (bei sinnvollem Einsatz des Homomorphieprinzips) günstiger, viele kleine Probleme anstatt eines einzigen großen zu lösen.

4.0.1 Satz:
Seien G und F Gruppen. G operiere auf einer Menge Ω und F operiere auf einer Menge Δ. $\Phi_\Omega : \Omega \longrightarrow \Delta$ sei surjektiv und $\Phi_G : G \longrightarrow F$ sei ein Epimorphismus. Weiter sei Φ_G ein Gruppenhomomorphismus und es gelte:

$$\forall g \in G \quad \forall \omega \in \Omega \ : \ \Phi_\Omega(\omega^g) = \Phi_\Omega(\omega)^{\Phi_G(g)}.$$

Dann folgt:

i) $\forall \delta \in \Delta, \forall f \in F : \Phi_\Omega^{-1}(\delta)$ und $\Phi_\Omega^{-1}(\delta^f)$ schneiden die gleichen Bahnen von G auf Ω.
ii) Falls $\omega_1, \omega_2 \in \Phi_\Omega^{-1}(\delta)$ und ${\omega_1}^g = \omega_2$ mit $g \in G$, so folgt $g \in \Phi_G^{-1}(C_F(\delta))$.
iii) Aus $\omega \in \Phi_\Omega^{-1}(\delta)$ folgt $C_G(\omega) \leq \Phi_G^{-1}(C_F(\delta))$.
iv) Sind Φ_G und Φ_Ω bijektiv, so heißt $\Phi := (\Phi_G, \Phi_\Omega)$ Isomorphismus von Gruppenoperationen und es gilt:
   - Φ_Ω induziert eine Bijektion zwischen der Menge der Transversalen von G auf Ω und der Menge der Transversalen von F auf Δ:

$$\mathcal{T}(G\backslash\backslash\Omega) \longrightarrow \mathcal{T}(F\backslash\backslash\Delta)$$

$$(\omega_1, \ldots, \omega_n) \longmapsto (\Phi_\Omega(\omega_1), \ldots, \Phi_\Omega(\omega_n))$$

   - Für alle $\omega \in \Omega$ gilt: $C_F(\Phi_\Omega(\omega)) = \Phi_G(C_G(\omega))$.

Beweis:

i) Sei $f \in F$, $\delta \in \Delta$. Dann gilt:
$\omega \in \Phi_\Omega^{-1}(\delta) \iff \Phi_\Omega(\omega) = \delta \iff \Phi_\Omega(\omega)^f = \delta^f \iff \Phi_\Omega(\omega^g) = \delta^f$ mit $g \in \Phi_G^{-1}(f)$
$\iff \omega^g \in \Phi_\Omega^{-1}(\delta^f)$.

ii) Seien $\omega_1, \omega_2 \in \Phi_\Omega^{-1}(\delta)$ und $g \in G$. Dann gilt:
${\omega_1}^g = \omega_2 \Longrightarrow \Phi_\Omega({\omega_1}^g) = \Phi_\Omega(\omega_2) \Longrightarrow \Phi_\Omega(\omega_1)^{\Phi_G(g)} = \Phi_\Omega(\omega_2) \Longrightarrow \delta^{\Phi_G(g)} = \delta \Longrightarrow$
$\Phi_G(g) \in C_F(\delta) \Longrightarrow g \in \Phi_G^{-1}(C_F(\delta))$.

iii) Sei $\omega \in \Phi_\Omega^{-1}(\delta), g \in C_G(\omega) \Longrightarrow \omega^g = \omega \Longrightarrow \Phi_\Omega(\omega^g) = \Phi_\Omega(\omega) \Longrightarrow \Phi_\Omega(\omega)^{\Phi_G(g)} =$
$\Phi_\Omega(\omega) \Longrightarrow \delta^{\Phi_G(g)} = \delta \Longrightarrow g \in \Phi_G^{-1}(C_F(\delta))$.

iv) – Seien $\delta_1, \delta_2 \in \Delta$ mit ${\delta_1}^f = \delta_2$ für ein $f \in F$. Nach Voraussetzung existieren $\omega_1, \omega_2 \in \Omega$ mit $\Phi_\Omega(\omega_i) = \delta_i (i = 1, 2)$, sowie $g \in G$ mit $\Phi_G(g) = f$. Es folgt:
$\Phi_\Omega({\omega_1})^{\Phi_G(g)} = \Phi_\Omega(\omega_2) \Longrightarrow \Phi_\Omega({\omega_1}^g) = \Phi_\Omega(\omega_2) \Longrightarrow {\omega_1}^g = \omega_2$.
Zusammen mit i) folgt die Behauptung.
– Aus iii) folgt $\Phi_G(C_G(\omega)) \leq C_F(\Phi_\Omega(\omega))$.
Es bleibt also $C_F(\Phi_\Omega(\omega)) \leq \Phi_G(C_G(\omega))$ zu zeigen: $f \in C_F(\Phi_\Omega(\omega)) \Longrightarrow \Phi_\Omega(\omega)^f = \Phi_\Omega(\omega) \Longrightarrow \Phi_\Omega(\omega^g) = \Phi_\Omega(\omega)$ mit $g \in \Phi_G^{-1}(f) \Longrightarrow \omega^g = \omega \Longrightarrow g \in C_G(\omega) \Longrightarrow f \in \Phi_G(C_G(\omega))$.

□

Will man Bahnenrepräsentanten und zugehörige Stabilisatoren einer Gruppenoperation berechnen, so möchte man gerne das Problem mittels einer homomorphen Abbildung auf eine einfacher zu lösende Gruppenoperation abbilden, berechnet dort Repräsentanten samt Stabilisator, und muß dann nur noch mit dem Urbild des gefundenen Stabilisators auf der Urbildmenge des Repräsentanten operieren. Hat man eine Folge solcher Homomorphismen, so wird im allgemeinen die operierende Gruppe immer kleiner, weil man jeweils nur mit dem Stabilisator weiteroperieren muß. Das Rückgängigmachen einer solchen Vereinfachung nennen wir dann „Aufspalten". Dieses Vorgehen ist unter Umständen sehr effektiv und wurde vom Autor auch schon erfolgreich zur Konstruktion schlichter Graphen angewandt. Leider ist es aber nicht immer möglich, eine solche Folge von Vereinfachunsschritten zu finden. Vielmehr gibt es manchmal eben nur einen einzigen großen Vereinfachunsschritt, wodurch dann wenig gewonnen wäre. In solchen Fällen kann man versuchen, zwischen den Vereinfachungsschritten auch Vergröberungen zu erlauben. In der Rücktransformation entsprechen Vergröberungen dem „Verschmelzen" von Lösungen.

4.1 Aufspalten

Seien G und F Gruppen. G operiere auf einer Menge Ω und F operiere auf einer Menge Δ. Sei $\Phi : \Omega \longrightarrow \Delta$ eine Abbildung. Weiter sei $\Phi_G : G \longrightarrow F$ ein Gruppenepimorphismus und es gelte:

$$\forall g \in G \quad \forall \omega \in \Omega \ : \ \Phi_\Omega(\omega^g) = \Phi_\Omega(\omega)^{\Phi_G(g)}.$$

$\mathcal{A}$ sei ein Algorithmus zur Berechnung von $\Phi^{-1}(\delta)$ für alle $\delta \in \Delta$.
Sei $\mathcal{B}$ ein Algorithmus, der zu $M \subset \Omega$ und $U \leq G$ mit $M^U = M$ folgendes berechnet:

– eine Transversale $\Gamma_M \in \mathcal{T}(U \backslash\backslash M)$.
– eine Abb. $f_M : M \longrightarrow U$ mit $m^{f_M(m)} \in \Gamma_M$ für alle $m \in M$.
– den Stabilisator $N_U(\delta)$ für alle $\delta \in \Gamma_M$.

Sei $\Gamma_\Delta \in \mathcal{T}(F \backslash\backslash \Delta)$ und $f_\Delta : \Delta \longrightarrow F$ mit $\delta^{f_\Delta(\delta)} \in \Gamma_\Delta$. Darüberhinaus sei für alle $\gamma \in \Gamma_\Delta$ der Stabilisator in F bekannt.

(1) $\Gamma_\Omega := \{\}$.
(2) **Für alle** $\gamma \in \Gamma_\Delta$ **tue**
(3) Berechne $\Phi^{-1}(\gamma)$ mit $\mathcal{A}$.
(4) Verwende $\mathcal{B}$ zur Berechnung einer Transversale $\Gamma_\gamma \in \mathcal{T}(\Phi_G^{-1}(N_F(\gamma))\backslash\backslash\Phi^{-1}(\gamma))$, sowie der zugehörigen Stabilisatoren in $\Phi_G^{-1}(N_F(\gamma))$.
(5) $\Gamma_\Omega := \Gamma_\Omega \cup \Gamma_\gamma$.
ende

Es ist nun $\Gamma_\Omega \in \mathcal{T}(G\backslash\backslash\Omega)$.
Wir erhalten $f_\Omega : \Omega \longrightarrow G$ mit $\omega^{f_\Omega(\omega)} \in \Gamma_\Omega$ für alle $\omega \in \Omega$ wie folgt:

(1) $\omega' := \Phi(\omega)$.
(2) $\gamma := \omega'^{f_\Delta(\omega')}$.
(3) $\delta := \omega^{\tilde{g}}$ mit $\tilde{g} \in \Phi_G^{-1}(f_\Delta(\omega'))$.
(4) Berechne $f_M(\delta)$ mit Hilfe von $\mathcal{B}$ für $M = \Phi^{-1}(\gamma)$.
(5) $f_\Omega(\omega) := \tilde{g} f_M(\delta)$.

4.2 Verschmelzen

Seien G und F Gruppen. G operiere auf einer Menge Ω und F operiere auf einer Menge Δ. Sei $\Phi : \Omega \longrightarrow \Delta$ eine surjektive Abbildung. Weiter sei $\Phi_G : G \longrightarrow F$ ein Gruppenepimorphismus und es gelte:

$$\forall g \in G \quad \forall \omega \in \Omega \ : \ \Phi_\Omega(\omega^g) = \Phi_\Omega(\omega)^{\Phi_G(g)}.$$

$\mathcal{A}$ sei ein Algorithmus zur Berechnung von $\Phi^{-1}(\delta)$ für alle $\delta \in \Delta$. Sei $\Gamma_\Omega \in \mathcal{T}(G\backslash\backslash\Omega)$. Sei $\mathcal{B}$ ein Algorithmus, der eine Funktion $f_\Omega : \Omega \longrightarrow G$ berechnet, wobei für alle $\omega \in \Omega$ gilt: $\omega^{f_\Omega(\omega)} \in \Gamma_\Omega$. Sei $\mathcal{N}$ ein Algorithmus, der zu $\gamma \in \Gamma_\Omega$ den Stabilisator $N_G(\gamma)$ berechnet. Dann erhalten wir $\Gamma_\Delta \in \mathcal{T}(F\backslash\backslash\Delta)$ wie folgt:

(1) $\Gamma_\Delta := \{\}$.
(2) $C := \cup_{\gamma \in \Gamma_\Omega} \Phi(\gamma)$.
(3) **Solange** $C \neq \{\}$ **tue**
(4) Wähle $\delta \in C$.
(5) $C := C \setminus \delta$.
(6) $\Gamma_\Delta := \Gamma_\Delta \cup \delta$.
(7) Berechne $\Phi^{-1}(\delta)$.
(8) **Für alle** $\eta \in \Phi^{-1}(\delta)$ **tue**
(9) Berechne $\eta^{f_\Omega(\eta)}$.
(10) $C := C \setminus \Phi(\eta^{f_\Omega(\eta)})$.
(11) $f_\Delta(\Phi(\eta^{f_\Omega(\eta)})) := \Phi_G(f_\Omega(\eta))^{-1}$.
ende
ende

Wir erhalten $f_\Delta : \Delta \longrightarrow F$ mit $\delta^{f_\Delta(\delta)} \in \Gamma_\Delta$ für alle $\delta \in \Delta$ folgendermaßen:

(1) Für $\delta \in \Delta$ berechne $\Phi^{-1}(\delta)$ mit Algorithmus $\mathcal{A}$.
(2) Wähle ein $\gamma \in \Phi^{-1}(\delta)$ und berechne $f_\Omega(\gamma)$.
(3) **Falls** $\Phi(\gamma^{f_\Omega(\gamma)}) \in \mathcal{T}(F\backslash\backslash\Delta)$, so $f_\Delta(\delta) := \Phi_G(f_\Omega(\gamma))$,
(4) **sonst** $f_\Delta(\delta) := \Phi_G(f_\Omega(\gamma)) f_\Delta(\Phi(\gamma^{f_\Omega(\gamma)}))$.

Zu $\delta \in \Delta$ berechnet man eine Menge von Erzeugern von $N_F(\delta)$ wie folgt:

(1) Berechne $M = \Phi^{-1}(\delta)$ mit Algorithmus $\mathcal{A}$.
(2) Wähle ein $m \in M$ mit $m \in \Gamma_\Omega$.
(3) Berechne eine Menge T von Erzeugern von $N_G(m)$ mit Algorithmus $\mathcal{N}$.
(4) **Für alle** $x \in M$ **tue**
(5) Berechne $x^{f_\Omega(x)}$ mit Algorithmus $\mathcal{B}$.
(6) **Falls** $x^{f_\Omega(x)} = m$, **dann** $T := T \cup \Phi_G(f_\Omega(x))^{-1}$.
(7) **ende**

4.2.1 Bemerkung:
Das Homomorphieprinzip erwies sich bei verschiedenen Konstruktionsproblemen als praxistauglich. Unter anderem wurde es eingesetzt für

- die Konstruktion auflösbarer Gruppen ([31]).
- die Konstruktion von Doppelnebenklassen ([47], [53]), insbesondere von Doppelnebenklassen mit Younguntergruppen. Hierbei wurde das sogenannte „Leiterspiel" verwendet, das aus einer Folge von Aufspaltungs- und Verschmelzungsschritten besteht. Auf diese Weise konnten große Doppelnebenklassenprobleme gelöst werden, allerdings hat dieser Algorithmus einen sehr großen Speicherverbrauch, d.h. mit wachsender Problemgröße steigt der Platzbedarf erheblich.
- die Konstruktion schlichter Graphen zu vorgegebener Gradpartition ([18],[19]). In diesem Fall wurden keine Verschmelzungsschritte benötigt, so daß insbesondere durch Berücksichtigung sog. *impliziter* Lösungen eine sehr hohe Generierungsgeschwindigkeit erreicht werden konnte.

5 Ordnungstreue Erzeugung

Da das Homomorphieprinzip wegen des hohen mathematischen Aufwandes bei gewissen Konstruktionsproblemen erst bei sehr großen Eingabeparametern effizient (d.h. schneller als alle anderen bekannten Konstruktionsmethoden) wird, ist es manchmal nicht ratsam oder gar unmöglich, das Homomorphieprinzip zu verwenden. In solchen Situationen kann die *ordnungstreue Erzeugung* eine sinnvolle Alternative sein. Es seien z.B. die Bahnen bzgl. der Operation einer Gruppe G auf 2^M gesucht. Dann erzeugt man (grob gesprochen) alle in Frage kommenden Teilmengen von M in lexikographisch aufsteigender Reihenfolge und wählt jeweils die kleinsten Elemente in jeder Bahn als Lösungen. Das Isomorphieproblem wird hier ziemlich direkt angegangen, d.h. zu einem Kandidaten K muß geprüft werden, ob dieser minimal in seiner Bahn unter G ist. Für diese Aufgabe verwenden wir einen sogenannten Minimalitätstest. Der Minimalitätstest liefert uns gleichzeitig die Automorphismengruppe des getesteten Kandidaten.

5.1 Erzeugungsalgorithmus und Minimalitätstest

Die Menge $(Z, \leq)$ sei total geordnet und die Gruppe G operiere auf Z. Wie bereits oben erwähnt, ist dann $rep_<(G\backslash\backslash Z) = \{z \in Z \mid \forall g \in G : z \leq z^g\}$ eine kanonische Transversale der Bahnen von G auf Z. Die ordnungstreue Erzeugung nach Read ([43]) beschreibt dann der folgende

5.1.1 Satz:
Sei $Z = \bigcup_{i=1}^{n} Z_i$, $Z_i{}^G \subseteq Z_i, Z_i \cap Z_j = \emptyset$ $(i \neq j)$, und P ein Algorithmus, der $\forall z \in rep_<(G\backslash\backslash Z_i)$ eine Menge $P(z) \subseteq Z$ erzeugt, so daß

$$\forall i : rep_<(G\backslash\backslash Z_{i+1}) \subseteq \bigcup_{z \in rep_<(G\backslash\backslash Z_i)} P(z).$$

Dann erhält man die gesuchte Transversale durch:

i) Erzeuge $rep_<(G\backslash\backslash Z_1)$ und setze $rep_<(G\backslash\backslash Z) := rep_<(G\backslash\backslash Z_1)$.
ii) Für $i = 1, ..., n$ durchlaufe $rep_<(G\backslash\backslash Z_i)$ mit z, und erzeuge dabei $P(z)$. Durchlaufe weiter $P(z)$ mit w und prüfe, ob w minimal in seiner Bahn ist. Falls ja, so setze $rep_<(G\backslash\backslash Z) := rep_<(G\backslash\backslash Z) \cup w$.

Häufig gilt zusätzlich:

$$\forall i \; \forall z \in rep_<(G\backslash\backslash Z_{i+1}) : \exists!\; z' \in rep_<(G\backslash\backslash Z_i) : z \in P(z').$$

Diese Bedingung stellt sicher, daß nicht zu viele überflüssige Kandidaten betrachtet werden.
Sei nun M eine Menge und die Gruppe G operiere auf M. Dann operiert G auch auf $Z = 2^M = \{T \mid T \subseteq M\}$ und wir können den obigen Satz auf diese Situation anwenden:

5.1.2 Satz:
Sei $(M, \leq)$ eine geordnete Menge und $Z = 2^M$ sei lexikographisch geordnet. Dann gilt: Falls $T \in rep_<(G\backslash\backslash Z)$ und $T_1 \subset T$ mit $T_1 < T$, so folgt $T_1 \in rep_<(G\backslash\backslash Z)$.

Beweis:
Sei $T = T_1 \cup T_2$ und $T_1 \notin rep_<(G\backslash\backslash Z)$ mit $T_1 < T$. Dann ist $T_1^g < T_1$ für ein $g \in G$. Falls $T_1^g = \{m_1, ..., m_l\}$ mit $m_1 < m_2 < ... < m_l$, dann existiert ein i mit $T_1 = \{m_1, ..., m_{i-1}, m_i', ..., m_l'\}$ mit $m_{i-1} < m_i' < ... < m_l'$ und $m_i < m_i'$. Aus $T^g = T_1^g \cup T_2^g \supseteq \{m_1, ..., m_i\}$ folgt $T^g < T_1 < T$. Dies ist ein Widerspruch zu $T \in rep_<(G\backslash\backslash Z)$.
□

5.1.3 Bemerkung:
Um Satz 5.1.1 anwenden zu können, müssen wir für einen Repräsentanten $T = \{m_1, ..., m_l\}$ mit $m_1 < m_2 < ... < m_l$ die Menge $P(T)$ definieren als

$$P(T) := \{\{m_1, ..., m_l, w\} \subseteq Z \mid w > m_l\}.$$

Obwohl es keine Komplexitätsanalyse für 5.1.1 gibt, ist dieser Ansatz doch recht effizient. Ein wichtiger Punkt dabei ist, daß man kaum überflüssige Kandidaten testen muß.
Bei der Erzeugung der Kandidatenmenge $P(T)$ für einen Repräsentanten T kann zur Erhöhung der Effizienz das folgende Lemma benutzt werden.

5.1.4 Lemma:
Sei $T = \{m_1, ..., m_l\}$ mit $m_1 < m_2 < ... < m_l$. Dann gilt:

i) Falls $y \in \bigcup_{i=1}^{t-1} \bigcup_{m=m_i+1}^{m_{i+1}-1} m^{N_G(\{m_1,...,m_i\})}$, so folgt $T \cup \{y\} \notin rep_<(G\backslash\backslash Z)$.
ii) Falls y für $i = t$ nicht minimal in seiner Bahn unter $N_G(T)$ ist, so ist $T \cup \{y\} \notin rep_<(G\backslash\backslash Z)$.

Beweis:

i) Sei $y \in \bigcup_{i=1}^{t-1} \bigcup_{m=m_i+1}^{m_{i+1}-1} m^{N_G(\{m_1,...,m_i\})}$. Dann existiert ein j mit $1 \leq j \leq t-1$ und ein k mit $m_j + 1 < k \leq m_{j+1} - 1$ mit $y \in k^{N_G(\{m_1,...,m_j\})}$. Also gibt es ein $g \in N_G(\{m_1, ..., m_j\})$ mit $y^g = k$. Betrachte nun $T \cup \{y\} = \{m_1 < ... < m_t < y\}$ und

$$\begin{aligned}\{T \cup \{y\}\}^g &= \{m'_1 < m'_2 < ... < m'_{t+1}\} = \\ &= \{m_1 < ... < m_j < m'_{j+1} < ... < m'_{t+1}\}, \text{ da } g \in N_G(\{m_1, ..., m_j\}) \\ &= \{m_1 < ... < m_j < ... < k < ... < m_{j+1} < ... < m'_{t+1}\}.\end{aligned}$$

Daraus folgt $\{T \cup \{y\}\}^g < \{T \cup \{y\}\}$ und deshalb $\{T \cup \{y\}\} \notin rep_<(G\backslash\backslash Z)$.
ii) Sei $y > m_t$ mit $y^g < y$ für ein $g \in N_G(T)$. Dann gilt

$$\begin{aligned}\{T \cup \{y\}\}^g &= \{m_1, ..., m_t, y\}^g = \{m_1^g, ..., m_t^g, y^g\} = \\ &= \{m_1, ..., m_t, y^g\} < \{m_1, ..., m_t, y\},\end{aligned}$$

da $y^g < y$. Somit folgt $\{T \cup \{y\}\} \notin rep_<(G\backslash\backslash Z)$.

□

5.1.5 Bemerkung:
Die Anwendbarkeit des Lemmas hängt natürlich davon ab, ob die verwendeten Normalisatoren bekannt sind. Allgemeiner kann man aber feststellen:
Ist $T = \{m_1, ..., m_t\}$ mit $m_1 < m_2 < ... < m_t$, $g \in G$ und $y > m_t$ mit

- $\exists i \leq t$ mit $T^g = \{m_1, ..., m_i, m'_{i+1}, ..., m'_t\}$ und
- $m_i + 1 \leq y^g \leq m_{i+1} - 1$,

so folgt $T \cup \{y\} \notin rep_<(G\backslash\backslash Z)$.

5.1.6 Minimalitätstest. Der schwierigste Teil der ordnungstreuen Erzeugung ist der Minimalitätstest. Zu einem Kandidaten K und einer Gruppe G muß die Frage beantwortet werden, ob K minimal in seiner Bahn bzgl. der Operation von G ist. Im Prinzip müssen also alle $g \in G$ auf K angewandt werden, um zu überprüfen, ob ein $h \in G$ mit $K^h < K$ existiert, allerdings ist klar, daß man nicht wirklich so vorgehen kann. Eine Beschreibung eines Minimalitätstests, bei dem die operierende Gruppe als labeled branching dargestellt ist, findet sich in [19]. Eine Spezialversion des Minimalitätstests für Graphen (S_n ist operierende Gruppe) wird vorgestellt in [16]. Vom Autor wurde darüberhinaus eine Spezialversion eines Minimalitätstests für Konfigurationen ($S_n \times S_n$ ist operierende Gruppe) implementiert.
Da der Aufbau eines Minimalitätstests z.B. in den obengenannten Arbeiten sehr genau dargestellt ist, kann an dieser Stelle auf Details verzichtet werden. Es sei lediglich noch erwähnt, daß es ohne zusätzlichen Zeitaufwand möglich ist, während des Minimalitätstest die Automorphismengruppe von K mitzuberechnen.

5.1.7 Lerneffekt. Bei Read's ordnungstreuer Erzeugung muß bei jeder Hinzunahme eines neuen Elementes ein Minimalitätstest durchgeführt werden. Das kann unnötigen Zeitaufwand bedeuten, weil man Kandidaten, die letztlich bestehen, quasi mehrmals testen muß. Noch ungünstiger wird die Situation, wenn zusätzliche Restriktionen berücksichtigt werden sollen, da man unter Umständen Kandidaten, die den Minimalitätstest bestanden haben, aufgrund der Nebenbedingungen verwerfen muß. Der Aufwand zum Testen der Minimalitätseigenschaft war bei diesen Kandidaten dann umsonst. Weil das Prüfen der zusätzlichen Restriktionen häufig sehr viel schneller ist als ein Minimalitätstest, kann man die Strategie verfolgen, in den Zwischenschritten nur die Nebenbedingungen zu testen und nur einen abschließenden Minimalitätstest durchzuführen. Da man so wiederum, ohne es zunächst zu merken, sehr früh zu nicht-minimalen Kandidaten kommen kann, ist es notwendig, aus einem nicht bestandenen abschließenden Minimalitätstest Schlüsse zu ziehen ([16]).

5.1.8 Lemma: *Lerneffekt*
Sei $T = \{m_1, ..., m_t\}, m_1 < m_2 < ... < m_t$ nicht minimal in seiner Bahn unter G, d.h. es existiert ein $g \in G$ mit $T^g < T$. Also gibt es ein $i < t$ mit $T^g = \{m_1, ..., m_i, m'_{i+1}, ..., m'_t\}$ und $m'_{i+1} < m_{i+1}$. Es ist $\{m_1^{g^{-1}}, ..., m_i^{g^{-1}}, m'^{g^{-1}}_{i+1}\} = \{m_{j_1}, ..., m_{j_{i+1}}\}$ mit $1 < j_l \leq t$ für alle $l = 1, ..., i+1$. Sei $k := max\{j_1, ..., j_{i+1}, i+1\}$. Dann sind alle $\tilde{T} \subseteq Z$ mit $\tilde{T} = \{m_1, ..., m_k, \tilde{m}_{k+1}, ..., \tilde{m}_s\}, m_1 < ... < m_k < \tilde{m}_{k+1} < ... < \tilde{m}_s$ ebenfalls nicht minimal in ihrer Bahn.

Beweis:
Da $k \geq i+1$ ist $\tilde{T} = \{m_1, ..., m_i, m_{i+1}, ..., m_k, \tilde{m}_{k+1}, ..., \tilde{m}_t\}$. Außerdem ist $\tilde{T}^g = \{m_1^g, ..., m_k^g, \tilde{m}_{k+1}^g, ..., \tilde{m}_t^g\} \supseteq \{m_{j_1}^g, ..., m_{j_{i+1}}^g\} = \{m_1, ..., m_i, m'_{i+1}\}$. Es folgt $\tilde{T}^g < \tilde{T}$ wegen $m'_{i+1} < m_{i+1}$. □

5.1.9 Bemerkung:
Der Lerneffekt sagt uns also, daß man bei der rekursiven Erzeugung der Kandidaten in der Rekursion so weit zurückgehen kann, bis maximal die ersten $k-1$ Elemente mit dem verworfenen Kandidaten übereinstimmen. Da man den aufwendigen Minimalitätstest normalerweise beim ersten $g \in G$ mit $T^g < T$ abbricht, findet man aber nicht unbedingt den größtmöglichen Rückwärtsschritt.

5.2 Restriktionen und Anwendungen

Insbesondere bei der Konstruktion von Graphen ist es wichtig, beliebige Nebenbedingungen berücksichtigen zu können, wie z.B. die Taillenweite, Planarität oder die Existenz eines bestimmten Teilgraphen. Bei der ordnungstreuen Konstruktion von Graphen muß eine rechte obere Dreiecksmatrix auf alle möglichen Weisen gefüllt werden, um alle Lösungen zu erhalten. Dabei ist die Numerierung der Plätze der Adjazenzmatrix frei wählbar und kann somit an die vorgegebenen Nebenbedingungen angepaßt werden.
Die naheliegenste Numerierung ist wohl, die rechte obere Dreiecksmatrix zeilenweise durchzunumerieren (siehe [16]). Konsistente Restriktionen wie Taillenweite oder Planarität können schon mit dieser einfachen Numerierung berücksichtigt werden.
Möchte man hingegen etwa die Existenz eines bestimmten Teilgraphen vorschreiben, so ist eine Umnumerierung der Plätze der Adjazenzmatrix nötig (siehe [13]). Es ist erlaubt,

den Atomen des gewünschten Teilgraphen von vornherein die niedrigsten Nummern zuzuweisen und die zum Teilgraphen gehörigen Plätze der Adjazenzmatrix vorzubesetzen, wie in 9.1.18 beschrieben. Eine sehr effektive Implementation dieser Methode wurde z.B. von S. Molodtsov ([?]) vorgenommen.
Alternativ kann die Vorgabe eines (oder mehrerer) Teilgraphen auch mittels einer Kombination aus Homomorphieprinzip und ordnungstreuer Erzeugung ([15]) realisiert werden.

5.2.1 Graphen. Bei der Anwendung der ordnungstreuen Erzeugung auf die Konstruktion von Graphen mit n Knoten ergibt sich die folgende Situation: S_n operiert auf der Menge der $(n \times n)$–Adjazenzmatrizen der Graphen mit n Knoten, indem Zeilen und Spalten simultan permutiert werden. Es genügt, die Plätze der rechten oberen Dreiecksmatrix der Adjazenzmatrix zeilenweise durchzunumerieren, und nur diese auf alle möglichen Weisen lexikographisch aufsteigend zu füllen. Die beim Minimalitätstest auftretenden Stabilisatoren kann man mit Hilfe von Partitionen darstellen, deshalb ist es nicht notwendig auf die Darstellung als Simskette oder labeled branching zurückzugreifen. Es existieren u.a. folgende Implementationen dieser Methode:

- Konstruktion aller schlichten Graphen mit bis zu 12 Knoten ([16]).
- Konstruktion von molekularen Graphen ([15]) zwecks Verwendung im MOLGEN-Generator Version 3.
- Konstruktion molekularer Graphen zwecks Verwendung im MOLGEN-Generator Version 4.0 (vorgenommen vom Autor).
- Konstruktion regulärer Graphen ([37], [38]).

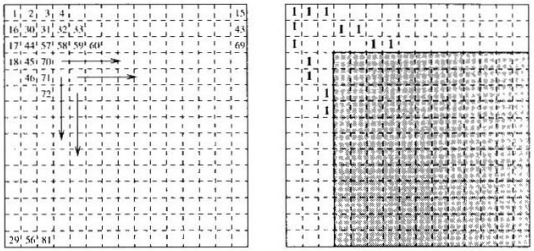
5.2.2 Konfigurationen. Eine Konfiguration (n_r, b_k) ist eine Inzidenzstruktur mit n Punkten und b Geraden, so daß gilt:

- Jede Gerade enthält k Punkte.
- Jeder Punkt liegt auf r Geraden.
- Zwei unterschiedliche Punkte werden höchstens durch eine Gerade verbunden.

Gilt $n = b$ (also auch $r = k$), dann ist die Konfiguration symmetrisch und wird mit n_k bezeichnet.

Numeriert man die Punkte und Linien einer Konfiguration K zu den Parametern n_k beliebig, so erhält man also eine $(n \times n)$–Matrix MK mit Einträgen 0 oder 1, sowie Zeilen- und Spaltensummen k. Darüberhinaus gibt es keine 4 Indizes z_1, z_2, s_1, s_2 mit $MK(z_1, s_1) = MK(z_1, s_2) = MK(z_2, s_1) = MK(z_2, s_2) = 1$. Umgekehrt wird durch eine Matrix, die den obigen Bedingungen genügt auch eine Konfiguration mit den Parametern n_k definiert. Sei $A_{n,k}$ definiert als die Menge der $(n \times n)$–Matrizen, die den obigen Bedingungen genügen.
Bei der Konstruktion der Konfigurationen n_k sind also die Bahnen bzgl. der kanonischen Operation der Gruppe $S_n \times S_n$ auf $A_{n,k}$ zu bestimmen. Zusätzlich zum Isomorphieproblem kann es hierbei, abhängig von den Parametern n und k, auch erhebliche Probleme machen, die Menge $A_{n,k}$ zu konstruieren. Die im folgenden skizzierte Methode zielt allerdings mehr auf die Lösung des Isomorphieproblems ab und hat sich insbesondere bei den Parametern n_3 als sehr effizient erwiesen.

Bild 1: *Generierungsstrategie für Konfigurationen*



Wir orientieren uns am Algorithmus zur ordnungstreuen Erzeugung von Graphen und möchten daher die Plätze der zu füllenden Matrix so durchnumerieren, daß wir von der Transitivität der operierenden Gruppe profitieren können. Die von uns gewählte Numerierung ist in Bild 1 (links) angedeutet. Im Fall der Konfigurationen 15_3 ergibt sich aufgrund der Transitivität von $S_{15} \times S_{15}$ z.B., daß alle Lösungen Fortsetzung des in Bild 1 (rechts) gezeigten Anfangsstücks sein müssen.
Die gewählte Numerierung der Plätze der Matrix erlaubt es also, bereits während der Konstruktion der Kandidaten die Transitivität der operierenden Gruppe auszunutzen und viele überflüssige Kandidaten, die den Minimalitätstest offensichtlich nicht bestehen werden, von vornherein zu verwerfen. Die benötigten Stabilisatoren kann man (wie bei der ordnungstreuen Erzeugung von Graphen) einfach als Partitionen speichern. Die Idee zur Implementation des Minimalitätstest lehnt sich stark an den Minimalitätstest für Graphen an. Zu beachten ist natürlich, daß eine andere operierende Gruppe vorliegt und Zeilen und Spalten nicht simultan permutiert werden. Prinzipiell stecken im Minimalitätstest aber keine neuen Ideen, so daß wir auf die technischen Details verzichten können. Die eigentliche Idee dieses Algorithmus ist, die Numerierung der Plätze der Matrix geschickt an die operierende Gruppe anzupassen.
Mit der oben dargestellten Methode ist es dem Autor gelungen, die Konfigurationen zu folgenden Parametersätzen *erstmals* zu berechnen (Pentium 90):

Tabelle 1: *Anzahlen von Konfigurationen zu den Parametersätzen* n_3

n	Anzahl	Zeit
15	245342	2h 26m
16	3004881	30h 26m
17	38904499	17d 6h 17m

5.2.3 Inzidenzstrukturen. Nun wollen wir die vorgestellten Methoden anwenden auf die Operation einer Permutationsgruppe G dargestellt als labeled branching auf der Menge der $(z \times s)$-Matrizen zu vorgegebenen Zeilen- und Spaltensummen und Einträgen 0 oder 1.

5.2.4 Satz: *Gale-Ryser*
Eine $(m \times n)$-Matrix I mit Einträgen 0 oder 1 und Zeilensummen $z_1, \ldots, z_m$ sowie Spaltensummen $s_1, \ldots, s_n, s_i \geq s_{i+1}$ $(i = 1, \ldots, n-1)$ existiert genau dann, wenn gilt:

i) $\sum_{i=1}^{m} z_i = \sum_{i=1}^{n} s_i$.

ii) $\sum_{i=1}^{k} s_i \leq \sum_{i=1}^{m} min\{z_i, k\}$, $(k = 1, ..., n)$.

Bild 2: *Anordnung der Plätze der Inzidenzmatrix*

1	1	2	3	4	5
2	6	7	8	9	10
3	11	12	13	14	15
4	16	17	18	19	20
	5	6	7	8	9

Es sei $z = 4, s = 5$ und $G < S_z \times S_s \leq S_{z+s}$. Die Plätze der Inzidenzmatrix werden zeilenweise numeriert. G permutiere, wie durch die Zeilen- und Spaltennummern angedeutet, die Zeilen bzw. Spalten der Matrix.

Mit Hilfe von 5.2.4 ist es kein Problem, alle 0/1-Matrizen mit z Zeilen, s Spalten und vorgegebenen Zeilen- und Spaltensummen zu erzeugen. Wir können uns also auf den Minimalitätstest konzentrieren.
Im Vergleich zu den vorherigen Anwendungen der ordnungstreuen Erzeugung ist die Situation nun komplizierter, da die operierende Gruppe viel allgemeiner ist, und wir die erforderlichen Stabilisatoren deshalb nicht mehr mit Hilfe von Partitionen darstellen können, sondern sie tatsächlich explizit als labeled branching berechnen müssen. Zu diesem Zweck verwenden wir den Basiswechsel für labeled branchings.
Der hier verwendete Minimalitätstest läßt sich kaum in wenigen Worten schildern: der interessierte Leser sei daher auf [19] verwiesen. Dort wurden Inzidenzstrukturen zur Konstruktion von Graphen zu vorgegebener Gradpartition benötigt. Darüberhinaus fand die im Rahmen von [19] entstandene Implementation schon eine Anwendung im Bereich der kombinatorischen Chemie (siehe [52]).

5.2.5 Doppelnebenklassen. Gegeben seien drei Permutationsgruppen A, B, G mit $A \leq G$ und $B \leq G$. Wir möchten die Doppelnebenklassen $A \setminus G/B$ berechnen.
Sei $G \leq S_n$. Dann entspricht jedes $g \in G$ einer $n \times n$-Matrix mit Einträgen 0 oder 1 und Zeilen- und Spaltensummen 1. Somit können wir die Operation von $A \times B$ auf G auffassen als Operation von $A \times B$ auf der Menge der zu G gehörigen 0/1-Matrizen. A permutiert dabei die Zeilen der Matrix und B permutiert die Spalten. Speichern wir die Permutationsgruppen als labeled branching, so erhalten wir einen Spezialfall von 5.2.3.
Diese Methode ist rein algorithmisch und verwendet kein gruppentheoretisches Wissen. Natürlich ist es aber leicht möglich, in die Erzeugung der Kandidaten von außen einzugreifen, und so im Spezialfall zusätzliches Wissen über die operierenden Gruppen einfließen zu lassen.
So wurden z.B. die 1858 Doppelnebenklassen $S_2 \wr S_{12} \setminus S_{24}/M_{24}$ mit diesem Algorithmus bestimmt. Dabei wurde die Transitivität von $S_2 \wr S_{12}$ ausgenutzt. Diese Doppelnebenklassen wurden bereits unter Verwendung des Homomorphieprinzips mit Hilfe des sog. Leiterspiels bestimmt ([53]), jedoch benötigte das damalige Programm erhebliche Ressourcen (ca. 3d cpu, HP 9000/735 mit 272 MB RAM). Der etwas direktere Zugang mit ordnungstreuer Erzeugung bestätigte die Lösungsanzahl, vor allem der Platzbedarf war aber bedeutend günstiger (ca. 3 MB RAM, ca. 1d cpu, Pentium 90).

Ein wichtiger Spezialfall ist, wenn G die volle symmetrische Gruppe und A eine Youngguntergruppe von G ist. Bei der Berechnung dieser Doppelnebenklassen vereinfacht sich unser Algorithmus, da die Operation von A auf G genau bekannt ist (siehe [19]). Diese Art von Doppelnebenklassen wurde wiederum schon mit Hilfe des Leiterspiels berechnet ([47]), allerdings mit beträchtlichem Speicherverbrauch. Werden die Eingabeparameter zu groß, besteht die Gefahr, daß der vom Leiterspiel benötigte Speicherplatz die Systemressourcen übersteigt, obwohl der Zeitaufwand noch unkritisch wäre. Die Leistungsfähigkeit der ordnungstreuen Erzeugung soll folgende Tabelle illustrieren:

Tabelle 2: *Doppelnebenklassen mit Younguntergruppen*

	Anzahl	Zeit
$S_{\{164,4\}} \setminus S_{168}/PSL_2(167)$	35	21 sec.
$S_{\{163,5\}} \setminus S_{168}/PSL_2(167)$	451	8 min. 57 sec.
$S_{\{162,6\}} \setminus S_{168}/PSL_2(167)$	12838	4h 20min
$S_{\{161,7\}} \setminus S_{168}/PSL_2(167)$	283551	4d 2h 56min

Die obigen Doppelnebenklassen wurden alle auf einem Pentium 90 mit einem Speicherverbrauch von ca. 4 MB berechnet.

5.2.6 Konjugationsklassen. Die Berechnung der Konjugationsklassen einer Gruppe G läßt sich in ebenso naheliegender Weise wie die Berechnung von Doppelnebenklassen auf die ordnungstreue Erzeugung von 0/1-Matrizen mit Zeilen- und Spaltensummen 1 zurückführen. Wir fassen die Operation von G auf G einfach auf als die Operation von G auf der Menge der zu G gehörigen 0/1-Matrizen. Zeilen und Spalten einer zu $h \in G$ gehörigen 0/1-Matrix H werden bei Anwendung von $g \in G$ auf H simultan permutiert, so daß die entstehende Matrix gHg^{-1} dem Gruppenelement ghg^{-1} entspricht.
Die Leistungsfähigkeit des entstandenen Programms ist angesichts des recht naiven Zugangs erstaunlich. So wurden z.B. in Zusammenarbeit mit A. Betten die Konjugationsklassen der Automorphismengruppen aller auflösbaren Gruppen der Ordnung < 128 mit diesem Programm berechnet.

6 Zielgerichtete Erzeugung

6.1 Generierungsprozedur

Zur Erzeugung diskreter Strukturen haben wir bisher das Homomorphieprinzip und die ordnungstreue Erzeugung kennengelernt. Beide Methoden, insbesondere das Homomorphieprinzip, zielen mehr auf die Lösung des Isomorphieproblems ab.
In der Praxis läß sich das Isomorphieproblem jedoch häufig durch kanonische Numerierung lösen, da die gewünschte Lösungsanzahl überschaubar bleiben soll. Das Problem besteht vielmehr schon darin, die numerierten Strukturen zu konstruieren. Im Fall der Konstruktion von 0/1-Matrizen zu vorgegebenen Zeilen- und Spaltensummen z.B. ist die Konstruktion der numerierten Strukturen kein Problem, da der Konstruktionsalgorithmus durch 5.2.4 quasi vorgegeben ist. Bei den meisten Konstruktionsproblemen jedoch liegen keine theoretischen Erkenntnisse vor, die es erlauben würden, die Fortsetzbarkeit einer Teillösung festzustellen.

Berechnet werden sollen alle Abbildungen $f \in Y^X$, die einer gewissen zusätzlichen Restriktion $\mathcal{R}$ genügen. Es gelte $X = \{x_1, ..., x_n\}$. Wie in 1.2.13 suchen wir alle Tupel $(f(x_1), f(x_2), ..., f(x_n))$, die $\mathcal{R}$ genügen. Für unsere *verallgemeinerte* Backtrackprozedur müssen wir für alle $Z \subseteq X$ Restriktionen $\mathcal{R}_Z$ definieren, so daß für $Z_1 \subset Z_2 \subset X$, $f_1 \in Y^{Z_1}$, $f_2 \in Y^{Z_2}$ mit $f_1 \subset f_2$ gilt:

$$\text{Aus } \{\mathcal{R}_{Z_2}(f_2) = \text{WAHR}\} \text{ folgt } \{\mathcal{R}_{Z_1}(f_1) = \text{WAHR}\}.$$

6.1.1 Algorithmus: *Zielgerichtete Erzeugung*

(1) $k := 0$, $Z_k := \emptyset$.
(2) Berechnen der Nachfolger: ($P_{Z_k}(f_k)$ ist erfüllt.)
Wähle $\epsilon \in X \setminus Z_k$ und definiere $Z_{k+1} := Z_k \cup \epsilon$.
Für $y \in Y$ definiere $f^y_{k+1} \in Y^{Z_{k+1}}$ durch
- $f^y_{k+1}(\epsilon) = y$ und
- $f^y_{k+1}(x) = f_k(x)$ für alle $x \in Z_k$, falls $k > 0$.

$S_k := \{y \mid R_{Z_{k+1}}(f^y_{k+1}) = \text{WAHR}\}$.
(3) **Falls** $|S_k| = 0$, gehe zu (6).
(4) Wähle $y \in S_k$, $S_k := S_k \setminus y$, $k := k + 1$, $f_k := f^y_k$.
(5) **Falls** $k < n$, gehe zu (2), **sonst** gebe $(f(x_1), ..., f(x_n))$ aus und gehe zu (6).
(6) $k := k - 1$. **Falls** $k \geq 0$, gehe zu (3), **sonst** beende die Suche.

6.2 Strategie

Es sei ein Anfangsstück $g \in Y^{Z_1}$ gegeben. Möchten wir g erweitern, so wählen wir i.a. ein $Z_2 \supset Z_1$ mit $|Z_2| = |Z_1| + 1$ und berechnen dann alle Fortsetzungen $f \in Y^{Z_2}$ von g, die mit den gegebenen Restriktionen verträglich sind. Für die Wahl von Z_2 haben wir mehrere Möglichkeiten. Die Wahl von Z_2 beeinflußt, je nachdem welche Nebenbedingungen gelten sollen, die Anzahl der möglichen Fortsetzungen von g. Um den Zeitaufwand eines Backtrackingalgorithmus zu minimieren, ist es i.a. erfolgversprechend, zuerst einen Knoten mit minimal vielen Söhnen zu besuchen. Es kommt also darauf an, Z_2 möglichst geschickt zu wählen. Man kann leider nie mit Sicherheit vorhersagen, welche Wahl die günstigste ist, jedoch ist es häufig leicht, plausible Heuristiken zu finden, die sich in der Praxis bewähren. Wenn „kompliziertere" Restriktionen vorgegeben sind, so ist diese *zielgerichtete Konstruktion* der ordnungstreuen Erzeugung vorzuziehen.

6.2.1 Beispiel:
Ein recht stark beachtetes Konstruktionsproblem ist z.B. die Konstruktion von Cages. Unter einem (k, g)-Cage versteht man einen k-regulären Graphen mit Taillenweite g und minimaler Knotenzahl. Es gibt nur wenige Parametersätze, zu denen alle Cages berechnet wurden. Eines der effektivsten Programme zur Konstruktion 3-regulärer Cages stammt von G. Brinkmann [6], der ordnungstreue Erzeugung verwendete und die Taillenweite während der Konstruktion prüfte.
Eine beachtliche Geschwindigkeitssteigerung bei der Konstruktion 3-regulärer Cages wurde von W. Myrvold erzielt [34], die keine ordnungstreue Erzeugung verwendete, sondern vor jedem Einsetzen einer zusätzlichen Kante des Graphen eine günstige Einsetzposition sucht. Dadurch ist es ihr geglückt, den Backtrackingbaum stärker zu lichten, wodurch Geschwindigkeitssteigerungen bis zu Faktor 100 im Vergleich zur ordnungstreuen Erzeugung

erreicht wurden. Beispielsweise konnten mit diesem neuen Algorithmus alle (3, 9)-cages mit ca. 5 Tagen cpu-Zeit bestimmt werden, der ordnungstreue Ansatz von G. Brinkmann benötigte zuvor 259 Tage cpu-Zeit auf einem vergleichbaren Rechner.

6.2.2 Vergleich der allgemeinen Konstruktionsstrategien. Wir haben bisher vier unterschiedliche Konstruktionsstrategien kennengelernt. Es handelte sich um

- Homomorphieprinzip,
- Ordnungstreue Erzeugung,
- Generierung mittels kanonischer Konstruktionsreihenfolge (McKay), sowie
- Zielgerichtete Erzeugung.

Mit Hilfe des Homomorphieprinzips läßt sich der Rechenaufwand zur Lösung des Isomorphieproblems unter gewissen Voraussetzungen logarithmieren ([30]). Sollen komplizierte Restriktionen berücksichtigt werden, dann lassen sich leider meistens keine geeigneten Homomorphismen mehr finden, so daß das Homomorphieprinzip nicht angewandt werden kann, bzw. alle Ergebnisse im Nachhinein gefiltert werden müssen, was extrem ineffizient sein kann. Falls geeignete Homomorphismen vorliegen, so stellt das Homomorphieprinzip ein sehr mächtiges Werkzeug zur Konstruktion diskreter Strukturen dar.
An dieser Stelle sei allerdings noch erwähnt, daß aufgrund des hohen mathematischen Aufwands ein Laufzeitvorteil etwa im Vergleich zur ordnungstreuen Erzeugung erst bei größeren Eingabeparametern zu erwarten ist.

Die ordnungstreue Erzeugung sowie die Generierung mittels kanonischer Konstruktionsreihenfolge erlauben im Vergleich zum Homomorphieprinzip i.a. eine einfachere Berücksichtigung zusätzlicher Restriktionen ([13]). Jedoch ist festzustellen, daß die Hauptzielrichtung dieser Konstruktionsstrategien trotzdem die Vereinfachung des Isomorphieproblems ist. Bei der Konstruktion nach McKay soll im Vergleich zur ordnungstreuen Erzeugung der Zeitaufwand zur Lösung des Isomorphieproblems durch Verwendung ausgeklügelter kombinatorischer Invarianten weiter reduziert werden.

Die zielgerichtete Erzeugung schließlich, welche nach dem Wissen des Autors in dieser Arbeit zum ersten mal explizit formuliert wurde, bildet das andere Extrem, in dem Sinne, daß die Berücksichtigung zusätzlicher Restriktionen absoluten Vorrang im Vergleich zum Isomorphieproblem hat. In der Praxis ist dies eine wichtige Eigenschaft, da man i.a. an riesigen Lösungsanzahlen bzw. an einer sehr großen Menge von Lösungen überhaupt nicht interessiert ist, so daß es akzeptabel ist, eine Liste aller bisher gefundenen Lösungen zu führen. Vielmehr ist es wichtig, in möglichst kurzer Zeit zu möglichst variablen Restriktionen die ersten Lösungen zu generieren. Dieses Ziel läßt sich am besten mit Hilfe der zielgerichteten Erzeugung erreichen.

III. Konstruktion molekularer Graphen

Ein von Chemikern seit langem verfolgtes Ziel ist die automatische Analyse unbekannter Substanzen. Zu diesem Zweck werden i.a. verschiedene Spektroskopiemethoden (Massenspektroskopie, NMR, Infrarot) verwendet. Seitens der Chemie werden seit längerer Zeit Anstrengungen unternommen, aus den Spektren einer unbekannten Substanz S automatisch, d.h. mit Hilfe eines Computerprogramms, Informationen über S zu gewinnen, die es im Idealfall ermöglichen sollen, die Struktur von S exakt zu berechnen (siehe auch [20]). Wir können also davon ausgehen, daß eine Vielzahl von Informationen über S zur Verfügung steht. Die Aufgabe, aus diesen Informationen alle für S in Frage kommenden Strukturen zu berechnen, ist ein Problem aus dem Bereich der Konstruktion diskreter Strukturen zu vorgegebenen Nebenbedingungen.
Es existieren bereits mehrere (kommerzielle) Programme, um dieses Problem zu lösen ([8],[9],[10],[11],[42],[49]). Eines davon ist der MOLGEN-Generator, entwickelt am Lehrstuhl II für Mathematik der Universität Bayreuth. Er beruht auf dem Prinzip der ordnungstreuen Erzeugung und kann auch schon verschiedene Restriktionen berücksichtigen. Eines der Probleme des bisherigen MOLGEN-Generators ([4],[17],[27]) besteht darin, daß entscheidende Restriktionen, die evtl. zu einer drastischen Verkleinerung der Lösungsmenge führen können, erst nach der Konstruktion der Kandidaten geprüft werden, so daß die Chance vergeben wird, den Backtrackingbaum zu lichten. Ein weiteres Problem ist, daß etwa im Vergleich zu anderen Generatoren, viele Informationen über S nicht verarbeitet werden können, so daß die Zahl der Lösungsvorschläge häufig unnötig hoch ist.
Durch die Entwicklung einer neuen Version des MOLGEN-Generators sollten folgende Ziele erreicht werden:

- Durch die Verwendung einer neuen Konstruktionsstrategie (siehe 6) sollen die vorgegebenen Restriktionen frühzeitig benutzt werden, um die Geschwindigkeit und Reichweite des Generators zu erhöhen.
- Die Datenstrukturen sollen dahingehend erweitert werden, daß auch Atomladungen und Radikalstellen berücksichtigt werden können.
- Es sollen verschiedene, möglichst variable Eingabemöglichkeiten für Nebenbedingungen geschaffen werden, die insbesondere die aktuellen Benutzerwünsche abdecken.
- Die Erweiterbarkeit des Generators soll gewährleistet sein, d.h. es muß möglich sein, ohne in den eigentlichen Konstruktionsprozeß einzugreifen, weitere Eingabemöglichkeiten zu schaffen.

8 Generatorinput

Die Eingabe des Generators ist im wesentlichen eine Bruttoformel sowie eine Menge von Nebenbedingungen. Es sollen dann *vollständig* und *redundanzfrei* alle möglichen molekularen Graphen konstruiert werden, die diese vorgegebenen Eigenschaften besitzen.

8.1 Füllalgorithmus

Es besteht die Wahlmöglichkeit zwischen *ordnungstreuer Erzeugung* und *zielgerichteter Konstruktion*.
Die ordnungstreue Erzeugung ist schnell, wenn nur wenige Zusatzbedingungen vorgegeben sind. Insbesondere ist die ordnungstreue Erzeugung zu empfehlen, wenn man nur an der (evtl. sehr großen) Lösungsanzahl interessiert ist, weil die Lösungen nicht abgespeichert werden müssen.
Die zielgerichtete Konstruktion sollte verwendet werden, wenn aus der riesigen Menge der zu einer Bruttoformel möglichen Molekülstrukturen **gezielt** einige wenige Strukturen konstruiert werden sollen, d.h. wenn viele Restriktionen vorliegen.

8.1.1 Konstruktions- und Speicherlimit. Eine sehr einfache, aber notwendige Eingabe ist die Vorgabe von Obergrenzen für die Anzahl der zu konstruierenden Strukturen sowie für die Anzahl der abzuspeichernden Strukturen.

8.1.2 Bruttoformel. In 1.2.12 haben wir bereits die Atomnamenverteilung und die Atomtypenverteilung kennengelernt. Bisher war es in MOLGEN3.5 möglich, die Atomnamenverteilung anzugeben. Die Atomnamenverteilung wurde ersetzt durch eine sog.

8.1.3 Definition: *Weiche Atomnamenverteilung*
Unter einer *weichen Atomnamenverteilung* verstehen wir eine Menge

$$WANV := \{(an_1, v_1, b_1), ..., (an_k, v_k, b_k)\}.$$

Dabei gilt:

- Für alle $i = 1, ..., k$ bezeichnet an_i einen Atomnamen.
  Für alle $i = 1, ..., k$ muß jeder zu konstruierende molekulare Graph $anz_i \in [v_i; b_i]$ Atome mit Atomname an_i enthalten.
- $an_i \neq an_j$ für $i \neq j$.

Da der Generator mit Atomtypen arbeitet ist es notwendig, für jeden Atomnamen eine Liste der erlaubten Atomtypen anzugeben.

8.1.4 Definition: *Atomtypliste*
Eine Menge

$$ATL := \{(at_1, v_1, b_1), ..., (at_k, v_k, b_k)\}$$

heißt *Atomtypliste* zum Atomnamen α, wenn gilt:

- $k \geq 1$.
- at_i bezeichnet für alle $i = 1, ..., k$ einen Atomtyp.
- $at_i.na = \alpha$ für alle $i = 1, ..., k$.
- Für $i = 1, ..., k$ bezeichnet $[v_i; b_i]$ ein Intervall für die Häufigkeit des Atomtyps at_i.
- $at_i \neq at_j$ für $i \neq j$.

Für jeden Namen der (weichen) Atomnamenverteilung muß eine Atomtypliste angegeben werden. Bei der Generierung werden dann alle mit der (weichen) Atomnamenverteilung und den Atomtyplisten verträglichen Atomtypverteilungen gebildet. Zu gegebener Atomtypverteilung atv werden dann alle molekularen Graphen mit dieser Atomtypverteilung berechnet.

Hat man von der Möglichkeit Gebrauch gemacht, bei der (weichen) Atomnamenverteilung echte Intervalle für die Häufigkeit des Auftretens von Atomnamen anzugeben, so macht es Sinn, optional ein Intervall für die *Gesamtatomanzahl* des Moleküls anzugeben.

Ebenso kann man optional ein Intervall für die *Gesamtkantenzahl* des gesuchten molekularen Graphen vorschreiben.

Um die Verwendung der weichen Bruttoformel variabler zu gestalten, kann man die gültigen Atomnamenverteilungen sowie Atomtypenverteilungen genauer spezifizieren.

8.1.5 Beschränkung der Atomnamenverteilungen:
Sei die weiche Atomnamenverteilung $\Phi = \{(\phi_1, v_1, b_1), ..., (\phi_i, v_i, b_i)\}$ gegeben. Sei $\Delta := \{\phi_1, ..., \phi_i\}$. Unter einer *Beschränkung der Atomnamenverteilungen* verstehen wir ein Tupel $\mathcal{B} = (A, min, max)$ mit $A \subseteq \Delta$. $\mathcal{B}$ bewirkt, daß jede zu bildende Atomnamenverteilung genau $\xi \in [min, max]$ Atome at enthalten muß mit $at.na \in A$.

8.1.6 Beschränkung der Atomtypenverteilungen:
Sei die weiche Atomnamenverteilung $\Phi = \{(\phi_1, v_1, b_1), ..., (\phi_i, v_i, b_i)\}$ gegeben. Für alle $j = 1, ..., i$ sei die Atomtypliste L_j zum Atomnamen ϕ_j gegeben. Definiere die Menge Δ von Atomtypen in folgender Weise: Ein Atomtyp ϱ ist genau dann in Δ enthalten, wenn zu ϱ ein $k \in [1; i]$ existiert, so daß ϱ in der Atomtypliste L_k enthalten ist.
Unter einer *Beschränkung der Atomtypenverteilungen* verstehen wir ein Tupel $\mathcal{B} = (A, min, max)$ mit $A \subseteq \Delta$. $\mathcal{B}$ bewirkt, daß jede zu bildende Atomtypenverteilung genau $\xi \in [min, max]$ Atome at enthalten muß, deren Atomtyp in A enthalten ist.

8.1.7 Gesamtladung. Da Atomladungen in unserer Modellbildung enthalten sind, ist es sinnvoll, für jeden der zu konstruierenden molekularen Graphen die *Gesamtladung* vorzuschreiben. Weil positive und negative Ladungen sich evtl. gegenseitig aufheben können, muß angegeben werden, ob *Ladungstrennung* erlaubt sein soll. Falls Ladungstrennung erlaubt ist, so kann man für jeden der zu konstruierenden molekularen Graphen die *maximale Anzahl geladener Atome* vorgeben.

8.1.8 Radikalstellen. Für jeden der zu konstruierenden molekularen Graphen muß die Anzahl von Radikalstellen vorgeschrieben werden.

Da beim Konstruktionsalgorithmus zunächst alle möglichen Atomtypenverteilungen gebildet werden, können wir im folgenden davon ausgehen, daß wir molekulare Graphen $G \in 4^{AM[2]}$ konstruieren möchten, wobei die Menge $AM = \{\vartheta_1, ..., \vartheta_n\}$ der Atome bereits festgelegt wurde.

8.1.9 Maximale Bindungsvielfachheit. Es ist möglich, die maximale Bindungsvielfachheit anzugeben, z.B. kann man also Dreifachkanten verbieten.

8.2 Wasserstoffverteilung

8.2.1 H-Anzahl pro Atomname:

Für jeden Atomnamen A kann man ein Intervall für die Anzahl von Wasserstoffatomen vorgeben, die zu einem Atom mit Namen A verbunden sind.

8.2.2 H-Anzahl pro Atomtyp:

Für jeden Atomtyp AT kann man ein Intervall für die Anzahl von Wasserstoffatomen vorgeben, die zu einem Atom ϱ mit Atomtyp AT verbunden sind.

8.2.3 H-Verteilung pro Atomname:

Unter einer *H-Verteilung für den Atomnamen A* mit den Parametern $\{(a_1, v_1, b_1), ..., (a_k, v_k, b_k)\}$, $a_i \geq 0$, $v_i \leq b_i$, $a_i, v_i, b_i \in \mathbb{N}$ für alle $i = 1, ..., k$, verstehen wir eine Restriktion

$$\mathcal{R}_{\{(a_1,v_1,b_1),...,(a_k,v_k,b_k)\}} : \bar{4}^{(\hat{2}^{AM^{[2]}})} \longrightarrow \{\text{WAHR}, \text{FALSCH}\} \text{ mit}$$

$$\mathcal{R}_{\{(a_1,v_1,b_1),...,(a_k,v_k,b_k)\}}(f) := \begin{cases} \text{WAHR, falls } \forall i = 1, ..., k : v_i \leq \alpha_{f,i} \leq b_i, \\ \text{FALSCH, sonst.} \end{cases}$$

Dabei sei $\alpha_{f,i} := |\Delta|$ mit $\Delta \subseteq AM$. Per Definition sei $\vartheta \in AM$ genau dann Element von Δ, wenn gilt:

– $\text{AN}(\vartheta) = A$ und
– es existieren genau a_i Elemente $\xi_1, ..., \xi_{a_i} \in AM$ mit $\text{AN}(\xi_j) =$ „H“ und $f(\vartheta, \xi_j) = 1$ für $j = 1, .., a_i$.

8.2.4 H-Verteilung pro Atomtyp:

Unter einer *H-Verteilung für den Atomtyp AT* mit den Parametern $\{(a_1, v_1, b_1), ..., (a_k, v_k, b_k)\}$, $a_i \geq 0$, $v_i \leq b_i$, $a_i, v_i, b_i \in \mathbb{N}$ für alle $i = 1, ..., k$ verstehen wir eine Restriktion

$$\mathcal{R}_{\{(a_1,v_1,b_1),...,(a_k,v_k,b_k)\}} : \bar{4}^{(\hat{2}^{AM^{[2]}})} \longrightarrow \{\text{WAHR}, \text{FALSCH}\} \text{ mit}$$

$$\mathcal{R}_{\{(a_1,v_1,b_1),...,(a_k,v_k,b_k)\}}(f) := \begin{cases} \text{WAHR, falls } \forall i = 1, ..., k : v_i \leq \beta_{f,i} \leq b_i, \\ \text{FALSCH, sonst.} \end{cases}$$

Dabei sei $\alpha_{f,i} := |\Delta|$ mit $\Delta \subseteq AM$. Per Definition sei $\vartheta \in AM$ genau dann Element von Δ, wenn gilt:

– $\text{AT}(\vartheta) = AT$ und
– es existieren genau a_i Elemente $\xi_1, ..., \xi_{a_i} \in AM$ mit $\text{AN}(\xi_j) =$ „H“ und $f(\vartheta, \xi_j) = 1$ für $j = 1, .., a_i$.

8.3 Hybridisierungen

8.3.1 Definition: *Hybridisierung*

Gegeben sei ein molekularer Graph $f \in 4^{AM^{[2]}}$ sowie ein Atom $A \in AM$ vom Atomtyp AT. Unter der *Hybridisierung* $Hyb(f, A) = (h_1, ..., h_m) \models AT.val$ von A in f verstehen wir eine Partition mit:

– $h_i \leq 3$ für alle $i = 1, ..., m$.
– Für $i = 1, 2, 3$ gilt $|\{j : \ h_j = i\}| = |\{\xi \in AM : \ f(\xi, A) = i\}|$.

8.3.2 Gesamthybridisierung:
Gegeben sei ein Atomname N, eine Hybridisierung H und ein Intervall I.
Unter der *Vorgabe der Gesamthybridisierung* zu den Parametern N, H, I verstehen wir eine Restriktion $\mathcal{R}_{\{N,H,I\}} : \bar{4}^{(\bar{2}^{AM^{[2]}})} \longrightarrow \{\text{WAHR}, \text{FALSCH}\}$ mit

$$\mathcal{R}_{\{N,H,I\}}(f) := \begin{cases} \text{WAHR, falls } |\{\vartheta \in \text{AM} : \ \text{AN}(\vartheta) = N \wedge \text{Hyb}(f, \vartheta) = H\}| \in I. \\ \text{FALSCH, sonst.} \end{cases}$$

Optional besteht die Möglichkeit, den zusätzlichen Parameter w anzugeben.
Unter der *Vorgabe der Gesamthybridisierung* zu den Parametern N, H, I, w verstehen wir eine Restriktion $\mathcal{R}_{\{N,H,I,w\}} : \bar{4}^{(\bar{2}^{AM^{[2]}})} \longrightarrow \{\text{WAHR}, \text{FALSCH}\}$ mit

$$\mathcal{R}_{\{N,H,I,w\}}(f) := \begin{cases} \text{WAHR, falls } U_{\{f,N,H,w\}} \in I. \\ \text{FALSCH, sonst.} \end{cases}$$

Dabei sei $U_{\{f,N,H,w\}} := |\Delta|$ mit $\Delta \subset AM$. Per Definition sei $\vartheta \in AM$ genau dann Element von Δ, wenn gilt:

– $\text{AN}(\vartheta) = N$,
– $\text{Hyb}(f, \vartheta) = H$ und
– es existieren genau w Elemente $\xi_1, ..., \xi_w \in AM$ mit $\text{AN}(\xi_j) =$ „H“ und $f(\vartheta, \xi_j) = 1$ für $j = 1, .., w$.

8.4 Ringe

8.4.1 Definition: *Cycle*
Sei $f \in \bar{4}^{AM^{[2]}}$ ein molekularer Graph und $AM = \{\vartheta_1, ..., \vartheta_n\}$. Unter einem *Cycle* der Länge l in f verstehen wir einen geschlossenen Pfad $P = (p_1, ..., p_l, p_1)$, $p_i \in AM$ für $i = 1, ..., l$ mit $f(p_i, p_{i+1}) > 0$ für $i = 1, ..., l-1$, wobei für alle $1 \leq i, j \leq l$ mit $i \neq j$ gilt $p_i \neq p_j$.

8.4.2 Definition: *Ring*
Sei $f \in \bar{4}^{AM^{[2]}}$ ein molekularer Graph mit $AM = \{\vartheta_1, ..., \vartheta_n\}$. Ein *Ring* der Länge l in f ist ein Cycle $P = (p_1, ..., p_l, p_1)$ in f, wobei zusätzlich gilt:

$$|\{f(p_i, p_j) \mid f(p_i, p_j) > 0, 1 \leq i < j \leq n\}| = l.$$

Diese Ringdefinition stimmt mit der in MOLGEN3.5 verwendeten Ringdefinition überein.

Bezüglich Ringen bzw. Cycles bestehen folgende Eingabemöglichkeiten:

8.4.3 Ringanzahlen:
Vorgegeben werden zwei Intervalle I_l und I_a. Unter der *Vorgabe von Ringanzahlen* zu den Parametern I_l, I_a verstehen wir eine Restriktion

$$\mathcal{R}_{\{I_l,I_a\}} : \bar{4}^{(\bar{2}^{AM^{[2]}})} \longrightarrow \{\text{WAHR}, \text{FALSCH}\} \text{ mit}$$

$$\mathcal{R}_{\{I_l,I_a\}}(f) := \begin{cases} \text{WAHR, falls genau } a \in I_a \text{ Ringe der Länge } l \in I_l \text{ in f enthalten sind.} \\ \text{FALSCH, sonst.} \end{cases}$$

8.4.4 Cycleanzahlen:
Vorgegeben werden zwei Intervalle I_l und I_a. Unter der *Vorgabe von Cycleanzahlen* zu den Parametern I_l, I_a verstehen wir eine Restriktion

$$\mathcal{R}_{\{I_l,I_a\}} : \bar{4}^{(\bar{2}^{AM^{[2]}})} \longrightarrow \{\text{WAHR}, \text{FALSCH}\} \text{ mit}$$

$$\mathcal{R}_{\{I_l,I_a\}}(f) := \begin{cases} \text{WAHR, falls genau } a \in I_a \text{ Cycles der Länge } l \in I_l \text{ in f enthalten sind.} \\ \text{FALSCH, sonst.} \end{cases}$$

8.5 Bindungen

8.5.1 Definition: *Zyklische Bindung*
Sei $f \in \bar{4}^{AM^{[2]}}$ ein molekularer Graph und $AM = \{\vartheta_1, ..., \vartheta_n\}$. Eine Bindung $\{v_i, v_j\}_f$, $v_i, v_j \in AM$ heißt *zyklisch*, wenn f einen Cycle $P = (p_1, ..., p_l, p_1)$ besitzt, der $\{v_i, v_j\}_f$ enthält.
Wir unterscheiden zwischen neun Typen von Bindungen:

1. Einfachbindungen.
2. Doppelbindungen.
3. Dreifachbindungen.
4. Zyklische Einfachbindungen.
5. Zyklische Doppelbindungen.
6. Zyklische Dreifachbindungen.
7. Azyklische Einfachbindungen.
8. Azyklische Doppelbindungen.
9. Azyklische Dreifachbindungen.

Wir sprechen in diesem Zusammenhang auch vom Bindungstyp 1, ... , Bindungstyp 9.

8.5.2 Definition: *Einschränkung der erlaubten Bindungen*
Sei

- N_1 eine Menge von Atomnamen,
- N_2 eine Menge von Atomnamen,
- $t \in \{1, 2, ..., 9\}$ ein Bindungstyp,
- I ein Intervall.

Unter der *Einschränkung der erlaubten Bindungen* zu den Parametern (N_1, N_2, t, I) verstehen wir die Restriktion $\mathcal{R}_{(N_1,N_2,t,I)} : \bar{4}^{(\bar{2}^{AM^{[2]}})} \longrightarrow \{\text{WAHR}, \text{FALSCH}\}$ mit

$$\mathcal{R}_{(N_1,N_2,t,I)}(f) := \begin{cases} \text{WAHR, falls } \beta_{(N_1,N_2,t,I)} \in I, \\ \text{FALSCH, sonst.} \end{cases}$$

wobei $\beta_{(N_1,N_2,t,I)} := |\{\ \{a,b\}_f\ :\ \{a,b\}_f \text{ ist vom Typ } t \wedge a.AN \in N_1 \wedge b.AN \in N_2\}|$.

8.6 Symmetrien

Die Automorphismengruppe des Gesamtmoleküls operiert in natürlicher Weise auf der Menge der Atome des Moleküls und teilt die Atome in Bahnen ein.

8.6.1 Bahnen auf Atomnamen:
Sei

– N ein Atomname,
– I ein Intervall.

Unter der Vorgabe der *Anzahl der Bahnen der Automorphismengruppe* zu den Parametern (N, I) verstehen wir die Restriktion $\mathcal{R}_{(N,I)} : \bar{4}^{(\bar{2}^{AM^{[2]}})} \longrightarrow \{\text{WAHR}, \text{FALSCH}\}$ mit

$$\mathcal{R}_{(N,I)}(f) := \begin{cases} \text{WAHR, falls } |Aut(f)\backslash\backslash AM_N| \in I, \\ \text{FALSCH, sonst.} \end{cases}$$

wobei $AM_N := \{\varrho \in AM \mid \varrho.AN = N\}$.

8.6.2 Bahnen auf Atomtypen:
Sei

– T ein Atomtyp,
– I ein Intervall.

Unter der Vorgabe der *Anzahl der Bahnen der Automorphismengruppe* zu den Parametern (T, I) verstehen wir die Restriktion $\mathcal{R}_{(T,I)} : \bar{4}^{(\bar{2}^{AM^{[2]}})} \longrightarrow \{\text{WAHR}, \text{FALSCH}\}$ mit

$$\mathcal{R}_{(T,I)}(f) := \begin{cases} \text{WAHR, falls } |Aut(f)\backslash\backslash AM_T| \in I, \\ \text{FALSCH, sonst.} \end{cases}$$

wobei $AM_T := \{\varrho \in AM \mid \varrho.AT = T\}$.

8.7 Substrukturen

8.7.1 Substrukturen zu gegebener Summenformel. 8.7.2 Vorgabe der Atomnamen:
Sei

– $F = \{(n_1, a_1), ..., (n_k, a_k)\}$ eine Bruttoformel.
n_i bezeichnet jeweils einen Atomnamen und a_i die Häufigkeit mit der n_i vorkommt.
– I ein Intervall.

Unter der *Vorgabe zusammenhängender Teilgraphen* zu den Parametern (F, I) verstehen wir die Restriktion $\mathcal{R}_{(F,I)} : \bar{4}^{(\bar{2}^{AM^{[2]}})} \longrightarrow \{\text{WAHR}, \text{FALSCH}\}$ mit

$$\mathcal{R}_{(F,I)}(f) := \begin{cases} \text{WAHR, falls genau } a \in I \text{ zusammenhängende Teilgraphen} \\ \qquad\qquad \text{mit Formel F in f enthalten sind,} \\ \text{FALSCH, sonst.} \end{cases}$$

8.7.3 Vorgabe der Atomtypen:
Sei

– $F = \{(t_1, a_1), ..., (t_k, a_k)\}$ eine „Bruttoformel aus Atomtypen".
  t_i bezeichnet jeweils einen Atomtypen und a_i dessen Häufigkeit.
– I ein Intervall.

Unter der *Vorgabe zusammenhängender Teilgraphen* zu den Parametern (F, I) verstehen wir die Restriktion $\mathcal{R}_{(F,I)} : \bar{4}^{(\bar{2}^{AM^{[2]}})} \longrightarrow \{\text{WAHR}, \text{FALSCH}\}$ mit

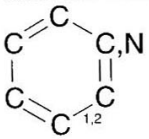
$$\mathcal{R}_{(F,I)}(f) := \begin{cases} \text{WAHR}, & \text{falls genau } a \in I \text{ zusammenhängende Teilgraphen} \\ & \text{mit Formel F in f enthalten sind,} \\ \text{FALSCH}, & \text{sonst.} \end{cases}$$

8.7.4 Substrukturen in Form von Teilgraphen. Die bei weitem variabelste und auch wichtigste Nebenbedingung bei der Generierung ist die Verwendung von Substrukturen.

8.7.5 Definition: *Substruktur*
Im Generator verwenden wir sog. *mehrdeutige Substrukturen*:
Eine Substruktur $S = (V, E)$ eines Moleküls ist ein Graph bestehend aus einer Knotenmenge $V = \{v_1, ..., v_n\}$ und einer Kantenmenge $E = \{e_{i,j} \mid 1 \leq i < j \leq n\}$.
Jeder Knoten $v_k = \{at_{k,1}, ..., at_{k,\epsilon_k}\}$ $(k = 1, ..., n)$ besteht aus einer nichtleeren Menge von Atomtypen. Jede Kante $e_{i,j} \subseteq \bar{4}$, $e_{i,j} \neq \{\}$ $(1 \leq i < j \leq n)$ ist aufzufassen als Menge der möglichen Bindungsvielfachheiten von v_i nach v_j.
Zusätzlich muß gelten:

– $\forall i, j \ : \ e_{i,j} = \{0\}$ oder $e_{i,j} \subseteq \{1, 2, 3\}$.
– Für $k \in \{1, ..., n\}$ mit $\epsilon_k = 1$ und $at_{k,1}.na =$ „H" existiert ein $l \in \{1, ..., n\}, l \neq k$ mit $e_{k,l} = \{1\}$. Weiterhin existiert ein $u \in v_l$ mit $u.na \neq$ „H".

8.7.6 Beispiel:
Bild 3: *Eine mehrdeutige Substruktur*



Diese Substruktur enthält ein Atom, dessen Atomtyp Kohlenstoff **oder** Stickstoff sein kann. Weiterhin ist eine Bindung der Vielfachheit 1 **oder** 2 enthalten.

8.7.7 Definition: *Einbettung einer Substruktur*
Sei $S = (V, E)$ eine Substruktur mit $V = \{v_1, ..., v_m\}$ und $E = \{e_{i,j} \mid 1 \leq i < j \leq m\}$. Weiterhin sei $AM = \{\vartheta_1, ..., \vartheta_n\}$ und $M \in \bar{4}^{AM^{[2]}}$ ein molekularer Graph. AM enthalte genau $h \geq 0$ Wasserstoffatome und $Adj(M)$ sei die reduzierte Adjazenzmatrix von M. Wie üblich nehmen wir an, daß $\{\vartheta_1, ..., \vartheta_{n-h}\}$ Heteroatome sind und für $i = 1, ..., n - h$ bezeichne h_i die Anzahl der zu ϑ_i benachbarten Wasserstoffatome.
Unter einer *Einbettung* von S nach M verstehen wir eine Abbildung

$$f : V \longrightarrow \{-(n-h), ..., -1, 1, 2, ..., n-h\}$$

für die gilt:

– Falls $f(v_i) > 0$, dann existiert $\alpha \in v_i$ mit $\vartheta_{f(v_i)} = \alpha$.

– Falls $f(v_i) < 0$, dann existiert $\alpha \in v_i$ mit $\alpha.na$="H"und $h_{-f(v_i)} > 0$.
– $v_i, v_j \in V,\ v_i \neq v_j,\ f(v_i) > 0,\ f(v_j) > 0 \Longrightarrow f(v_i) \neq f(v_j)$.
– Für alle $i < j$ mit $e_{i,j} \subseteq \{1, 2, 3\}$ gilt:
  – Aus $f(v_i) < 0$ folgt $f(v_j) = -f(v_i)$ und $1 \in e_{i,j}$.
  – Aus $f(v_j) < 0$ folgt $f(v_i) = -f(v_j)$ und $1 \in e_{i,j}$.
  – Ist $f(v_i) > 0$ und $f(v_j) > 0$, so gilt:
  $f(v_i) \neq f(v_j)$ und $M(\{\vartheta_{f(v_i)}, \vartheta_{f(v_j)}\}) \in e_{i,j}$.
– $|\{v \in V \mid f(v) = -i\}| \leq h_i$ für alle $i \in \{1, ..., n-h\}$.

8.7.8 Definition:
Sei S eine Substruktur und M ein molekularer Graph. Wir bezeichnen die Menge aller möglichen Einbettungen von S nach M mit $\mathcal{E}_{S,M}$.

8.7.9 Definition:
Es gelten die Voraussetzungen von 8.7.7. Zusätzlich seien $f, g \in \mathcal{E}_{S,M}$.
$v_f = (v_{f,1}, ..., v_{f,n-h})$ und $v_g = (v_{g,1}, ..., v_{g,n-h})$ seien wie folgt definiert:

– $v_{f,i} := |\{v \in V \mid f(v) = -i\}|$ für alle $i \in \{1, ..., n-h\}$.
– $v_{g,i} := |\{v \in V \mid g(v) = -i\}|$ für alle $i \in \{1, ..., n-h\}$.

f und g heißen *wesentlich verschieden* (kurz: $f \neq g$), wenn gilt:

– $\{i \mid 1 \leq i \leq n-h \wedge \exists\, v \in V:\ f(v) = i\} \neq \{i \mid 1 \leq i \leq n-h \wedge \exists\, v \in V:\ g(v) = i\}$

oder

– $\exists\, 1 \leq i \leq n-h:\ v_{f,i} \neq v_{g,i}$.

8.7.10 Definition: *Vielfachheit einer Einbettung*
Es gelten die Voraussetzungen von 8.7.7 und es sei $f \in \mathcal{E}_{S,M}$ eine Einbettung von S nach M. Weiterhin sei v_f definiert wie in 8.7.9. Die *Vielfachheit* f_m von f ist dann definiert als

$$f_m := \prod_{i=1}^{n-h} \binom{h_i}{v_{f,i}}.$$

8.7.11 Definition: *Konkrete Einbettung*
Es gelten die Voraussetzungen von 8.7.7 und es sei $f \in \mathcal{E}_{S,M}$ eine Einbettung von S nach M. Unter einer *konkreten f-Einbettung* von S nach M verstehen wir eine Abbildung

$$\hat{f} : V \longrightarrow AM$$

mit:

– $\hat{f}(v_i) := \vartheta_{f(v_i)}$, falls $f(v_i) > 0$.
– $\hat{f}(v_i) := \vartheta_j$, mit $M(\{\vartheta_{-f(v_i)}, \vartheta_j\}) = 1$ und $j > n-h$, falls $f(v_i) < 0$.
– $\hat{f}(v_i) \neq \hat{f}(v_j)$ für $1 \leq i < j \leq n$.

Es gibt genau f_m verschiedene konkrete f-Einbettungen.

8.7.12 Definition: *Substrukturrestriktion*
Gegeben sei eine Substruktur S. Unter einer *Substrukturrestriktion* für S verstehen wir eine Abbildung

$$\mathcal{R}^{\text{sub}} : \bigcup_{M \text{ molek. Graph}} \mathcal{E}_{S,M} \longrightarrow \{\text{WAHR}, \text{FALSCH}\}.$$

8.7.13 Erlaubte Atomtypenkombinationen:
Sei $V = \{v_1, ..., v_m\}$ und $S = (V, E)$ eine Substruktur. Weiterhin sei $T \subseteq V$, TM eine nichtleere Menge von Atomtypen und I ein Intervall. Unter der *Einschränkung der Atomtypenkombinationen* von S zu den Parametern T, TM, I verstehen wir die Substrukturrestriktion

$$\mathcal{R}^{\text{sub}}_{\{T,TM,I\}} : \bigcup_{M \text{ molek. Graph}} \mathcal{E}_{S,M} \longrightarrow \{\text{WAHR}, \text{FALSCH}\}.$$

mit

$$\mathcal{R}^{\text{sub}}_{\{T,TM,I\}}(f) := \begin{cases} \text{WAHR, falls } (\alpha(T, TM, f) + \beta(T, TM, f)) \in I, \\ \text{FALSCH, sonst.} \end{cases}$$

wobei gelte:

- $f \in \mathcal{E}_{S,\mathbf{M}}$ mit $AM = \{\vartheta_1, ..., \vartheta_n\}$ und $\mathbf{M} \in \bar{4}^{AM^{[2]}}$.
- $\alpha(T, TM, f) := |\{t \in T \mid f(t) > 0 \wedge \exists\, \xi \in TM : \vartheta_{f(t)} = \xi\}|$.
- $\beta(T, TM, f) := |\{t \in T \mid f(t) < 0 \wedge \exists\, \xi \in TM : \xi.na = \text{„H}^{\alpha}\text{"}\}|$.

8.7.14 Erlaubte Kombinationen von Bindungen:
Sei $V = \{v_1, ..., v_m\}$ und $S = (V, E)$ eine Substruktur. Weiterhin sei $P \subseteq E$ eine Teilmenge der Bindungen mit $e \subseteq \{1, 2, 3\}$ für alle $e \in P$, I ein Intervall und $idx \in \{1, 2, 3\}$. Unter der *Einschränkung der Bindungskombinationen* von S zu den Parametern P, I, idx verstehen wir die Substrukturrestriktion

$$\mathcal{R}^{\text{sub}}_{\{P,I,idx\}} : \bigcup_{M \text{ molek. Graph}} \mathcal{E}_{S,M} \longrightarrow \{\text{WAHR}, \text{FALSCH}\}.$$

mit

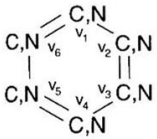
$$\mathcal{R}^{\text{sub}}_{\{P,I,idx\}}(f) := \begin{cases} \text{WAHR, falls } \alpha(P, idx, f) \in I, \\ \text{FALSCH, sonst.} \end{cases}$$

wobei gelte:

- $f \in \mathcal{E}_{S,\mathbf{M}}$ mit $AM = \{\vartheta_1, ..., \vartheta_n\}$ und $\mathbf{M} \in \bar{4}^{AM^{[2]}}$.
- $\alpha_1(P, idx, f) := \begin{cases} 0, \text{ falls } idx > 1, \\ |\{e_{i,j} \in P \mid f(v_i) < 0\}| + |\{e_{i,j} \in P \mid f(v_j) < 0\}|, \text{ sonst.} \end{cases}$
- $\alpha_2(P, idx, f) := |\{e_{i,j} \in P \mid f(v_i) > 0 \wedge f(v_j) > 0 \wedge \mathbf{M}(\vartheta_{f(v_i)}, \vartheta_{f(v_j)}) = idx\}|$.
- $\alpha(P, idx, f) := \alpha_1(P, idx, f) + \alpha_2(P, idx, f)$.

8.7.15 Beispiel:

Bild 4: *Eine Einschränkung von Atomtypenkombinationen*



$T := \{v_1, ..., v_6\}$.
$TM := \{(„C“, 4, \text{FALSCH}, 0)\}$.
$I := [2; 2]$.
$\mathcal{R}^{sub}_{\{T,TM,I\}}$ wäre eine Einschränkung der Atomtypenkombinationen, die bewirkt, daß S genau zwei Kohlenstoffatome enthalten muß.

Da es in der Praxis sehr häufig der Fall ist, daß dem Chemiker Informationen über die **Umgebung der Fragmente** eines Moleküls vorliegen, müssen wir gewährleisten, daß unser Generator alle vorliegenden Informationen verarbeiten kann, da diese die Anzahl der Lösungen drastisch reduzieren. Es wurde Wert darauf gelegt, diese Nebenbedingungen einer Substruktur so allgemein zu formulieren, daß der Generator eine echte Obermenge der momentan benötigten Restriktionen verarbeiten kann, so daß zukünftige Anforderungen eventuell von vornherein abgedeckt werden.
Wenden wir uns zunächst den *Abständen* innerhalb einer Substruktur zu:

8.7.16 Abstände zwischen Atomen:
Sei $V = \{v_1, ..., v_m\}$ und $S = (V, E)$ eine Substruktur. Weiterhin seien $T_1 \subseteq V$ und $T_2 \subseteq V$ zwei Teilmengen der Knotenmenge von S und I ein Intervall. Unter der *Vorgabe von Abständen* für S zu den Parametern T_1, T_2, I verstehen wir die Substrukturrestriktion

$$\mathcal{R}^{\text{sub}}_{\{T_1,T_2,I\}} : \bigcup_{M \text{ molek. Graph}} \mathcal{E}_{S,M} \longrightarrow \{\text{WAHR}, \text{FALSCH}\}$$

mit

$$\mathcal{R}^{\text{sub}}_{\{T_1,T_2,I\}}(f) := \begin{cases} \text{WAHR, falls } \text{dist}_f(T_1, T_2) \in I, \\ \text{FALSCH, sonst.} \end{cases}$$

wobei gelte:

- $f \in \mathcal{E}_{S,\mathbf{M}}$ mit $AM = \{\vartheta_1, ..., \vartheta_n\}$ und $\mathbf{M} \in \bar{4}^{AM[2]}$.
- $\hat{f}$ sei eine konkrete f-Einbettung von S nach $\mathbf{M}$.
- $\text{dist}_f(T_1, T_2) := dist_{\mathbf{M}}(\{\hat{f}(v) \mid v \in T_1\}, \{\hat{f}(v) \mid v \in T_2\})$.

8.7.17 Enthaltensein in Cycles:
Sei $V = \{v_1, ..., v_m\}$, $S = (V, E)$ eine Substruktur und $T \subseteq V$. Des weiteren wird ein Intervall I sowie ein boolescher Wert b vorgegeben. Unter der *Cyclebedingung* für S zu den Parametern T, I, b verstehen wir die Substrukturrestriktion

$$\mathcal{R}^{\text{sub}}_{\{T,I,b\}} : \bigcup_{M \text{ molek. Graph}} \mathcal{E}_{S,M} \longrightarrow \{\text{WAHR}, \text{FALSCH}\}.$$

Dabei gelte:

- $f \in \mathcal{E}_{S,\mathbf{M}}$ mit $AM = \{\vartheta_1, ..., \vartheta_n\}$ und $\mathbf{M} \in \bar{4}^{AM[2]}$.
- $\hat{f}$ sei eine konkrete f-Einbettung von S nach $\mathbf{M}$.

– $KM := \{\hat{f}(v) \mid v \in T\}$.

Dann sei

$$\mathcal{R}^{\text{sub}}_{\{T,I,b\}}(f) := \begin{cases} \text{WAHR, falls } b = \text{WAHR und es in } \mathbf{M} \text{ einen Cycle } C \\ \qquad \text{der Länge } l \in I \text{ gibt, der } KM \text{ enthält,} \\ \qquad \text{oder } b = \text{FALSCH und es in } \mathbf{M} \textbf{ keinen} \text{ Cycle } C \\ \qquad \text{der Länge } l \in I \text{ gibt, der } KM \text{ enthält.} \\ \text{FALSCH, sonst.} \end{cases}$$

8.7.18 Hybridisierungen:
Sei $V = \{v_1, ..., v_m\}$, $S = (V, E)$ eine Substruktur und $T \subseteq V$. Des weiteren wird ein Intervall I sowie eine Menge $\Phi = \{H_1, ..., H_j\}$ von Hybridisierungen vorgegeben. Unter der *Vorgabe von Hybridisierungen* von S zu den Parametern T, Φ, I verstehen wir die Substrukturrestriktion

$$\mathcal{R}^{\text{sub}}_{\{T,\Phi,I\}} : \bigcup_{M \text{ molek. Graph}} \mathcal{E}_{S,M} \longrightarrow \{\text{WAHR}, \text{FALSCH}\}.$$

Dabei gelte:

– $f \in \mathcal{E}_{S,\mathbf{M}}$ mit $AM = \{\vartheta_1, ..., \vartheta_n\}$ und $\mathbf{M} \in 4^{AM[2]}$.
– $\hat{f}$ sei eine konkrete f-Einbettung von S nach $\mathbf{M}$.

Dann sei

$$\mathcal{R}^{\text{sub}}_{\{T,\Phi,I\}}(f) := \begin{cases} \text{WAHR, falls } |\{v \in T : Hyb(\mathbf{M}, \hat{f}(v)) \in \Phi\}| \in I. \\ \text{FALSCH, sonst.} \end{cases}$$

8.7.19 Benachbarte Atome:
Sei $V = \{v_1, ..., v_m\}$, $S = (V, E)$ eine Substruktur und $T \subseteq V$. Gegeben sei weiterhin:

– Ein Intervall $I_d = [id_1; id_2]$ mit $id_1 > 0$.
– Ein Intervall I_a.
– Eine Menge $\mathcal{AT} = \{at_1, ..., at_j\}$ von Atomtypen ($j \geq 1$).
– Eine Menge $\mathcal{EH} = \{h_1, ..., h_k\}$ von Hybridisierungen ($k \geq 0$).
– Eine Menge $\mathcal{DT} = \{dt_1, ..., dt_l\}$ von Atomtypen ($l \geq 1$).

Unter der *Vorgabe benachbarter Atome* von S zu den Parametern $T, I_d, I_a, \mathcal{AT}, \mathcal{EH}, \mathcal{DT}$ verstehen wir die Substrukturrestriktion

$$\mathcal{R}^{\text{sub}}_{\{T,I_d,I_a,\mathcal{AT},\mathcal{EH},\mathcal{DT}\}} : \bigcup_{M \text{ molek. Graph}} \mathcal{E}_{S,M} \longrightarrow \{\text{WAHR}, \text{FALSCH}\}.$$

Dabei gelte:

– $f \in \mathcal{E}_{S,\mathbf{M}}$ mit $AM = \{\vartheta_1, ..., \vartheta_n\}$ und $\mathbf{M} \in 4^{AM[2]}$.
– $\hat{f}$ sei eine konkrete f-Einbettung von S nach $\mathbf{M}$.

Die Menge $EA \subseteq AM$ enthalte alle Atome α mit den folgenden Eigenschaften:

– $dist_{\mathbf{M}}(\{\hat{f}(v) \mid v \in T\}, \{\alpha\}) \in I_d$,
– $\alpha.AT \in \mathcal{AT}$,

– Falls $k > 0$, dann gilt $Hyb(\mathbf{M}, \alpha) \in \mathcal{EH}$.
– Falls $l > 0$, dann existiert ein $\tilde{\alpha} \in AM$ mit
  – $dist_{\mathbf{M}}(\{\hat{f}(v) \mid v \in T\}, \{\tilde{\alpha}\}) = dist_{\mathbf{M}}(\{\hat{f}(v) \mid v \in T\}, \{\alpha\}) - 1$,
  – $\mathbf{M}(\{\alpha, \tilde{\alpha}\}) > 0$,
  – $\tilde{\alpha}.AT \in \mathcal{DT}$.

Dann sei

$$\mathcal{R}^{\text{sub}}_{\{T,I_a,I_d,\mathcal{AT},\mathcal{EH},\mathcal{DT}\}}(f) := \begin{cases} \text{WAHR, falls } |EA| \in I_a, \\ \text{FALSCH, sonst.} \end{cases}$$

8.7.20 Bindungsvielfachheiten:
Sei $V = \{v_1, ..., v_m\}$, $S = (V, E)$ eine Substruktur und $T \subseteq V$. Außerdem sei ein Intervall I und eine Menge $\mathcal{B} \subseteq \{1, 2, 3\}$, $|\mathcal{B}| > 0$ von Bindungsvielfachheiten gegeben.
Unter der Vorgabe von *Bindungsvielfachheiten* von S zu den Parametern $T, I, \mathcal{B}$ verstehen wir die Substrukturrestriktion

$$\mathcal{R}^{\text{sub}}_{\{T,I,\mathcal{B}\}} : \bigcup_{M \text{ molek. Graph}} \mathcal{E}_{S,M} \longrightarrow \{\text{WAHR, FALSCH}\}.$$

Dabei gelte:

– $f \in \mathcal{E}_{S,\mathbf{M}}$ mit $AM = \{\vartheta_1, ..., \vartheta_n\}$ und $\mathbf{M} \in \bar{4}^{AM^{[2]}}$.
– $\hat{f}$ sei eine konkrete f-Einbettung von S nach $\mathbf{M}$.

Dann sei

$$\mathcal{R}^{\text{sub}}_{\{T,I,\mathcal{B}\}}(f) := \begin{cases} \text{WAHR, falls } |\{k \in AM^{[2]} \mid \mathbf{M}(k) \in \mathcal{B} \wedge \exists\, v \in T : \hat{f}(v) \in k\}| \in I, \\ \text{FALSCH, sonst.} \end{cases}$$

8.7.21 Definition: *Substrukturtypen*

– Eine Substruktur S, die Restriktionen vom Typ 8.7.13-8.7.20 tragen *kann*, heißt **Substruktur vom Typ I**.
– Eine Substruktur S, die Restriktionen vom Typ 8.7.13-8.7.20 sowie vom Typ 8.7.22 und 8.7.23 tragen *kann*, heißt **Substruktur vom Typ II**.

8.7.22 Benachbarte Substrukturen:
Sei $V = \{v_1, ..., v_m\}$, $S = (V, E)$ eine Substruktur vom Typ II und $T \subseteq V$. Gegeben sei weiterhin:

– Ein Intervall I_a.
– Ein Intervall I_d.
– Eine Menge $\mathcal{SS} = \{str_1, ..., str_j\}$ von Substrukturen vom Typ I ($j > 0$).
– Eine Menge $\mathcal{B} \subseteq \{1, 2, 3\}$, $|\mathcal{B}| > 0$ von Bindungsvielfachheiten.

Unter der Vorgabe *benachbarter Substrukturen* von S zu den Parametern $T, I_a, I_d, \mathcal{SS}, \mathcal{B}$ verstehen wir die Substrukturrestriktion

$$\mathcal{R}^{\text{sub}}_{\{T,I_a,I_d,\mathcal{SS},\mathcal{B}\}} : \bigcup_{M \text{ molek. Graph}} \mathcal{E}_{S,M} \longrightarrow \{\text{WAHR, FALSCH}\}.$$

Dabei gelte:

– $f \in \mathcal{E}_{S,\mathbf{M}}$ mit $AM = \{\vartheta_1, ..., \vartheta_n\}$ und $\mathbf{M} \in \bar{4}^{AM^{[2]}}$.

- $\hat{f}$ sei eine konkrete f-Einbettung von S nach $\mathbf{M}$.

Für $i = 1, ..., j$ enthalte die Menge

$$\Delta \subseteq \bigcup_{str \in \mathcal{SS}} \mathcal{E}_{str,\mathbf{M}}$$

das Element $g \in \mathcal{E}_{str_i,\mathbf{M}}$ genau dann, wenn gilt:

- Für jede konkrete g-Einbettung $\hat{g}$ von $str_i = (V_i, E_i)$ nach $\mathbf{M}$ ist $dist_{\mathbf{M}}(\{\hat{f}(v) \mid v \in V\}, \{\hat{g}(v) \mid v \in V_i\}) \in I_d$.
- Es existiert ein Pfad $P = (p_1, ..., p_{\alpha+1})$ mit
  - $\alpha := dist_{\mathbf{M}}(\{\hat{f}(v) \mid v \in V\}, \{\hat{g}(v) \mid v \in V_i\})$.
  - $p_1 \in \{\hat{f}(v) \mid v \in V\}$.
  - $p_{\alpha+1} \in \{\hat{g}(v) \mid v \in V_i\}$.
  - $\mathbf{M}(\{p_u, p_{u+1}\}) \in \mathcal{B}$ für alle $u = 1, ..., \alpha$.

Dann sei

$$\mathcal{R}^{\text{sub}}_{\{T, I_a, I_d, \mathcal{SS}, \mathcal{B}\}}(f) := \begin{cases} \text{WAHR, falls } \sum_{g \in \Delta} g_m \in I_a, \\ \text{FALSCH, sonst.} \end{cases}$$

8.7.23 Branches:
Sei $V = \{v_1, ..., v_m\}$, $S = (V, E)$ eine Substruktur vom Typ II und $T \subseteq V$. Gegeben sei weiterhin:

- Ein Intervall I_a.
- Ein Intervall I_d.
- Eine Menge $\mathcal{BR} = \{br_1, ..., br_j\}$ von Bruttoformeln ($j > 0$).
- Eine Menge $\mathcal{B} \subseteq \{1, 2, 3\}$, $|\mathcal{B}| > 0$ von Bindungsvielfachheiten.

Unter der Vorgabe *benachbarter Branches* von S zu den Parametern $T, I_a, I_d, \mathcal{BR}, \mathcal{B}$ verstehen wir die Substrukturrestriktion

$$\mathcal{R}^{\text{sub}}_{\{T, I_a, I_d, \mathcal{BR}, \mathcal{B}\}} : \bigcup_{M \text{ molek. Graph}} \mathcal{E}_{S,M} \longrightarrow \{\text{WAHR}, \text{FALSCH}\}.$$

Dabei gelte:

- $f \in \mathcal{E}_{S,\mathbf{M}}$ mit $AM = \{\vartheta_1, ..., \vartheta_n\}$ und $\mathbf{M} \in \bar{4}^{AM[2]}$.
- $\hat{f}$ sei eine konkrete f-Einbettung von S nach $\mathbf{M}$.

Die Menge Δ enthalte den zusammenhängenden Teilgraphen γ von $\mathbf{M}$ genau dann, wenn gilt:

- γ besitzt eine Bruttoformel aus $\mathcal{BR}$.
- $dist_{\mathbf{M}}(\{\hat{f}(v) \mid v \in V\}, \gamma) \in I_d$.
- Es existiert ein Pfad $P = (p_1, ..., p_{\alpha+1})$ mit
  - $\alpha := dist_{\mathbf{M}}(\{\hat{f}(v) \mid v \in V\}, \gamma)$.
  - $p_1 \in \{\hat{f}(v) \mid v \in V\}$.
  - $p_{\alpha+1} \in \gamma$.
  - $\mathbf{M}(\{p_u, p_{u+1}\}) \in \mathcal{B}$ für alle $u = 1, ..., \alpha$.

Dann sei

$$\mathcal{R}^{\text{sub}}_{\{T, I_a, I_d, \mathcal{BR}, \mathcal{B}\}}(f) := \begin{cases} \text{WAHR, falls } |\Delta| \in I_a, \\ \text{FALSCH, sonst.} \end{cases}$$

8.7.24 Makroatome. Gegeben seien die Substrukturen $M_1 = (V_1, E_1), ..., M_t = (V_t, E_t)$ (vom Typ II). Desweiteren seien $\mathcal{R}_1, ..., \mathcal{R}_t$ Mengen von Substrukturrestriktionen. Unter der Vorgabe der *Makroatome* $(M_1, \mathcal{R}_1), ..., (M_t, \mathcal{R}_t)$ verstehen wir die Restriktion

$$\mathcal{R}_{\{(M_1,\mathcal{R}_1),...,(M_t,\mathcal{R}_t)\}} : \bar{4}^{(\bar{2}^{AM^{[2]}})} \longrightarrow \{\text{WAHR}, \text{FALSCH}\}.$$

Dabei gelte:

– $AM = \{\vartheta_1, ..., \vartheta_n\}$ und $\mathbf{M} \in \bar{4}^{(\bar{2}^{AM^{[2]}})}$.

Die Menge $\Delta \subseteq \mathcal{E}_{M_1,\mathbf{M}} \times ... \times \mathcal{E}_{M_t,\mathbf{M}}$ enthalte $(f_1, ..., f_t) \in \mathcal{E}_{M_1,\mathbf{M}} \times ... \times \mathcal{E}_{M_t,\mathbf{M}}$ genau dann, wenn gilt:

– Für alle $i = 1, ..., t$: f_i erfüllt alle Substrukturrestriktionen aus $\mathcal{R}_i$.
– Es existiert $(\hat{f}_1, ..., \hat{f}_t)$, wobei $\hat{f}_i$ eine konkrete f_i-Einbettung von M_i nach $\mathbf{M}$ ist, so daß für $1 \leq j < k \leq t$ gilt: $\{\hat{f}_j(v) \mid v \in V_j\} \cap \{\hat{f}_k(v) \mid v \in V_k\} = \emptyset$.

Dann sei

$$\mathcal{R}_{\{(M_1,\mathcal{R}_1),...,(M_t,\mathcal{R}_t)\}}(\mathbf{M}) := \begin{cases} \text{WAHR}, & \text{falls } |\Delta| > 0, \\ \text{FALSCH}, & \text{sonst.} \end{cases}$$

8.7.25 Substrukturvorgabe. Gegeben seien die Substrukturen $M_1 = (V_1, E_1), ..., M_t = (V_t, E_t)$ (vom Typ II). Desweiteren seien $\mathcal{R}_1, ..., \mathcal{R}_t$ Mengen von Substrukturrestriktionen. Unter der *Substrukturvorgabe* zu den Parametern
$\{(M_1, \mathcal{R}_1, v_1, b_1), ..., (M_t, \mathcal{R}_t, v_t, b_t), v, b\}, v_i, b_i, v, b \in \mathbb{N}$,
$v_i \leq b_i$ für alle $i = 1, ..., t$, $v \leq b$ verstehen wir die Restriktion

$$\mathcal{R}_{\{(M_1,\mathcal{R}_1,v_1,b_1),...,(M_t,\mathcal{R}_t,v_t,b_t),v,b\}} : \bar{4}^{(\bar{2}^{AM^{[2]}})} \longrightarrow \{\text{WAHR}, \text{FALSCH}\}.$$

Dabei gelte:

– $AM = \{\vartheta_1, ..., \vartheta_n\}$ und $\mathbf{M} \in \bar{4}^{(\bar{2}^{AM^{[2]}})}$.

Die Menge $\Delta \subseteq \{1, ..., t\}$ enthalte $i \in \{1, ..., t\}$ genau dann, wenn gilt:

$$v_i \leq \sum_{f \in \mathcal{E}_{M_i,\mathbf{M}}:\, f \text{ erfüllt } \mathcal{R}_i} f_m \leq b_i.$$

Dann sei

$$\mathcal{R}_{\{(M_1,\mathcal{R}_1,v_1,b_1),...,(M_t,\mathcal{R}_t,v_t,b_t),v,b\}}(\mathbf{M}) := \begin{cases} \text{WAHR}, & \text{falls } v \leq |\Delta| \leq b, \\ \text{FALSCH}, & \text{sonst.} \end{cases}$$

9 Testen der Restriktionen

9.1 Vorberechnungen

9.1.1 Goodlisteinträge als Makroatome. 9.1.2 Definition: *Eindeutige Substruktur* Die Substruktur $S = (V, E)$ mit $V = \{v_1, ..., v_n\}$ und $E = \{e_{i,j} \mid 1 \leq i < j \leq n\}$ heißt *eindeutig*, falls gilt:

– $\forall k = 1, ..., n: \ v_k = \{at_{k,1}\}$,
d.h. für jeden Knoten ist nur **ein** Atomtyp möglich.
– $\forall e \in E: \ |e| = 1$,
d.h. die Kantenvielfachheiten sind eindeutig.

9.1.3 Definition: *Goodlist*
Unter einem *Goodlisteintrag* verstehen wir eine Substrukturvorgabe zu den Parametern $\{(M, \mathcal{R}, v_1, b_1), 1, 1\}$, wobei M eine Substruktur (vom Typ II) und $\mathcal{R}$ eine Menge von Substrukturrestriktionen ist. Zusätzlich muß $v_1 \geq 1$ gelten, d.h. die Substruktur M muß im gesuchten molekularen Graphen mindestens einmal enthalten sein.
Eine *Goodlist* besteht aus einer Menge von Goodlisteinträgen.
Makroatome sind grob gesprochen Goodlisteinträge, die sich gegenseitig nicht überlappen dürfen. Im Fall der *ordnungstreuen Erzeugung* werden Makroatome nicht anders behandelt als gewöhnliche Goodlisteinträge.
Im Fall der *zielgerichteten Erzeugung* jedoch ist es offenbar möglich, bei der Generierung der gesuchten Moleküle die Adjazenzmatrix von vornherein mit einem bestimmten Goodlisteintrag bzw. mit den Makroatomen vorzubesetzen. Mit zwei (oder mehr) beliebigen Goodlisteinträgen ist dies i.a. schon nicht mehr möglich, da diese sich gegenseitig überlappen könnten. Jedoch möchten wir gerne Situationen entdecken, in denen sich Goodlisteinträge (aufgrund ihrer Atomtypen, Bindungsverhältnisse oder Substrukturrestriktionen) nicht gegenseitig überlappen können, so daß sie als Makroatome betrachtet werden können. Unser Ziel ist es, möglichst viele Plätze der Adjazenzmatrix durch Goodlisteinträge oder Makroatome vorzubesetzen.

9.1.4 Prozedur: *Nachbarschaftsvergleich*
Gegeben seien die Substrukturen S_1, S_2 sowie $a_1 \in S_1$ und $a_2 \in S_2$.

(1) **Für** alle „eindeutigen" Nachbarschaften von a_1 in S_1
(2) **Für** alle „eindeutigen" Nachbarschaften von a_2 in S_2
(3) best := WAHR.
(4) **Falls** a_1 mit α_1 Atomen vom Atomtyp t in S_1 v-fach verbunden ist
(5) Berechne die Anzahl α_2 der Atome vom Atomtyp t, die mit a_2 in S_2 v-fach verbunden sind.
(6) **Falls** a_2 weniger als $v * (\alpha_1 - \alpha_2)$ freie Valenzen hat, **tue** best := FALSCH.
ende
(7) **Falls** best=WAHR, so return(WAHR).
ende
ende
(8) return(FALSCH).

Da in einer Substruktur für Atomtypen und Bindungsvielfachheiten Alternativen möglich sind, durchlaufen wir in den Zeilen (1) und (2) der obigen Prozedur zunächst alle Möglichkeiten für die Atomtypen der zu a_1 (bzw. a_2) benachbarten Atome sowie der zugehörigen Bindungen.

9.1.5 Prozedur: *Umgebungsvergleich*
Die folgende Prozedur prüft, ob zwei Atome aus verschiedenen Substrukturen miteinander identifiziert werden können, indem die Atomumgebungen verglichen werden.
Gegeben seien die Substrukturen S_1, S_2 sowie $a_1 \in S_1$ und $a_2 \in S_2$.

(1) **Falls** für a_1 und a_2 kein gleicher Atomtyp möglich ist, return(FALSCH).
(2) **Falls** der Nachbarschaftsvergleich von a_1 und a_2 (9.1.4) FALSCH liefert,
return(FALSCH).
(3) t sei der maximale Wert $\in [0;3]$, so daß für $i = 1,2$ alle Atome $A_i \in S_i$ mit
$dist_{S_i}(A_i, a_i) < t$ keine freien Valenzen mehr haben.
(4) Identifiziere $a_1 \in S_1$ mit $a_2 \in S_2$.
(5) **Durchlaufe** *breadth-first* von a_1 ausgehend alle Atome aus S_1, die von a_1
Abstand $\leq t$ haben.
(6) **Falls** alle zu durchlaufenden Atome aus S_1 identifiziert sind,
return(WAHR).
(7) Ω bezeichne die Menge der Atome aus S_2 mit denen
bereits ein Atom aus S_1 identifiziert wurde.
(8) **Falls** bereits α Atome aus S_1 mit Atomen aus S_2 identifiziert sind und
nun $\vartheta \in S_1$ identifiziert werden soll, **tue**
(9) **Durchlaufe** alle Atome $\tilde{\vartheta}$ aus S_2,
- mit denen noch kein Atom aus S_1 identifiziert wurde,
- die einen gleichen Atomtyp wie ϑ haben können,
- die den Nachbarschaftsvergleich mit ϑ bestehen und
- deren Bindungen zu den Atomen aus Ω verträglich sind zu den
Bindungen von ϑ zu den bereits identifizierten Atomen.
(10) Identifiziere ϑ mit $\tilde{\vartheta}$ und besuche das in der
breadth-first Reihenfolge nächste Atom.
ende
(11) Gehe in der breadth-first Reihenfolge eine Schritt zurück.
ende
ende
(12) return(FALSCH).

9.1.6 Prozedur: *Überlappungstest*

Gegeben seien zwei Substrukturen S_1 und S_2. Es soll getestet werden, ob sich S_1 und S_2 überlappen können.

(1) **Für** alle Atome $s_1 \in S_1$ **tue**
(2) **Für** alle Atome $s_2 \in S_2$ **tue**
(3) **Falls** der Umgebungsvergleich 9.1.5 von $s_1 \in S_1$ mit $s_2 \in S_2$
WAHR liefert, so return(WAHR).
ende
ende
(4) return(FALSCH).

9.1.7 Algorithmus: *Berechnung eindeutiger Substrukturen*

Gegeben sei eine (mehrdeutige) Substruktur $S = (V, E)$, wobei $V = \{v_1, ..., v_n\}$ mit $v_k = \{at_{k,1}, ..., at_{k,\iota_k}\}$ und $E = \{e_{i,j} \mid 1 \leq i < j \leq n\}$ mit $e_{i,j} \subseteq \bar{4}$. Es sollen alle sich aus S ergebenden *eindeutigen* Substrukturen $\tilde{S}$ berechnet werden:

(1) **Für** alle möglichen Auswahlen von Atomtypen **tue**
(2) **Falls** diese Atomtypenkombination erlaubt ist, **tue**
(3) **Für** alle möglichen Auswahlen von Kantenvielfachheiten **tue**

(4) **Falls** diese Auswahl von Kantenvielfachheiten erlaubt ist und falls kein Widerspruch zu den Valenzen der aktuellen Atomtypauswahl besteht **tue**

(5) Füge die aktuell repräsentierte eindeutige Substruktur zur Lösungsmenge hinzu.

ende

ende

ende

ende

Eine Auswahl von Atomtypen entspricht einem Typel $(a_1, ..., a_n) \in \underline{\epsilon_1} \times ... \times \underline{\epsilon_n}$. Wenn für *S Einschränkungen der Atomtypenkombinationen* vorliegen, so vergleichen wir mit diesen Einschränkungen. Ebenso werden alle möglichen Kombinationen von Kantenvielfachheiten erstellt und mit etwaigen *Einschränkungen der Bindungskombinationen* verglichen.

9.1.8 Algorithmus: *Goodlist/Makro Austausch*
L_m bezeichne die Liste der tatsächlich verwendeten Makroatome. Initialisiere L_m mit den vorgegebenen Makroatomen.

(1) **Durchlaufe** die Liste der Goodlststrukturen nach absteigender Summe der Bindungsvielfachheiten mit G

(2) **Falls** sich G mit keinem Element von L_m überschneiden (9.1.6) kann **tue**

(3) **Falls** die Anzahl der sich aus G ergebenden eindeutigen Makroatome (9.1.7) eine vorgegebene Grenze (z.B. 100) nicht übersteigt, **tue** $L_m := L_m \cup G$.

ende

ende

ende

(4) Entferne überflüssige Einträge aus der Goodlist.

9.1.9 Bemerkung:
Von Thomas Wielandt [52] wurde vorgeschlagen, zuerst **alle** nichtisomorphen Überlappungen der Goodlisteinträge zu berechnen, und dann mit jeder Makrostruktur einen gesonderten Generatorlauf durchzuführen. Davon wurde bei dieser Implementation aus folgenden Gründen kein Gebrauch gemacht: Da es erlaubt ist, mehrdeutige Substrukturen (d.h. mehrere Alternativen für Atomtypen, Bindungsvielfachheiten, ...) mit vielen weiteren Restriktionen vorzugeben, wird die Berechnung aller möglichen Überlappungen schwieriger. Sind mehrere Goodlisteinträge vorgeschrieben, so kann es leicht passieren, daß es eine sehr große Anzahl von Überlappungen gibt. Außerdem können die zu verschiedenen Makroatomen gefundenen Lösungen leider isomorph sein.

9.1.10 Atomtypen- und H-Verteilung. Vor dem eigentlichen Konstruktionsprozeß müssen die Atomnamen, die Atomtypen sowie die Wasserstoffe verteilt werden. Wir brauchen also einen Algorithmus, der alle möglichen „Beschriftungen der Adjazenzmatrix" erzeugt. Natürlich gehen wir wieder depth-first vor, verzichten aber aus Gründen der Übersichtlichkeit auf Laufvariable:

9.1.11 Algorithmus: *H-Verteilung*

(1) Festlegung der *Atomnamenverteilung*.

(Berücksichtige Einschränkungen für die *Gesamtatomanzahl*)
(2) Vergleiche mit *Beschränkungen der Atomnamenverteilung.*
(3) Lege die *Atomtypenverteilung* fest.
(4) Vergleiche mit der *Gesamtkantenzahl.*
(5) Vergleiche mit *Beschränkungen der Atomtypenverteilung.*
(6) Die aktuelle Atomtypenverteilung liefert uns eine Gradpartition $d = (d_1, ..., d_p) \vdash 2k$.
(7) **Prüfe** ob
(8) - $d_1 \leq k$ und
(9) - $k \geq p - 1$.
(10) Falls die maximale Bindungsvielfachheit 1 ist, rufe 1.2.8.
(11) Vergleiche mit *Gesamtladung, Anzahl geladener Atome* sowie *Anzahl von Radikalstellen.*
(12) **Falls** die aktuelle Atomnamenverteilung Wasserstoffe enthält, **tue**
(13) Verteile die H-Anzahlen auf die Atomtypen. Berücksichtige dabei Vorgaben für die *H-Anzahl pro Atomtyp.*
(14) Vergleiche mit Vorgaben für die *H-Anzahl pro Atomname.*
(15) Verteile die Anzahlen benachbarter H-Atome auf den Atomtypen. Berücksichtige dabei Vorgaben für die *H-Verteilung pro Atomtyp.*
(16) Vergleiche mit Vorgaben für die *H-Verteilung pro Atomname.*
(17) Teste, ob die um die Wasserstoffe reduzierte Partition gültig ist.
(18) **ende**

9.1.12 GoodOr-Einträge als Makroatome. Ist als Konstruktionsstrategie die *zielgerichtete Generierung* gewählt, dann möchte man die Adjazenzmatrix mit einem möglichst großen Goodlisteintrag vorbelegen, weil dadurch die Konstruktionsgeschwindigkeit i.a. ansteigt. In 9.1.1 haben wir bereits versucht, aus den vorgegebenen Goodlisteinträgen eine möglichst große Substruktur, die in den zu konstruierenden molekularen Graphen enthalten sein muß, zu konstruieren. Für den für die Praxis relevanten Spezialfall, daß eine *Substrukturvorgabe* in der Form

$$\{(S_1, \mathcal{R}_1, v_1, b_1), ..., (S_n, \mathcal{R}_n, v_n, b_n), 1, n\}$$

mit $v_i \geq 1$ für alle $i = 1, ..., n$ vorliegt, gehen wir einen Schritt weiter. Eine Substrukturvorgabe dieser Art bedeutet praktisch, daß aus einer vorgegebenen Menge $S = \{S_1, ..., S_n\}$ von Goodliststrukturen mindestens eine Struktur im molekularen Graphen enthalten sein muß. Haben wir bereits die mehrdeutigen Makroatome $M = \{M_1, ..., M_l\}$, dann testen wir für alle $k = 1, ..., n$, ob S_k in M aufgenommen werden kann, d.h. ob sich S_k mit einem Element aus M überlappen kann. Ist für alle $k = 1, ..., n$ keine Überlappung möglich, so kann man anstatt eines Generatorlaufs mit den Makroatomen aus M n Generatorläufe mit jeweils einem zusätzlichen Makroatom durchführen. Um einen Zeitvorteil zu erhalten, sollten wir nur so vorgehen, falls n nicht **zu** groß und $min := min\{|S_1|, ..., |S_n|\}$ nicht **zu** klein ist.
Sind mehrere mit ODER verknüpfte Mengen von Goodliststrukturen gegeben, so wenden wir das obige Vorgehen zunächst nur einmal an, damit sich die Anzahl der notwendigen Generatorläufe nicht multipliziert. Zweckmäßigerweise wählen wir die erfolgversprechendste Menge S aus (n möglichst klein, $min\{|S_1|, ..., |S_n|\}$ möglichst groß).

9.1.13 Erzeugung der eindeutigen Makroatome. Die eindeutigen Makroatome werden nach 9.1.7 erzeugt.

9.1.14 Beschriftung der Adjazenzmatrix. Falls Makroatome vorgegeben wurden, dann verwenden wir automatisch die zielgerichtete Generierung und es müssen zunächst auf alle verschiedenen Weisen evtl. vorhandene Wasserstoffe auf die Atome der Makroatome verteilt werden. Die Wasserstoffverteilung liefert uns einen Verteilungsvektor

$$v = \{(at_1, h_1, v_1), ..., (at_n, h_n, v_n)\}.$$

Dabei bedeutet (at_i, h_i, v_i), daß die im folgenden zu konstruierenden molekularen Graphen genau v_i Atome vom Atomtyp at_i enthalten sollen, die jeweils zu genau h_i Wasserstoffen benachbart sind.
Wir notieren uns für jedes Heteroatom der Makroatome die minimal und maximal mögliche Zahl benachbarter Wasserstoffe. In bewährter depth-first Reihenfolge können wir nun leicht alle möglichen Zuweisungen von Wasserstoffanzahlen auf die Heteroatome der Makroatome bestimmen. Ist eine Zuordnung gefunden, so erhalten wir auch den zugehörigen reduzierten Verteilungsvektor

$$\tilde{v} = \{(at_1, h_1, \tilde{v}_1), ..., (at_n, h_n, \tilde{v}_n)\}.$$

$\tilde{v}$ berechnet sich, indem man von v die den Makroatomen zugewiesenen Elemente abzieht. Nun beschriften wir die Zeilen der Adjazenzmatrix, indem wir den Heteroatomen der Makroatome die niedrigsten Nummern zuordnen und die verbleibenden Atomtypen aus $\tilde{v}$ auf die restlichen Zeilen verteilen.

Werden keine Makroatome verwendet, so gibt es nur eine Beschriftung der Adjazenzmatrix. Wir beschriften die Zeilen der Adjazenzmatrix so, daß die Atome aus v blockweise (d.h. Atome gleichen Atomtyps mit gleicher H-Anzahl folgen aufeinander) angeordnet werden. Wir wählen lediglich eine günstig erscheinende Reihenfolge dieser Blöcke:

- Die Atome mit restlicher Valenz 1 erhalten die höchsten Nummern.
  Dies wirkt sich bei der ordnungstreuen Erzeugung positiv aus, weil wir nur zusammenhängende Moleküle konstruieren wollen. Ist das Molekülanfangsstück bestehend aus den Atomen mit restlicher Valenz > 1 zusammenhängend, so können die Atome mit restlicher Valenz 1 auf jeden Fall hinzugefügt werden, so daß ein zusammenhängender Graph entsteht. (Bei einer anderen Anordnung kann man beträchtliche Schwierigkeiten bekommen.)
- Die verbleibenden Atome werden so sortiert, daß diejenigen Atomtypen, für die viele Good- oder Badlisteinträge existieren, möglichst niedrige Nummern bekommen. Im Fall der ordnungstreuen Erzeugung soll dadurch erreicht werden, daß Good- und Badlistrestriktionen frühzeitig getestet werden können.

9.1.15 Isomorphietest der Makroatome. Durch die Verwendung mehrdeutiger Substrukturen kann es vorkommen, daß das gleiche eindeutige Makroatom mehrmals erzeugt wird. Also ist es sinnvoll, an dieser Stelle mittels eines Isomorphietests doppelte Arbeit zu vermeiden. Dies tun wir allerdings nur dann, wenn das Makroatom keine Substrukturrestriktionen besitzt, weil die Situation andernfalls erheblich schwieriger ist.

9.1.16 Hybridisierungen festlegen. Bisher haben wir die Zeilen der Adjazenzmatrix mit den Atomtypen sowie den Anzahlen der benachbarten Wasserstoffe beschriftet. Würden Vorgaben für die Gesamthybridisierung der zu konstruierenden Moleküle gemacht, dann können wir die Beschriftung der Adjazenzmatrix evtl. verfeinern.
Was unter einer *Vorgabe der Gesamthybridisierung* zu verstehen ist, wurde bereits definiert. Grob gesprochen wird zu einem Atomnamen N und einer Hybridisierung H ein Intervall $I = [min; max]$ für die Anzahl der Atome des molekularen Graphen mit Namen N und Hybridisierung H vorgeschrieben. Ist $min > 0$, so kann man von vornherein festlegen, welche Atome die Hybridisierung H besitzen müssen und stellt damit sicher, daß die Intervalluntergrenze auf jeden Fall erreicht wird. Wir gehen bisher allerdings nur dann so vor, wenn keine Makroatome verwendet werden.

Es seien nun n Vorgaben $\mathcal{R}_1, ..., \mathcal{R}_n$ für die Gesamthybridisierung gegeben. Die zugehörigen Intervalle seien mit $I_1 = [v_1; b_1], ..., I_n = [v_n; b_n]$ bezeichnet. Man beachte, daß es erlaubt ist, zu einer Hybridisierung zusätzlich die Anzahl der benachbarten Wasserstoffe des Atoms vorzuschreiben. Wir dürfen annehmen, daß mit $0 \leq i \leq n$ für $\mathcal{R}_1, ..., \mathcal{R}_i$ die Anzahl der benachbarten Wasserstoffe vorgeschrieben ist.

9.1.17 Algorithmus: *Festlegen der Hybridisierungen*

(1) Berechne für $j = 1, ..., n$ unter Berücksichtigung der Atomtypenverteilung die maximale Anzahl ma_j von Atomen mit Hybridisierung passend zu $\mathcal{R}_j$.
(2) **Für** alle $(a_1, ..., a_i) \in [v_1; ma_1] \times ... \times [v_i; ma_i]$ **tue**
(3) **Für** alle mit $(a_1, ..., a_i)$ *verträglichen* $(a_{i+1}, ..., a_n) \in [v_{i+1}; b_{i+1}] \times ... \times [v_n; b_n]$ **tue**
(4) **Für** $j = 1, ..., i$ suche die a_j Atome mit den niedrigsten Nummern, die in Atomname und H-Anzahl mit $\mathcal{R}_j$ übereinstimmen und lege für diese die Hybridisierung entsprechend fest.
(5) Berechne $(\tilde{a}_{i+1}, ..., \tilde{a}_n)$, indem gleiche bereits festgelegte Hybridisierungen mit vorgeschriebener H-Anzahl gegebenenfalls von $(a_{i+1}, ..., a_n)$ abgezogen werden.
(6) Berechne alle verschiedenen Festlegungen der zu $(\tilde{a}_{i+1}, ..., \tilde{a}_n)$ gehörigen Hybridisierungen auf die Atome, deren Hybridisierung bisher noch unbestimmt ist.

ende

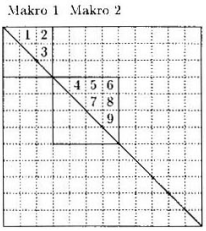
ende

Durch den obigen Algorithmus wird unser Konstruktionsproblem in disjunkte Teilprobleme zerlegt, d.h. je zwei Teilprobleme unterscheiden sich bereits anhand ihrer Hybridisierungen (bzw. sogar schon durch die Atomtypen- oder H-Verteilung). Deshalb darf während der Konstruktion kein Atom, dessen Hybridisierung nicht explizit vorgeschrieben wurde, eine Hybridisierung, für die in $\mathcal{R}_1, ..., \mathcal{R}_n$ ein Intervall vorgegeben wurde, erhalten.

9.1.18 Generierung der Adjazenzmatrizen. Bei der eigentlichen Generierung müssen wir eine Adjazenzmatrix A auf alle möglichen, mit den gegebenen Nebenbedingungen verträglichen Weisen füllen. Wie bereits erwähnt, besteht die Wahlmöglichkeit zwischen ordnungstreuer Erzeugung und der sog. *zielgerichteten Erzeugung*.

- Bei der zielgerichteten Erzeugung werden die Plätze von A **nicht** fest numeriert, d.h. abhängig vom jeweiligen Füllzustand von A wird die nächste Einfügeposition bestimmt.

Dabei werden zuerst die zu Makroatomen gehörigen Plätze von A gefüllt. Der Rest der Matrix wird zeilenweise gefüllt, wobei die jeweils als nächstes zu füllende Zeile in Abhängigkeit vom aktuellen Molekülanfangsstück sowie der verwendeten Restriktionen bestimmt wird. Die folgende Zeichnung veranschaulicht, wie die zu Makroatomen gehörigen Plätze numeriert werden:
Bild 5: *Generierung mit Makroatomen*



Natürlich berücksichtigen wir bei der zielgerichteten Erzeugung (wie bei der ordnungstreuen Erzeugung gelernt) offensichtliche Symmetrien der zu konstruierenden molekularen Graphen bei der Wahl der möglichen Einfügepositionen („Semikanonizität"). Der Isomorphietest erfolgt mit Hilfe von kanonischer Numerierung und einer Hashtabelle.

Bei der ordnungstreuen Erzeugung numerieren wir wie üblich die Plätze der rechten oberen Dreiecksmatrix der Adjazenzmatrix A zeilenweise aufsteigend und füllen diese ordnungstreu mit der lexikographisch kleinsten Füllung beginnend. Die Numerierung der Plätze ist in diesem Fall also fest und unabhängig von den Restriktionen immer gleich. Der Isomorphietest erfolgt durch den üblichen *Minimalitätstest*. Der Spezialfall schlichter regulärer Graphen wird gesondert berücksichtigt (siehe z.B. [37]).

Vor jedem Einfügen einer Kante k wird geprüft, ob der entstehende molekulare Graph mit den verwendeten Restriktionen verträglich ist. Falls dem nicht so ist, so wird k nicht eingefügt. In der Praxis wird vor jedem Legen einer Kante eine Funktion gerufen, die WAHR oder FALSCH liefert. An dieser Stelle können beliebige weitere Restriktionstypen berücksichtigt werden.

9.2 Implementation

Jede Restriktionsklasse R muß über die folgenden sechs Memberfunktionen verfügen:

- BOOLEAN **IsErl**(short s, short z, char *wert*);
  Überprüft, ob das Einfügen einer Kante der Vielfachheit *wert* von z nach s erlaubt ist. Evtl. werden hier recht aufwendige Berechnungen durchgeführt, so daß u.U. nicht bei jedem Aufruf von **IsErl** wirkliche Tests stattfinden sondern stattdessen einfach WAHR zurückgegeben wird.
- BOOLEAN **IsValid()**:
  Überprüft, ob die nun vollständig gefüllte Adjazenzmatrix mit den zugehörigen Restriktionen verträglich ist.

- void **Set**(short z, short s, char *wert*);
  Teilt der Restriktionsklasse mit, daß eine Kante der Vielfachheit *wert* von z nach s eingefügt wurde.
- void **Del**(short z, short s, char *wert*);
  Teilt der Restriktionsklasse mit, daß die Kante der Vielfachheit *wert* von z nach s gelöscht wurde.
  Set und **Del** ermöglichen es der Restriktionsklasse, sich auf den aktuellen Füllzustand der Adjazenzmatrix beziehende Datenstrukturen zu führen.
- int **NextLine()**;
  Liefert die Nummer des Atoms, dessen freie Valenzen als nächstes verbunden werden sollen (aus Sicht der jeweiligen Instanz der Restriktionsklasse R).
  Wie bereits erwähnt, ist **NextLine** nur für den Fall der *zielgerichteten Erzeugung* relevant.
- int **Priority()**;
  Die zu **NextLine()** gehörige Priorität.
  Existieren mehrere Instanzen von Restriktionsklassen, so werden die freien Valenzen desjenigen Atoms als nächstes verbunden, das mit der höchsten Priorität von einer NextLine Funktion vorgeschlagen wurde.

9.3 Hybridisierungen

Wiederum seien nun n Vorgaben für die Gesamthybridisierung $\mathcal{R}_1, ..., \mathcal{R}_n$ gegeben. Die zugehörigen Intervalle seien mit $I_1 = [v_1; b_1], ..., I_n = [v_n; b_n]$ bezeichnet. Wir dürfen annehmen, daß mit $0 \leq i \leq n$ für $\mathcal{R}_1, ..., \mathcal{R}_i$ die Anzahl der Wasserstöffe vorgeschrieben ist. In den Funktionen **Set** und **Del** werden u.a. die folgenden Datenstrukturen geführt:

- Für $j = 1, ..., n$ wird die aktuelle Anzahl anz_j von Atomen mit dieser Hybridisierung mitgezählt.
- Falls Makroatome verwendet werden, dann wird für $j = 1, ..., n$ die aktuelle Anzahl max_poss_j von Atomen mitgezählt, die hinsichtlich ihres Atomnamens und ihrer Wasserstoffanzahl noch $\mathcal{R}_j$ erfüllen könnten.

9.3.1 Prozedur: *IsErl*

(1) **Falls** keine Makroatome verwendet werden, **tue**
(2) Die Obergrenzen der für die Hybridisierungen vorgeschriebenen Intervalle dürfen nicht überschritten werden.
(3) Wurde für ein Atom α eine Hybridisierung H vorgeschrieben, so darf α nicht anders hybridisiert sein.
(4) Wurde für ein Atom α *keine* Hybridisierung vorgeschrieben, so darf α keine Hybridisierung H haben, die für andere Atome bereits vorgeschrieben wurde.
(5) **sonst**
(6) Die Obergrenzen der für die Hybridisierungen vorgeschriebenen Intervalle dürfen nicht überschritten werden.
(7) Zähle für jede Kombination $(name, h\_anz)$ von Atomname und Wasserstoffanzahl, wieviele Atome mit diesem Atomnamen und dieser H-Anzahl noch fehlen und vergleiche mit der maximal noch möglichen Anzahl.
(8) Überprüfe anhand der Adjazenzmatrix für $j = 1, ...n$,

ob die Untergrenze v_j noch erreicht werden kann.
ende

9.3.2 Prozedur: *NextLine*

(9) **Falls** keine Makroatome verwendet werden, **tue**
(10) Überprüfe, ob es noch ein Atom a gibt, dessen Hybridisierung vorgeschrieben wurde und das noch freie Valenzen hat.
(11) **Falls** a existiert, return(a).
(12) **sonst**
(13) Bestimme $k \in [1; n]$ so, daß es für die zu $\mathcal{R}_k$ noch fehlenden $b_k - anz_k$ Atome möglichst wenige Auswahlmöglichkeiten gibt.
(14) Suche ein Atom a, das noch freie Valenzen hat und $\mathcal{R}_k$ erfüllen kann.
(15) **Falls** a existiert, return(a).
ende

9.4 Ringe

Vor Beginn der Generierung wird die Taillenweite bestimmt. Die Taillenweite wird dann bei jedem Einsetzen einer Kante überprüft (da dieser Test ziemlich billig ist, kann man sich das leisten).
Die Prozedur **IsErl** prüft in bestimmten Abständen, ob sich im molekularen Graphen schon *zuviele* Kreise befinden. Die geforderten Mindestanzahlen bestimmter Kreise werden während der Generierung nicht geprüft, weil sich in den meisten molekularen Graphen *viele* Kreise befinden, so daß es wohl wahrscheinlicher ist, daß man eine Höchstzahl vorgeben will, die dann auch effektiver geprüft werden kann.

9.4.1 Prozedur: *IsErl*

(1) Für $i := 1$ bis *dim* suche Kreise, die nur aus Knoten $\geq i$ bestehen.
(2) Schätze die maximale Länge *max_l* eines solchen Kreises ab.
(3) Falls Knoten i weniger als 2 Nachbarn hat, so tue nichts.
(4) Aus *max_l* ergibt sich der maximal mögliche Abstand *maxabst* eines Knotens, der auf einem gesuchten Kreis liegt.
(5) Suche alle Knoten x, die von i mit höchstens *maxabst* Kanten über Knoten $\geq i$ zu erreichen sind und notiere jeweils deren Abstand von i.
(6) Suche jetzt depth-first alle Ringe und Cycles (der ges. Längen).
(7) Im Fall von Ringen prüfe, ob es unerlaubte Kanten gibt und verfolge entsprechende Äste (des Suchbaums) nicht weiter.
(8) Ist ein Ring bzw. cycle gefunden, so vergleiche mit den Obergrenzen.
(9) Handelt es sich um einen abschließenden Test, so vergleiche mit den Untergrenzen.

Durch die Vorgabe von Ring- bzw. Cycleanzahlen wird kein Einfluß auf die Füllreihenfolge der Adjazenzmatrix genommen.

9.5 Bindungen

9.5.1 Berechnung des Kantentyps. Sei $f \in \bar{4}^{(\bar{2}^{AM[2]})}$ ein molekularer Graph mit $AM = \{\vartheta_1, ..., \vartheta_n\}$ und $k = (k_1, k_2)$ eine Kante in f. Per Definition ist k vom Typ

- BOND_CYCLIC, falls k auf einem Kreis in f liegt.
- BOND_ACYCLIC, falls k in keiner Fortsetzung von f auf einem Kreis liegen kann.
- PERHAPS_BOND_CYCLIC, sonst.

9.5.2 Algorithmus: *Kantentyp*

Zu Berechnen sei der Typ der Kante $k = (k_1, k_2)$ im molekularen Graphen f.

(1) $\tilde{f} := f$.
(2) Entferne die Kante k aus $\tilde{f}$.
(3) V_{k_1} bezeichne die Zusammenhangskomponente von k_1 in $\tilde{f}$.
(4) V_{k_2} bezeichne die Zusammenhangskomponente von k_2 in $\tilde{f}$.
(5) **Falls** $k_2 \in V_{k_1}$, return(BOND_CYCLIC).
(6) **Falls** keines der Atome aus $\{V_{k_1} \cup V_{k_2}\} \setminus \{k_1, k_2\}$
noch freie Valenzen hat, return(BOND_ACYCLIC).
(7) **sonst** return(PERHAPS_BOND_CYCLIC).

9.5.3 Abschätzung der Anzahl zusätzlicher Bindungen. Sei $f \in \bar{4}^{(\bar{2}^{AM[2]})}$ ein molekularer Graph mit $AM = \{\vartheta_1, ..., \vartheta_n\}$. Gegeben seien weiterhin zwei Mengen von Atomtypen $T_x = \{tx_1, ..., tx_a\}$ und $T_y = \{ty_1, ..., ty_b\}$. Wir möchten die Anzahl der Kanten zwischen Atomen mit Typ aus T_x und Atomen mit Typ aus T_y mit Vielfachheit δ, die noch in f eingefügt werden können, nach oben abschätzen. Diese obere Grenze bezeichnen wir mit Ω.

9.5.4 Algorithmus: *Abschätzen zusätzlicher Bindungen*

(1) Suche alle Knoten $V_x = \{vx_1, ..., vx_{\tilde{a}}\}$ mit einem Typ aus T_x,
die noch mindestens δ freie Valenzen haben.
(2) Bestimme analog $V_y = \{vy_1, ..., vy_{\tilde{b}}\}$.
(3) Bilde die Mengen $V_{xy} = V_x \cap V_y$ sowie
$\tilde{V}_x := V_x \setminus V_{xy}$ und $\tilde{V}_y := V_y \setminus V_{xy}$.
(4) $fv[v]$ bezeichne die Anzahl der freien Valenzen des Knotens v.
(5) $\omega_{xy} := \sum_{v \in V_{xy}} \{\lfloor \frac{fv[v]}{\delta} \rfloor\}$.
(6) $\omega_x := \sum_{v \in \tilde{V}_x} \{\lfloor \frac{fv[v]}{\delta} \rfloor\}$.
(7) $\omega_y := \sum_{v \in \tilde{V}_y} \{\lfloor \frac{fv[v]}{\delta} \rfloor\}$.
(8) $\Omega := min\{\omega_x, \omega_y\}$.
(9) $\nu := min\{\omega_{xy}, max\{\omega_x, \omega_y\} - \Omega\}$, $\Omega := \Omega + \nu$.
(10) $\mu := \omega_{xy} - \nu$, $\Omega := \Omega + \lfloor \frac{\mu}{2} \rfloor$.

9.5.5 Die Testfunktion. Zu einer gegebenen *Einschränkungen der erlaubten Bindungen* $\mathcal{R}_{N_1,N_2,d,l}$ geben wir nun die zugehörige Testfunktion an:

9.5.6 Prozedur: *IsErl*

(1) $z := 0$
(2) **Für** alle Kanten (k_1, k_2) im Anfangsstück **tue**

(3) **Falls** (k_1, k_2) zu N_1 und N_2 paßt, **tue**
(4) Berechne den Typ von (k_1, k_2) mit 9.5.2
(5) **Falls** (k_1, k_2) mit Vielfachheit und Typ zu
$\mathcal{R}_{N_1,N_2,d,I}$ paßt, so $z := z + 1$.
(6) **Falls** z grösser als die Obergrenze von I ist, return(FALSCH).
ende
ende
(7) $z := 0$
(8) **Falls** $\mathcal{R}$ zyklische Kanten restringiert, **tue**
(9) Belege z mit der Anzahl der zu N_1, N_2 passenden Kanten
vom Typ BOND_CYCLIC und PERHAPS_BOND_CYCLIC.
(10) **sonst wenn** $\mathcal{R}$ azyklische Kanten restringiert, **tue**
(11) Belege z mit der Anzahl der zu N_1, N_2 passenden Kanten
vom Typ BOND_ACYCLIC und PERHAPS_BOND_CYCLIC.
(12) **sonst**
Belege z mit der Anzahl aller zu N_1, N_2 passenden Kanten.
ende
(13) Berechne Ω mit Algorithmus 9.5.4, $z := z + \Omega$.
(14) **Falls** z kleiner als die Untergrenze von I ist, return(FALSCH);
(15) return(WAHR).

9.6 Symmetrien

9.6.1 Konstruktion maximaler Cliquen. Gegeben sei ein molekularer Graph $f \in \bar{4}^{(\bar{2}^{AM[2]})}$ mit $AM = \{\vartheta_1, ..., \vartheta_a\}$, ein Atom $v \in AM$ und eine Menge $N = \{n_1, ..., n_a\}$ von Nachbarn von v in f. Gesucht seien alle Cliquen $\{v, e_1, ..., e_k\}$ mit $\{e_1, ..., e_k\} \subset N$, $0 \leq k \leq a$, die sich durch kein weiteres Element aus N vergrößern lassen. Für großes a können wir nicht alle Möglichkeiten durchlaufen. Da sich die Zahl der Lösungen hoffentlich in Grenzen halten wird, gehen wir folgendermaßen vor:

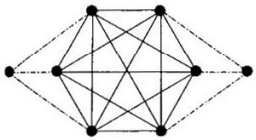
9.6.2 Algorithmus: *Cliquensuche*

(1) L sei eine Liste aller bisherigen Lösungen. $L := \emptyset$.
(2) $k := 1$, $next_1 := \{n_1, ..., n_a\}$; $bis_1 := a$, $idx_1 := 1$.
(3) **Solange** $k \geq 1$
(4) $e_k := next_k[idx_k]$.
(5) Belege $next_{k+1}$ mit allen Knoten $u \notin \{e_1, ..., e_k\}$, $u \in N$,
wobei u für alle $i = 1, ..., k$ mit e_i verbunden ist.
(6) $bis_{k+1} := |next_{k+1}|$.
(7) **Für** alle $\sigma \in L$ mit $\{e_1, ..., e_k\} \subseteq \sigma$ **tue**
(8) Berechne die Menge $next_\sigma$ der Knoten $u \notin \sigma$, $u \in N$,
wobei u für alle $i = 1, ..., k$ mit e_i verbunden ist.
(9) **Falls** $|next_\sigma| = 0$, so $bis_{k+1} := 0$ und break.
(10) **Falls** $|next_\sigma| < |next_{k+1}|$,
(11) $next_{k+1} := next_\sigma$, $bis_{k+1} := |next_{k+1}|$.
ende
ende
(12) **Falls** $bis_{k+1} > 0$, $k := k + 1 \wedge idx_k := 1$.

(13) **sonst**
(14) **Falls** $\{v, e_1, ..., e_k\}$ nicht Teilmenge eines Elements aus L ist, **tue**
(15) Füge $\{v, e_1, ..., e_k\}$ in L ein.
ende
(16) **Solange** $idx_k = bis_k$
(17) $k := k - 1$.
(18) **Falls** $k = 0$, return.
ende
(19) $idx_k := idx_k + 1$.
ende
ende

9.6.3 Minimale Überdeckung mit Cliquen. Gegeben sei ein schlichter Graph KG auf σ Punkten. Zu berechnen sei die minimale Anzahl ν von Cliquen, mit der man KG überdecken kann. ν kann man nicht berechnen, indem man sukzessive die größte vorhandene Clique entfernt, wie das folgende Gegenbeispiel zeigt:

Bild 6: *Beispiel zur Überdeckung mit Cliquen*



Wählt man zuerst die 6-er-Clique in der Mitte, so bleiben die Knoten rechts und links allein und wir erhalten $\nu = 3$. Offensichtlich ist es aber auch möglich, zwei 4-er-Cliquen zu wählen.

Dadurch wird leider Backtracking nötig:

9.6.4 Algorithmus: *Überdeckung mit Cliquen*

v_i sei der Knoten, von dem aus die i-te Clique gesucht wird. $v_1 := 1$.
In Cl_i werden die zur i-ten Clique gehörigen Knoten gespeichert.
(1) $\nu := \infty$, $stack := \emptyset$.
(2) $\alpha_i := \sum_{r=1}^{i} |Cl_r|$; $\alpha_0 := 0$, $\mu := 1$; $Cl_\mu := \{\}$.
(3) **Solange** $\mu \geq 1$
(4) $N_\mu := \{n_{\mu,1}, ...n_{\mu,\eta_\mu}\}$, wobei gilt:
$n_{\mu,i}$ ist benachbart zu v_μ und $n_{\mu,i} \notin stack$.
(5) **Falls** es eine neue Auswahl von k Elementen $\{e_1, ..., e_k\} \subset N_\mu$ gibt mit
- $1 \leq k \leq \eta_\mu$, falls $\eta_\mu > 0$,
- $k = 0$, falls $\eta_\mu = 0$,
- $\{v_\mu, e_1, ..., e_k\}$ ist eine „maximale Clique“,

(6) **tue**
(7) $stack := stack \setminus Cl_\mu$, $Cl_\mu := \{v_\mu, e_1, ..., e_k\}$.
(8) $stack := stack \cup Cl_\mu$, $\alpha_\mu := \alpha_{\mu-1} + (k+1)$.
(9) **Falls** $\alpha_\mu = \sigma$, so $\nu := \mu$.
(10) **sonst**

(11) **Falls** $\mu + 1 < \nu$, so $\mu := \mu + 1$; $Cl_\mu := \{\}$; Belege v_μ.
ende
(12) **sonst tue**
(13) $stack := stack \setminus Cl_\mu$, $\mu := \mu - 1$.
ende
ende
(14) return(ν).

9.6.5 Bemerkung:
Es genügt, maximale Cliquen zu betrachten. Angenommen, wir haben eine Überdeckung von KG mit Cliquen $Cl_1, ..., Cl_\mu$, wobei Cl_1 nicht maximal ist. Dann existiert also ein Knoten $v \notin Cl_1$, der zu allen Knoten aus Cl_1 benachbart ist. Wir können v dann zu Cl_1 hinzunehmen ohne daß μ größer wird.

9.6.6 Die Testfunktion. Gegeben sei ein molekularer Graph $f \in \bar{4}^{(2^{AM^{[2]}})}$ mit $AM = \{\vartheta_1, ..., \vartheta_n\}$. Die Anzahl der Bahnen von $Aut(f)$ auf den Atomen mit einer vorgegebenen Eigenschaft E muß in $I = [min; max]$ enthalten sein.

Handelt es sich um einen abschließenden Test, so liegt $Aut(f)$ als labeled branching vor. Durch entsprechenden Basiswechsel können wir die gesuchten Bahnen direkt ablesen.

Bei einem Zwischentest gehen wir folgendermaßen vor: Da fast alle Moleküle eine triviale Automorphismengruppe besitzen, haben wir nur dann eine starke Einschränkung der Lösungsmenge, wenn Automorphismen gefordert werden; d.h. wenn die Obergrenze max kleiner als die Anzahl der Atome mit Eigenschaft E ist. Also wird bei einem Zwischentest auch nur abgeschätzt, ob die Obergrenze überschritten wurde.
Sind zwei Atome in der gleichen Bahn der Automorphismengruppe, so müssen sie den gleichen Atomtyp haben und die jeweiligen Nachbaratome müssen sich auch aufeinander abbilden lassen. Dies gilt natürlich auch für Nachbaratome im Abstand $a > 1$.

9.6.7 Prozedur: *IsErl*

$X := \{\varrho \in AM \mid \varrho$ hat Eigenschaft $E\}$.
(1) Stelle den Kompatibilitätsgraph $KG \in \{0,1\}^{X^{[2]}}$ auf:
(2) **Für** alle $\{A, B\} \subseteq X$ **tue**
(3) **Falls** A kompatibel mit B ist (9.1.4), **tue**
(4) Füge Kante $\{A, B\}$ in KG ein.
ende
ende
(5) Berechne die minimale Anzahl ν von Cliquen,
mit der man KG überdecken kann (9.6.4).
(6) **Falls** $\nu > max$, so return(FALSCH).
(7) return(WAHR).

9.6.8 Bemerkung:
Der geschilderte Algorithmus berücksichtigt keine aromatischen Bindungen, so daß bei der Konstruktion aromatischer Strukturen Vorsicht geboten ist, denn es können sich in solchen Fällen mehr topologisch verschiedene Atome ergeben, als es unter Berücksichtigung der Aromatizität tatsächlich der Fall ist.

9.7 Substrukturen

9.7.1 Substrukturen zu gegebener Summenformel. Sei $f \in \bar{4}^{(\bar{2}^{AM[2]})}$ ein molekularer Graph mit $AM = \{\vartheta_1, ..., \vartheta_n\}$,
$F = \{(X_1, \alpha_1), ..., (X_k, \alpha_k)\}$ eine Bruttoformel und I ein Intervall.
Substruktursuche

Gegeben sei eine Bruttoformel $F = \{(X_1, \alpha_1), ..., (X_k, \alpha_k)\}$, die keine H-atome enthält. Darüberhinaus seien ξ Wasserstoffatome vorgegeben.
anz bezeichne die Anzahl der gefundenen Substrukturen.

9.7.2 Algorithmus: *Einbettungsvielfachheit*

Gegeben sei eine zusammenhängende Substruktur S von f.
(1) Definiere ζ als die Anzahl der an S gebundenen H-atome.
(2) return($\binom{\zeta}{\xi}$).

9.7.3 Algorithmus: *Substruktursuche*

(1) Wähle u so, daß von Atomen mit Name X_u minimal viele Kanten ausgehen.
(2) **Für** alle Atome ϑ_ν mit $AN(\vartheta_\nu) = X_u$ **tue**
(3) $z_{1,u} := 1$; $z_{1,j} := 0$ für alle $j \neq u$.
($z_{1,j}$ ist die momentane Anzahl von Atomen mit Name X_j)
(4) $schicht_1 := \{\vartheta_\nu\}$.
(5) **Falls** dies eine Lösung ist **tue**
(6) Erhöhe *anz* entsprechend 9.7.2, Gehe zu (2).
ende
(7) $schicht\_sum_1 := schicht_1$.
(8) $nachb_1 := \{v \in AM \mid v \in N(schicht_1) \setminus schicht\_sum_1\}$, wobei $AN(v)$ in F enthalten ist und für $\vartheta_t \in$ nachb$_1$ mit $AN(\vartheta_t) = X_u$ gilt: $t > \nu$.
ende
(9) $schicht\_sum_2 := schicht\_sum_1 \cup nachb_1$.
(10) $\tilde{\alpha}_{1,j} := |\{v \in nachb_1 \mid \mathrm{AN}(v) = X_j\}|$ für alle $j = 1, ..., k$.
(11) $\hat{\alpha}_{1,j} := \alpha_j - z_{1,j}$, $\gamma_{1,j}^{max} := min\{\tilde{\alpha}_{1,j}, \hat{\alpha}_{1,j}\}$.
(12) Schätze $\gamma_{1,j}^{min}$, die Anzahl der Atome mit Namen X_j, die mindestens hinzugenommen werden müssen ab.
(13) $i := 1$.
(14) **Solange** $i \geq 1$
(15) **Falls** es eine neue Auswahl von k ($\gamma_{1,j}^{min} \leq k \leq \gamma_{1,j}^{max}$) Atomen aus $nachb_i$ gibt, **tue**
(16) Belege $schicht_{i+1}$ mit den ausgewählten Atomen.
(17) Belege $z_{i+1,j}$ für alle $j = 1, ..., k$.
(18) **Falls** dies eine Lösung ist, erhöhe *anz* entsprechend 9.7.2.
(19) **sonst**
(20) $i := i + 1$, Belege $nachb_i$.
(21) $schicht\_sum_{i+1} := schicht\_sum_i \cup nachb_i$.
(22) Belege $\tilde{\alpha}_{i,j}$, $\hat{\alpha}_{i,j}$, $\gamma_{i,j}^{max}$, $\gamma_{i,j}^{min}$ wie im Fall $i = 1$.
ende

ende

(23) $i := i - 1$.

ende

ende

Die Testfunktion

Gegeben sei die Vorgabe zusammenhängender Teilgraphen zu den Parametern (F, I).

9.7.4 Algorithmus: *IsErl*

(1) Durchsuche den molekularen Graphen nach zusammenhängenden Substrukturen mit der Summenformel F nach 9.7.3 und zähle die Häufigkeit *anz* des Vorkommens.

(2) **Falls** die Obergrenze von I überschritten wird, return(FALSCH).

(3) **Falls** *anz* kleiner als die Untergrenze von I ist, **tue**

(4) **Falls** bei der Substruktursuche 9.7.3 keines der Anfangsstücke noch freie Valenzen hatte, return(FALSCH).

ende

(5) return(WAHR).

9.7.5 Teilgraphen. Sei $S = (V, E)$ eine Substruktur mit $V = \{v_1, ..., v_l\}$. $\mathcal{R}$ eine zugehörige Menge von Substrukturrestriktionen und $f \in \bar{4}^{(2^{AM^{[2]}})}$ ein Molekülanfangsstück mit $AM = \{\vartheta_1, ..., \vartheta_n\}$

Im folgenden benutzen wir wiederholt:

- Das Tupel *einbettung* $= (eb_1, ..., eb_l)$, der eine teilweise Einbettung von S in f repräsentiert. Für $i = 1, ..., l$ gilt dann:
  - $eb_i > 0 \implies v_i$ wird abgebildet auf ϑ_{eb_i}.
  - $eb_i = 0 \implies v_i$ wird auf ein Wasserstoffatom abgebildet.
  - $eb_i = -1 \implies v_i$ wurde noch nicht abgebildet.
- Das Tupel *einbettungsfolge* $= (ebf_1, ..., ebf_k)$. Dabei repräsentiert ebf_i für $i = 1, ..., k$ eine teilweise Einbettung von S in f. Für $j = 1, ..., k-1$ ist außerdem ebf_{j+1} eine Fortsetzung von ebf_j, d.h. aus $ebf_j(a) \neq -1$ folgt $ebf_{j+1}(a) = ebf_j(a)$.

Für die Rückgabewerte der anschließend dargestellten Prozeduren *TypTest*, *EdgeTest*, *DistTest*, *CycleTest*, *HybrTest*, *ExtTypesTest*, *BondVielfTest*, *SubResTest*, *BranchTest* und *SubTest* gilt:

- WAHR, wenn bzgl. *einbettung* alle Fortsetzungen des aktuellen Molekülanfangsstücks den jeweiligen Test bestehen.
- VIELLEICHT, wenn es mindestens eine Fortsetzung des aktuellen Molekülanfangsstücks geben könnte, die den jeweiligen Test besteht.
- FALSCH, wenn es keine Fortsetzung des aktuellen Molekülanfangsstücks gibt, die den jeweiligen Test bestehen kann.

9.7.6 Prozedur: *TypTest*

(1) *ret* := WAHR.

(2) **Für** alle *Einschränkungen der Atomtypenkombinationen* von S **tue**

(3) Seien T, TM, I die Parameter der aktuellen *Einschränkung der Atomtypenkombinationen* mit

- $T \subseteq V$,
- TM ist eine nichtleere Menge von Atomtypen,
- $I = [min; max]$ ist Intervall.

(4) Belege die boolesche Variable *was_in_last_einbettung* so, daß *was_in_last_einbettung* = WAHR genau dann wenn T von eb_k schon komplett eingebettet wurde.

(5) **Falls** *was_in_last_einbettung* = FALSCH oder ($(k > 0)$ und $tr_k \neq$ WAHR) **tue**

(6) Belege die boolesche Variable *is_in_einbettung* so, daß *is_in_einbettung* = WAHR genau dann wenn T von *einbettung* komplett abgebildet wird.

(7) Berechne nun die Anzahl α der Atome von T, die bzgl. *einbettung* mit TM verträglich sind.

(8) **Falls** $\alpha > max$, so return(FALSCH).

(9) **Falls** $\alpha < min$ und *is_in_einbettung* = WAHR, so return(FALSCH).

(10) **Falls** $\alpha < min$ und *is_in_einbettung* = FALSCH, so *ret* := VIELLEICHT.

ende

ende

(11) return(*ret*).

9.7.7 Prozedur: *EdgeTest*

(1) *ret* := WAHR.

(2) **Für** alle *Einschränkungen der Bindungskombinationen* von S **tue**

(3) Seien P, I, idx die Parameter der aktuellen *Einschränkung der Bindungskombinationen* mit

- $P \subseteq E$,
- $I = [min; max]$ ist Intervall,
- $idx \in \{1, 2, 3\}$.

(4) Belege die boolesche Variable *was_in_last_einbettung* so, daß *was_in_last_einbettung* = WAHR genau dann wenn P von eb_k schon komplett eingebettet wurde.

(5) **Falls** *was_in_last_einbettung* = FALSCH oder ($(k > 0)$ und $er_k \neq$ WAHR) **tue**

(6) Belege die boolesche Variable *is_in_einbettung* so, daß *is_in_einbettung* = WAHR genau dann wenn P von *einbettung* komplett abgebildet wird.

(7) Berechne nun die Anzahl α der Bindungen aus P, die bzgl. *einbettung* Vielfachheit idx haben.

(8) **Falls** $\alpha > max$, so return(FALSCH);

(9) **Falls** $\alpha < min$ und *is_in_einbettung* = WAHR, **tue**

(10) **Falls** im Molekül schon alle betreffenden Bindungen gesetzt wurden, so return(FALSCH).

(11) **sonst** *ret* := VIELLEICHT.

ende

ende

ende
(12) return(ret).

9.7.8 Prozedur: *DistTest*

(1) ret := WAHR.
(2) **Für** alle *Vorgaben von Abständen* von S **tue**
(3) Seien T_1, T_2, I die Parameter der aktuellen *Vorgabe von Abständen* mit
- $T_1 \subseteq V$,
- $T_2 \subseteq V$,
- $I = [min; max]$ ist Intervall.

(4) Belege die boolesche Variable *was_in_last_einbettung* so, daß *was_in_last_einbettung* = WAHR genau dann wenn T_1 und T_2 von eb_k schon komplett eingebettet wurden.
(5) **Falls** *was_in_last_einbettung* = FALSCH oder
($(k > 0)$ und $dr_k \neq$ WAHR) **tue**
(6) Belege die boolesche Variable *is_in_einbettung* so, daß *is_in_einbettung* = WAHR genau dann wenn T_1 und T_2 von *einbettung* komplett abgebildet werden.
(7) Berechne nun den Abstand α von T_1 und T_2 bzgl. *einbettung*.
(8) Prüfe dabei, ob sich α durch Einfügen weiterer Kanten in das Molekül verkleinern könnte.
(9) **Falls** $\alpha < min$, so return(FALSCH).
(10) **Falls** $\alpha > max$ und α sich nicht mehr verkleinern kann, return(FALSCH).
(11) **Falls** $\alpha > max$ und α sich verkleinern kann, ret := VIELLEICHT.
(12) **Falls** $min > 1$ und α sich verkleinern kann, ret := VIELLEICHT.
ende
ende
(13) return(ret).

9.7.9 Prozedur: *CycleTest*

(1) ret := WAHR.
(2) **Für** alle *Cyclebedingungen* von S **tue**
(3) Seien T, I, b die Parameter der aktuellen *Cyclebedingung* mit
- $T \subseteq V$,
- $I = [min; max]$ ist ein Intervall.
- b ist boolescher Wert.

(4) Belege die boolesche Variable *was_in_last_einbettung* so, daß *was_in_last_einbettung* = WAHR genau dann wenn T von eb_k schon komplett eingebettet wurde.
(5) **Falls** *was_in_last_einbettung* = FALSCH oder

((k > 0) und $cr_k \neq$ WAHR) **tue**
(6) Belege die boolesche Variable *is_in_einbettung* so, daß
is_in_einbettung = WAHR genau dann wenn
T von *einbettung* komplett abgebildet wird.
(7) **Falls** die Atome aus T bzgl. *einbettung* auf einem Kreis
der Länge $l \in I$ liegen, *ask* := WAHR.
(8) **sonst falls** es eine Fortsetzung von f geben könnte,
so daß die Atome aus T auf einem Kreis der
Länge $l \in I$ liegen, *ask* := VIELLEICHT.
(9) **sonst** *ask* := FALSCH.
(10) **Falls** b =WAHR **tue**
(11) **Falls** *ask* = FALSCH, so return(FALSCH).
(12) **Falls** *is_in_einbettung* = WAHR und *ask* = VIELLEICHT,
ret := VIELLEICHT.
(13) **sonst**
(14) **Falls** *is_in_einbettung* = WAHR und *ask* = WAHR,
return(FALSCH).
(15) **Falls** *is_in_einbettung* = WAHR und *ask* = VIELLEICHT,
ret := VIELLEICHT.
ende
ende
ende
(16) return(*ret*).

9.7.10 Prozedur: *HybrTest*

(1) *ret* := WAHR.
(2) **Für** alle *Vorgaben von Hybridisierungen* von S **tue**
(3) Seien T, Φ, I die Parameter der aktuellen
Vorgabe von Hybridisierungen mit
- $T \subseteq V$,
- Φ ist eine Menge von Hybridisierungen,
- $I = [min; max]$ ist Intervall.
(4) Belege die boolesche Variable *was_in_last_einbettung* so, daß
was_in_last_einbettung = WAHR genau dann wenn
T von eb_k schon komplett eingebettet wurde.
(5) **Falls** *was_in_last_einbettung* = FALSCH oder
((k > 0) und $hr_k \neq$ WAHR), **tue**
(6) Belege die boolesche Variable *is_in_einbettung* so, daß
is_in_einbettung = WAHR genau dann wenn
T von *einbettung* komplett abgebildet wird.
(7) Berechne die Anzahl a_1 der Atome aus T, die bzgl. *einbettung* eine
Hybridisierung aus Φ besitzen.
(8) Berechne die Anzahl a_2 der Atome aus T, die bzgl. *einbettung* eine
Hybridisierung aus Φ bekommen könnten.
(9) **Falls** $a_1 < min$, **tue**
(10) **Falls** *is_in_einbettung* = FALSCH, **tue**

(11) **Falls** min noch erreicht werden kann,
$ret :=$ VIELLEICHT.
(12) **sonst** return(FALSCH).
(13) **sonst**
(14) **Falls** $a_2 < min$, return(FALSCH).
(15) **sonst** $ret :=$ VIELLEICHT.
ende
ende
(16) **Falls** max noch überschritten werden kann, $ret :=$ VIELLEICHT.
ende
ende
(17) return(ret).

9.7.11 Prozedur: *ExtTypesTest*

(1) $ret :=$ WAHR.
(2) **Für** alle *Vorgaben benachbarter Atome* von S **tue**
(3) Seien $T, I_d, I_a, \mathcal{AT}, \mathcal{EH}, \mathcal{DT}$ die Parameter der aktuellen *Vorgabe benachbarter Atome* mit
- $T \subseteq V$,
- $I_d = [id_1; id_2]$ ist Intervall,
- $I_a = [ia_1; ia_2]$ ist Intervall,
- $\mathcal{AT}$ ist eine Menge von Atomtypen,
- $\mathcal{EH}$ ist eine Menge von Hybridisierungen,
- $\mathcal{DT}$ ist eine Menge von Atomtypen.

(4) Belege die boolesche Variable *was_in_last_einbettung* so, daß *was_in_last_einbettung* = WAHR genau dann wenn T von cb_k schon komplett eingebettet wurde.
(5) **Falls** *was_in_last_einbettung* = FALSCH oder
($(k > 0)$ und $er_k \neq$ WAHR), **tue**
(6) Belege die boolesche Variable *is_in_einbettung* so, daß *is_in_einbettung* = WAHR genau dann wenn T von *einbettung* komplett abgebildet wird.
(7) Im sei definiert als das Bild von S bzgl. *einbettung*.
(8) Berechne die Menge BA der Atome des aktuellen Molekülanfangsstücks, die von Im Abstand $x \in I_d$ haben.
(9) **Falls** sich BA in einer von f vergrößern könnte, **tue**
ext_atoms_may_extend = WAHR.
(10) **sonst** *ext_atoms_may_extend* = FALSCH.
(11) Berechne die Anzahl a_1 der Atome aus BA, die die Bedingungen $\mathcal{AT}, \mathcal{EH}, \mathcal{DT}$ erfüllen.
(12) Berechne die Anzahl a_2 der Atome aus BA, die die Bedingungen $\mathcal{AT}, \mathcal{EH}, \mathcal{DT}$ noch erfüllen können.
(13) **Falls** $a_1 > max$, so return(FALSCH).
(14) **Falls** $a_1 < min$, **tue**
(15) **Falls** *is_in_einbettung* = WAHR und $a_2 < min$ und *ext_atoms_may_extend* = FALSCH, return(FALSCH).
(16) **sonst** $ret :=$ VIELLEICHT.

(17) **Falls** $max < \infty$, **tue**
(18) **Falls** *is_in_einbettung* = FALSCH oder $a_2 > max$ oder
ext_atoms_may_extend = WAHR, *ret* := VIELLEICHT.
ende
ende
ende
ende
(19) return(*ret*).

9.7.12 Prozedur: *BondVielfTest*

(1) *ret* := WAHR.
(2) **Für** alle *Vorgaben von Bindungsvielfachheiten* von S **tue**
(3) Seien $T, I, \mathcal{B}$ die Parameter der aktuellen *Vorgabe von Bindungsvielfachheiten*.
(4) Belege die boolesche Variable *was_in_last_einbettung* so, daß
was_in_last_einbettung = WAHR genau dann wenn
T von cb_k schon komplett eingebettet wurde.
(5) **Falls** *was_in_last_einbettung* = FALSCH oder
$((k > 0)$ und $br_k \neq$ WAHR), **tue**
(6) Belege die boolesche Variable *is_in_einbettung* so, daß
is_in_einbettung = WAHR genau dann wenn
T von *einbettung* komplett abgebildet wird.
(7) Berechne die Anzahl a_1 der Kanten, die bzgl. *einbettung* mit
einem Atom aus T verbunden sind und eine Vielfachheit aus
$\mathcal{B}$ besitzen.
(8) Berechne die max. Anzahl a_2 zusätzlicher, bzgl. *einbettung* mit
einem Atom aus T verbundener Kanten, die eine
Vielfachheit aus $\mathcal{B}$ besitzen.
(9) **Falls** $a_1 > max$, return(FALSCH).
(10) **Falls** $a_1 < min$, **tue**
(11) **Falls** *is_in_einbettung* = WAHR und $a_1 + a_2 < min$.
return(FALSCH),
(12) **sonst** *ret* := VIELLEICHT.
ende
(13) **Falls** $max < \infty$
(14) **Falls** *is_in_einbettung* = FALSCH oder $a_1 + a_2 > max$.
(15) *ret* := VIELLEICHT.
ende
ende
ende
(16) return(*ret*).

9.7.13 Prozedur: *SubResTest*

(1) *ret* := WAHR.
(2) **Für** alle *Vorgaben benachbarter Substrukturen* von S **tue**
(3) Seien $T, I_a, I_d, \mathcal{SS}, \mathcal{B}$ die Parameter der aktuellen
Vorgabe benachbarter Substrukturen.
(4) Belege die boolesche Variable *was_in_last_einbettung* so, daß

was_in_last_einbettung = WAHR genau dann wenn
T von cb_k schon komplett eingebettet wurde.
(5) **Falls** *was_in_last_einbettung* = FALSCH, **tue**
(6) Belege die boolesche Variable *is_in_einbettung* so, daß
is_in_einbettung = WAHR genau dann wenn
T von *einbettung* komplett abgebildet wird.
(7) **Falls** *is_in_einbettung* = WAHR **tue**
(8) *startpoints* enthalte alle Atome, die bzgl. *einbettung*
Abstand $x \in I_d$ von T haben.
(9) *forbiddenpoints* enthalte alle Atome, die bzgl. *einbettung*
Abstand $x < min_d$ haben.
(10) Gegeben sei ein Algorithmus $\mathcal{A}$, der zu einer Substruktur S
alle möglichen Einbettungen in ein Molekuel M berechnet.
Beschränke die Anfangspunkte der Suche auf Elemente von
startpoints und besuche keine Atome aus *forbiddenpoints*.
(11) Führe Alg. $\mathcal{A}$ nacheinander für alle Strukturen aus $\mathcal{SS}$ durch.
($\mathcal{B}$ wird erst geprüft, wenn eine Einbettung gefunden wurde)
(12) Sei *anz* die Anzahl der benachbarten Substrukturen mit
den gewünschten Eigenschaften.
(13) **Falls** $anz > max_a$, return(FALSCH).
(14) *fv_flag* := FALSCH.
(15) Mit N bezeichnen wir die Menge aller Knoten, die während
der Berechnung von *startpoints* oder während Algorithmus
$\mathcal{A}$ besucht wurden.
(16) **Falls** ein Atom aus N freie Valenzen hat, *fv_flag* := WAHR.
(17) **Falls** $anz < min_a$ und *fv_flag* = FALSCH, return(FALSCH).
(18) **Falls** $anz < min_a$ und *fv_flag* = WAHR, *ret* := VIELLEICHT.
(19) **Falls** $max_a < \infty$ und *fv_flag* = WAHR, *ret* := VIELLEICHT.
ende
ende
ende
(20) return(*ret*).

Testen benachbarter branches BranchTest läuft analog zu SubResTest. Lediglich Algorithmus $\mathcal{A}$ muß ausgetauscht werden gegen den Algorithmus aus 9.7.1.

Testen aller Substrukturrestriktionen 9.7.14 Prozedur: *SubstrRestrTest*

(1) *ret* :=WAHR.
(2) b := *TypTest*(*einbettung*, *einbettungsfolge*).
(3) **Falls** b =FALSCH oder (b =VIELLEICHT und *ret* =WAHR),
(4) *ret* := b.
(5) b := *EdgeTest*(*einbettung*, *einbettungsfolge*).
(6) **Falls** b =FALSCH oder (b =VIELLEICHT und *ret* =WAHR),
(7) *ret* := b.
(8) b := *DistTest*(*einbettung*, *einbettungsfolge*).
(9) **Falls** b =FALSCH oder (b =VIELLEICHT und *ret* =WAHR),
(10) *ret* := b.

(11) $b := RingTest(einbettung, einbettungsfolge)$.
(12) **Falls** b =FALSCH oder (b =VIELLEICHT und ret =WAHR),
(13) $ret := b$.
(14) $b := HybrTest(einbettung, einbettungsfolge)$.
(15) **Falls** b =FALSCH oder (b =VIELLEICHT und ret =WAHR),
(16) $ret := b$.
(17) $b := ExtTypesTest(einbettung, einbettungsfolge)$.
(18) **Falls** b =FALSCH oder (b =VIELLEICHT und ret =WAHR),
(19) $ret := b$.
(20) $b := BondVielfTest(einbettung, einbettungsfolge)$.
(21) **Falls** b =FALSCH oder (b =VIELLEICHT und ret =WAHR),
(22) $ret := b$.
(23) $b := SubResTest(einbettung, einbettungsfolge)$.
(24) **Falls** b =FALSCH oder (b =VIELLEICHT und ret =WAHR),
(25) $ret := b$.
(26) $b := BranchTest(einbettung, einbettungsfolge)$.
(27) **Falls** b =FALSCH oder (b =VIELLEICHT und ret =WAHR),
(28) $ret := b$.
(29) return(ret).

Fortsetzbarkeit einer teilweisen Einbettung Gegeben sei eine Substruktur $S = (V, E)$, eine zugehörige Menge $\mathcal{R}$ von Substrukturrestriktionen, $AM = \{\vartheta_1, \ldots, \vartheta_n\}$ sowie ein molekularer Graph $f \in \bar{4}^{(\mathbb{P}^{AM[2]})}$. Weiterhin sei $sub\_el \in V$, $mol\_el \in AM$ sowie *einbettung* eine teilweise Einbettung von S in f. Nun soll getestet werden, ob die Identifikation von sub_el mit mol_el zu einer vollständigen Einbettung von S in f führen kann.

9.7.15 Algorithmus: *Fortsetzbarkeit*

(1) ret :=WAHR.
(2) **Falls** nicht genügend viele Wasserstoffe zu mol_el benachbart sind,
(3) return(FALSCH).
(4) **Falls** bzgl. der Atomtypen keine Übereinstimmung möglich ist,
(5) return(FALSCH).
(6) Zähle unterschieden bzgl. Atomname, Bindungsvielfachheit und Wasserstoffanzahl die Nachbarn von sub_el.
(7) **Falls** mol_el in f nicht genug Nachbarn dieses Typs hat (und auch nicht mehr bekommen kann), return(FALSCH).
(8) Für alle bereits identifizierten Atome α **tue**
(9) **Falls** sub_el mit α a-fach ($a > 0$), und mol_el mit $einbettung(\alpha)$ b-fach ($b > 0$) verbunden ist mit $a \neq b$,
(10) return(FALSCH).
(11) **Falls** sub_el mit α a-fach ($a > 0$), und mol_el mit $einbettung(\alpha)$ 0-fach verb. ist, **tue**
(12) **Falls** mol_el und $einbettung(\alpha)$ jeweils noch mind. a freie Valenzen besitzen, ret := VIELLEICHT.
(13) **sonst** return(FALSCH).
ende

ende

(14) $bool := SubstrRestrTest(einbettung)$.

(15) **Falls** $bool$ =FALSCH oder ($bool$ =VIELLEICHT und ret =WAHR),
$ret := bool$.

(16) return(ret).

Finden aller Einbettungen einer Substruktur Gegeben sei eine Substruktur $S = (V, E)$, eine zugehörige Menge $\mathcal{R}$ von Substrukturrestriktionen, $AM = \{\vartheta_1, ..., \vartheta_n\}$ sowie ein molekularer Graph $f \in \bar{4}^{(2^{AM[2]})}$. Wir möchten nun alle Einbettungen von S nach f finden. Da es allgemein bekannt ist, wie ein Algorithmus zur Substruktursuche funktioniert, sollen an dieser Stelle nur einige Bemerkungen über Besonderheiten unseres Vorgehens gemacht werden:

- Wir numerieren die Atome von S fortlaufend, indem wir mit einem uns „charakteristisch" erscheinenden Atom beginnen, und die restlichen Atome breadth-first durchlaufen. Falls S unzusammenhängend ist, gehen wir für jede Zusammenhangskomponente so vor. Die Atome sollen dann in der Reihenfolge dieser Numerierung eingebettet werden. Wir benutzen 9.7.5, um festzustellen, ob eine bestimmte Identifikation möglich ist.
- Wir führen eine Liste aller bisher gefundenen Einbettungen, um nicht eine Einbettung mehrfach zu zählen.

9.7.16 Die Testfunktion einer Substruktur. Gegeben sei eine Substruktur $S = (V, E)$, eine zugehörige Menge $\mathcal{R}$ von Substrukturrestriktionen, $AM = \{\vartheta_1, ..., \vartheta_n\}$, ein molekularer Graph $f \in \bar{4}^{(2^{AM[2]})}$, sowie das Intervall $I = [min; max]$ für die Anzahl der Einbettungen von S nach f.

9.7.17 Prozedur: *IsErl*

(1) **Falls** es ein Anfangsstück $\tilde{f}$ von f gibt, bei dem
S.IsErl einen Wert $ret \neq$ VIELLEICHT lieferte, return(ret).

(2) Zähle die Anzahl anz der Einbettungen von S nach f.

(3) **Falls** $anz > max$, return(FALSCH).

(4) **Falls** $anz \geq min$ und $max = \infty$, return(WAHR).

(5) **Falls** $anz < min$ und sich die Anzahl der Einbettungen für keine
Fortsetzung von f erhöhen kann, return(FALSCH).

(6) **Falls** sich die Anzahl der Einbettungen für eine Fortsetzung von f
erhöhen kann, **tue**

(7) **Falls** $anz < min$ oder $max < \infty$, return(VIELLEICHT).

ende

(8) return(WAHR).

Die Testfunktion einer Substrukturvorgabe Gegeben seien die Substrukturen $f_1 = (V_1, E_1), ..., f_n = (V_n, E_n)$ (vom Typ II). Des weiteren seien $\mathcal{R}_1, ..., \mathcal{R}_n$ Mengen von Substrukturrestriktionen. Wir betrachten nun die Funktion **IsErl** der *Substrukturvorgabe* zu den Parametern
$\{(f_1, \mathcal{R}_1, v_1, b_1), ..., (f_n, \mathcal{R}_n, v_n, b_n), v, b\}$.

9.7.18 Prozedur: *IsErl*

(1) $z_1 := 0$ und $z_2 := 0$.

(2) **Für** $i = 1$ bis n **tue**

(3) $ret := f_i.IsErl()$.
(4) **Falls** ret =WAHR, so $z_1 := z_1 + 1$ und $z_2 := z_2 + 1$.
(5) **Falls** ret =VIELLEICHT, so $z_2 := z_2 + 1$.
(6) **Falls** $z_1 > b$, return(FALSCH).
(7) **Falls** $z_2 \geq v$ und $b = n$, return(WAHR).
ende
(8) return(WAHR).

Steuerung der Konstruktionsreihenfolge durch Substrukturen Gegeben sei eine Substruktur $S = (V, E)$, eine zugehörige Menge $\mathcal{R}$ von Substrukturrestriktionen, $AM = \{\vartheta_1, ..., \vartheta_n\}$, und ein molekularer Graph $f \in \bar{4}^{(\bar{2}^{AM^{[2]}})}$. Wir möchten nun eine Zeile der Adjazenzmatrix von f ermitteln, die „aus der Sicht von S" als nächstes gefüllt werden sollte, mit der Absicht, daß wir frühzeitig möglichst viele Einbettungen von S nach f finden. Wir gehen folgendermaßen vor:

- Falls beim letzten Aufruf von **IsErl** WAHR oder FALSCH zurückgegeben wurde, so return(-1).
- Falls beim letzten Aufruf von **IsErl** eine Einbettung von S nach f aufgrund einer Substrukturrestriktion $\varrho \in R$ noch nicht gültig war, so prüfen wir zunächst, ob es aus der Sicht von ϱ einen vernünftig erscheinenden Vorschlag gibt.
- Andernfalls prüfen wir, ob beim letzten Aufruf von **IsErl** während der Substruktursuche eine teilweise Einbettung von S nach f gefunden wurde, die sich evtl. zu einer vollständigen Einbettung von S in eine Erweiterung von f ausbauen läßt. Falls dem so ist, erhalten wir dadurch einen Vorschlag für die als nächstes zu füllende Zeile.

9.8 Aromatizität

Bei der Generierung molekularer Graphen ist das Auffinden sog. aromatischer Strukturen an mehreren Stellen nötig. Zum einen ist es wünschenswert, aus der Menge der generierten Moleküle aromatische Dubletten herauszufiltern. Zu diesem Zweck müssen zunächst die aromatischen Bindungen markiert werden, sodann können mit Hilfe von kanonischer Numerierung und einer Hashtabelle die aromatischen Dubletten eliminiert werden.
Zum anderen möchte man auch die Möglichkeit haben, eine aromatische Substruktur S bei der Generierung vorzugeben (Makroatom), wobei allerdings durch die Kennzeichnung dieser Substruktur als aromatisch gewisse sich ergebende Restriktionen bzgl. der Umgebung von S im Gesamtmolekül überprüft werden.
Zur mathematischen Seite der Erkennung von Aromatizitäts- und Tautomerie-Überlagerungen siehe [14]. Das Problem, aromatische Bindungen in einem gegebenen molekularen Graphen zu markieren ist leider aus chemischer Sicht erheblich komplizierter als die zu bewältigenden mathematischen Schwierigkeiten, da Aromatizität rein phänomenologisch definiert ist. Im MOLGEN Generator bis Version 3.5 wurde (ein auch vom Autor implementierter) Algorithmus eingesetzt, der lediglich (ggf. kondensierte) Kohlenstoffringe der Längen 6,10,14,... mit alternierenden Einfach-/Doppelbindungen als aromatisch erkannte. In MOLGEN 4.0 wird nun ein Algorithmus verwendet, der wesentlich mehr chemisches Wissen benutzt, so daß nun auch Heteroaromaten sowie aromatische Ringe beliebiger Größe erkannt werden. Dabei werden auch die sich aus der Aromatizität ergebenden Distanzbedingungen geprüft.

9.8.1 Algorithmus: *Finden möglicher aromatischer Ringe*
Suche alle Ringe ohne Dreifachbindungen, wobei gilt:

- Die 4n+2 Regel ist erfüllt: Jede Doppelbindung gibt Summanden 2, (Heteroatome mit freien Elektronenpaaren (S,O,N...) oder C ohne Doppelbindungen im Ring steuern 2 Elektronen bei. Die Gesamtsumme muß kongruent 2 mod 4 sein.
- Jede Einfachbindung muß in jedem Ring an beiden Enden entweder mit einer Doppelbindung oder mit einem Heteroatom verbunden sein.
- Doppelbindungen dürfen nicht aneinanderstoßen, benachbarte Ringatome dürfen nicht beide ein Elektronenpaar beisteuern.

9.8.2 Algorithmus: *Prüfen der Distanzbedingungen*
Sei $f \in \bar{4}^{AM[2]}$ ein molekularer Graph mit $AM = \{\vartheta_1, ..., \vartheta_n\}$, sowie $\mathcal{AR} \subseteq AM$ ein Ringsystem, d.h. ein oder mehrere kondensierte Ringe, das den Bedingungen aus 9.8.1 genügt.

Hier wird getestet, ob die freien Valenzen, die $\mathcal{AR}$ besitzt, Bindungen eingehen, die bei einem aromatischen System nicht möglich sind. Es gelten folgende Voraussetzungen:

- Aromatische Ringsysteme sind planar, die beteiligten Ringe sind symmetrisch.
- Alle Bindungen haben die gleiche Länge.
- Freie Valenzen stehen „senkrecht" nach außen, d.h. sind mittelpunktssymmetrisch.
- Der Bindungswinkel zwischen einer Einfach- und einer Doppelbindung beträgt 120 Grad.
- Der Bindungswinkel zwischen einer Einfach- und einer Dreifachbindung beträgt 180 Grad.
- Der Bindungswinkel zwischen zwei Doppelbindungen beträgt 180 Grad.
- Der Bindungswinkel zwischen zwei Einfachbindungen beträgt 109,5 Grad.

(1) **Für** alle Paare $\{v_1, v_2\} \subset \mathcal{AR}$ von Atomen, die jeweils
Bindungen zu Atomen außerhalb von $\mathcal{AR}$ besitzen, **tue**
(2) $\tilde{f} := f$.
(3) Entferne aus $\tilde{f}$ alle Bindungen, die mit 9.8.1 markiert wurden.
(4) Berechne (unter den obigen Voraussetzungen) die Mindestdistanz m
von v_1 und v_2 in $\tilde{f}$.
(5) **Falls** $dist_{\tilde{f}}(v_1, v_2) < m$, return(FALSCH).
ende
(6) return(WAHR).

9.9 Beispiele

An dieser Stelle sollen nun einige Beispiele gezeigt werden, die die Leistungsfähigkeit der neuen Generierungsstrategie, also der zielgerichteten Erzeugung unterstreichen. Da wir leider zur Zeit noch über kein geeignetes Beispielmaterial aus der Praxis verfügen, müssen wir im folgenden mit theoretischen Konstruktionsproblemen, die in dieser Art aus chemischer Sicht sicher keinen Sinn ergeben, vorliebnehmen. Aufgrund unseres Konstruktionsalgorithmus ist es klar, daß die Lösungsmenge unseres Generators vollständig und redundanzfrei ist. Die Gesamtlösungsanzahl der Beispieleingaben ist daher für uns nicht so

interessant; vielmehr soll gezeigt werden, daß bei den dargestellten Konstruktionsproblemen mit vertretbarem Zeitaufwand überhaupt irgendwelche Lösungen gefunden werden, indem die geforderten Restriktionen während der Konstruktion getestet werden und gegebenenfalls die Konstruktionsreihenfolge entsprechend angepaßt wird. In der Praxis hätte man dann die Chance, durch Hinzufügen weiterer Nebenbedingungen die Lösungsmenge auf eine überschaubare Anzahl zu verkleinern. Die folgenden Beispiele wurden alle auf einem Pentium 90 gerechnet.

9.9.1 Beispiel:

- Summenformel: C_{20}
- Badlist: C=C=C

Dies ist ein recht einfaches Beispiel, das man natürlich nur behandeln kann, wenn man Badlisteinträge während des Konstruktionsprozesses berücksichtigt, denn insbesondere bei der ordnungstreuen Erzeugung werden zuerst nur solche molekularen Graphen konstruiert, die diese Teilstruktur enthalten.

Wir erhalten bereits nach einigen Sekunden mehrere hundert Lösungen.

9.9.2 Beispiel:

- Summenformel: C_{20}
- Maximale Bindungsvielfachheit: 2
- Keine Kreise der Länge 3 bis 7

Es existieren genau drei Lösungen, die wir in ca. 1 min berechnen können.

9.9.3 Beispiel:

- Summenformel: C_{20}
- Maximale Bindungsvielfachheit: 2
- Keine Kreise der Länge 4.

Nach 30 sec. sind bereits die ersten 100 Lösungen berechnet.

9.9.4 Beispiel:

- Summenformel: $C_{25}\ O_5$
- Keine Kreise der Länge 3 bis 4.
- Maximal 4 Doppelbindungen.

Nach 30 sec. sind die ersten 300 Lösungen berechnet.

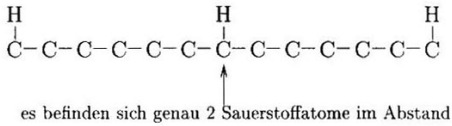
9.9.5 Beispiel:

- Summenformel: $C_{25}\ O_4\ H_8$
- Makroatom: Benzol
- Goodlist: CH_2CH_2C

Nach 30 sec. sind bereits die ersten 300 Lösungen berechnet. Man beachte, daß der Goodlisteintrag mit dem Makroatom überlappen darf, und man die Adjazenzmatrix deshalb **nicht** mit dem Goodlisteintrag vorbesetzen kann. Vielmehr wird während der Generierung geprüft, ob das aktuelle Molekülanfangsstück noch zu einer gültigen Lösung erweitert werden kann.

9.9.6 Beispiel:

– Summenformel: $C_{60}\ O_3\ N_4\ H_{20}$
– Makroatome: 3 mal



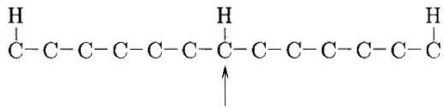
es befinden sich genau 2 Sauerstoffatome im Abstand 2

– Maximale Bindungsvielfachheit: 2
– closed substructures, d.h. keine zusätzlichen Bindungen zwischen Atomen eines Makroatoms.

Bereits nach wenigen Sekunden sind die ersten 100 Lösungen berechnet. Man beachte, daß sich im Gesamtmolekül nur drei Sauerstoffatome befinden, so daß sich die zu den Makroatomen benachbarten Sauerstoffatome also überschneiden müssen. Selbstverständlich kann man in diesem Fall auch Lösungen per Hand konstruieren, jedoch ist zu bemerken, daß es notwendig ist, eine günstige Konstruktionsreihenfolge zu benutzen, weil man mit einer unangepaßten Konstruktionsstrategie keine Lösungen finden wird.

9.9.7 Beispiel:

– Summenformel: $C_{60}\ O_3\ N_4\ H_{20}$
– Makroatome: 3 mal



es befindet sich mind. eine zshg. Substruktur
mit Formel NO im Abst. 1

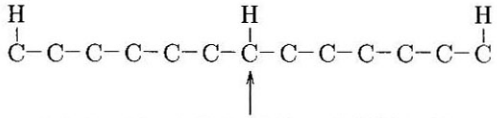
– Maximale Bindungsvielfachheit: 2
– closed substructures.

Bereits nach wenigen Sekunden sind die ersten 100 Lösungen berechnet.

9.9.8 Beispiel:

– Summenformel: $C_{50}\ O_8\ N_4\ H_{10}$

– Makroatome: 2 mal



es befindet sich mind. eine Substr. O-N-N im Abstand 1

– Maximale Bindungsvielfachheit: 2
– closed substructures.

Bereits nach wenigen Sekunden sind die ersten 100 Lösungen berechnet.

Literaturverzeichnis

1. L. Babai, P. Erdös, S. Selkow :
Random Graph Isomorphism. SIAM J. Comput., Vol.9, No.3, August 1980
2. L. Babai, L. Kucera :
Canonical labeling of graphs in linear average time. Proceedings, IEEE Symp. on Foundations of Computer Science 20 (1979), 39-46
3. C. Benecke:
Objektorientierte Darstellung und Algorithmen zur Klassifizierung bewerteter endlicher Strukturen. Dissertation Bayreuth, Prof. Dr. A. Kerber, Prof. Dr. R. Laue (1997)
4. C. Benecke, R. Grund, R. Hohberger, R. Laue, A. Kerber , Th. Wieland:
MOLGEN+, a generator of connectivity isomers and stereoisomers for molecular structure elucidation. Anal. Chim. Acta 314 (1995), 141-147
5. A. Betten:
Gruppenaktionen auf Verbänden und die Konstruktion kombinatorischer Objekte. Diplomarbeit Bayreuth, Juni 1995
6. G. Brinkmann:
Fast Generation of Cubic Graphs. Journal of Graph Theory Vol.23, No.2 (1996), 139-149
7. J. Dugundji, I. Ugi:
An Algebraic Model Of Constitutional Chemistry As A Basis For Chemical Computer Programs. Topics In Current Chemistry, Vol.39
8. M.E. Elyashberg, L.A. Gribov, Y.Z. Karasev, E.R. Martirosian:
IR Quantitative Analysis Of Organic Mixtures Without Any Callibration. Analytica Chimica Acta 353 (1997), 105-114
9. M.E. Elyashberg, Y.Z. Karasev, E.R. Martirosian, H. Thiele, H. Somberg:
Expert Systems As A Tool For The Molecular Structure Elucidation By Spectral Methods. Strategies Of Solution To The Problems. Analytica Chimica Acta 348 (1997), 443-463
10. M.E. Elyashberg, E.R. Martirosian, Y.Z. Karasev, H. Thiele, H. Somberg:
X-PERT: A User Friendly Expert System For Molecular Structure Elucidation By Spectral Methods. Analytica Chimica Acta 337 (1997), 265-286
11. K. Funatsu, N. Miyabayaski, S. Sasaki:
Further Development Of Structure generation In The Automated Structure Elucidation System CHEMICS. J. Chem. Inf. Comput. Sci., 28 (1988), 18-28
12. C.A. Brown, L. Finkelstein, P.W. Purdom:
A new Base Change Algorithm for Permutation Groups. SIAM J. COMPUT. Vol.18, No. 5 (1989), 1037-1047
13. C.J. Colbourn, R.C. Read:
Orderly Algorithms For Generating Restricted Classes Of Graphs. Journal of Graph Theory, Vol.3 (1979), 187-195

14. M. Fischer:
Algorithmen zur Erkennung von Aromatizitäts- und Tautomerie-Überlagerungen in organischen Strukturen. Diplomarbeit Bayreuth, Prof. Dr. R. Laue, in Zusammenarbeit mit IBM Heidelberg (1996)
15. R. Grund:
Konstruktion molekularer Graphen mit gegebenen Hybridisierungen und überlappungsfreien Fragmenten. Bayreuther Mathem. Schriften, 49 (1995), 1-113
16. R. Grund:
Konstruktion schlichter Graphen mit gegebener Gradpartition. Bayreuther Mathematische Schriften, Heft 44 (1993)
17. R. Grund, A. Kerber, R. Laue:
Construction of discrete structures, especially isomers. Discr. Appl. Math. 67 (1996), 115-126
18. T. Grüner, R. Laue, M. Meringer:
Applications For Group Actions Applied To Graph Generation. FINKELSTEIN L., KANTOR C., Hrsg., DIMACS Series in Discrete Mathematics And Theoretical Computer Science, Providence, RI, 1995
19. T. Grüner:
Ein neuer Ansatz zur rekursiven Erzeugung von schlichten Graphen. Diplomarbeit Bayreuth (1995)
20. T. Grüner, A. Kerber, R. Laue, M. Liepelt, M. Meringer, K. Varmuza, W. Werther:
Bestimmung von Summenformeln aus Massenspektren durch Erkennung überlagerter Isotopenmuster. MATCH 37 (1998), 163-177
21. F. Harary:
Graphentheorie. R. Oldenbourg Verlag, München Wien 1974.
22. F. Harary, E.M. Palmer:
Graphical Enumeration. Academic Press, New York/London, 1973.
23. R. Hager, A. Kerber, R. Laue, D. Moser, W. Weber:
Construction of orbit representatives. Bayreuther Mathematische Schriften 35 (1991), 157-169.
24. M. Jerrum:
A compact representation for permutation groups. J. Algorithms 7 (1986), 60-78
25. D. Jungnickel:
Graphen, Netzwerke und Algorithmen. BI-Wissenschaftsverlag (1994)
26. A. Kerber:
Algebraic Combinatorics Via Finite Group Actions. BI-Wissenschaftsverlag, Mannheim, Wien, Zürich (1991)
27. A. Kerber, R. Laue, D. Moser:
Ein Strukturgenerator für molekulare Graphen. Anal. Chim. Acta, 235 (1990), 221-228
28. D.E. Knuth:
Estimating The Efficiency Of Backtrack Programs. Mathematics of Computations. Vol.29,No.129 (1975), 121-136
29. E. Lang:
Datenstrukturen und Algorithmen für Permutationsgruppen. Diplomarbeit Bayreuth, 1992.

30. R. Laue:
Construction of Combinatorial Objects - A Tutorial. Bayreuther Mathematische Schriften 43 (1993), 53-96
31. R. Laue:
Zur Konstruktion und Klassifikation endlicher auflösbarer Gruppen. Bayreuther Mathematische Schriften 9 (1982)
32. E.G. Luks:
Isomorphism of Graphs of Bounded Valence Can Be Tested in Polynomial Time. Journal of Computer and System Science 25 (1986), 42-65
33. B.D. McKay:
Computing automorphisms and canonical labelings of graphs. Proc. Internat. Conf. on Combinatorial Theory. LN in Math. 686 Springer, Berlin (1977).
34. B.D. McKay, W. Myrvold, J. Nadon:
Fast backtracking principles applied to find new cages. Ninth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA) (1998), 188-191
35. B.D. McKay:
Isomorph-free exhaustive Generation. J. Algorithms 26, No.2 (1998), 306-324
36. B.D. McKay:
nauty User's Guide (version 1.5). Tech. Rpt. TR-CS-90-02, Dept. Computer Science, Austral. Nat. Univ. (1990)
37. M. Meringer:
Konstruktion regulärer Graphen. Diplomarbeit Bayreuth, 1995.
38. M. Meringer:
Fast Generation of Regular Graphs and Construction of Cages. erscheint im Journal of Graph Theory.
39. G. Miller:
Graph isomorphism, general remarks. J. Comp. Sys. Sci. 18 (1979), 128-142
40. S.G. Molodtsov:
The generation of molecular graphs with obligatory, forbidden and desirable fragments.. MATCH 37 (1998), 157-162
41. J. Müller:
Algorithmen zur Isomorphieerkennung bei mathematischer Darstellung chemischer Strukturen. Diplomarbeit Bayreuth, Prof. Dr. R. Laue(1994)
42. M.E. Munk:
ASSEMBLE: User Manual. Version April 1995
43. R.C. Read:
Everyone a winner. Ann. Discrete Math. 2 (1978), 107-120.
44. M. Schimmel:
Algorithmen zur Strukturerkennung, insbesondere bei Molekülen. Diplomarbeit Bayreuth, Prof. Dr. R. Laue (1993)
45. W.A. Schnyder:
Algorithmen für Normalformen von Graphen. Doktorarbeit ETH Zürich, 1983
46. C.C. Sims:
Computation with permutation groups. Proc. Second Symp. on Symbolic and Algebraic Manip. (S.R. Petrick, ed), New York 1971

47. B. Schmalz:
The t-Designs with prescribed automorpism group, new simple 6-designs. J. Combinatorial Designs 1 (1993), 125-170
48. I. Ugi, A. Dömling, B. Gruber, M. Heilingbrunner, C. Heiss, W. Hörl:
Formale Unterstützung bei Multikomponentenreaktionen - Automatisierung der Synthesechemie. MOLL R., Hrsg., Software-Entwicklung in der Chemie 9, 114-128,GDCh, Frankfurt am Main, 1995
49. K. Varmuza, W. Werther, F. Stancl, A. Kerber, R. Laue:
Computer-Assisted Structure Elucidation Of Organic Compounds, Based On Mass Spectra Classification And Exhaustive Isomer Generation. Software Development in Chemistry 10. Gesellschaft Deutscher Chemiker, Frankfurt am Main 1996
50. K. Varmuza, H. Scsibrany:
Common substructures in groups of compounds exhibiting similar mass spectra. Fresenius J. Anal. Chem. 344 (1992), 220-222
51. K. Varmuza, H. Scsibrany:
ToSiM: PC-Software for the investigation of topological similarities in molecules. Software development in Chemistry (1994), Vol.8
52. T. Wieland:
Konstruktionsalgorithmen bei molekularen Graphen und deren Anwendung. Dissertation Bayreuth (1996)
53. S. Weinrich:
Konstruktionsalgorithmen für diskrete Strukturen und ihre Implementierung. Diplomarbeit Bayreuth (1993)