Effiziente
Enumerationsalgorithmen für
Common Subtree Probleme

Andre Droschinsky

Algorithm Engineering Report
TR14-1-002
Dezember 2014
ISSN 1864-4503
Diplomarbeit

Effiziente Enumerationsalgorithmen für Common Subtree Probleme

Andre Droschinsky
24. Februar 2014

Betreuer:
Prof. Dr. Petra Mutzel
Dipl.-Inform. Nils Kriege

Fakultät für Informatik
Algorithm Engineering (Ls11)
Technische Universität Dortmund
Inhaltsverzeichnis

1 Einleitung

2 Das Maximum Common Subtree Problem
   2.1 Theoretische Grundlagen ........................................ 3
   2.2 Der Algorithmus von Edmonds .................................... 7
      2.2.1 Baumzerlegung .............................................. 8
      2.2.2 Dynamische Programmierung .................................. 9
      2.2.3 Zusammenfügen der Teillösungen .......................... 11
      2.2.4 Laufzeit und Speicherplatz .................................. 12
   2.3 Bestimmung eines MCST oder MCSTI .............................. 13
   2.4 Maximum Weight Bipartite Matching .............................. 14
      2.4.1 MaxWBM' durch Aufzählung aller maximalen Matchings .... 15
      2.4.2 Reduktion auf Maximum Weight Bipartite Perfect Matching .... 17

3 Enumeration von Common Subtrees und CSTI .......................... 23
   3.1 Grundlagen der Enumeration .................................... 23
   3.2 Verwandte Arbeiten .............................................. 25
      3.2.1 Enumeration von maximalen CCISGI .......................... 27
      3.2.2 Enumeration von Maximum Weight Bipartite Matchings ........ 28
   3.3 Enumeration von Maximum Common Subtree Isomorphismen ........ 32
      3.3.1 Enumeration von Maximum CSTI auf gewurzelten Bäumen ...... 34
      3.3.2 Mehrfache Ausgabe gleicher Isomorphismen vermeiden ........ 38
      3.3.3 Laufzeit und Speicherverbrauch ............................ 41
   3.4 Enumerationsvarianten und -kriterien ............................ 43
      3.4.1 Maximale Common Subtree Isomorphismen ..................... 43
      3.4.2 Maximale CSTI mit Mindestgröße ............................ 45
      3.4.3 Enumerationskriterien ...................................... 49
   3.5 Knoten- und Kantenbezeichner .................................. 49
      3.5.1 Enumeration von Common Weighted Subtree Isomorphismen ...... 51
      3.5.2 Enumeration von Common Labeled Subtree Isomorphismen ........ 53
3.6 Enumeration von Common Subtrees .............................................. 54
   3.6.1 Baumkanonisierung ............................................................. 55
   3.6.2 Einfügen einer Baumkanonisierung in einen sortierten Binärbaum .... 56
   3.6.3 Dynamische Programmierung ............................................... 57
   3.6.4 Zusammenfügen der Teillösungen ....................................... 57
   3.6.5 Maximale CST und maximale CST mit Mindestgröße .................. 60
   3.6.6 Laufzeit und Speicherverbrauch ....................................... 62
   3.6.7 Enumeration von Teilbäumen, auf denen ein Isomorphismus existiert 62
   3.6.8 Beschleunigungstechniken ................................................ 65
3.7 Parallelisierung ................................................................. 66

4 Experimentelle Resultate ......................................................... 69
   4.1 Common Subtree Isomorphismen ............................................. 69
      4.1.1 Ergebnisse der Programms ............................................. 70
      4.1.2 Analyse und Bewertung ............................................... 75
   4.2 Common Subtrees .............................................................. 78
      4.2.1 Ergebnisse der Programms ............................................. 78
      4.2.2 Analyse und Bewertung ............................................... 80
   4.3 Teilbaumpaare, auf denen ein Isomorphismus existiert .................. 82
      4.3.1 Ergebnisse der Programms ............................................. 82
      4.3.2 Analyse und Bewertung ............................................... 83

5 Zusammenfassung und Ausblick ................................................. 85

A Weitere Informationen .............................................................. 89
   A.1 Implementierung ............................................................. 89
   A.2 Konfigurationsdatei ......................................................... 90
   A.3 Bedienungsanleitung ........................................................ 91

Abbildungsverzeichnis .............................................................. 94

Algorithmenverzeichnis ............................................................ 95

Tabellenverzeichnis ............................................................... 97

Index ....................................................................................... 101

Literaturverzeichnis ................................................................. 103

Erklärung .................................................................................. 105
Kapitel 1

Einleitung


Kapitel 2

Das Maximum Common Subtree Problem


2.1 Theoretische Grundlagen


**Definition 2.1 (Graph [12]).** Ein Graph ist ein Paar \( G = (V, E) \) von Mengen mit der Eigenschaft \( E \subseteq [V]^2 \).

Die Elemente in \( V \) werden Knoten genannt. Die Elemente der Menge \( E \), die eine Teilmenge aller zweielementigen Teilmengen von \( V \) ist, werden als Kantens bezeichnet. Eine andere übliche Notation für die Menge \([V]^2\) ist \([V]^2 := \mathcal{P}_2(V)\). Für eine Kante \( e = \{u, v\} \) werden die Knoten \( u \) und \( v \) als inzident zu \( e \) bezeichnet und die Kante häufig abkürzend \( uv \) geschrieben. Für einen Graphen \( G = (V, E) \) sei \( V[G] := V \) und \( E[G] := E \). Ein Graph ohne Knoten wird als leerer Graph bezeichnet. Ein Graph mit \( E = [V]^2 \) wird vollständiger Graph genannt. Die Größe \( |G| = |V_G| \) eines Graphen \( G \) ist definiert als die Anzahl der Knoten des Graphen. Die Notation für die Anzahl der Kanten lautet \( ||G|| = |E_G| \). Der
Grad eines Knotens \( v \) ist durch die Anzahl der mit diesem Knoten inzidenten Kanten definiert [12].

**Bemerkung 2.2.** Der in Definition 2.1 definierte Graph wird in anderen Literaturquellen auch als *ungerichteter Graph* bezeichnet. Wenn im Folgenden der Begriff *Graph* verwendet wird, ist dabei immer, sofern nicht anders erwähnt, der ungerichtete Graph nach Definition 2.1 gemeint.

Neben ungerichteten Graphen existieren noch *gerichtete Graphen*.

**Definition 2.3 (Gerichteter Graph [5]).** Ein *gerichteter Graph* ist ein Paar \( G = (V, E) \) von Mengen mit der Eigenschaft \( E \subseteq \{(u, v) \in V \times V \mid u \neq v\} \).

Im Unterschied zum ungerichteten Graphen ist beim gerichteten Graphen eine Kante ein *geordnetes Paar* von Knoten. Die Menge der Kanten muss dabei nicht symmetrisch sein. Für eine Kante \( (u, v) \) kann die Kante \( (v, u) \) vorhanden sein, muss aber nicht. Nach der Definition von Diestel sind in einem gerichteten Graphen auch Kanten \( (u, u) \) mit \( u \in V \) erlaubt. Diese werden dort als *Schleifen* bezeichnet. Außerdem sieht die Definition in [12] vor, dass die Kantenmenge eine Multimenge ist. Das bedeutet, dass gleiche Elemente in \( E \) mehrmals vorhanden sein können. Diese Kanten werden von Diestel als *parallele Kanten* bezeichnet. In dieser Arbeit werden weder Schleifen noch parallele Kanten benötigt. Definition 2.3 impliziert, dass beides nicht vorkommt.

Eine Teilmenge aller Graphen ist die Menge der *Bäume*. Um diese definieren zu können, müssen zunächst weitere Begriffe eingeführt werden.

**Definition 2.4 (Teilgraph, induzierter Teilgraph [12]).**
Seien \( G_1 = (V_1, E_1) \) und \( G_2 = (V_2, E_2) \) Graphen. Gilt \( V_1 \subseteq V_2 \) und \( E_1 \subseteq E_2 \), so wird \( G_1 \) *Teilgraph* von \( G_2 \) genannt. Dieser Zusammenhang wird als \( G_1 \subseteq G_2 \) notiert und informell mit \( G_2 \) enthält \( G_1 \) bezeichnet. Die Notation \( G_1 \subseteq G_2 \) bedeutet, dass \( G_1 \) in \( G_2 \) enthalten ist, und \( G_2 \) mindestens einen Knoten oder eine Kante mehr als \( G_1 \) hat.

Gilt weiterhin \( E_1 = \{uv \in E_2 \mid u, v \in V_1\} \), so wird \( G_1 \) *induzierter Teilgraph* von \( G_2 \) genannt. Die Notation dazu lautet \( G_1 =: G[V_1] \).

Für einen induzierten Teilgraphen gilt somit, dass es keinen anderen Teilgraphen mit derselben Knotenmenge und mehr Kanten gibt. Anders ausgedrückt bedeutet dies, dass in einem induzierten Teilgraphen genau die Kanten fehlen, die zu fehlenden Knoten inzident sind.

**Definition 2.5 (Pfad, Kreis, azyklisch [12]).**
Ein durch die Mengen \( V = \{v_0, v_1, \ldots, v_k\}, k \geq 0 \) und \( E = \{v_0v_1, v_1v_2, \ldots, v_{k-1}v_k\} \) definierter Graph \( G \) wird als *Pfad* bezeichnet.

Die *Länge* eines Pfades ist definiert durch die Anzahl der Kanten des Pfades.
2.1. THEORETISCHE GRUNDLAGEN

Gilt $v_0 = v_k$ und ist $k \geq 3$, wird $G$ Kreis genannt.
Ein Graph, der keinen Kreis enthält, wird azyklisch genannt.


**Definition 2.6 (Zusammenhängender Graph, Zusammenhangskomponente [12]).**
Sei $G = (V, E)$ ein Graph. Wenn $G$ für alle Knoten $u, v \in V$ einen Pfad von $u$ nach $v$ enthält, wird $G$ als zusammenhängender Graph bezeichnet.

Mit diesen Begriffen lassen sich Bäume definieren.

**Definition 2.7 (Baum, Wald [12]).** Ein azyklischer Graph wird Wald genannt. Ein Wald, der zusammenhängt, wird Baum genannt.


**Definition 2.8 (Isomorphismus [12]).** Zwei Graphen $G_1 = (V_1, E_1)$ und $G_2 = (V_2, E_2)$ werden isomorph genannt, wenn eine bijektive Abbildung $\varphi : V_1 \rightarrow V_2$ existiert mit $uv \in E_1 \iff \varphi(u)\varphi(v) \in E_2$ für alle $u, v \in V_1$. Die Notation lautet dann $G_1 \simeq G_2$ und $\varphi$ wird Isomorphismus genannt.

Die Größe $|\varphi|$ eines Isomorphismus $\varphi : V_1 \rightarrow V_2$ wird als $|\varphi| := |V_1| = |V_2|$ definiert.

Falls $G_1$ zu einem Teilgraphen von $G_2$ isomorph ist, wird dies abkürzend mit $G_1$ in $G_2$ enthalten bezeichnet. Für das in der Klasse $\mathsf{NP}$ enthaltene Entscheidungsproblem, ob zwei Graphen isomorph sind, ist zum Zeitpunkt dieser Arbeit nicht bekannt, ob es in $\mathsf{P}$ enthalten ist. Es ist ebenfalls unbekannt, ob es $\mathsf{NP}$-vollständig ist [14]. Die Komplexitätsklasse $\mathsf{P}$ beinhaltet alle Probleme, die in polynomieller Zeit deterministisch entschieden werden können. In der Klasse $\mathsf{NP}$ sind die Probleme enthalten, die in polynomieller Zeit nichtdeterministisch entschieden werden können. Weiterführende Informationen zu Komplexitätsklassen finden sich beispielsweise in [3]. Ein Maximum Common Subtree lässt sich wie folgt definieren.
KAPITEL 2. DAS MAXIMUM COMMON SUBTREE PROBLEM

Definition 2.9 (Common Subtree (CST), Maximum CST [21]). Seien $T_1$ und $T_2$ Bäume. Ein Baum $T$ ist ein Common Subtree von $T_1$ und $T_2$, wenn $T$ isomorph zu Teilbäumen von $T_1$ und $T_2$ ist.

$T$ wird Maximum Common Subtree genannt, wenn es keinen anderen Common Subtree von $T_1$ und $T_2$ gibt, der größer als $T$ ist.


Definition 2.10 (Common Subtree Isomorphismus (CSTI), Maximum CSTI). Seien $R$ und $S$ Bäume, und $R' = (V'_R, E'_R)$ und $S' = (V'_S, E'_S)$ Teilbäume dieser Bäume. Wenn $R'$ und $S'$ isomorph sind, wird ein zugehöriger Isomorphismus $\varphi : V'_R \rightarrow V'_S$ (vgl. Def. 2.8) als Common Subtree Isomorphismus (CSTI) bezeichnet.

Falls $\varphi$ grösstmöglich ist, wird $\varphi$ Maximum Common Subtree Isomorphismus (MCSTI) genannt.

Common Subtree und Common Subtree Isomorphimus stehen nah zusammen. Aus einem CSTI lässt sich ein CST direkt ablesen. Mit den Bezeichnungen aus Definition 2.10 ist es ein zu $R'$ bzw. $S'$ isomorpher Baum. Andererseits lässt sich zu jedem Common Subtree $T = (V, E)$ mindestens ein CSTI finden. Abbildung 2.1 zeigt diesen Zusammenhang. Zu dem dort dargestellten Maximum CSTI $\varphi$ zwischen $R$ und $S$, definiert durch $\varphi(r_1) = s_1$, $\varphi(r_2) = s_2$, $\varphi(r_3) = s_3$, ist der zugehörige Maximum CST $T$ dargestellt. Zu diesem Maximum CST gibt es neben $\varphi$ beispielsweise noch den MCSTI $\varphi'(r_3) = s_2$, $\varphi'(r_2) = s_3$, $\varphi'(r_4) = s_4$. Offensichtlich gilt, dass die Größe eines MCST und MCSTI identisch ist.

Neben Maximum CSTI existieren auch maximale CST, die wie folgt definiert sind.

Definition 2.11 (Maximaler CST, Maximaler CSTI [17]). Sei $\varphi$ ein Common Subtree Isomorphismus von Bäumen $T_1$ und $T_2$. Dieser wird maximaler CSTI genannt, wenn er nicht zu einem größeren CSTI von $T_1$ und $T_2$ erweiterbar ist.
Sei $T$ ein Common Subtree. $T$ wird \textit{maximaler CST} genannt, wenn es keinen anderen Common Subtree $T'$ von $T_1$ und $T_2$ gibt mit $T \subset T'$.


Graphen können neben ihrer Struktur aus Knoten und Kanten weitere Informationen enthalten. Ein Beispiel dazu sind \textit{Bezeichner}, die auch \textit{Label} genannt werden.

\textbf{Definition 2.12 (Gelabelter Graph \cite{9}).} Ein \textit{gelabelter Graph} ist ein Graph $G = (V, E)$ zusammen mit einer Abbildung $l : V \cup E \to \Sigma$.

Die Elemente der endlichen Menge $\Sigma$ werden \textit{Label} oder \textit{Bezeichner} genannt. Für einen Knoten $v \in V$ ist $l(v)$ der Bezeichner von $v$. Für eine Kante $e \in E$ ist $l(e)$ der Bezeichner von $e$. Die Abbildung $l$ wird im Folgenden \textit{Labelfunktion} genannt. Der Begriff des gelabelten Graph lässt sich auf Teilklassen von Graphen, beispielsweise Bäume, übertragen. Die Definitionen werden entsprechend um die Abbildung $l$ erweitert.

Zu erwähnen sind die vielen unterschiedlichen Definition von gelabelten Graphen in der Literatur. So muss der Graph nach Bunke \cite{6} vollständig sein. In \cite{2} und vielen anderen Quellen haben nur Knoten Bezeichner. Sollen nur die Knoten oder nur die Kanten unterschiedlich sein, so kann jeweils allen Kanten oder allen Knoten das gleiche Label gegeben werden. Ein Beispiel für die Menge $\Sigma$ ist die Menge aller Atome und Bindungstypen zwischen Atomen. Auf diese Art lässt sich ein Molekül dann durch einen gelabelten Graphen darstellen. Die Begriffe Isomorphismus und Common Subtree Isomorphismus lassen sich wie folgt auf gelabelten Graphen definieren.

\textbf{Definition 2.13 (Isomorphismus auf gelabelten Graphen \cite{6}).} Seien $G_1 = (V_1, E_1)$ und $G_2 = (V_2, E_2)$ Graphen. Seien weiterhin $l_1 : V_1 \cup E_1 \to \Sigma$ und $l_2 : V_2 \cup E_2 \to \Sigma$ Labelfunktionen. $G_1$ und $G_2$ werden \textit{isomorph bezüglich} $l_1$ und $l_2$ genannt, wenn eine Isomorphismus $\varphi : V_1 \to V_2$ existiert mit $l_1(v) = l_2(\varphi(v))$ für alle Knoten $v \in V_1$ und $l_1(uv) = l_2(\varphi(u)\varphi(v))$ für alle Kanten $uv \in E_1$.

Bei einem Isomorphismus auf gelabelten Graphen müssen folglich die aufeinander abgebildeten Knoten und Kanten die gleichen Bezeichner haben. Bunke \cite{6} verwendet in Bezug auf gelabelte Graphen für Knoten und Kanten getrennte Labelfunktionen. Die Definition des Isomorphismus auf gelabelten Graphen unterscheidet sich somit marginal von der hier gegebenen Definition.

\section{2.2 Der Algorithmus von Edmonds}

In diesem Abschnitt wird der Algorithmus von Edmonds und Matula \cite{21} vorgestellt. Bei Eingabe zweier Bäume $R = (V_R, E_R)$ und $S = (V_S, E_S)$ bestimmt dieser Algorithmus die
Größe eines MCST bzw. MCSTI der gegebenen Bäume. Matula schreibt „A polynomial bounded procedure for solving the subtree isomorphism problem was independently discovered by Edmonds\textsuperscript{1} and Matula [20] in 1968 [...] Edmond’s solution [...] resolved the more general question of determining the largest subtree of $T$ isomorphic to a subtree of $S$.“ [21]. Da sich diese Arbeit vorwiegend mit der generelleren Methode von Edmonds beschäftigt, wird der Algorithmus im Folgenden mit Algorithmus von Edmonds bezeichnet.

Im Algorithmus wird die Baumeigenschaft, dass ein Baum bei Wegnahme einer Kante in genau zwei Zusammenhangskomponenten zerfällt, ausgenutzt. Mit Hilfe von dynamischer Programmierung auf diesen Komponenten wird daraus die Größe eines MCST für die Bäume der Eingabe bestimmt. Die einzelnen Schritte werden im Folgenden vorgestellt.

2.2.1 Baumzerlegung


**Definition 2.14 (Gewurzelter Baum [12]).** Ein *gewurzelter Baum* ist ein Baum mit einem ausgezeichneten Wurzelknoten.

Für zwei benachbarte Knoten $u$, $v$ in einem gewurzelten Baum herrscht eine *Elter-Kind-Beziehung*. Dabei ist $u$ Elter von $v$, wenn die Länge des Pfades von $u$ zur Wurzel kürzer ist als die Länge des Pfades von $v$ zur Wurzel, sonst ist $u$ Kind von $v$. Knoten, die keine Kinder besitzen, werden als Blatt bezeichnet.

In Abbildung 2.2 ist ein gewurzelter Baum mit Wurzel $r_1$ dargestellt. Der Knoten $r_2$ ist Kind von $r_1$ und Elter von $r_3$ und $r_4$. Die Knoten $r_3$ und $r_4$ sind Blätter des gewurzelten Baumes.

**Definition 2.15 (Gewurzelter Teilbaum [21]).** Sei $T$ ein Baum und $e = uv$ eine beliebige Kante des Baumes, die aus diesem entfernt wird. Die Zusammenhangskomponenten werden als *gewurzelte Teilbäume* $T^u_v$ bzw. $T^w_u$ bezeichnet. Die Komponente $T^u_v$ besitzt den Knoten $v$ als Wurzel, $T^w_u$ den Knoten $u$.

\textsuperscript{1}laut Matulas persönlicher Kommunikation mit Edmonds [21]
2.2. DER ALGORITHMUS VON EDMONDS

Abbildung 2.3: Der gewurzelte Teilbaum $T^u_v$ ist hellgrün dargestellt. $T^u_v$ ist dunkelgrün dargestellt. Die gestrichelte Linie in rot ist die Trennungskante.

In Abbildung 2.3 ist die Zerlegung eines Baumes $T$ in die zwei gewurzelten Teilbäume $T^u_v$ und $T^v_u$ dargestellt. Ein CSTI auf zwei gewurzelten Teilbäumen wird im Folgenden manchmal abkürzend Isomorphismus genannt. Dabei ist aber stets klar, dass ein maximaler CSTI bzw. MCSTI gemeint ist.

2.2.2 Dynamische Programmierung

Im zweiten Schritt des Algorithmus von Edmonds wird für alle Paare $(R^u_v, S^w_x)$ von gewurzelten Teilbäumen die Größe $D(R^u_v, S^w_x)$ des zugehörigen MCSTI bestimmt. Einschränkend gilt dabei, dass in den Isomorphismen jeweils $v$ auf $x$ abgebildet wird. Es gibt $4 \cdot |E_R| \cdot |E_S|$ solcher Paare. Diese Anzahl ergibt sich aus $2 \cdot |E_R|$ gewurzelten Teilbäumen für $R$ und $2 \cdot |E_S|$ für $S$. Zur Bestimmung der Werte $D(R^u_v, S^w_x)$ kann auf ein Maximum Weight Bipartite Matching zurückgegriffen werden. In Abschnitt 2.4 wird dieses Matching beschrieben und verschiedene Lösungsmöglichkeiten vorgestellt.

Seien $\{v_1, v_2, \ldots, v_k\}$ die Kinder von $v$ und $\{x_1, x_2, \ldots, x_l\}$ die Kinder von $x$. Sei $M$ ein Maximum Weight Bipartite Matching auf dem vollständigen bipartiten Graphen zwischen den Kindern von $v$ und $x$ mit den Kantengewichten $w(v_i x_j) = D(R^u_v, S^w_x)$ für alle $i \in \{1, \ldots, k\}$ und $j \in \{1, \ldots, l\}$ (vgl. Definition 2.19). Falls $k = 0$ oder $l = 0$ ist, also mindestens einer der beiden gewurzelten Teilbäume aus nur einem Blatt besteht, gilt $D(R^u_v, S^w_x) = 1$. Ansonsten gilt $D(R^u_v, S^w_x) = 1 + W(M)$ für das oben genannte Matching. Jeder gewurzelte Teilbaum $R^u_v$ besitzt mindestens einen Knoten weniger als $R^w_u$ und jeder Teilbaum $S^w_x$ mindestens einen weniger als $S^w_x$. $D$ ist somit wohldefiniert.

In Abbildung 2.4 sind zwei gewurzelte Teilbäume $R^u_v$ und $S^w_x$ und der zugehörige bipartite Graph dargestellt, auf dem das Maximum Weight Bipartite Matching bestimmt werden soll. Die Kantengewichte zwischen den Kindern von $v$ und $x$ lauten $w(v_2 x_2) = 3$, $w(v_2 x_3) = 2$, sowie $w(v_i x_j) = 1$ für alle anderen Kanten. Daraus ergibt sich $D(R^u_v, S^w_x) = 5$. Dies wird durch $\varphi(v) = x$ und beispielsweise $\varphi(v_1) = x_1$ und $\varphi(v_2) = x_2$ realisiert.
KAPITEL 2. DAS MAXIMUM COMMON SUBTREE PROBLEM

Abbildung 2.4: Der Wert $D(R_u^v, S_w^x)$ wird mit Hilfe eines MaxWBM bestimmt.

Algorithmus 2.1: GetD($R_u^v, S_w^x$)


Falls der Wert $D(R_u^v, S_w^x)$ bereits berechnet wurde, wird dieser von Algorithmus GetD sofort ausgegeben. Ansonsten wird der Wert auf 1 gesetzt, falls einer der Teilbäume ein Blatt ist, oder auf $1 + W(M)$ für das weiter oben beschriebene Maximum Weight Bipartite Matching. Die Struktur des Algorithmus GetD stellt sicher, dass trotz Rekursion kein Matching mehrmals berechnet wird, unabhängig davon, wie oft GetD aufgerufen wird


### 2.2.3 Zusammenfügen der Teillösungen

Im vorherigen Abschnitt 2.2.2 wird die Größe eines MCSTI für zwei gewurzelte Teilbäume mit der Einschränkung, dass die Wurzeln aufeinander abgebildet werden, berechnet. Im letzten Schritt des Algorithmus von Edmonds wird daraus die Größe eines MCSTI für die gegebenen Bäume $R$ und $S$ der Eingabe bestimmt. Dies wird im Folgenden beschrieben.

Seien $R^v_u$ und $S^w_x$ zwei gewurzelte Teilbäume. Dann liefert, wie in Abschnitt 2.2.2 beschrieben, $\text{GetD}(R^v_u, S^w_x)$ die Größe eines MCSTI der beiden Teilbäume $R^v_u$ und $S^w_x$ mit der Einschränkung, dass $v$ auf $x$ abgebildet wird. $\text{GetD}(R^v_u, S^w_x)$ ist die Größe eines MCSTI der beiden Teilbäume $R^v_u$ und $S^w_x$ mit der Einschränkung, dass $u$ auf $w$ abgebildet wird. Zusammengenommen ergibt das die Größe eines MCSTI der Bäume $R$ und $S$ mit der Einschränkung, dass $v$ auf $x$ und $u$ auf $w$ abgebildet wird.

Abbildung 2.5 zeigt diesen Zusammenhang. Dort gilt $D(R^v_u, S^w_x) = 2$, dargestellt durch die hellgrünen und hellblauen Knoten. Weiterhin gilt $D(R^v_u, S^w_x) = 2$. Das ist durch die dunkel gefärbten Knoten dargestellt. Ein MCSTI mit $\phi(v) = x$ und $\phi(u) = w$ hat folglich die Größe 4. Zu beachten ist, dass die Größe eines MCSTI ohne Einschränkung an den Isomorphismus bei den dort dargestellten Bäumen 5 beträgt. Letztgenannte Größe lässt sich nach Edmonds mit dem folgenden Algorithmus 2.2 berechnen.

In Zeile 1 bis 4 werden die Randfälle, dass einer der beiden Bäume aus einem oder keinem Knoten besteht, behandelt. Anschließend wird das Maximum über alle wie oben beschriebenen eingeschränkten MCSTI gebildet und ausgegeben. Dieses entspricht der Größe eines MCSTI der Bäume $R$ und $S$ der Eingabe. Wenn in Zeile 7 ein Paar $(R^v_u, S^w_x)$ gewählt wurde, muss natürlich das Paar $(R^v_u, S^w_x)$ nicht mehr gewählt werden, da der Lösungswert offensichtlich identisch ist.
KAPITEL 2. DAS MAXIMUM COMMON SUBTREE PROBLEM

Abbildung 2.5: Ein Maximum CSTI mit der Einschränkung $\varphi(u) = w$ und $\varphi(v) = x$

**Eingabe:** Zwei Bäume $R$ und $S$

**Ausgabe:** Größe eines MCSTI von $R$ und $S$

1. if $R$ oder $S$ ist leer then
2. return 0
3. else if $R$ oder $S$ besteht aus nur einem Knoten then
4. return 1
5. else
6. $m \leftarrow 0$ // Bisher gefundenes Maximum
7. for all Paare $(R_u, S_w)$ von gewurzelten Teilbäumen do
8. $s \leftarrow \text{GetD}(R_u, S_w) + \text{GetD}(R_v, S_x)$
9. if $s > m$ then
10. $m \leftarrow s$
11. end if
12. end for
13. return $m$
14. end if

Algorithmus 2.2: SizeMCSTI($R, S$)

### 2.2.4 Laufzeit und Speicherplatz

Die Laufzeit von Algorithmus 2.2 (SizeMCSTI) wird durch die Schleife über die Zeilen 7 bis 12 bestimmt. Diese wird $O(|E_R| \cdot |E_S|)$ mal durchlaufen. Nach den obigen Überlegungen muss in Algorithmus 2.1 (GetD) $O(|E_R| \cdot |E_S|)$ mal ein MaxWBM bestimmt werden. Nach Abschnitt 2.4.2 ist die Berechnungszeit dabei jeweils durch $O(\max\{|E_R|, |E_S|\}^3)$ beschränkt. Der Speicherbedarf liegt für die Werte von $D$ bei $O(|E_R| \cdot |E_S|)$. Ein MaxWBM kann nach der Berechnung verworfen werden, da nur die Größe relevant ist. Für die Berechnung eines Matchings reicht $O(|E_R| \cdot |E_S|)$ Speicherplatz aus, wenn die bipartiten Graphen in einer Adjazenzmatrix gespeichert werden. Daraus ergibt sich folgender Satz.
2.3 BESTIMMUNG EINES MCST ODER MCSTI

Abbildung 2.6: Bild zur Laufzeitabschätzung in Satz 2.16

Satz 2.16. Die Größe eines Maximum Common Subtree Isomorphismus zweier Bäume $R = (V_R, E_R)$ und $S = (V_S, E_S)$ lässt sich in Zeit $O(|E_R| \cdot |E_S| \cdot \max\{|E_R|, |E_S|\})$ bei zusätzlichem Speicherbedarf von $O(|E_R| \cdot |E_S|)$ bestimmen.

Eine bessere Abschätzung in Satz 2.16 ist nicht möglich, wie Abbildung 2.6 zu entnehmen ist. Nach Definition 2.24 gilt $|V| = |X| = \max\{k, l\}$ für die den $k \cdot l$ Paaren $(R^i_r, S^j_s)$, $i \in \{1, \ldots, k\}$, $j \in \{1, \ldots, l\}$ zugehörigen bipartiten Graphen.

2.3 Bestimmung eines MCST oder MCSTI


In Zeile 8 in Algorithmus 2.1 wird das Gewicht eines Maximum Weight Bipartite Matchings bestimmt. Wie in Abschnitt 2.4 dargestellt, ergibt sich das Gewicht aus einem konkreten Matching. Dieses wird in dem modifizierten Algorithmus für jedes Paar von gewurzelten Bäumen gespeichert. Der dazu nötige Speicherplatz beträgt $O(|R| \cdot |S| \cdot \min\{|R|, |S|\})$. Die ersten beiden Faktoren ergeben sich aus der Anzahl der Paare gewurzelter Teilbäume, der letzte Faktor aus der maximalen Anzahl an Matchingkanten für ein Paar von gewurzelten Teilbäumen.

Um ein MCSTI auszugeben, wird auf Algorithmus 2.2 zurückgegriffen. Dieser wird insofern erweitert, dass neben dem Maximum $m$ auch die Knoten $u, v, w, x$ gespeichert werden, die zu diesem Maximum geführt haben. In Zeile 13 wird dann nicht $m$ ausgegeben sondern zunächst $\varphi(u) = w, \varphi(v) = x$. Anschließend werden rekursiv entlang beider Paare gewurzelter Bäume (vgl. Zeile 8 von Algorithmus 2.2) die durch die gespeicherten Matchings definierten Knotenzuordnungen ausgegeben. Es ist offensichtlich, dass auf diese Weise ein Maximum Common Subtree Isomorphismus ausgegeben werden kann. Die Zeitschranke entspricht der des ursprünglichen Algorithmus von Edmonds.

Ein Maximum Common Subtree lässt sich sehr ähnlich bestimmen. Im auf Definition 2.10 folgenden Absatz wurde bereits erwähnt, wie sich aus einem MCSTI ein MCST ge-
winnen lässt. Es muss also nur die Ausgabe verändert werden, der Rest des modifizierten Algorithmus bleibt identisch. Dies wird an dieser Stelle deshalb nicht weiter ausgeführt.

Es ist möglich, den zusätzlichen Speicherverbrauch von $O(|R| \cdot |S| \cdot \min\{|R|, |S|\})$ zu vermeiden. Dazu wird zunächst der unmodifizierte Algorithmus 2.2 aufgerufen, die Matchings werden also nicht gespeichert. Die Ausgabe des Isomorphismus erfolgt dann rekursiv wie oben beschrieben. Da die Zuordnungen nicht gespeichert wurden, werden die entsprechenden Matchings ein zweites mal berechnet und die Zuordnungen dann direkt ausgegeben. Die Gesamtzeit des Algorithmus bleibt damit bezüglich O-Notation identisch. Analog lässt sich ein Maximum Common Subtree ausgeben.

2.4 Maximum Weight Bipartite Matching

Im folgenden wird das in Abschnitt 2.2.2 erwähnte Maximum Weight Bipartite Matching behandelt. Dazu werden zunächst die Begriffe bipartiter Graph und Matching definitiert.

**Definition 2.17 (Bipartiter Graph [12])**. Sei $G = (V, E)$ ein Graph. Wenn disjunkte Mengen $V_1$ und $V_2$ existieren, so dass $V = V_1 \cup V_2$ und alle Kanten $e \in E$ nur zwischen diesen Mengen verlaufen, wird $G$ bipartit genannt.

Ein bipartiter Graph $G$ ist vollständig, wenn gilt $E = V_1 \times V_2$.

Häufig wird in einem bipartiten Graphen die Zerlegung von $V$ direkt angegeben. $G = (V_1 \cup V_2, E)$ bedeutet dann, dass Kanten nur zwischen $V_1$ und $V_2$ verlaufen, ohne dass dies explizit erwähnt wird.

**Definition 2.18 (Matching [12])**. Sei $G = (V, E)$ ein Graph. Eine Teilmenge $M \subseteq E$ der Kanten des Graphen wird Matching genannt, wenn die Kanten in $M$ unabhängig sind, das heißt für zwei beliebige Kanten $uv, wx$ aus $M$ gilt, dass die Knoten $u, v, w, x$ paarweise voneinander verschieden sind.


**Definition 2.19 (Maximum Weight Bipartite Matching (MaxWBM) [8])**. Sei $G = (V_1 \cup V_2, E)$ ein bipartiter Graph mit $E \subseteq V_1 \times V_2$ und $w : E \rightarrow \mathbb{Q}$ eine Gewichtsfunktion, die jeder Kante ein Gewicht zuordnet. Das Gewicht $W(M)$ eines Matchings $M \subseteq E$ ist definiert als $W(M) = \sum_{e \in M} w(e)$.

Ein Matching $M$ im bipartiten Graphen $G$ wird Maximum Weight Bipartite Matching genannt, wenn es kein anderes Matching $M'$ in $G$ gibt mit $W(M') > W(M)$.
2.4. MAXIMUM WEIGHT BIPARTITE MATCHING

Da im Rahmen dieser Arbeit nur Matchings auf bipartiten Graphen berechnet werden, wird ein MaxWBM auch als Maximum Matching bezeichnet.

Satz 2.20. Wenn alle Gewichte in einem gegebenen bipartiten Graphen mit Gewichtsfunktion positiv sind, ist jedes Maximum Weight Bipartite Matching in diesem Graphen maximal.

Beweis. Beweis durch Widerspruch. Angenommen $M$ sei ein Maximum Weight Bipartite Matching, das nicht maximal ist. Dann existiert eine Kante $e$, die zu $M$ hinzugefügt werden kann, so dass $M \cup \{e\}$ ein Matching ist. Dann gilt $W(M \cup \{e\}) - W(M) = w(e) > 0$. Dies ist ein Widerspruch dazu, dass $M$ ein MaxWBM ist. Die Annahme, dass $M$ nicht maximal ist, ist also falsch. □

In Zusammenhang mit Algorithmus 2.1 entspricht ein MaxWBM einer Zuordnung der Kinder, so dass der CSTI für die gewurzelten Teilbäume ein MCSTI ist. Im Folgenden werden zwei Möglichkeiten vorgestellt, ein MaxWBM zu bestimmen. Das erste Verfahren nimmt Bezug auf Algorithmus 2.1 und stellt kein Lösungsverfahren für beliebige bipartite Graphen dar. Dieses auf Algorithmus 2.1 zugeschnittene MaxWBM wird deshalb im Folgenden $MaxWBM'$ genannt. Der zugehörige bipartite Graph ist insbesondere vollständig.

2.4.1 $MaxWBM'$ durch Aufzählung aller maximalen Matchings

Der im Folgenden vorgestellt Algorithmus zum systematischen Aufzählen aller maximalen Matchings, um ein $MaxWBM'$ zu finden, stammt vom Verfasser dieser Arbeit.

Sei $G = (V \cup X, E)$ ein vollständiger bipartiter Graph mit $1 \leq |V| \leq |X|$. Die letzte Bedingung ist keine Einschränkung. Sowohl $V$ als auch $X$ sind im Algorithmus von Edmonds niemals leer, denn dort wird nur dann ein MaxWBM bestimmt, wenn beide gewurzelten Teilbäume mindestens ein Kind haben. Sollte $k > l$ (vgl. Algorithmus 2.1) sein, können $V$ und $X$ vertauscht werden. Nach Satz 2.20 müssen nur maximale Matchings aufgezählt werden, da alle Gewichte positiv sind. Für jedes maximale Matching $M \subseteq E$ gilt, dass alle Knoten aus $V$ zu einer Kante aus $M$ inzident sind. Dies folgt unmittelbar aus $E = V \times X$ und $|V| \leq |X|$. Im Folgenden soll ein Matching $M$ durch eine Folge $(z_i)_{1 \leq i \leq k := |V|}$ von natürlichen Zahlen $z_i \in \{1, 2, \ldots, l := |X|\}$ beschrieben werden. Die Folge $(z_i)$ steht für das Matching $\{v_1x_{z_1}, v_2x_{z_2}, \ldots, v_kx_{z_k}\}$. Aufgrund der Matchingeigenschaft gilt $z_i \neq z_j$ für $i \neq j$. Das Problem der Aufzählung aller maximalen Matchings kann somit reduziert werden auf die Aufzählung aller möglicher injektiver Folgen $(z_i)$. Diese soll lexicographisch erfolgen.

Beispiel 2.21. Sei $k = 3$ und $l = 4$. Die Folgen in lexikographischer Ordnung lauten dann $((1,2,3),(1,2,4),(1,3,2),(1,3,4),\ldots,(4,2,3),(4,3,1),(4,3,2))$. Diese stehen für die Matchings $\{v_1x_1, v_2x_2, v_3x_3\}, \{v_1x_1, v_2x_2, v_3x_4\}, \ldots, \{v_1x_4, v_2x_3, v_3x_2\}$.
Satz 2.22. Das Gewicht eines MaxWBM' kann durch lexikographische Aufzählung in Zeit $O\left(\frac{n}{(l-k)!}\right)$ bei zusätzlichem Speicherbedarf von $O(k+l)$ bestimmt werden.

Beweis. Zur Berechnung des Gewichts ab dem zweiten Matching werden nicht alle Kantengewichte aufsummiert. Stattdessen werden nur die Veränderungen zum vorherigen Matching berücksichtigt. Sei $(z'_j)$ lexikographisch auf $(z_j)$ folgend. Wenn dann $z_j$ durch $z'_j$ für ein oder mehrere $j$ ersetzt wird, wird zum aktuellen Gewicht $w(v_jx'_j) - w(v_jx_j)$ für jedes dieser $j$ addiert. Wie in Beispiel 2.21 zu sehen, ändern sich vom letzten Glied beginnend eine oder mehrere Zahlen. Beispielsweise ändern sich zwischen den Folgen $(4,2,3)$ und $(4,3,1)$ die letzten beiden Folgenglieder. Der Zeitaufwand pro Veränderung beträgt $O(1)$ bei einmaliger Initialisierungszeit $O(k+l)$ (vgl. letzter Absatz dieses Beweises). Die Laufzeit hängt damit linear von $k$ für die erste Folge, $k + l$ für die Initialisierung, sowie der Anzahl der Veränderungen zwischen allen Folgen ab. Die Anzahl der Änderungen wird im Folgenden abgeschätzt.

Die Anzahl verschiedener maximaler Matchings beträgt $\frac{n}{(l-k)!}$. Dies ergibt sich kombinatorisch daraus, dass der Knoten $v_1$ mit $l$ Knoten aus $X$ verbunden werden kann, $v_2$ mit $l-1$ Knoten und so weiter. Beim Wechsel von $(z_j)$ zu $(z'_j)$ verändert sich das letzte Folgenglied immer. Falls $k = l$ ändert sich auch das vorletzte Folgenglied immer. Das drittletzte Folgenglied wechselt im worst case $k = l$ bei jeder zweiten Folge. Ein Beispiel sei $k = l = 3$ und die Folgen $\{(1,2,3),(1,3,2),(2,1,3),\ldots\}$. Im Fall $k < l$ sind es mehr als zwei Folgen, bis sich das drittletzte Glied ändert, vgl. Beispiel 2.21. Allgemein braucht es mindestens $n!$ Folgen für das $(k-n)$-te Folgenglied, bis sich dieses ändert. Dies lässt sich mit vollständiger Induktion beweisen. Die Gesamtzahl der Änderungen ist damit durch $\frac{n}{(l-k)!} \cdot (1 + 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \ldots) \leq \frac{n}{(l-k)!} \cdot (2 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \ldots) \leq \frac{n}{(l-k)!} \cdot 3$ beschränkt.

Mit der konstanten Zeit $O(1)$ pro Änderung (vgl. den folgenden Absatz) ergibt dies eine Zeitschanke von $O\left(\frac{n}{(l-k)!}\right)$. Die Initialisierungszeit und die Zeit für das erste Matching, $O((k+l) + k)$, ist darin enthalten.

Die Grundidee zur Veränderung der Folgenglieder in $O(1)$ liegt darin, in einem Feld $N[0..l]$ den jeweils nächsten exponentierten Knoten aus $X$ festzuhalten. $N[0]$ steht dabei für den ersten (kleinster Index) exponentierten Knoten. $N[m] = n$ bedeutet dabei, dass der nächste exponentierte Knoten nach $x_m$ der Knoten $x_n$ ist. Falls $N[m] = 0$, gibt es keinen exponentierten Knoten $x_n$ mit Index $n > m$. Pro Veränderung eines Folgenglieds müssen zwei Einträge in $N$ aktualisiert werden. Wenn $z_j =: m$ auf $n := z'_j > z_j$ gesetzt wird, muss $N[m] = n$ auf $N[n]$ gesetzt werden. Dies liegt daran, dass der nächste exponentierte auf $x_m$ folgende Knoten dann nicht mehr der Knoten $x_n = x'_j$ sondern $x_{N[n]}$ ist. Andererseits ist durch die Änderung von $z_j$ zu $z'_j$ der Knoten $x_m$ exponentiert. Dieser muss also in $N$ wieder gefunden werden können. Dazu wird in einem Feld $V[1..k]$ der Index desjenigen Knoten $x_r$ gespeichert, für den $N(r) = m$ gilt, bevor $x_m$ gematched wurde. Für $z_j$ wird dabei der Wert $r$ in $V[j]$ gespeichert. $N[V[j]]$ wird deshalb auf $m$ gesetzt. Auf diese Weise steht in $N$
immer der korrekte Index des nächsten exponierten Knoten aus \( X \). Wie in diesem Absatz beschrieben sind dazu jeweils 2 Änderungen an \( N \) und eine an \( V \) nötig, der Zeitbedarf ist folglich \( O(1) \) pro Veränderung eines Folgenglieds. \( \square \)


Die Laufzeit in Satz 2.22 ist offenbar nicht durch ein Polynom in \( k \) und \( l \) beschränkt. Es existieren allerdings Algorithmen für MaxWBM mit polynomieller Laufzeit, wie Abschnitt 2.4.2 zu entnehmen ist. Es gibt aber zwei Gründe, weshalb dieser Algorithmus dennoch vorgestellt wurde. Zum einen wird in Kapitel 3 die Aufzählung aller maximalen Common Subtree Isomorphismen behandelt. Ein Teilproblem in diesem Algorithmus ist die Aufzählung aller maximalen Matchings. Der in diesem Abschnitt vorgestellte Algorithmus kann so modifiziert werden, dass jedes Matching ausgegeben bzw. in einer geeigneten Datenstruktur gespeichert wird. Die dazu nötige Zeit beträgt im Durchschnitt lediglich \( O(1) \) pro weiterem Matching. Zum anderen sind viele Graphen in der Praxis dünn besetzt. Beispielsweise gilt für ein Straßenetz mit den Knoten als Kreuzungen und Kanten als Straßen, dass der Knotengrad bis auf sehr wenige Ausnahmen kleiner als 5 ist. Ein anderes Beispiel sind chemischen Struktur-Graphen. Diese sind nach J. Raymond dünn besetzt: „Since 2D chemical graphs are very sparse (i.e., the constituent vertices are of low degree), the number of edges is approximately equal to the number of vertices (i.e., \( |E(G)| \approx O(|V(G)|) \)“ [25]. Ebenso weisen die dieser Arbeit zur Verfügung stehenden Feature Trees häufig einen niedrigen Knotengrad auf. In all diesen Fällen ist \( \frac{n}{(l-k)!} \) sehr häufig „klein“.

### 2.4.2 MaxWBM durch Reduktion auf Maximum Weight Bipartite Perfect Matching

Zunächst werden die Begriffe Perfektes Matching und Maximum Weight Bipartite Perfect Matching definiert.

**Definition 2.23 (Perfektes Matching [27]).** Existieren in einem Graphen \( G \) für ein Matching \( M \) keine \( M \)-exponierten Knoten, wird dieses perfekt genannt.
Definition 2.24 (Maximum Weight Bipartite Perfect Matching [8]).

Sei $G = (V \cup X, E)$ ein bipartiter Graph mit $|V| = |X|$ und einer zugehörigen Gewichtsfunktion. Ein Matching $M$ wird Maximum Weight Bipartite Perfect Matching (MaxWBPM) genannt, wenn es kein anderes perfektes Matching $M'$ mit größerem Gewicht gibt.

Neben der Aufzählung aller maximalen Matchings besteht eine weitere Möglichkeit, ein MaxWBM zu finden, darin, den bipartiten Graphen in einen anderen bipartiten Graphen zu transformieren, auf dem dann ein MaxWBPM anstelle eines MaxWBM gesucht wird. Die Transformation stellt sicher, dass das Gewicht für beide Matchings gleich groß ist. Die Transformation sowie die Bestimmung eines MaxWBPM wird im Folgenden beschrieben.

Ein perfektes Matching in einem bipartiten Graphen $G = (V \cup X, E)$ kann nur dann existieren, wenn $|V| = |X|$, da Knoten aus $V$ nur mit Knoten aus $X$ verbunden werden können und umgekehrt. Sei $V = \{v_1, v_2, \ldots, v_k\}$ und $X = \{x_1, x_2, \ldots, x_l\}$ mit $k \leq l$ und $E = V \times X$. Die Voraussetzung $k \leq l$ ist, wie in Abschnitt 2.4.1 beschrieben, keine Einschränkung. Die gegebene Gewichtsfunktion sei $w : E \rightarrow \mathbb{Q}$. Der transformierte Graph $G' = (V' \cup X, E')$ sei definiert durch $V' = V \cup \{v_{k+1}, \ldots, v_l\}$ mit $E' = V' \times X$. Die zugehörige Gewichtsfunktion $w'$ ist mit $w$ identisch, wobei den neu hinzugefügten Kanten das Gewicht 0 zugeordnet wird. Das maximale Gewicht eines Matchings in $G'$ entspricht dann offensichtlich dem maximalen Gewicht eines Matchings in $G$. Des Weiteren gilt offensichtlich $|V'| = |X|$. Im Folgenden wird ein Verfahren beschrieben, dass auf $G'$ ein MaxWBPM findet.


2.4. MAXIMUM WEIGHT BIPARTITE MATCHING

Maximiere \[ \sum_{e \in E} w(e) \cdot x(e) \]

Nebenbedingungen
\[ \sum_{e \in \delta(v)} x(e) = 1 \quad \forall v \in V \cup X \]
\[ x(e) \in \{0, 1\} \quad \forall e \in E \]

Die letzte Bedingung, \( x(e) \in \{0, 1\} \), kann relaxiert werden zu \( x(e) \geq 0 \quad \forall e \in E \). Auf einem bipartiten Graphen führt diese Relaxierung zu einem korrekten Ergebnis \([8]\). Das dazu duale Programm (D) \([13]\) lautet

Minimiere \[ \sum_{v \in V} y(v) + \sum_{x \in X} y(x) \]

Nebenbedingung \[ y(v) + y(x) \geq w(e) \quad \forall e = vx \in E \]

Für eine Lösungsvektor \( y^* \in \mathbb{R}^{V \cup X} \) des dualen Programms wird die zulässige Kantenmenge definiert durch \( E(y) := \{ e = vx \in E \mid y(v) + y(x) = w(e) \} \) und der zulässige Teilgraph durch \( G(y) := (V \cup X, E(y)) \) \([13]\). Zu beachten ist, dass auf den Kanten von \( G(y) \) keine Gewichtsfunktion definiert ist. Der wohlbekannte Satz des komplementären Schlupfs impliziert folgende Aussage \([13]\).

**Satz 2.25.** Sei \( y^* \in \mathbb{R}^{V \cup X} \) eine optimale Lösung des dualen Programms (D). Dann hat ein perfektes Matching \( M \) in \( G \) maximales Gewicht genau dann, wenn \( M \subseteq E(y^*) \)

Eine optimale Lösung \( y^* \) von (D) lässt sich in Zeit \( O(|V|^3) \) finden. Dabei ergibt sich gleichzeitig ein Matching größten Gewichts sowie der Graph \( G(y^*) \) \([13]\). Auf diesen wird in Abschnitt 3.2 zurückgegriffen. Die Idee, eine optimale Lösung zu finden, wird im Folgenden beschrieben \([18]\). Dabei sei zunächst vorausgesetzt, dass \( G \) vollständig ist, was im Zusammenhang mit dem Algorithmus von Edmonds gegeben ist. Das von Kuhn \([18]\) beschriebene Verfahren lässt sich auch auf nicht vollständigen bipartiten Graphen anwenden. Die dazu nötigen Änderungen werden am Ende dieses Abschnittes beschrieben. Zunächst folgt das Vom Verfasser dieser Arbeit gegenüber \([18]\) modifizierte Verfahren, das einen vollständigen bipartiten Graphen voraussetzt. Dazu wird der Begriff des \( M \)-augmentierenden Pfades eingeführt.

**Definition 2.26 (M-augmentierender Pfad \([26]\)).** Sei \( M \) ein Matching in einem Graphen \( G = (V, E) \) und ein \( P \) ein durch \( V' := \{v_1, v_2, \ldots, v_k\} \) und \( E' := \{v_1v_2, \ldots, v_{k-1}v_k\} \) definierter Pfad in \( G \) mit \( v_i \neq v_j \) für alle \( i \neq j \). Wenn \( v_1 \) und \( v_k \) M-exponiert sind, für gerades \( i \) die Kanten \( v_iv_{i+1} \in M \) sind und für ungerades \( i \) die Kanten \( v_iv_{i+1} \notin M \) sind, wird dieser Pfad \( M \)-augmentierend genannt.

Wenn die Kanten entlang eines \( M \)-augmentierenden Pfades bezüglich des Matchings ausgetauscht werden, entsteht ein neues Matching \( M' \) mit \( |M'| = |M| + 1 \). Dieser Austausch wird Augmentierung genannt. In Abbildung 2.7 ist für das Matching \( M_1 \) der \( M_1 \)-augmentierenden Pfad \( P = \{v_5, v_1, v_6, v_3\} \) dargestellt. Durch die Augmentierung mit \( P \) entsteht das Matching \( M_2 \).
KAPITEL 2. DAS MAXIMUM COMMON SUBTREE PROBLEM

Abbildung 2.7: Augmentierung eines Matchings durch den Pfad $P = \{v_5, v_1, v_6, v_3\}$

Abbildung 2.8: Erweiterung von $T = (\{r\}, \emptyset)$ um die Kanten $rx_1$ und $x_1v$ sowie anschließende Augmentierung durch den Pfad von $r$ nach $x$.

Für den Algorithmus sei $y$ ein beliebiger zulässiger Lösungsvektor. Der Algorithmus versucht in $G(y)$ ein perfektes Matching $M$ durch Augmentierung zu finden. Falls dies möglich ist, ist $M$ nach dem Satz des komplementären Schlups ein perfektes Matching maximalen Gewichts in $G$, denn $G(y)$ enthält die gleichen Knoten wie $G$, $y$ ist eine zulässige Lösung für das duale Programm (D) und der Vektor $x$ des linearen Programms (P), definiert durch $x(e) = 1$, falls $e \in M$ und $x(e) = 0$, falls $e \notin M$, stellt eine Lösung für (P) dar [8, 22]. Ansonsten wird der Lösungsvektor $y$ modifiziert, bis $M$ wieder augmentiert werden kann. Jede einzelne Modifizierung an $y$ stellt sicher, dass $G(y)$ um mindestens eine Kante erweitert wird und $y$ zulässig bleibt. Die einzelnen Schritte werden im Folgenden genauer beschrieben.

Ein initial zulässiger Lösungsvektor in dem bipartiten Graphen $G = (V \cup X, E)$ lässt sich beispielsweise durch $y(v) = 0$ für alle $v \in V$ und $y(x) = \max\{w(vx) \mid vx \in E\}$ für alle $x \in X$ definieren. Anschließend führt der Algorithmus genau $|V|$ Augmentierungen durch. Die Augmentierungen finden dabei entlang eines Baumes $T = (V_T, E_T)$, der in $G(y)$ enthalten ist und schrittweise erweitert wird, statt. Zu Beginn sei $T = (\{r\}, \emptyset)$ mit $r \in V$ als exponiertem Knoten. Sei weiterhin $B(T) := V_T \cap V$ und $A(T) := V_T \cap X$. In jedem Baumweiterungsschritt wird eine Kante $e = vx \in E(y)$ mit $v \in B(T)$ und $x \notin A(T)$ gesucht.

Falls $x$ exponiert ist, wird der Pfad von $r$ nach $x$ augmentiert. Wenn dann $|V|$ Matchingkanten gefunden wurden, ist der Algorithmus fertig, da dieses Matching, wie oben beschrieben, ein MaxWBPM in $G$ ist. Ansonsten wird der Baum $T$ wieder durch einen neuen exponierten Knoten $r \in V$ mit $T = (\{r\}, \emptyset)$ initialisiert. Falls $x$ nicht exponiert ist,
sei \( z \) der Knoten, der mit \( x \) durch eine Matchingkante verbunden ist. \( T \) wird dann um die Kanten \( vx \) und \( xz \) erweitert.

An Abbildung 2.8 lassen sich diese Schritte demonstrieren. Der Graph \( G(y) \) sei durch die dort dargestellten verschiedenenfarbigen Kanten und Knoten definiert. Sei weiterhin \( T = (\{ r \}, \emptyset) \) und \( rx_1 \) die erste gewählte Kante und \( x_1v \in M \). Da \( x_1 \) gematched ist, ergibt sich nach Hinzunahme von \( rx_1 \) und \( x_1v \) zu \( T \) die in Abbildung 2.8 (a) dargestellte Situation mit \( B(T) = \{ r, v \} \) und \( A(T) = \{ x_1 \} \). Die zweite gewählte Kante sei \( vx \) mit exponiertem Knoten \( x \). Das Matching wird augmentiert, ein neuer exponierter Knoten \( r \in V \) ausgewählt und \( T \) zurückgesetzt, wie in Abbildung 2.8 (b) dargestellt.

Falls keine Kante \( e \) mit den oben genannten Eigenschaften existiert, wird \( y \) durch
\[
y(v) \leftarrow y(v) + \varepsilon \text{ für alle } v \in B(T) \text{ und } y(x) \leftarrow y(x) - \varepsilon \text{ für alle } x \in A(T)
\]
modifiziert. Dabei wird \( \varepsilon > 0 \) größtmöglich gewählt, so dass \( y \) zulässig bleibt. Dies wird durch \( \varepsilon = \min \{ y(v) + y(x) - w(vx) \mid vx \in E, v \in B(T) \text{ und } x \notin A(T) \} \) erreicht und hat zur Folge, dass sowohl alle Matchingkanten als auch der Baum \( T \) in \( G(y) \) enthalten bleiben. Außerdem kommt mindestens eine Kante zu \( G(y) \) hinzu. Anschließend kann das Matching augmentiert und/oder \( T \) erweitert werden.

Die Laufzeit hängt im Wesentlichen von den Baumerweiterungsschritten ab. Nach jeweils höchstens \(| V| \) Baumerweiterungsschritten wird \( M \) augmentiert. Außerdem ist nach \(| V| \) Augmentierungen ein perfektes Matching in \( G(y) \) gefunden. Änderungen an \( y \) sind in Zeit \( \mathcal{O}(|V|) \) möglich. Insgesamt ergibt sich daher eine Zeitschranke von \( \mathcal{O}(|V|^3) \).

Für den allgemeinen Fall, dass \( G \) nicht vollständig ist, muss das oben beschriebene Verfahren an zwei Stellen modifiziert werden und entspricht damit dem Verfahren von Kuhn [18]. Zum einen kann der exponierte Knoten \( r \) beliebig aus \( V \cup X \) gewählt werden. Falls \( r \) aus \( X \) gewählt wird, werden die Mengen \( A(T) \) und \( B(T) \) vertauscht. Außerdem ist es in einem nicht vollständigen bipartiten Graphen möglich, dass dieser gar kein perfektes Matching enthält. Das lässt sich unmittelbar vor der Modifizierung von \( y \) feststellen. Wenn für jede Kante \( vx \in E \) mit \( v \in B(T) \) gilt, dass \( x \in A(T) \) ist, dann existiert kein perfektes Matching in \( G \). An dieser Stelle ist dann auch keine Modifizierung von \( y \) durch ein \( \varepsilon > 0 \) möglich.
Kapitel 3

Enumeration von Common Subtrees und Common Subtree Isomorphismen


3.1 Grundlagen der Enumeration

Je nach Problemstellung möchten die Anwender möglicherweise nicht nur eine Lösung haben, sondern einige oder alle Lösungen, die den Anforderungen entsprechen. In der Praxis gibt es mehrere Beispiele. Im öffentlichen Nahverkehr existieren häufig verschiedene Wege, die zum Ziel führen, sich aber in Zeit, Kosten, Transportmittel und so weiter unterscheiden können. Der Anwender definiert dann Kriterien wie Startzeit oder Zielstation und das Programm liefert dann mehrere Lösungen, die diesen Anforderungen entsprechen. Ein weiteres Beispiel sind chemische Strukturen, die auf Ähnlichkeit untersucht werden. Hier können Anwender beispielsweise die Aufgabe stellen, dass aus einer Menge von 50 Strukturen jene 10 Paare gefunden werden sollen, die die größte Übereinstimmung haben. Sollen mehrere gültige Lösungen gefunden werden, wird dies als Enumeration oder Aufzählung bezeichnet. Die Algorithmen, die diese Aufgabe erfüllen, werden entsprechend Enumerationsalgorithmen genannt.

Für Algorithmen der ersten drei Stufen kann häufig eine genaue (niedrigste) Laufzeitsschranke in Abhängigkeit der Eingabegröße angegeben werden. Bei Enumerationsalgorithmen ist eine Abhängigkeit allein von der Eingabegröße wenig sinnvoll, wie Abbildung 3.1 zu entnehmen ist. Bei den MCSTI gilt \( \varphi(r) = s \), die anderen Knoten werden beliebig aufeinander abgebildet. Die Anzahl der Maximum CSTI beträgt somit \( k! \) und ist deshalb nicht durch ein Polynom in \( k \) beschränkt. Selbst, wenn jede einzelne Lösung in \( O(1) \) berechnet werden könnte, wäre die Gesamtzeit dennoch nicht polynomial. Aus diesem Grund werden Enumerationsalgorithmen anders klassifiziert. Im Folgenden werden einige Klassen, basierend auf [15], vorgestellt.


(b) Polynomial delay. Die Zeit zwischen den einzelnen Lösungen, bis zur ersten Ausgabe und zwischen der letzten Ausgabe und dem Programmende ist durch ein Polynom in der Größe der Eingabe beschränkt. Ein Beispiel dazu ist der Algorithmus von Uno [27] zur Bestimmung aller perfekten Matchings in einem bipartiten Graphen. Dieser Hilfsalgo-
3.2. VERWANDTE ARBEITEN

rithmus wird unter anderem in Abschnitt 3.3 benötigt und in Abschnitt 3.2.2 vorgestellt. Jeder Algorithmus aus dieser Klasse ist natürlich auch in (a) enthalten.

(c) Specific order. Die Ausgabe erfolgt in einer bestimmten Reihenfolge, beispielsweise sortiert nach einem bestimmten Kriterium. Der Algorithmus zur Berechnung eines Max-WBM' aus Abschnitt 2.4.1 kann als Enumerationsalgorithmus aufgefasst werden, wenn alle während des Algorithmus berechneten maximalen Matchings ausgegeben werden. Die Ausgabe erfolgt dabei in lexikographischer Ordnung.


In modernen Publikationen, wie beispielsweise in [10], ist diese Unterteilung teils präziser und formal genau definiert. Für diese Arbeit soll die oben vorgestellte Unterteilung genügen.

3.2 Verwandte Arbeiten


Definition 3.1 (Common Subgraph (CSG), CSG Isomorphismus). Seien \(G_1\) und \(G_2\) Graphen. Ein Graph \(G\) wird Common Subgraph (CSG) von \(G_1\) und \(G_2\) genannt, wenn \(G\) isomorph zu Teilgraphen \(G'_1\) und \(G'_2\) von \(G_1\) und \(G_2\) ist [19]. Ein Graph \(G\) wird Common Induced Subgraph (CISG) von \(G_1\) und \(G_2\) genannt, wenn \(G\) isomorph zu induzierten Teilgraphen \(G'_1\) und \(G'_2\) von \(G_1\) und \(G_2\) ist [11]. Ein maximaler CSG bzw. CISG ist in keinem anderen CSG bzw. CISG von \(G_1\) und \(G_2\) enthalten [17]. Ein Maximum CISG ist ein CISG mit höchstmöglichster Knotenanzahl [11]. Falls nur Graphen \(G\) gesucht werden, die zusammenhängen, wird dies mit connected bezeichnet und mit einem vorangestellten \(C\) abgekürzt. Beispielsweise ist ein CCISG ein Connected Common Induced Subgraph. Ein zugehöriger Isomorphismus \(\varphi : V(G'_1) \rightarrow V(G'_2)\) wird durch ein nachgestelltes I gekennzeichnet, analog zu den CSTI.

Auf den gegebenen Graphen kann zusätzlich eine Labelfunktion definiert werden (vgl. Definition 2.12). In diesem Fall muss dann Isomorphie bezüglich der Label gegeben sein (vgl. Definition 2.13).

In Zusammenhang mit Enumeration beschränkt sich die Suche gewöhnlich auf maximale CSGI oder CISGI. Um Maximum CISGI zu enumerieren, müsste zunächst die Größe
bestimmt werden. Das Problem, ob für zwei Graphen und eine natürliche Zahl \( k \) ein CISGI der Mindestgröße \( k \) existiert, ist NP-schwer [11]. Entsprechend schwierig ist die Enumeration von Maximum CISGI.


**Definition 3.2 (Clique [17]).** Sei \( G \) ein Graph. Ein vollständiger Teilgraph \( G' \) von \( G \) wird *Clique* genannt. Falls es keinen anderen Clique \( G'' \) von \( G \) mit \( G' \subset G'' \) gibt, wird \( G \) maximale Clique genannt.

In einer Clique sind alle Knoten paarweise miteinander verbunden.

**Definition 3.3 (Vertex Product Graph (VPG) [17]).** Seien \( G_1 = (V_1, E_1) \) und \( G_2 = (V_2, E_2) \) gelabelte Graphen. Der Vertex Product Graph \( H_v = G_1 \circ_v G_2 = (V_H, E_H) \) ist wie folgt definiert. In \( V_H \) sind die Knotenpaare \( (u, v) \) aus \( V_1 \times V_2 \) enthalten, die das gleiche Label besitzen. Eine Kante \( u_Hv_H \in E_H \) mit \( u_H = (u_1, u_2) \) und \( v_H = (v_1, v_2) \) existiert genau dann, wenn \( u_1 \neq v_1 \) und \( u_2 \neq v_2 \) und die Kanten \( u_1v_1 \) und \( u_2v_2 \) das gleiche Label haben oder beide nicht vorhanden sind.

Von Levi [19] wurde gezeigt, dass eine maximale Clique, definiert durch die Knoten \( (u_1, v_1), \ldots, (u_k, v_k) \) in \( H_v \), einem maximalen CISGI von \( G_1 \) und \( G_2 \) entspricht. Der Isomorphismus ist dabei durch \( \varphi(u_i) = v_i \) für \( i \in \{1, \ldots, k\} \) gegeben. Die Enumeration aller maximalen CISGI kann somit auf die Enumeration aller Cliquen auf dem VPG zurückgeführt werden. Von Koch [17] wird außerdem noch der Edge Product Graph definiert, der bei der Enumeration von maximalen CSGI behilflich ist. Maximale CCSGI und CCISGI korrespondieren zu maximalen \( c \)-zusammenhängenden Cliquen im entsprechenden Produktgraphen [17].

In Abschnitt 3.2.1 wird der Algorithmus zur Enumeration von maximalen CCISGI von Cazals und Karande [7], basierend auf der Enumeration von maximalen CCSGI von Koch [17], beschrieben. Für zwei Bäume \( T_1 \) und \( T_2 \) entspricht die Enumeration aller maximalen CCISGI auf dem zugehörigen VPG der Enumeration aller maximalen CSTI, denn Teilbäume sind insbesondere induzierte zusammenhängende Teilgraphen eines Baumes. Der
3.2. VERWANDTE ARBEITEN


3.2.1 Enumeration von maximalen CCISGI

Sei $G = (V, E)$ ein gelabelter Graph mit Kantenbezeichnern (vgl. Definition 2.12) und $\Sigma = \{c, d\}$. Der Graph $G$ sei dabei als VPG aus zwei ungelabelten Graphen $G_1$ und $G_2$ entstanden. Eine Kante $uv \in E$ mit $u = (u_1, u_2)$ und $v = (v_1, v_2)$, für die $u_1v_1 \notin E(G_1)$ und $u_2v_2 \notin E(G_2)$ gilt, bekommt den Bezeichner $d$ für disconnected. Ansonsten bekommt die Kante den Bezeichner $c$ für connected. Im ersten Fall sind jeweils die beiden Knoten aus $V(G_1)$ und $V(G_2)$ nicht miteinander verbunden, im zweiten Fall sind sie verbunden. Kanten mit Bezeichner $c$ bzw. $d$ werden $c$-Kanten bzw. $d$-Kanten genannt.

**Definition 3.4 (c-zusammenhängende Clique [7]).** Eine über $c$-Kanten verbundene Clique ist eine $c$-zusammenhängende Clique.

Die Idee zur Enumeration der maximalen CCISGI besteht darin, maximale $c$-zusammenhängende Cliquen zu enumerieren. Maximale Cliquen entsprechen einem maximalen CISGI von $G_1$ und $G_2$. Die Eigenschaft des $c$-Zusammenhangs garantiert, dass der maximale CISGI zusammenhängend ist. Im Folgenden wird der Algorithmus zur Enumeration aller maximalen $c$-zusammenhängenden Cliquen beschrieben.


In der Menge $Q$ sind Knoten gespeichert, die zu allen Knoten aus $R$ ausschließlich über $d$-Kanten verbunden sind. Zu Beginn sind das genau die Knoten, die mit $u_i$ über eine $d$-Kante verbunden sind. Wenn $R$ um einen Knoten $u_i$ erweitert wird, können alle Knoten

---

*Dies hat keinen Einfluss auf den Algorithmus und betrifft nur die Konstruktion des VPG.*
KAPITEL 3. ENUMERATION VON COMMON SUBTREES UND CSTI

$u_j \in Q$, die über eine $c$-Kante mit $u_i$ verbunden sind, in die Menge $P$ verschoben werden. Die Knoten aus $R$ bilden mit $u_j$ eine Clique, die wegen der $c$-Kante zwischen $u_i$ und $u_j$ dann auch eine $c$-zusammenhängende Clique ist.

Immer dann, wenn $R$ um einen Knoten $u_i$ erweitert wird, müssen alle Knoten aus $P$ und $Q$ gelöscht werden, die nicht mit $u_i$ verbunden sind. Denn diese Knoten bilden dann mit den Knoten aus $R$ keine Clique mehr, so dass $R$ nicht mehr um diese Knoten erweitert werden kann.

Um Cliques nicht mehrfach aufzuzählen, beispielsweise, wenn in einer Rekursion (a) zuerst $u_2$, dann $u_3$ und in einer anderen Rekursion (b) erst $u_3$, dann $u_2$ hinzugenommen wird, gibt es Mengen von verbotenen Knoten. In diesen wird gespeichert, welche Knoten bereits besucht und abgearbeitet wurden. In dem obigen Beispiel würde vor der Hinzunahme von $u_3$ in der Rekursion (b) der Knoten $u_2$ in die Menge der verbotenen Knoten verschoben werden und steht dann in der Rekursion (b) nicht mehr zur Verfügung. In [7] ist der Algorithmus in Pseudocode angegeben.

3.2.2 Enumeration von Maximum Weight Bipartite Matchings

In Abschnitt 2.4.1 wurde beschrieben, wie das Gewicht eines MaxWBM' auf einem vollständigen bipartiten Graphen bestimmt werden kann. Dazu wurden alle maximalen Matchings enumeriert und dabei jeweils das grösste Gewicht gespeichert. Dieser Algorithmus kann so modifiziert werden, dass er im ersten Durchlauf die Größe eines MaxWBM' bestimmt und im zweiten Durchlauf genau die Matchings ausgibt, die dieses Gewicht haben. Somit ist ein einfacher Enumerationsalgorithmus für alle MaxWBM in einem vollständigen bipartiten Graphen gegeben. Wenn der relative Anteil der MaxWBM' bezogen auf alle maximalen Matchings nicht zu klein ist, der Graph wenig Knoten hat oder der Knotengrad allgemein beschränkt ist, können MaxWBM' auf diese Weise effizient enumeriert werden. Die in Abschnitt 2.4.1 angegebene Laufzeit gilt auch für die Enumeration aller MaxWBM'.


Sei $G' = (V' \cup X', E')$ ein vollständiger bipartiter Graph mit Kantengewichten, wie diese im Algorithmus von Edmonds konstruiert werden. Nach Abschnitt 2.4.2 lässt sich daraus ein Graph $G = (V \cup X, E)$ mit $|V| = |X| = \max\{|V'|, |X'|\}$ konstruieren, indem Kanten mit Gewicht 0 hinzugenommen werden. In Abschnitt 2.4.2 wurde außerdem der zulässige Teilgraph $G(y)$ für einen Lösungsvektor $y$ des dualen Programms (D) vorgestellt. Für eine optimale Lösung $y^*$ hat nach Satz 2.25 ein perfektes Matching $M$ in $G$ maximales Gewicht genau dann, wenn $M \subseteq E(y^*)$. Von Uno [27] wird beschrieben, wie alle perfekten Matchings in einem bipartiten Graphen ohne Gewichtsfunktion enumeriert werden können.

**Definition 3.5 (Alternierender Kreis [27]).** Sei $M$ ein Matching in einem bipartiten Graphen $G = (V \cup X, E)$ und $K$ ein durch $V_K := \{v_1, x_1, \ldots, v_k, x_k\}$ und $E_K := \{v_1x_1, x_1v_2, v_2x_2, \ldots, x_{k-1}v_k, v_kx_k, x_kv_1\}$ definierter Kreis in $G$. Wenn die Kanten $v_ix_i \in M$ für alle $i \in \{1, \ldots, k\}$ sind, wird dieser Kreis alternierender Kreis genannt.


Dazu werden zwei neue Graphen generiert, die mit $G^+(e)$ und $G^-(e)$ bezeichnet werden. In $G^+(e)$ werden $v$ und $x$ und alle dazu inzidenten Kanten entfernt. Nach Uno beinhalten alle perfekten Matchings in $G^+(e)$ die Kante $e$. In $G^-(e)$ wird die Kante $e$ entfernt. Damit existiert in $G^-(e)$ kein Matching, dass $e$ enthält. Es folgen zwei rekursive Aufrufe, einer auf dem Graphen $G^+(e)$ mit dem Matching $M \setminus e$, und einer auf dem Graphen $G^-(e)$ mit dem Matching $M'$. In Abbildung 3.2 ist zum einen ein Graph mit einem Matching $M$, einem
alternierenden Kreis (gestrichelte Linien) und der Kante $e$ dargestellt, zum anderen die sich daraus ergebenden Graphen $G^+(e)$ mit dem Matching $M$ und $G^-(e)$ mit dem Matching $M'$. Die gelöschten Knoten und Kanten sind jeweils mit hellen Farbwerten dargestellt.


Der nötige Speicherplatzverbrauch liegt nach Uno bei $O(m)$, wobei $m$ der Anzahl der Kanten des bipartiten Graphen entspricht. Dazu wird bei einem Rekursionsaufruf nicht der Graph $G^+(e)$ bzw. $G^-(e)$ übergeben, sondern die gelöschten Kanten.


Satz 3.6. Der Algorithmus von Uno zur Enumeration perfekter Matchings ist ein polynomial-delay-Algorithmus.
3.2. VERWANDTE ARBEITEN


Im Folgenden wird noch die nötige Modifikation beschrieben, um MaxWBM auf dem aus dem Algorithmus von Edmonds gegebenen Graphen $G' = (V' \cup X', E')$ zu enumerieren.

Falls $|V'| = |X'|$ ist, muss nichts weiter beachtet werden, da in dem Fall $G'$ und der nach Abschnitt 2.4.2 konstruierte Graph $G = (V \cup X, E)$ mit $|V| = |X| = \max\{|V'|, |X'\}$ identisch sind. Sei im Folgenden $|V'| < |X'|$. Das Verfahren im Fall $|X'| < |V'|$ wird analog durchgeführt. Die Idee liegt darin, Matchingkanten, die zu Knoten aus $V \setminus V' := V''$ inzident sind, zu verwerfen, denn Knoten aus $V''$ stellen nur Hilfsknoten dar und sind in den Bäumen der Eingabe nicht vorhanden. Ein enumieriertes perfektes Matching $M$ in $G(y^*)$ ist ein MaxWBM in $G$. Das Matching $M \cap E'$ ist ein MaxWBM in $G'$, da nur Matchingkanten mit Gewicht 0 entfernt wurden. Auf diese Weise können alle MaxWBM in $G'$ aufgezählt werden.


Die Laufzeit im Algorithmus von Uno [27] ergibt sich aus der Summe $|E(G_x)| + |V(G_x)|$ über alle Rekursionen $x$. Im Beweis wird gezeigt, dass der Durchschnitt aller $|E(G_x)|$ durch $O(n)$ beschränkt ist. Dazu wird eine Kostenanalyse durchgeführt und die Kosten $|E(G_x)|$ auf die nachfolgenden Rekursionsaufrufe verteilt. Wegen der obigen Modifikation
KAPITEL 3. ENUMERATION VON COMMON SUBTREES UND CSTI

3.3 Enumeration von Maximum CSTI

In diesem Abschnitt wird ein im Rahmen dieser Arbeit entwickelter Enumerationsalgorithmus für Maximum Common Subtree Isomorphismen vorgestellt. Als Eingabe dienen zwei Bäume \( R = (V_R, E_R) \) und \( S = (V_S, E_S) \). Wenn mindestens ein Baum höchstens einen Knoten hat, sind die Lösungen trivial - entweder ist der einzige Maximum CSTI die leere Abbildung \( \varphi : \emptyset \rightarrow \emptyset \), wenn einer der Bäume leer ist, oder die Isomorphismen bilden den Knoten des Baumes, der nur einen Knoten beinhaltet, auf jeweils einen Knoten des anderen Baumes ab. Dies kann durch eine einfache Fallunterscheidungen zu Beginn geprüft werden. Im Folgenden werden die trivialen Fälle nicht mehr besprochen. Stattdessen wird vorausgesetzt, dass \( R \) und \( S \) jeweils aus mindestens zwei Knoten bestehen.

Der Enumerationsalgorithmus für die Maximum Common Subtree Isomorphismen basiert auf dem Algorithmus von Edmonds [21]. Die dort vorgestellten gewurzelten Teilbäume (siehe Definition 2.15) bilden den Ausgangspunkt für die Enumeration. Die Grundidee ist...
3.3. ENUMERATION VON MAXIMUM COMMON SUBTREE ISOMORPHISMEN

Eingabe: Zwei Bäume R und S
Ausgabe: Alle Maximum Common Subtree Isomorphismen von R und S

1. \( m \leftarrow \text{SizeMCSTI}(R,S) \) // Algorithmus 2.2
2. \textbf{for all} \( r \in RST_R \text{ mit } I(r) < I(\overline{r}) \) \textbf{do}
3. \textbf{for all} \( s \in RST_S \) \textbf{do}
4. \textbf{if} \( D(r,s) + D(\overline{r}, \overline{s}) = m \) // alle Werte für \( D \) wurden in Zeile 1 berechnet \textbf{then}
5. \textbf{for all} Maximum CSTI \( \varphi_1 \) von \( (r,s) \) // Wurzelknoten aufeinander abbilden \textbf{do}
6. \textbf{for all} Maximum CSTI \( \varphi_2 \) von \( (\overline{r}, \overline{s}) \) // Wurzelknoten aufeinander abb. \textbf{do}
7. Verbinde die Isomorphismen \( \varphi_1 \) und \( \varphi_2 \) der gewurzelten Teilbäume zu einem Maximum CSTI \( \varphi \) von \( R \) und \( S \).
8. Gebe \( \varphi \) aus, falls \( \varphi \) bisher nicht gefunden wurde.
9. \textbf{end for}
10. \textbf{end for}
11. \textbf{end if}
12. \textbf{end for}
13. \textbf{end for}

Algorithmus 3.1: EnumMaximumCSTI\((R,S)\)

es, für alle Paare \((uv \in E_R, xw \in E_S)\) von Kanten den CSTI zunächst durch \( \varphi(u) = x \) und \( \varphi(v) = w \) beziehungsweise \( \varphi(u) = w \) und \( \varphi(v) = x \) zu definieren. Falls sich dieser CSTI zu einem Maximum CSTI erweitern lässt, werden sukzessive Knoten hinzugenommen, solange diese nicht die Erweiterung zu einem Maximum CSTI verhindern. Das lässt sich mit Algorithmus 2.1 (GetD) feststellen. Die Enumeration besteht darin, alle Möglichkeiten, Knoten zu \( \varphi \) hinzuzufügen, systematisch so zu durchlaufen, dass alle Maximum CSTI genau einmal gefunden werden. Die nötigen Formalisierungen werden im Folgenden beschrieben.

Zur Kantenmenge \( E_R \) wird die Menge \( E'_R \) der gerichteten Kanten definiert durch \( E'_R := \{(u,v), (v,u) \mid uv \in E_R\} \). Die Kantenmenge \( E'_S \) ist analog definiert. Die Menge der gewurzelten Teilbäume \( RST_R \) wird durch \( RST_R := \{R'_u \mid (u,v) \in E'_R\} \) definiert, entsprechendes gilt für \( RST_S \). Für einen gewurzelten Teilbaum \( t = T'_u \) mit \( T \in \{R,S\} \) sei \( l := T'_u \) der durch die andere Zusammenhangskomponente definierte gewurzelte Teilbaum. Außerdem wird eine bijektive Abbildung \( I : RST_R \to \{1,2,\ldots,|RST_R|\} \) definiert, die im Folgenden mit \textit{Ordnungsfunktion} bezeichnet wird. Für \( I \) gelten gewisse Einschränkungen, die in Abschnitt 3.3.2 diskutiert werden. Mit Hilfe der Ordnungsfunktion wird sichergestellt, dass derselbe Isomorphismus nicht mehrmals ausgegeben wird.

Im Hauptalgorithmus 3.1 (EnumMaximumCSTI), Zeile 2 bis 4, werden alle Paare \((r,s)\) von gewurzelten Teilbäumen gesucht, für die ein Maximum CSTI existieren kann, wie dies in Abschnitt 2.2.3 beschrieben wurde. Das sind genau die Paare, für die in Algorithmus 2.2 (SizeMCSTI, Zeile 8) \( m = \text{GetD}(R'_u, S'_w) + \text{GetD}(R'_u, S'_w) \) gilt. Die Voraussetzung \( I(r) <
$I(\pi)$ wird in Abschnitt 3.3.2 besprochen. Anschließend werden alle Maximum CSTI von $(r,s)$ enumeriert und diese nacheinander mit allen Maximum CSTI von $(\tau,\pi)$ verbunden (Zeile 5 bis 7). Verbinden bedeutet dabei, dass in $\varphi$ genau die Zuordnungen aus $\varphi_1$ und $\varphi_2$ enthalten sind. In diesen Isomorphismen werden die Wurzelknoten jeweils aufeinander abgebildet, wie in Abschnitt 2.2.3 beschrieben. Der genaue Ablauf in Zeile 5 und 6 wird in Abschnitt 3.3.1 vorgestellt. Warum Lösungen überhaupt mehrmals gefunden werden und wie sichergestellt wird, dass diese nur einmal ausgegeben werden (Zeile 8), wird in Abschnitt 3.3.2 beschrieben.

### 3.3.1 Enumeration von Maximum CSTI auf gewurzelten Bäumen

In diesem Abschnitt wird ein Verfahren beschrieben, wie in Zeile 5 und 6 von Algorithmus 3.1 die Maximum CSTI aufgezählt werden können. Das Verfahren lässt sich auf beliebigen Paaren von gewurzelten Teilbäumen anwenden, insbesondere auf $(r,s)$ und $(r,s)$ aus Algorithmus 3.1.

Sei $(r,s) = (R_u^v, S_w^x)$ ein beliebiges Paar von gewurzelten Teilbäumen, auf dem alle Maximum CSTI aufgezählt werden sollen. Aus dem Algorithmus von Edmonds ist bekannt, dass sich die Größe eines Maximum CSTI mit Hilfe von Maximum Weight Bipartite Matchings berechnen lässt. Konkret wird dort zu zwei gewurzelten Bäumen ein bipartiter Graph erstellt, auf dem dann das Gewicht eines MaxWBM bestimmt wird. Das Verfahren wurde in Abschnitt 2.2.2 vorgestellt. Die Matchingkanten entsprechen dabei den Zuordnungen der Kinder von $v$ und $w$.

Die Idee zur Enumeration der Maximum CSTI auf $(r,s)$ besteht darin, die Menge $M$ aller Maximum CSTI von $r$ und $s$ mit $\varphi(v) = x$ disjunkt in Mengen $M_1, \ldots, M_m$ zu zerlegen, in denen jeweils weitere Vorgaben für $\varphi$ existieren, und für diese Mengen dann jeweils alle Maximum CSTI aufzuzählen. Bei den Teilmengen $M_i$ wird das Verfahren dann rekursiv angewendet. Das genaue Vorgehen wird im Folgenden beschrieben. Dabei meint der Begriff „Rekursionsebene“ die aktuelle Ebene oder Tiefe der Rekursion, in der sich der Algorithmus befindet, und steht jeweils für ein Paar von gewurzelten Teilbäumen.

Falls $v$ oder $x$ keine Kinder hat, gibt es nur eine Lösung, und zwar $\varphi(v) = x$. In diesem Fall ist keine Rekursion erforderlich. Voraussetzung ist im Folgenden, dass $v$ und $x$ Kinder haben. Seien $V := \{v_1, v_2, \ldots, v_k\}$ und $X := \{x_1, x_2, \ldots, x_l\}$ diese Kinder. Außerdem sei $k \leq l$ vorausgesetzt. Ansonsten können für die Berechnung die Rollen von $V$ und $X$ vertauscht werden, wie dies in Abschnitt 2.4 durchgeführt wurde. Auf dem bipartiten Graphen $G = (V \cup X, V \times X)$ mit Kantengewichten, wie in Abschnitt 2.2.2 beschrieben, sei $M := \{M_1, \ldots, M_m\}$ die Menge aller MaxWBM. Wie diese Matchings enumeriert werden können, wurde in Abschnitt 3.2.2 beschrieben. Sei $M_i = \{v_1^{(i)}, v_2^{(i)}, \ldots, v_k^{(i)}\}$ das $i$-te numerierte Matching mit $x_j^{(i)} \in X$. Für alle $i \in \{1, 2, \ldots, m\}$ werden der Menge $M_i$ dann alle Isomorphismen mit $\varphi(v) = w$ sowie $\varphi(v_j) = x_j^{(i)}$ für $j \in \{1, 2, \ldots, k\}$ zugeordnet. Die
3.3. ENUMERATION VON MAXIMUM COMMON SUBTREE ISOMORPHISMEN

Abbildung 3.5: Beispiel zur Enumeration auf zwei gewurzelten Teilbäumen

Enumeration aller Maximum CSTI auf \((r, s)\) entspricht somit der aufeinanderfolgenden Enumeration aller Maximum CSTI aus \(M_1\) bis \(M_m\). Die Enumeration auf den Mengen \(M_i\) läuft wie folgt ab.

Sei \(\varphi(v) = x\) und \(\varphi(v_j) = x_j := x_j^{(i)}\) für \(j \in \{1, \ldots, k\}\) die Festlegung von \(\varphi\) in \(M_i\). Sei \(M_j'\) die Menge aller Maximum CSTI auf dem Paar \((R_v^{x_j}, S_w^{x_j})\) von gewurzelten Teilbäumen. In \(M_i\) sind genau \(\prod_{j=1,2,\ldots,k} |M_j'|\) verschiedene Maximum CSTI für das Paar \((R_v^u, S_w^w)\) von gewurzelten Teilbäumen enthalten. Das sind genau die Isomorphismen, die die Vorgaben von \(M_i\) an \(\varphi\) erfüllen. Diese lassen sich enumerieren, indem alle Kombinationen von Isomorphismen aus \(M_j'\) bis \(M_k'\) miteinander und mit \(\varphi\) verbunden werden. Diese Kombinationen können durch Enumeration der Maximum CSTI auf den Mengen \(M_j'\) gebildet werden. An dieser Stelle erfolgt der rekursive Aufruf, denn dabei handelt es sich um die Enumeration aller Maximum CSTI auf einem Paar von gewurzelten Teilbäumen. Der Aufruf auf \((R_v^{x_j}, S_w^{x_j})\) erfolgt allerdings nur dann, wenn beide Teilbäume \(R_v^{x_j}\) und \(S_w^{x_j}\) Kinder haben. Ansonsten kann der Isomorphismus über diese Rekursion nicht mehr erweitert werden.

Die Kombinationen der Maximum CSTI aus den Mengen \(M_1'\) bis \(M_k'\), also der Maximum CSTI auf den Paaren \((R_v^{x_1}, S_w^{x_1})\) bis \((R_v^{x_k}, S_w^{x_k})\) von gewurzelten Teilbäumen, lassen sich beispielsweise bilden, indem zunächst aus jeder Menge der erste Maximum CSTI berechnet wird. Anschließend werden alle Isomorphismen in \(M_k'\) enumeriert. Danach wird der zweite Isomorphismus in \(M_{k-1}'\) enumeriert, anschließend wieder alle Isomorphismen aus \(M_k'\) usw. Die letzte Kombination wurde bestimmt, wenn in den Mengen \(M_1'\) bis \(M_k'\) jeweils der letzte Isomorphismus enumeriert wurde. Aufgrund der Tatsache, dass immer wieder auf den gleichen Mengen enumeriert wird, bietet es sich an, die MaxWBM nach der ersten Berechnung in einer geeigneten Datenstruktur zu speichern. Zu beachten ist, dass die Anzahl der MaxWBM für einen bipartiten Graphen im Allgemeinen nicht polynomial beschränkt ist. Es sollte also eine Obergrenze für die Anzahl der zu speichernden MaxWBM festgelegt werden.

Die Enumeration aller Maximum CSTI zweier gewurzelter Teilbäume wird folgend anhand Abbildung 3.5 illustriert. Es sollen alle Maximum CSTI von \((r, s) = (R_v^u, S_w^w)\) enumeriert werden. Zunächst ergibt sich \(\varphi(v) = x\), \(V := \{v_1, v_2\}\) und \(X := \{x_1, x_2, x_3\}\). Die Enumeration der MaxWBM erzeuge zunächst das Matching \(M_1 = \{v_1x_1, v_2x_2\}\). Der
Menge $M_1$ werden also die Maximum CSTI zugeordnet, für die zusätzlich $\varphi(v_1) = x_1$ und $\varphi(v_2) = x_2$ gilt. Anschließend werden diese enumeriert. Dabei ergeben sich die Mengen $M'_1$ als die Menge aller Maximum CSTI auf $(R^v, S^v_1)$ und $M'_2$ als die Menge aller Maximum CSTI auf $(R^v_2, S^v_2)$. Alle Kombinationen von Isomorphismen aus diesen beiden Mengen werden nun miteinander und mit $\varphi$ verbunden. Zunächst wird jeweils rekursiv der erste Isomorphismus aus beiden Mengen bestimmt. Für $M'_1$ führt dies zunächst zur Zuordnung $\varphi(v_3) = x_4$. Da mindestens einer der Knoten $v_3$ und $x_4$ keine Kinder hat (in diesem Beispiel beide), ergibt sich somit insgesamt eine erste Lösung für $M'_1$. Um den ersten Isomorphismus aus $M'_2$ zu erhalten, werden die MaxWBM auf den Kindern von $v_2$ und $x_2$ enumeriert. Das sei zunächst das Matching $\{v_2x_5, v_5x_6\}$, das zur Festlegung von $\varphi(v_1) = x_5$ und $\varphi(v_5) = x_6$ führt. Da es keine weiteren Kinder gibt, ist der erste Isomorphismus für $M'_2$ vollständig berechnet. Durch Verbindung der Isomorphismen ergibt sich der erste MCSTI $\varphi(v) = x$, $\varphi(v_1) = x_1$, $\varphi(v_3) = x_4$, $\varphi(v_2) = x_2$, $\varphi(v_4) = x_5$ und $\varphi(v_5) = x_6$ von $(r,s)$. Die nächste Kombination, die gebildet wird, belässt den Isomorphismus aus $M'_1$ und enumeriert den nächsten aus $M'_2$ und verbindet diese dann mit $\varphi$. Dabei ergibt sich $\varphi(v_4) = x_6$ und $\varphi(v_5) = x_5$, die anderen Zuordnungen von $\varphi$ stimmen mit dem ersten MCSTI überein. Ein weiterer Isomorphismus auf $M'_2$ existiert nicht, weshalb der nächste Isomorphismus aus $M'_1$ mit allen Isomorphismen von $M'_2$ verbunden werden soll. Aber auch für $M'_1$ gibt es keine Lösung mehr, weshalb alle MCSTI für das MaxWBM $M_1$ aufgezählt worden sind. Das nächste MaxWBM $M_2 = \{v_1x_3, v_2x_2\}$ der Kinder von $v$ und $w$ führt auf die gleiche Art zu zwei weiteren Isomorphismen. Da es nur zwei MaxWBM zu den Kindern von $v$ und $w$ gibt, sind damit alle Maximum CSTI von $(r,s)$ aufgezählt.

Die technische Umsetzung im Programm wurde so realisiert, dass jeder Isomorphismus $\varphi_1$ bzw. $\varphi_2$ ausgehend von dem Paar $(r,s) = (R^v_1, S^v_1)$ bzw. $(r,s)$, das in Zeile 2 und 3 von Algorithmus 3.1 (EnumMaximumCSTI) gewählt wurde, gesucht wird. In Zeile 5 wird dazu $\varphi(v) = x$ gesetzt und GetMaximumCSTI$(r,s, \varphi_1)$ (Algorithmus 3.2) aufgerufen. In Zeile 6 wird entsprechend $\varphi(u) = w$ gesetzt und GetMaximumCSTI$(r,s, \varphi_2)$ aufgerufen. Mit Hilfe einer lokalen booleschen Variablen $f$ ($f$ steht für first isomorphism), die zu Beginn „true“ ist, wird dann der erste bzw. nächste MCSTI auf dem übergebenen Paar von gewurzelten Teilbäumen bestimmt. Der Rückgabewert ist 1, wenn ein nächster MCSTI enumeriert werden konnte, und 0, wenn es keinen weiteren mehr gibt. Zu beachten ist, dass auch in Algorithmus 3.2 alle Parameter als Referenzen übergeben werden.

Im Folgenden sind die einzelnen Schritte von Algorithmus 3.2 zusammengefasst. Falls bereits ein MCSTI berechnet wurde, wird zunächst versucht, rekursiv die nächste Kombination von MCSTI auf den Kindern zu bestimmen, wie in diesem Abschnitt beschrieben wurde (Zeile 2). Falls dies erfolgreich war, wurde ein weiterer MCSTI auf $(R^v_1, S^v_1)$ enumeriert, und der Wert 1 wird zurückgegeben (Zeile 3 bis 5). In Zeile 7 können zwei verschiedene Situationen vorliegen. Falls noch kein MCSTI bestimmt wurde, dann muss das erste MaxWBM bestimmt werden (Zeile 9). Dabei wird $\varphi$ entsprechend festgelegt. Ansonsten wurde
3.3. ENUMERATION VON MAXIMUM COMMON SUBTREE ISOMORPHISMEN

**Eingabe:** Referenzen auf $R_v^u$, $S_x^w$ und $\varphi$

**Ausgabe:** 1, falls ein weiterer Isomorphismus aufgezählt werden konnte; 0, sonst

```plaintext
1: if $f = \text{false} /* Es wird nicht der erste MCSTI auf } R_v^u, S_x^w \text{ berechnet */ then
2: Berechne die nächste Kombination von MCSTI auf $M_1'$ bis $M_k'$
3: if es gibt eine weitere Kombination then
   return 1
4: end if
5: end if
6: if $f = \text{true} /* Bisher noch kein MCSTI auf } R_v^u, S_x^w \text{ bestimmt */ then
7: $f \leftarrow \text{false}$
8: Berechne erstes MaxWBM und erweitere $\varphi$ um die durch das Matching bestimmte Zuordnung der Kinder
9: else
10: /* Alle Kombinationen auf letztem Matching wurden aufgezählt
   Berechne nächstes MaxWBM, verändere $\varphi$ entsprechend
11: if es gab kein nächstes MaxWBM, da bereits alle enumeriert wurden then
12: $f \leftarrow \text{true} /* Es gibt keinen weiteren MCSTI auf } R_v^u, S_x^w$ der in Zeile 2 erfolgt, aber im Algorithmus nicht explizit dargestellt ist, in diesem Abschnitt allerdings beschrieben wurde, den Wert 0 zurückliefert hat. In diesem Fall muss dann das nächste MaxWBM bestimmt werden, und $\varphi$ entsprechend verändert werden (Zeile 11). Falls es kein weiteres Matching mehr gab, können auch keine weiteren MCSTI aufgezählt werden. In diesem Fall wird $f$ auf „true“ gesetzt (Zeile 12-15), damit beim nächsten Aufruf wieder mit der Enumeration des ersten MCSTI begonnen wird. In Zeile 17, die erreicht wird, wenn das erste oder ein weiteres Matching bestimmt wurde, wird die erste Kombination von Maximum Common Subtree Isomorphismen auf $M_1'$ bis $M_k'$ berechnet. Hier erfolgen rekursive Aufrufe von GetMaximumCSTI auf den bis zu $k$ Kind-Paaren ($R_v^u, S_x^w$), in denen $v_i$ und $x_i$ keine Blätter sind, und die den Mengen $M_1'$ bis $M_k'$ entsprechen.
```

Algorithmus 3.2: GetMaximumCSTI($R_v^u, S_x^w, \varphi$)
3.3.2 Mehrfache Ausgabe gleicher Isomorphismen vermeiden

Wenn in Zeile 2 und 3 von Algorithmus 3.1 die gewurzelten Teilbäume \( r \) und \( s \) ausgewählt werden, werden in Zeile 6 die Maximum CSTI von \((r, s)\) (Zeile 5) verbunden. Ohne die Einschränkung \( I(r) < I(\bar{r}) \) in Zeile 2 würde die Auswahl von \( \bar{r} \) und \( \bar{s} \) in Zeile 2 und 3 dazu führen, dass in Zeile 6 die Maximum CSTI von \((\bar{r}, \bar{s})\) aufgezählt würden, so dass in dem Fall genau die gleichen Isomorphismen bestimmt werden, mit vertauschten Rollen von \( \varphi_1 \) und \( \varphi_2 \).

Die Einschränkung in Zeile 2 ist allein noch unzureichend, bei der Aufzählung identische Isomorphismen zu vermeiden. Zu den in Abbildung 3.6 dargestellten Bäumen beträgt die Größe eines MCSTI 5. Der Wert der gerichteten Kante \((r_i, r_j)\) in der Abbildung steht für den Wert \( I(R_{r_j}) \). Die Festlegung der Ordnungsfunktion \( I \) wird im nächsten Abschnitt behandelt. Wird im Algorithmus \((r, s) = (R_{r_4}^4, S_{s_3}^3)\) gewählt, führt dies zu den (einzigen) Lösungen \( \varphi_1(r_3) = s_3, \varphi_1(r_2) = s_2, \varphi_1(r_1) = s_1 \) und \( \varphi_2(r_4) = s_4, \varphi_2(r_5) = s_5 \). Insgesamt ergibt sich somit \( \varphi_i(r_i) = s_i \) für alle \( i \in \{1, \ldots, 5\} \). Die Wahl von \((r, s) = (R_{r_2}^1, S_{s_2}^1)\) führt zu genau dem gleichen Isomorphismus. Gleiches gilt für \((r, s) = (R_{r_2}^2, S_{s_2}^2)\) und \((r, s) = (R_{r_4}^5, S_{s_4}^5)\). Allgemein gilt, dass ein Maximum CSTI der Größe \( n \) mit dem Test in Zeile 2 über \( n - 1 \) verschiedene Paare \((r, s)\) von gewurzelten Bäumen gefunden werden kann. Im Folgenden werden zwei Möglichkeiten vorgestellt, diese identischen Isomorphismen dennoch nur einmal auszugeben.

(a) Vermeidung identischer Isomorphismen mit Hilfe der Ordnungsfunktion

Die Idee, diese identischen Isomorphismen zu vermeiden, besteht darin, einen dieser Isomorphismen als denjenigen auszuwählen, der ausgegeben werden soll, und die anderen entsprechend nicht auszugeben.

Sei \( n + 1 \) die Größe eines MCSTI \( \varphi \) und \( R := \{r_1, \ldots, r_n\} \) die Menge der gewurzelten Teilbäume, durch die \( \varphi \) in Algorithmus 3.1, zusammen mit jeweils einem gewurzelten Baum aus \( S \), enumeriert wird. Da die Ordnungsfunktion \( I \) bijektiv ist, gibt es genau ein \( r \in R \) mit \( I(r) = \min\{I(r') \mid r' \in R\} \). Ein berechneter Isomorphismus wird genau dann ausgegeben, wenn in Zeile 2 genau dieses \( r \) gewählt wurde.

Realisieren lässt sich das, indem die Enumeration der Maximum CSTI auf den gewurzelten Bäumen aus Abschnitt 3.3.1 um eine entsprechende Prüfung erweitert wird. Sei dazu...
3.3. Enumeration von Maximum Common Subtree Isomorphismen

\( M \) das zuletzt ausgewählte Matching in irgendeiner Rekursionsebene und \( V' := \{ v' \in V \mid v' \text{ ist durch } M \text{ gematched} \} \). Falls \( I(r) > I(r') \) für ein \( r' \in \{ R^v_r \cup R^v_s \mid v' \in V' \} \) ist, gehört der bisher berechnete Teil des Maximum CSTI zu einem Isomorphismus, der nicht ausgegeben wird. In dem Fall findet keine Rekursion statt und das nächste Matching wird gebildet, das dann auch entsprechend geprüft wird. Falls \( V \) und \( X \) getauscht wurden, muss dies für die Tests berücksichtigt werden.

Es gibt Möglichkeiten, die Anzahl dieser Überprüfungen zu verringern. Sollte vor einer eventuellen Vertauschung von \( V \) und \( X \) gelten, dass \( |V| \leq |X| \) ist, so werden wegen Satz 2.20 in jedem Fall alle Knoten von \( V \) gematched. Wenn nach der Enumeration des ersten MaxWBM festgestellt wird, dass der derzeit berechnete Isomorphismus nicht ausgegeben wird, kann auf die weitere Enumeration der Matchings verzichtet werden, denn die Menge \( V' \) aus dem vorherigen Absatz ist wegen \( |V| \leq |X| \) für alle weiteren MaxWBM identisch. Die Rekursion kann auf dieser Ebene folglich abgebrochen werden.

Durch geschickte Festlegung von \( I \) kann die Zahl der Vergleiche weiter reduziert werden. Wenn für \( I \) zusätzlich gilt, dass \( I(R^v_{r_i}) = I(R^v_{r_i}) \pm 1 \) für benachbarte Knoten \( r_i, r_j \) ist, wie dies in Abbildung 3.6 dargestellt ist, ist es ausreichend, \( I(r) < I(r') \) für \( r' \in \{ R^v_r \mid v' \in V' \} \) sicherzustellen. Die Anzahl der Vergleiche kann somit halbiert werden. Die Maximum CSTI von \( R \) und \( S \), die in Abbildung 3.6 entlang der farbigen Knoten enumeriert werden, werden von der Ordnungsfunktion verworfen, bis auf jener, der vom gewurzelten Teilbaum \( r = R^v_{r_2} \) beginnend enumeriert wird. Falls \( r \neq R^v_{r_2} \) ist, gibt es jeweils einen gewurzelten Teilbaum \( r' \) mit \( I(r') < I(r) \), der während der Erweiterung des Isomorphismus ausgewählt wird. Die entsprechenden Rekursionen werden dann abgebrochen.

Bezogen auf Algorithmus 3.2 (GetMaximumCSTI) findet diese Prüfung immer dann statt, nachdem in Zeilen 9 und 11 ein Matching bestimmt wurde. Falls dann ein \( r' \) mit \( I(r') < I(r) \) existiert, werden sofort weitere MaxWBM enumeriert, bis \( I(r') > I(r) \) für alle \( r' \) ist oder kein weiteres Matchings existiert. Es ist möglich, dass in Zeile 17 für eine oder mehrere der Mengen \( M_1' \) bis \( M_k' \) kein einziger MCSTI existiert, der die Bedingung an \( I \) erfüllt. In diesem Fall wird dann nicht 1 zurückgegeben, sondern der Programmablauf in Zeile 11 mit der Bestimmung eines nächsten MaxWBM fortgesetzt.

(b) Vermeidung identischer Isomorphismen durch Kantenlöschung

Sei \( m \) die Größe eines Maximum CSTI von \( R \) und \( S \) und \( r = R^v_u \) der zuletzt gewählte gewurzelte Teilbaum in Zeile 2 von Algorithmus 3.1. Dort wird genau dann ein neuer gewurzelter Teilbaum gewählt, wenn die Enumeration auf \( (r, s) \) für alle gewurzelten Teilbäume \( s \) von \( S \) abgeschlossen wurde. Insbesondere bedeutet das, dass alle Maximum CSTI von \( R \) und \( S \), in denen gleichzeitig \( u \) und \( v \) durch \( \varphi \) abgebildet werden, enumeriert wurden. Folglich kann die Kante \( uv \) aus \( R \) gelöscht werden. Der so modifizierte Graph wird im folgenden mit \( R' \) bezeichnet. Die Lösung der Kante erfolgt zwischen Zeile 12 und
13 in Algorithmus 3.1. Anschließend werden die in $D$ gespeicherten Werte verworfen, alle gespeicherten MaxWBM gelöscht und $m' \leftarrow \text{SizeMCSTI}(R', S)$ bestimmt. Das hat zur Folge, dass der Algorithmus von Edmonds die neuen Größenwerte der Maximum CSTI für alle Paare von gewurzelten Bäumen sowie für $R'$ und $S$ neu berechnet. Wenn $m' < m$ ist, kann die Enumeration an dieser Stelle beendet werden. Ansonsten werden im Verlauf des Algorithmus weitere Kanten gelöscht, bis $m' < m$ erfüllt ist. Alle Maximum CSTI von $R$ und $S$, die nach der Kantenlöschung berechnet werden, bilden nicht mehr gleichzeitig die Knoten $u$ und $v$ durch $\varphi$ ab. Demzufolge unterscheiden sich diese Isomorphismen von allen zuvor gefundenen Isomorphismen. Die Ordnungsfunktion $I$ wird folglich nicht mehr benötigt.

Der Baum $R$ mutiert durch die Kantenlösung zu einem Wald. Für den Algorithmus von Edmonds stellt das aber kein Problem dar. Voraussetzung für diesen Algorithmus ist, dass der Graph azyklisch ist. Dies folgt aus der Berechnung von $D(r, s)$, die von den Blättern aus beginnt. Die Berechnung auf einem Wald ist dagegen identisch mit der aufeinanderfolgenden Berechnung auf den einzelnen Zusammenhangskomponenten.


Sei $e = xy$ die Kante, die gelöscht wird. Eine weitere Verbesserung des Laufzeit lässt sich erzielen, wenn nur diejenigen Werte $D(R_v^*, s)$ verworfen werden, für die $x$ oder $y$ in $V[R_v^*]$ enthalten ist. Denn wenn weder $x$ noch $y$ darin enthalten sind, kann sich auch der Wert $D(R_v^*, s)$ nicht ändern. Der Zusammenhangskomponente $Z$ sind $2|E(Z)|$ gewurzelte Bäume zugeordnet, auf der Hälfte davon wird der in $D$ gespeicherte Wert verworfen. Für die gelöschte Kante erfolgt natürlich auch keine Berechnung mehr. Die Anzahl der nötigen
3.3. ENUMERATION VON MAXIMUM COMMON SUBTREE ISOMORPHISMEN

Abbildung 3.7: Wahl des gewurzelten Teilbaums $r$ in Algorithmus 3.1 in der Variante „Kanten löschen“

Neuberechnungen ist somit durch $|E(Z)| - 1$ beschränkt. Falls MaxWBM gespeichert wurden, wie in Abschnitt 3.3.1 beschrieben, müssen entsprechend nur jene gelöscht werden, die in Verbindung zu den bipartiten Graphen stehen, für die der Wert in $D$ verworfen wurde. Falls der gewurzelte Teilbaum $r$ zu keinen Maximum CSTI führt, kann die Lösung der Kante übersprungen werden, da diese dann kein Teil irgendeines Maximum CSTI ist. Somit muss in diesem Fall $D$ auch nicht aktualisiert werden.

Vergleich der Verfahren

Der Vorteil des Vergleichs mit Hilfe der Ordnungsfunktion $I$ ist, dass dieses Verfahren zu den in Abschnitt 3.4 vorgestellten Enumerationsvarianten und -kriterien, sowie den Bezeichnern für Knoten und Kanten aus Abschnitt 3.5 kompatibel ist.


In Abschnitt 4.1 erfolgen Laufzeitvergleiche zwischen den beiden Verfahren. Dabei wird auch die Reihenfolge der Kantenauswahl variiert.

3.3.3 Laufzeit und Speicherverbrauch

Im Folgenden wird die Laufzeit von Algorithmus 3.1 abgeschätzt. Die Größe eines MaxWBM im Algorithmus von Edmonds wird mit dem Verfahren aus Abschnitt 2.4.2 berechnet. Dies liefert den zulässigen Teilgraphen und ein erstes perfektes Matching für die Enumeration der MaxWBM mit dem Verfahren aus Abschnitt 3.2.2. Mehrfache Lösungen sollen durch Kantenlösung vermieden werden. Seien die Bäume $R$ und $S$ gegeben, $m$ die Größe der MCSTI und $N_m$ die Anzahl der MCSTI.

Die Laufzeit des Algorithmus von Edmonds beträgt $O(|E_R| \cdot |E_S| \cdot \max\{|E_R|, |E_S|\}^3)$, wie in Satz 2.16 bewiesen wurde. Der Algorithmus kann bis zu $|V_R| + 1 - m$ mal wegen Neuberechnungen von $D$ aufgerufen werden. Ein Beispiel dafür ist ein Graph $R$ wie in
Abbildung 3.1 gegeben und ein Graph $S = (\{u, v\}, \{uv\})$. Obwohl beim Kantenlöschen nicht alle $D$-Werte neu berechnet werden müssen, bleibt der Zeitaufwand in diesem Beispiel bezüglich $O$-Notation gleich.

Die MCSTI werden über die Matchings erweitert. Im Folgenden wird abgeschätzt, wie oft Matchings bestimmt werden müssen und wie viel Zeit dies jeweils benötigt. Die Anzahl der MaxWBM in jedem Rekursionsschritt hängt von den gegebenen Bäumen $R$ und $S$ ab. Im worst case ist das jeweils nur ein MaxWBM. Außerdem kann die Struktur von $R$ und $S$ bedingen, dass jede enumerierte Matchingkante nur zu höchstens einem MCSTI gehört. Eine Speicherung der MaxWBM bringt dann keinen Vorteil. Ein Beispiel dazu sind Bäume $R$ und $S$, wobei $R$ ein Pfad ist und in $S$ nur genau ein Pfad dieser Länge enthalten ist. Die Zeit zur Bestimmung eines Matchings hängt im Wesentlichen vom zugehörigen zulässigen Teilgraphen ab. Für einen einzelnen MCSTI sind die Knoten all dieser zulässigen Teilgraphen, auf denen während der Bestimmung des MCSTI ein perfektes Matching enumeriert wird, paarweise disjunkt. Sei $m_i := \max\{r_i, s_i\}$ die Anzahl der Kanten des $i$-ten Matchings, das während der Bestimmung eines MCSTI enumeriert wird, wobei $r_i$ der Anzahl der Knoten aus $V_R$ und $s_i$ der Anzahl der Knoten aus $V_S$ entspricht, die in dem zulässigen Teilgraphen enthalten sind. Es gilt $\sum_i r_i < |V_R|$ und $\sum_i s_i < |V_S|$, wegen der paarweisen Disjunktheit der Knoten in den zulässigen Teilgraphen und weil zwei Knotenzuordnungen bereits durch die Wahl der gewurzelten Teilbäume $(r, s)$ festgelegt werden. Nach Abschnitt 3.2.2 beträgt die amortisierte Zeit zur Bestimmung des oben erwähnten $i$-ten Matchings $O(m_i^2)$. Es gilt $\sum_i m_i^2 = \sum_i \max\{r_i, s_i\}^2 = \sum_{\{i|r_i > s_i\}} r_i^2 + \sum_{\{i|r_i \leq s_i\}} s_i^2 < |V_R|^2 + |V_S|^2$. Die amortisierte Zeit pro MCSTI ist damit durch $O(|V_R|^2 + |V_S|^2)$ beschränkt.

Weil der modifizierte Algorithmus von Uno nach Satz 3.7 ein polynomial-delay-Algorithmus ist, kann jeder MCSTI in polynomialer Zeit bestimmt werden. Der Algorithmus von Edmonds benötigt ebenfalls nur polynomial viel Zeit und wird nicht häufiger als $|V_R| + 2 - m$ mal aufgerufen. Daraus ergibt sich, dass der vorgestellte Algorithmus zur Enumeration aller MCSTI ein polynomial-delay-Algorithmus ist. Wenn die zulässigen Teilgraphen über den Lösungsvektor $y^*$ gespeichert werden, wird dazu pro Teilgraph $O(|V_R| + |V_S|)$ Speicherplatz benötigt. In die selbe Schranke fällt ein initiales perfektes Matching. Für alle Teilgraphen und initialen Matchings ergibt sich eine Schranke von $O((|V_R| + |V_S|) \cdot |V_R| \cdot |V_S|)$. Der Speicherplatzverbrauch des modifizierten Algorithmus von Uno ist durch $O(|V_R| \cdot |V_S|)$ beschränkt. Für ein MCSTI der Größe $m$ ist der Bedarf durch $O(m \cdot |V_R| \cdot |V_S|)$ beschränkt. Der Platzbedarf für den Algorithmus von Edmonds ist nach Satz 2.16 durch $O(|V_R| \cdot |V_S|)$ beschränkt. Mit $m \in O(|V_R| + |V_S|)$ lässt sich folgendes Ergebnis ableiten.

**Theorem 3.8.** Der in Abschnitt 3.3 beschriebene Algorithmus zur Enumeration aller $N_m$ MCSTI mit Größe $m$ auf zwei Bäumen $R$ und $S$ ist ein polynomial-delay- und polynomial-
3.4. ENUMERATIONSVARIANTEN UND -KRITERIEN

Die Gesamtzeit ist durch \( O(|E_R| \cdot |E_S| \cdot \max\{|E_R|, |E_S|\}^3 \cdot (|V_R| + 2 - m) + (|V_R|^2 + |V_S|^2) \cdot N_m) \) beschränkt. Der benötigte Speicherbedarf ist durch \( O((|V_R| + |V_S|) \cdot |V_R| \cdot |V_S|) \) beschränkt.

3.4 Enumerationsvarianten und -kriterien

Die Enumeration in Abschnitt 3.3 kann aufgefasst werden als Enumeration von Common Subtree Isomorphismen in der Variante „Maximum“. Ein andere Variante ist die Enumeration aller maximalen CSTI (vgl. Definition 2.11).

Im folgenden Abschnitt 3.4.1 wird beschrieben, wie maximale CSTI enumeriert werden können. Im darauffolgenden Abschnitt 3.4.2 wird beschrieben, wie alle maximalen CSTI mit einer vom Benutzer vorgegebene Mindestgröße aufgezählt werden können. Die dort vorgestellten Algorithmen wurden vom Verfasser dieser Arbeit entwickelt. In Abschnitt 3.4.3 werden ergänzende Kriterien zu den vorgestellten Varianten besprochen.

3.4.1 Maximale Common Subtree Isomorphismen

Die Enumeration von maximalen CSTI (Algorithmus 3.3) ähnelt der Enumeration von Maximum CSTI. Der wesentliche Unterschied liegt darin, dass anstelle von Maximum Weight Bipartite Matchings alle maximalen Matchings aufgezählt werden. Solch eine Aufzählung wurde in Abschnitt 3.2.2 beschrieben. Die Aufzählung von maximalen Matchings stellt sicher, dass die sich daraus ergebenden Isomorphismen auch maximal sind. Das ist unmittelbar klar, denn die Erweiterung der Isomorphismen erfolgt über die berechneten Matchings. Wenn nur maximale, das heißt nicht erweiterbare, Matchings gebildet werden, ist auch der zugehörige Isomorphismus nicht erweiterbar, also maximal. Dementsprechend entfällt der Größentest aus Algorithmus 3.1, Zeile 4. Der Aufruf in Zeile 3 und 4 von Algorithmus 3.3 ruft die entsprechende Variante des Algorithmus aus Abschnitt 3.3.1 auf, wobei in diesem Fall maximale statt Maximum Matchings enumeriert werden. Mit Hilfe der Ordnungsfunktion \( I \) wird dabei sichergestellt, dass keine Isomorphismen mehrfach ausgegeben wird, wie dies in Abschnitt 3.3.2 beschrieben wurde.

Das Verfahren der Kantenlöschung lässt sich nicht übernehmen. Isomorphismen, die zuvor über eine gelöschte Kante erweitert werden konnten, würden fälschlicherweise als maximal erkannt werden, wie dies in Abbildung 3.8 zu sehen ist. Die Enumeration über das Paar \( (R_{r_2}^3, S_{s_2}^1) \) führt nach Löschung der Kante \( e \) zu dem Isomorphismus \( \varphi(r_2) = s_2 \), \( \varphi(r_3) = s_3 \). Dieser Isomorphismus ist, bezogen auf die Bäume der Eingabe, nicht maximal, denn er lässt sich mit \( \varphi(r_1) = s_1 \) erweitern. Im Algorithmus zur Enumeration der Maximum CSTI stellte dies wegen des Vergleichs mit der Größe eines Maximum CSTI der Bäume der Eingabe, in denen keine Kanten gelöscht sind, kein Problem dar.
KAPITEL 3. ENUMERATION VON COMMON SUBTREES UND CSTI

Eingabe: Zwei Bäume $R$ und $S$
Ausgabe: Alle maximalen Common Subtree Isomorphismen von $R$ und $S$

1: for all $r \in RST_R$ mit $I(r) < I(\tau)$ do
2: for all $s \in RST_s$ do
3: while GetMaximalCSTI($r, s, \varphi_1$) = 1 // Alg. 3.2 mit maximalen Matchings do
4: while GetMaximalCSTI($\bar{r}, \bar{s}, \varphi_2$) = 1 do
5: Verbinde die Isomorphismen $\varphi_1$ und $\varphi_2$ der gewurzelten Teilbäume zu einem
maximalen CSTI $\varphi$ von $R$ und $S$.
6: Gebe $\varphi$ aus.
7: end while
8: end while
9: end for
10: end for

Algorithmus 3.3: EnumMaximalCSTI($R, S$)

Abbildung 3.8: Kantenlöschung in $R$ führt zu nicht maximalen Isomorphismen

Laufzeit und Speicherverbrauch

Die Laufzeit von Algorithmus 3.3 lässt sich wie folgt abschätzen. Seien die Bäume $R$ und $S$ gegeben, $m$ die Größe eines MCSTI und $N_m$ die Anzahl der CSTI. Mehrfache Lösungen werden mit der Ordnungsfunktion $I$ erkannt. Da maximale CSTI enumeriert werden sollen, muss der Algorithmus von Edmonds nicht aufgerufen werden. Analog zu Abschnitt 3.3.3 muss im worst case für jede Erweiterung eines CSTI ein neues maximales Matching enumeriert werden. Die amortisierte Zeit dazu beträgt nach Abschnitt 2.4.1 pro Matching $O(1)$. Ein CSTI $\varphi$ der Größe $k + 1$ kann nach Abschnitt 3.3.2 (a) ausgehend von $k$ Paaren $(r, s)$ gewurzelter Teilbäume gefunden werden. Auch wenn $k - 1$ davon über die Ordnungsfunktion verworfen werden, muss für diese $k - 1$ CSTI die Zeit mit jeweils $O(k)$ abgeschätzt werden. Ein Beispiel dazu sind zwei Pfade $R$ und $S$, in denen die Ordnungsfunktion entlang des Pfades in eine Richtung aufsteigend definiert ist und die Erweiterung von $\varphi_1$ in diese Richtung durchgeführt wird. Mit $k \leq m$ lässt sich die amortisierte Zeit pro CSTI durch $O(m^2)$ beschränken.

Je nach Definition der Ordnungsfunktion und der gegebenen Bäume ist es möglich, dass mehr als polynomial viele aufeinander folgende maximale CSTI verworfen werden. Das passiert beispielsweise dann, wenn bei der Enumeration der maximalen Matchings in einem bipartiten Graphen mit $|V| = |X|$ für die gewurzelten Teilbäume $r'$ der zugeordneten
Kinder $I(r') > I(r)$ gilt, aber für jede Erweiterung entlang der Kinder der Kinder ein gewurzelter Teilbaum $r''$ mit $I(r'') < I(r)$ existiert. Deshalb handelt es sich um keinen polynomial-delay-Algorithmus. Der Speicherverbrauch zur Enumeration der MaxWBM auf einem vollständigen bipartiten Graphen beträgt nach Satz 2.22 $O(|V_R| + |V_S|)$. Die gleiche Schranke gilt auch für alle bipartiten Graphen zusammen, auf denen entlang eines CSTI enumeriert wird, denn alle Knoten in diesen bipartiten Graphen sind paarweise disjunkt. Die Bestimmung der Kombination in Algorithmus 3.2 (GetMaximumCSTI, hier mit maximalen Matchings) wird über die booleschen Variablen $f$ realisiert. Diese können jeweils in der aufrufenden Rekursion gespeichert und übergeben werden, so dass nicht mehr als $O(m)$ dieser Variablen gleichzeitig gespeichert werden müssen. Mit $m \in O(|V_R| + |V_S|)$ ergibt sich folgendes Resultat.

Satz 3.9. Der in Abschnitt 3.4.1 beschriebene Algorithmus zur Enumeration aller $N_m$ maximalen CSTI auf zwei Bäumen $R$ und $S$, mit $m$ als Größe eines MCSTI, ist ein polynomial-space- und polynomial-total-time-Algorithmus mit Zeitschranke $O(m^2 N_m)$ und Platzschranke $O(|V_R| + |V_S|)$.

3.4.2 Maximale CSTI mit Mindestgröße

Im Folgenden wird ein Algorithmus angegeben, der alle maximalen CSTI mit einer vom Benutzer festgelegten Mindestgröße enumeriert. Der Algorithmus greift Techniken aus den beiden Algorithmen 3.1 und 3.3 auf. Sei dazu $m$ die vom Benutzer geforderte Mindestgröße des CSTI. Die Idee zur Enumeration liegt darin, für einen Isomorphismus, der gerade berechnet wird, festzuhalten, welche Größe dieser durch Erweiterung höchstens noch erreichen kann. Die Erweiterungen erfolgen dann nur entlang solcher Kanten, die die vom Benutzer festgelegte Grenze $m$ nicht unterschreiten.

Der grundlegende Ablauf von Algorithmus 3.4 zur Enumeration von maximalen CSTI der Mindestgröße $m$ gestaltet sich wie folgt. Zunächst wird geprüft, ob es überhaupt einen CSTI der Mindestgröße $m$ gibt. Falls nicht, kann das Programm beendet werden (Zeile 1 bis 3). Anschließend werden, wie in den beiden vorherigen Algorithmen, alle Paare $(r,s)$ von gewurzelten Teilbäumen nacheinander aufgelistet (Zeile 4 und 5). In Zeile 6 wird sichergestellt, dass die Suche nach maximalen CSTI nur dann beginnt, wenn die Mindestgröße $m$ erreicht werden kann. Dies lässt sich, wie bekannt, mit Hilfe der aus dem Algorithmus von Edmonds gewonnenen Informationen feststellen. In Zeile 7 wird sichergestellt, dass $\varphi_1$ so groß ist, dass es mindestens einen Isomorphismus $\varphi_2$ auf den Teilbäumen $(r,s)$ gibt, so dass $|\varphi_1| + |\varphi_2| \geq m$ ist. In Zeile 8 werden dann alle Isomorphismen $\varphi_2$ aufgelistet, die diese Bedingung erfüllen. Anschließend wird der verbundene Isomorphismus ausgegeben, wenn er bisher nicht gefunden wurde. Die Prüfung erfolgt mit der Ordnungsfunktion $I$, wie dies in Abschnitt 3.3.2 beschrieben wurde. Die Aufzählung von maximalen Isomorphismen
**Eingabe:** Zwei Bäume $R$ und $S$, Mindestgröße $m$

**Ausgabe:** Alle maximalen CSTI von $R$ und $S$ mit einer Mindestgröße von $m$.

1: if SizeMCSTI($R, S$) < $m$ // Algorithmus 2.2 then
2: return Es gibt keinen CSTI mit einer Mindestgröße von $m$.
3: end if
4: for all $r \in RST_R$ mit $I(r) < I(\overline{r})$ do
5: for all $s \in RST_S$ do
6: if $D(r, s) + D(\overline{r}, \overline{s}) \geq m$ then
7: for all Maximalen CSTI $\varphi_1$ von $(r, s)$ mit $|\varphi_1| \geq m - D(\overline{r}, \overline{s})$ do
8: for all Maximalen CSTI $\varphi_2$ von $(\overline{r}, \overline{s})$ mit $|\varphi_2| \geq m - |\varphi_1|$ do
9: Verbinde die Isomorphismen $\varphi_1$ und $\varphi_2$ der gewurzelten Teilbäume zu einem maximalen CSTI $\varphi$ von $R$ und $S$.
10: Gebe $\varphi$ aus, falls $\varphi$ bisher nicht gefunden wurde.
11: end for
12: end for
13: end if
14: end for
15: end for

Algorithmus 3.4: EnumMinSizeMaximalCSTI($R, S, m$)

mit festgelegter Mindestgröße auf einem Paar von gewurzelten Teilbäumen (Zeilen 7 und 8) wird im Folgenden beschrieben.

**Enumeration von maximalen CSTI auf gewurzelten Teilbäumen mit vorgegebener Mindestgröße**

Dieser Abschnitt orientiert sich an Abschnitt 3.3.1. Hier werden im Wesentlichen nur die Änderungen gegenüber des dort vorgestellten Verfahrens angegeben. Sei der Aufruf mit den gewurzelten Teilbäumen $(r', s')$ und der Mindestgröße $m$ als Parameter erfolgt. Die Idee, nur CSTI von $(r', s')$ der Mindestgröße $m$ zu enumerieren, besteht darin, in einer globalen Variablen $p$ festzuhalten, wie groß der aktuell generierte Isomorphismus höchstmäßig werden kann. Zu Beginn, direkt nach dem Aufruf aus Algorithmus 3.4, sei daher $p$ durch $p := D(r, s)$ definiert. Die weiteren Zuordnungen der Knoten erfolgen dann so, dass stets $p \geq m$ gilt.

Die Erweiterung von $\varphi$ erfolgt, wie in Abschnitt 3.3.1 beschrieben, über Matchings. Falls $p = m$ gilt, läuft die weitere Rekursion, genau wie in Abschnitt 3.3.1, über die Maximum Weight Bipartite Matchings. Ansonsten wird versucht, den Isomorphismus über maximale Matchings zu erweitern.

---

2Es gibt je eine globale Variable für $\varphi_1$ und $\varphi_2$. 

KAPITEL 3. ENUMERATION VON COMMON SUBTREES UND CSTI
Sei \((r, s) = (R^u, S^w)\) das Paar von gewurzelten Teilbäumen, über die der Isomorphismus auf der aktuellen Rekursionsebene erweitert werden soll. Daraus folgt, dass in jedem Fall \(\varphi(v) = x\) zum Isomorphismus hinzugefügt wird. Sei weiterhin \(p'\) definiert durch den Wert \(p\) zu Beginn der Rekursion auf dieser Ebene. Vor der Aufzählung des ersten Matchings wird \(p\) um \(D(r, s) - 1\) verringert. Sei \(M = \{v_1 x_1, v_2 x_2, \ldots, v_k x_k\}\) das letzte enumerierte Matching und \(M'\) das vorherige Matching. Falls \(M\) das erste Matching ist, gilt \(M' = \emptyset\). Zu \(p\) wird der Wert \(p(M) := \sum_{v_j x_j \in M} D(R^u_{v_j}, S^w_{x_j})\) addiert, anschließend wird \(p\) um \(p(M')\) verringert. Diese Änderungen an \(p\) haben zur Folge, dass \(p\) dann der höchstmögliche Größe entspricht, die der Isomorphismus mit den weiteren Zuordnungen \(\varphi(v_j) = x_j\) erreichen kann. Sollte das maximale Matching \(M\) auch ein Maximum Matching sein, gilt \(p = p'\). Ansonsten gilt \(p < p'\). Die Aussagen in diesem Absatz ergeben sich aus der Definition von \(D\) in Abschnitt 2.2.2.

Gilt weiterhin \(p < m\), kann das enumerierte maximale Matching \(M\) auf keinen Fall zu einem Isomorphismus der Mindestgröße \(m\) führen. In diesem Fall werden weitere Matchings enumeriert, bis \(p \geq m\) gilt. Laut Abschnitt 3.3.1 müssen nach der Bestimmung eines Matchings alle Kombinationen von Isomorphismen auf den Teilbäumen \((R^u_{v_j}, S^w_{x_j})\) mit \(j \in \{1, \ldots, k\}\) gebildet werden. Zunächst wird, wie dort beschrieben, auf all diesen Teilbäumen der jeweils erste Isomorphismus berechnet. Zu beachten ist, dass, nachdem rekursiv ein Isomorphismus auf \((R^u_{v_j}, S^w_{x_j})\) für ein \(j\) berechnet wurde, sich der Wert in der globalen Variable \(p\) verringert haben kann. Dabei gilt aber stets \(m \leq p \leq p'\). Nachdem überall eine erste Lösung bestimmt wurde, steht in \(p\) die exakte Größe des Isomorphismus \(\varphi_1\) bzw. \(\varphi_2\) bezogen auf Zeile 7 bzw. 8 in Algorithmus 3.4. Die Aufzählung aller Kombinationen von Isomorphismen auf den gewurzelten Teilbäumen \((R^u_{v_j}, S^w_{x_j})\) erfolgt wie in Abschnitt 3.3.1 beschrieben. Der Wert \(p\) entspricht dabei während der gesamten Enumeration der Größe des Isomorphismus \(\varphi_1\) bzw. \(\varphi_2\), falls dieser vollständig berechnet wurde, also maximal ist, bzw. der höchstmöglichen Größe, die dieser noch erreichen kann, falls er noch nicht maximal ist, wie aus der weiteren Beschreibung hervorgeht.

Nachdem alle Kombinationen gebildet wurden, werden für den nächsten Isomorphismus \(\varphi_1\) bzw. \(\varphi_2\) zunächst die Zuordnungen der letzten Kombination entfernt, die entsprechenden Rekursion auf deren Kindern also beendet (vgl. nächster Absatz). Die Aufzählung auf dem Matching \(M\) ist dann abgeschlossen. An dieser Stelle wird dann, wie oben beschrieben, das nächste Matching gesucht, das zu einem Isomorphismus der Mindestgröße \(m\) führen kann.

Damit der Wert in \(p\) korrekt bleibt, muss, direkt bevor die Rekursion auf der aktuellen Ebene beendet wird, \(p\) zunächst um \(p(M)\) für das letzte Matching verringert und um \(D(r, s) - 1\) erhöht werden, genau andersherum wie ganz zu Beginn. Damit gilt dann wieder \(p = p'\).

Die Aktualisierung von \(p\) wird anhand Abbildung 3.9 illustriert. Mit \((r, s) = (R^u, S^w)\) gilt dort \(D(R^u, S^w) = 6\). Beim Aufruf aus Algorithmus 3.4 wird somit \(p\) auf 6 gesetzt. Sei außerdem \(m = 5\) die geforderte Mindestgröße. Vor der Aufzählung der Matchings wird \(p\)
KAPITEL 3. ENUMERATION VON COMMON SUBTREES UND CSTI

48

Abbildung 3.9: Beispiel zur Aktualisierung von $p$

nach der obigen Beschreibung um $D(R^u_v, S^w_x) - 1$ verringert, so dass $p = 1$ gilt. Sei $M_1 = \{v_1x_1, v_2x_2\}$ das erste enumerierte Matching. Für dieses wird $p$ um $p(M_1) = D(R^u_v, S^w_x) + D(R^u_v, S^w_x) = 2 + 3$ erhöht. Damit gilt wieder $p = 6$. Da $M_1$ ein Maximum Matching auf dem zugehörigen bipartiten Graphen ist, ist das genau der Wert, der in $p$ stehen muss. Sobald die Rekursion auf den gewurzelten Teilbäumen $(R^u_{v_1}, S^w_{x_1})$ und $(R^u_{v_2}, S^w_{x_2})$ abgeschlossen ist, wird das nächste maximale Matching gewählt. Sei dieses $M_2 = \{v_1x_1, v_2x_3\}$. Dann wird $p$ um $p(M_1) = 5$ verringert und um $p(M_2) = D(R^u_{v_1}, S^w_{x_1}) + D(R^u_{v_2}, S^w_{x_3}) = 2 + 1 = 3$ erhöht, so dass anschließend $p = 4$ gilt. Das entspricht genau der höchstmöglichen Größe, auf die der Isomorphismus $\varphi$ mit $\varphi(v) = x$, $\varphi(v_1) = x_1$ und $\varphi(v_2) = x_3$ erweitert werden kann. Dieses Matching wird übersprungen, da $4 = p < m = 5$ gilt. Nachdem alle Matchings und in allen Matchings, die zu der geforderten Mindestgröße führen können, auf den zugehörigen Paaren von gewurzelten Teilbäumen alle Kombinationen von Isomorphismen enumeriert wurden, muss der Wert in $p$ wieder zurückgesetzt werden. Für dieses Beispiel werden die anderen maximalen Matchings ausgelassen und angenommen, dass $M_2$ das letzte Matching sei. Dann wird $p$ um $p(M_2) = 3$ auf 1 verringert und um $D(R^u_v, S^w_x) - 1 = 5$ erhöht, so dass in $p$ wieder der Wert 6 steht. Zu jeder Zeit war in $p$ offenbar der korrekte Wert gespeichert. Das gilt auch für alle tiefere Rekursionsebenen.

Laufzeit und Speicherverbrauch

Dieser Algorithmus ist kein polynomial-total-time Algorithmus. Sei $m$ die geforderte Mindestgröße. Das Problem stellt die Enumeration der maximalen Matchings in einem bipartiten Graphen mit Mindestgewicht dar. Diese erfolgt immer dann, wenn die zu erreichende Größe des aktuellen CSTI größer als $m$ ist. Es ist leicht, bipartite Graphen mit Gewichten anzugeben, in denen das Verhältnis der Anzahl aller maximalen Matchings zu maximalen Matchings mit einem bestimmten Mindestgewicht nicht polynomiell beschränkt ist. Wenn diese bipartiten Graphen maßgeblich für die Gesamtmenge aller maximalen CSTI mit Mindestgröße $m$ verantwortlich sind, ist das Verhältnis verworrender Matchings zur Gesamtzahl der maximalen CSTI mit Mindestgröße nicht durch ein Polynom beschränkt. Ein Beispiel sind die Sterngraphen aus Abbildung 3.1, wobei jeweils die Hälfte der „Strahlen“ aus zwei
statt einer Kante besteht. Wenn dann \( m \) um eins geringer als die Größe eines MCSTI gewählt wird, führt das zu der hier beschriebenen Situation. Die Größe \( m \) wird nur dann erreicht, wenn, bis auf höchstens eine Ausnahme, jeweils Strahlen gleicher Länge einander zugeordnet werden.

Der Platzbedarf ist durch \( O\left((|V_R| + |V_S|) \cdot |V_R| \cdot |V_S|\right) \) beschränkt und entspricht der Schranke zur Enumeration der Maximum CSTI. Dies schließt die Enumeration von maximalen Matchings ein.

### 3.4.3 Enumerationskriterien

In diesem Abschnitt werden einige Kriterien vorgestellt, die sich mit den vorgestellten Varianten verbinden lassen.

#### Einzelne feste Knoten oder Kanten

Die Bäume der Eingabe repräsentieren häufig Strukturen aus der echten Welt. Deshalb können einige Knoten oder Kanten als besonders wichtig gelten, die auf jeden Fall im Common Subtree Isomorphismus vorhanden sein müssen. Für eine einzelne Kante oder einen einzelnen Knoten lässt sich das Problem sehr einfach lösen. Im Fall einer Kante beginnt die Enumeration ausgehend von dieser und nur dieser Kante. Bei einem Knoten beginnt die Enumeration dann von allen Kanten, zu denen der Knoten inzident ist.

#### Knotengrad


#### Zusätzlich nicht-maximale CSTI

Bisher wurden nur maximale CSTI aufgezählt, also solche, die in keinem anderen enthalten sind. Nicht-maximale CSTI lassen sich aufzählen, wenn bei der Bestimmung der Matchings alle möglichen Matchingsenumeriert werden und nicht nur solche, die maximal sind oder gar ein Maximum darstellen. Hier handelt es sich um ein nicht schwieriges kombinatorisches Problem, weshalb dazu kein konkreter Algorithmus angegeben wird.

### 3.5 Knoten- und Kantenbezeichner

In Abschnitt 3.4.3 wurde besprochen, dass Knoten oder Kanten auch gewisse Bedeutungen haben können. Beispielsweise gilt das für die in der Einleitung erwähnten Feature Trees
KAPITEL 3. ENUMERATION VON COMMON SUBTREES UND CSTI

(a) Zwei gelabelte Bäume

(b) Gewichtsfunktion $w$

Abbildung 3.10: Common Weighted Subtree Isomorphismus mit Gewicht $W(\varphi) = 10$

[24]. In Abschnitt 2.1 wurden gelabelte Graphen und ein Isomorphismus auf gelabelten Graphen beschrieben. Darauf aufbauend kann ein CSTI auf gelabelten Bäumen definiert werden.

Definition 3.10 (Common Labeled Subtree Isomorphismus, MCLSTI).

Seien $R = (V_R, E_R)$ und $S = (S_R, E_S)$ Bäume, und $R' = (V'_R, E'_R)$ und $S' = (V'_S, E'_S)$ Teilbäume dieser Bäume. Seien weiterhin $l_1 : V_R \cup E_R \rightarrow \Sigma$ und $l_2 : V_R \cup E_R \rightarrow \Sigma$ Labelfunktionen. Wenn $R'$ und $S'$ isomorph bezüglich $l_1$ und $l_2$ sind, wird der zugehörige Isomorphismus $\varphi : V'_R \rightarrow V'_S$ als Common Labeled Subtree Isomorphismus (CLSTI) bezeichnet.

Falls $\varphi$ grösstmöglich ist, wird $\varphi$ Maximum Common Labeled Subtree Isomorphismus (MCLSTI) genannt.

In Anlehnung an die gewichteten Matchings lässt sich auch ein gewichteter CSTI definieren.

Definition 3.11 (Common Weighted Subtree Isomorphismus, MCWSTI).

Seien $R = (V_R, E_R)$ und $S = (S_R, E_S)$ Bäume, und $R' = (V'_R, E'_R)$ und $S' = (V'_S, E'_S)$ Teilbäume dieser Bäume. Seien weiterhin $l_1 : V_R \cup E_R \rightarrow \Sigma$ und $l_2 : V_R \cup E_R \rightarrow \Sigma$ Labelfunktionen und $w : \Sigma \times \Sigma \rightarrow \mathbb{Q}$ eine symmetrische Gewichtsfunktion. Wenn $R'$ und $S'$ isomorph sind, wird der zugehörige Isomorphismus $\varphi : V'_R \rightarrow V'_S$ als Common Weighted Subtree Isomorphismus (CWSTI) mit Gewicht $W(\varphi)$, definiert durch $W(\varphi) := \sum_{v \in V'_R} w(l_1(v), l_2(\varphi(v))) + \sum_{uv \in E'_R} w(l_1(uv), l_2(\varphi(uv)))$, bezeichnet.

Falls $W(\varphi)$ grösstmöglich ist, wird $\varphi$ Maximum Common Weighted Subtree Isomorphismus (MCWSTI) genannt.

In Abbildung 3.10 ist ein CWSTI dargestellt. Die Buchstaben in den Knoten und an den Kanten stellen die Bezeichner dar. Die aufeinander abgebildeten Knoten geben zusammen ein Gewicht von $2 + 3 - 1 = 4$, die Kanten ergeben $4 + 2 = 6$, insgesamt also $W(\varphi) = 10$. Der CWSTI ist allerdings kein CLSTI, denn dort wird ein Knoten mit Bezeichner b auf einen Knoten mit Bezeichner c abgebildet.

*Es wird keine Isomorphie bezüglich $l_1$ und $l_2$ gefordert.*
CLSTI und CWSTI unterscheiden sich insofern, dass im ersten Fall die Bezeichner in jedem Fall übereinstimmen müssen. Im zweiten Fall dürfen sie verschieden sein. Wie gut sie „zueinander passen“, hängt von der Gewichtsfunktion $w$ ab. Beispielsweise könnten „wichtige Knoten“ ein bestimmtes Label erhalten und $w$ so definiert werden, dass dieses Label zusammen mit anderen Labeln einen besonders hohen Wert unter $w$ hat. Die Ähnlichkeit von Feature Trees wird im Rahmen dieser Arbeit mit Hilfe von MCWSTI bestimmt. Die Knoten in Feature Trees repräsentieren ein Fragment eines Moleküls. Die in [24] beschriebene Vergleichsfunktion (comparison function) $c$ ist definiert durch $c(0,0) = 1$ und $c(a,b) = \frac{2 \min(a,b)}{a+b}$ für $a + b > 0$, wobei $a$ und $b$ der Anzahl der Nicht-Wasserstoff-Atome des entsprechenden Fragments entsprechen. Übertragen auf CWSTI stehen die Label für die Anzahl dieser Atome, mit $\Sigma \subseteq \mathbb{N}$. Die Gewichtsfunktion $w$ wird definiert durch $w := c$. Dieser Arbeit steht eine Datenbank mit 51415 Feature Trees zur Verfügung. Ähnlichkeitsanalysen auf den Feature Trees, mit Hilfe von MCWSTI, erfolgen in Abschnitt 4.1. In den folgenden beiden Abschnitten 3.5.1 und 3.5.2 wird beschrieben, wie sich CWSTI und CLSTI finden lassen. Dabei werden die Varianten „Maximum“, „maximal“ und „Mindestgröße“ bzw. „Mindestgewicht“ betrachtet.

### 3.5.1 Enumeration von Common Weighted Subtree Isomorphismen


#### Knotenbezeichner

Seien $R$ und $S$ zwei Bäume, sowie $\varphi$ ein CWSTI mit den Bezeichnungen aus Definition 3.11. Seien $v$ und $x$ Knoten mit $\varphi(v) = x$. Dann wird das Gewicht $W(\varphi)$ für den Knoten $v$ um $w(l_1(v), l_2(x))$ erhöht. Dieses Gewicht lässt sich durch zwei Änderungen an Algorithmus 2.1 (GetD) berücksichtigen. In Zeile 3 und 8 wird die „1“ durch $w(l_1(v), l_2(x))$ ersetzt. Mit diesen Änderungen steht dann in $D(r,s)$ das Gewicht eines MCWSTI auf dem Paar von gewurzelten Teilbäumen $(r,s)$. So lang die symmetrische Gewichtsfunktion $w$ positiv ist, also $w(a,b) > 0$ für alle $a,b \in \Sigma$, muss nichts weiter beachtet werden. Die Algorithmen 3.1 bis 3.4 können dann unverändert übernommen werden.

Falls $a,b \in \Sigma$ mit $w(a,b) \leq 0$ existieren, muss ein MCWSTI nicht mehr zwingend maximal sein, wie Abbildung 3.11 zu entnehmen ist. Die dort dargestellten Zahlen repräsentieren den Wert $w(l_1(r_i), l_2(s_i))$. So ist der Isomorphismus $\varphi_1$, definiert durch $\varphi_1(r_2) = s_2$, ein nicht-maximaler MCWSTI. Das gleiche gilt für $\varphi_2$, definiert durch $\varphi_2(r_1) = s_1$ und $\varphi_2(r_2) = s_2$, sowie für $\varphi_3(r_i) = s_i$ für $i \in \{2,3,4\}$. Problematisch ist in diesem Fall
KAPITEL 3. ENUMERATION VON COMMON SUBTREES UND CSTI

Abbildung 3.11: Der Isomorphismus $\varphi(r_2) = s_2$ ist ein nicht-maximaler Maximum CWSTI.

zum einen, dass MCWSTI $\varphi$ mit $|\varphi| = 1$ existieren können. Diese werden dann generell nicht gefunden, wenn die Suche stets von einem Kantenpaar ausgehend startet. Der Enumerationsalgorithmus müsste damit um die Suche nach Isomorphismen $\varphi$ mit $|\varphi| = 1$ erweitert werden. Zum anderen führen nicht-maximale MCWSTI in Verbindung mit Kantenlöschen (vgl. Abschnitt 3.3.2) dazu, dass diese mehrfach gefunden werden können. Beispielsweise wird $\varphi_3$ gefunden, wenn keine Kante entfernt wurde, aber auch dann, wenn die Kante $r_1r_2$ gelöscht wurde. Diese Probleme lassen sich vermeiden, wenn zum einen Maximalität für alle CWSTI vorausgesetzt wird und zum anderen die Ordnungsfunktion $I$ genutzt wird, um mehrfach gleiche Isomorphismen zu erkennen. In dem dieser Arbeit zugehörigen Programm wurde das genau so umgesetzt und lässt sich für Algorithmus 3.1 und 3.4 gleichermaßen verwenden. Maximale CSTI (Algorithmus 3.3) und CWSTI unterscheiden sich nicht, da die Größe bzw. das Gewicht für diese nicht relevant ist.

Die Ausgabe kann in allen drei Enumerationsvarianten um $W(\varphi)$ erweitert werden. Dieser Wert lässt sich einerseits aus $\varphi$ in $O(|\varphi|)$ Zeit berechnen. Andererseits steht er in der globalen Variablen $p$ zur Verfügung, die in Abschnitt 3.4.2 zur Aufzählung aller maximalen CSTI mit Mindestgewicht beschrieben wurde. In der dieser Arbeit beiliegenden Implementierung wird $p$ in allen drei Varianten stets aktuell gehalten. So kann die Größe im Fall von CSTI bzw. das Gewicht im Fall von CWSTI in $O(1)$ ausgegeben werden.

**Kantenbezeichner**

Kantenbezeichner werden auf eine recht ähnliche Art berücksichtigt. Seien $(r, s) = (R_v^w, S_x^w)$ zwei gewurzelte Teilbäume, für die das Gewicht eines MCWSTI bestimmt werden soll. Im Programm wurde die Entscheidung getroffen, das Gewicht $w(l_1(uv), l_2(wx))$ zu $D(r, s)$ hinzuzufügen. Die folgende Beschreibung richtet sich nach dieser Entscheidung. Analog zu den Knotenbezeichnern wird in Algorithmus 2.1 die Berechnung in Zeile 3 und 8 verändert. Dort wird jeweils das Gewicht $w(l_1(uv), l_2(wx))$ hinzugefügt. Wenn dann in Zeile 6 die Gewichte der Matchingkanten festgelegt werden, ist durch obige Entscheidung das Gewicht $w(l_1(vv_i), l_2(xx_j))$ in $D(R_v^w, S_x^w)$ enthalten, so dass an dieser Stelle dann nichts addiert werden muss.
Zu beachten ist, dass in der Summe \( D(r,s) + D(\tau, \pi) \) das Gewicht \( w(l_1(uv), l_2(wx)) \) zweimal addiert wurde. In den Enumerationsalgorithmen für die MCWSTI und die maximalen CWSTI mit Mindestgewicht muss dies entsprechend berücksichtigt werden. Gleiches gilt für Zeile 8 in Algorithmus 2.2 (SizeMCSTI). Das doppelt gezählte Gewicht muss entsprechend einmal abgezogen werden, um das korrekte Gewicht des verbundenen Isomorphismus zu erhalten.

Die Aussagen in Bezug auf mehrfach gleiche Isomorphismen und Kantenlösungen lassen sich aus dem vorherigen Abschnitt übernehmen. In der Implementierung wird das Gewicht durch die Kantenbezeichner in der globalen Variablen \( p \) berücksichtigt, so dass die Ausgabe des Gewichts mit Kantenbezeichnern in \( O(1) \) erfolgen kann.

**Laufzeit und Speicherverbrauch**

Die Laufzeit und der Speicherverbrauch zur Enumeration von maximalen CWSTI, maximalen CWSTI mit Mindestgewicht und Maximum CWSTI stimmen mit den Laufzeiten und dem Speicherverbrauch zur Enumeration der entsprechenden Verfahren ohne Bezeichner überein, da Bezeichner im Wesentlichen nur Einfluss auf die Bestimmung der Werte \( D \) haben. Die Zeit- und Platzschranken bleiben dabei unverändert.

### 3.5.2 Enumeration von Common Labeled Subtree Isomorphismen

In Verbindung mit Common Labeled Subtree Isomorphismen muss das Label von aufeinander abgebildeten Knoten und Kanten übereinstimmen. Es macht deshalb Sinn, im Algorithmus von Edmonds bei der in Abschnitt 2.2.2 beschriebenen dynamischen Programmierung anzusetzen.

Sei dazu \((r,s) = (R^u_v, S^w_x)\) das Paar von gewurzelten Bäumen, für das die Größe eines MCLSTI ermittelt werden soll. Algorithmus 2.1 (GetD) wird dazu folgendermaßen modifiziert. Zwischen Zeile 1 und 2 wird ein Test eingefügt. Falls \( l_1(u) \neq l_2(w) \) oder \( l_1(v) \neq l_2(x) \), wird \( D(R^u_v, S^w_x) \) auf \(-1\) gesetzt. Dieser Wert signalisiert, dass ein CLSTI \( \varphi \) mit \( \varphi(u) = w \) und \( \varphi(v) = x \) nicht möglich ist. In Abschnitt 2.2.2 wurde der \((r,s)\) zugehörige bipartite Graph beschrieben, auf dem das Gewicht eines Maximum Weight Bipartite Matching bestimmt werden soll. In Zusammenhang mit CLSTI wird die Kante \( v_i x_j \) aus dem bipartiten Graphen entfernt, falls \( D(R^u_v, S^w_x) = -1 \) ist, denn es gilt dann \( l_1(v_i) \neq l_2(x_j) \) oder \( l_1(v_i) \neq l_2(x_j) \). Zeile 8 in Algorithmus 2.1 wird dementsprechend modifiziert. In Abschnitt 2.4.2 wurde beschrieben, wie das Gewicht eines MaxWBM auf einem bipartiten Graphen bestimmt werden kann. Als Eingabe für das dort vorgestellte Verfahren sind insbesondere auch nicht vollständige bipartite Graphen erlaubt.

Die Enumerationsalgorithmen 3.1 bis 3.4 greifen auf die Enumeration von Maximum bzw. maximalen Matchings zurück. Die Enumeration von MaxWBM wurde in Abschnitt 3.2.2 beschrieben. Ein Verfahren zur Enumeration von maximalen Matchings auf nicht voll-

**Laufzeit und Speicherverbrauch**


Maximale Matchings auf einem nicht notwendig vollständigen bipartiten Graphen lassen sich mit Platzbedarf \( O(k) \) in amortisierter Zeit \( O(n) \) pro maximalem Matching berechnen, wobei \( n \) der Knoten- und \( k \) der Kantenzahl entspricht [27]. Mit den Bezeichnungen aus Abschnitt 3.3.3 ergibt sich \( \sum_i m_i < \sum_i r_i + s_i < |V_R| + |V_S| \). Unter Zuhilfenahme der Ordnungsfunktion zur Erkennung identischer maximaler CLSTI ergibt dies eine Zeitschranke von \( O((|V_R| + |V_S|)^2 \cdot N_m) \) zur Enumeration von maximalen CLSTI. Da die Knoten in allen bipartiten Graphen entlang des CLSTI paarweise disjunkt sind, ist der Speicherbedarf für die Enumeration der maximalen Matchings durch \( O(|V_R| \cdot |V_S|) \) beschränkt. Das ist auch gleichzeitig die Platzschranke für die Enumeration der maximalen CLSTI, da der Platzbedarf für die Enumeration von maximalen CSTI darin enthalten ist.

Die Aussagen zu maximalen CSTI mit Mindestgröße lassen sich auf maximale CLSTI mit Mindestgröße übertragen.

### 3.6 Enumeration von Common Subtrees


Stattdessen wurde ein Idee verfolgt, die unter Einsatz von zusätzlichem Speicherplatz mit Hilfe von dynamischer Programmierung zunächst maximale bzw. Maximum CST auf

### 3.6.1 Baumkanonisierung

Um gewurzelte Bäume speichern zu können, bietet sich der von Valiente definierte isomorphism code an. Dieser code wird im Folgenden Baumkanonisierung genannt und unterscheidet sich bezüglich der Sortierung.

**Definition 3.12 (Baumkanonisierung [28]).** Sei $T = (V, E)$ ein gewurzelter Baum mit Wurzel $r$ und Kindern $r_1, \ldots, r_k$ von $r$. Die Baumkanonisierung $K(T) := K(r)$ ist eine Folge von natürlichen Zahlen und definiert durch $K(r) := (|V|); K(r_1); \ldots; K(r_k)$. Die Kinder von $r$ seien dabei in nicht aufsteigender\footnote{Valiente definiert die Ordnung als nicht absteigend [28].} lexicographischer Ordnung bezogen auf ihre Baumkanonisierung sortiert.

Satz 3.13. Seien \( R \) und \( S \) zwei Bäume mit Wurzeln \( r \) und \( s \). Dann existiert ein Isomorphismus \( \varphi : V(R) \to V(S) \) mit \( \varphi(r) = s \) genau dann, wenn \( K(R) = K(S) \) ist.


Satz 3.14. \( K(T) \) lässt sich mit Zeitschranke \( O(|T|^2) \) und Platzschranke \( O(|T|) \) bestimmen.

Der Satz 3.14 zugrunde liegende Algorithmus [28] berechnet rekursiv die Kanonisierungen für die Kinder, und sortiert diese dann mit dem wohlbekannten Sortieralgorithmus Radixsort. Die Zahlenfolgen werden anschließend aneinander gehängt, so dass daraus \( K(T) \) entsteht.

3.6.2 Einfügen einer Baumkanonisierung in einen sortierten Binärbaum

Die berechneten Maximum bzw. maximalen CST werden für jedes Paar \((r, s)\) von gewurzelten Teilbäumen in Form ihrer Baumkanonisierung in einem Binärbaum \( B(r, s) \) gespeichert. Die Sortierung erfolgt dabei lexikographisch bezogen auf die einzelnen Zahlen der Folge, von vorn beginnend. Sei \(|B(r, s)|\) definiert als die Anzahl der Zahlenfolgen in dem Binärbaum. Die Zeit zum Einfügen von \( K(T) \) in \( B(r, s) \) ist durch \( O(|T|\log(|B(r, s)|)) \) beschränkt. Der Faktor \(|T|\) ist die Zeitschranke zum Vergleich von \( K(T) \) mit einer anderen Folge. Der zweite Faktor \( \log(|B(r, s)|) \) ergibt sich als Zeitschranke für die binäre Suche, wenn der Baum ausgeglichen gehalten wird, wie dies beispielsweise bei AVL- oder Rot-Schwarz-Bäumen der Fall ist. Die Folge \( K(T) \) wird nur dann in \( B(r, s) \) eingefügt, wenn sie dort noch nicht vorhanden ist.

Beim Verbinden der CST von \((r, s)\) und \((\overline{r}, \overline{s})\) erfolgt die Speicherung auf die selbe Art und Weise. Die dazu nötige Wahl der Wurzel wird in Abschnitt 3.6.4 beschrieben. Die Bestimmung der Kanonisierung \( K(T) \) von \( T \) und die Einfügung in einen Binärbaum \( B \) lässt sich als Satz formulieren.

Satz 3.15. Sei \( B \) ein Binärbaum mit Zahlenfolgen als Elementen, \( m := |B| \) die Anzahl der gespeicherten Zahlenfolgen und \( T \) ein gewurzelter Baum mit \( n \) Knoten. Die Zeit zum Einfügen von \( K(T) \) in \( B \) ist durch \( O(n^2 + n\log m) \) beschränkt.

Bemerkung 3.16. Es gibt mehrere Algorithmen, die für zwei Bäume der Größe \( n \) in \( O(n) \) Zeit entscheiden können, ob diese isomorph sind. Ein solcher Algorithmus wird beispielsweise von Aho und Hopcroft angegeben [1]. Die Kanonisierung nach Valiente ermöglicht
allerdings einen sehr schnellen Isomorphietest gegen eine Menge von Bäumen, wie gezeigt wurde. Des Weiteren wird die Kanonisierung $K(T)$ für alle CST auf einem Paar $(r, s)$ von gewurzelten Teilbäumen schrittweise berechnet und Zwischenergebnisse wiederverwendet. Es ist daher nicht unmittelbar klar, ob ein Isomorphietest in $O(n)$, der dann möglicherweise gegen alle gespeicherten Bäume durchgeführt werden muss und möglicherweise keine Zwischenergebnisse wiederverwenden kann, schneller ist. Ein anderer Vorteil der Kanonisierung nach Valiente ist die Tatsache, dass aus der Zahlenfolge der zugehörige Baum in $O(n)$ Zeit erstellt werden kann.

### 3.6.3 Dynamische Programmierung

In diesem Abschnitt wird beschrieben, wie für ein Paar $(r, s)$ von gewurzelten Teilbäumen alle Maximum CST bestimmt werden können. Falls $r$ oder $s$ ein Blatt ist, gibt es nur einen Maximum CST $T$. Dieser besteht aus genau einem Knoten. Wegen $K(T) = (1)$ gilt somit $B(r, s) = \{(1)\}$. Ansonsten können die CST mit Hilfe von MaxWBM bestimmt werden, ähnlich dem Vorgehen aus Abschnitt 3.3.1 zur Enumeration von Maximum CSTI auf gewurzelten Teilbäumen. Wie dort beschrieben, lässt sich die Menge aller Maximum CSTI disjunkt zerlegen, wobei jedem Matching dann eine Teilmenge der Maximum CSTI zugeordnet ist. Um alle Maximum CST von $r$ und $s$ zu bestimmen, werden, genau wie in Abschnitt 3.3.1, die Maximum Matchings enumeriert und auf diesen jeweils alle Maximum CST bestimmt. Das geschieht analog zu Abschnitt 3.3.1, indem alle Kombinationen der Maximum CST auf den einander zugeordneten Kindern unter einer neuen Wurzel verbunden und in $B(r, s)$ eingefügt werden, falls sie dort noch nicht vorhanden sind. Die Maximum CST der Kinder werden allerdings nicht rekursiv immer wieder enumeriert, sondern nur genau einmal berechnet, und stehen dann, in einem Binärbaum gespeichert, zur Verfügung.

Der wesentliche Vorteil gegenüber der zu Beginn von Abschnitt 3.5 beschriebenen Idee, alle Maximum CSTI zu berechnen und daraus die Maximum CST zu bestimmen, liegt darin, dass hier in jeder Stufe der dynamischen Programmierung identische CST verworfen werden und nicht erst dann, wenn ein Maximum CST für die Bäume der Eingabe bestimmt ist. Die Laufzeitvergleiche in Abschnitt 4.2 belegen, dass die Enumeration von Maximum CST mit diesem Verfahren deutlich schneller ist, als die Maximum CST aus den Maximum CSTI zu generieren. In Algorithmus 3.5 (GetB) ist das in diesem Abschnitt beschriebene Verfahren festgehalten. In Zeile 7 und 8 wird die Kanonisierung des verbundenen Maximum CST berechnet. GetD($R_u^v, S_x^w$) entspricht der Anzahl der Knoten dieses Maximum CST. Die Binärbäume $B(r, s)$ seien zu Beginn alle leer.

### 3.6.4 Zusammenfügen der Teillösungen

Die Hauptalgorithmus zur Bestimmung aller Maximum CST von zwei Bäumen $R$ und $S$ orientiert sich an Algorithmus 3.1 zur Enumeration aller Maximum CSTI. Der wesentliche
Eingabe: $R^u_v, S^w_x$
Ausgabe: Verweis auf $B(R^u_v, S^w_x)$

1: if $B(R^u_v, S^w_x) = \emptyset$ // $B(R^u_v, S^w_x)$ noch nicht berechnet then
2: if $R^u_v$ oder $S^w_x$ ist ein Blatt then
3: $B(R^u_v, S^w_x) \leftarrow \{(1)\}$
4: else
5: for all MaxWBM $M = \{v_1x_1, \ldots, v_kx_k\}$ auf dem durch die Kinder von $v$ und $x$ definierten bipartiten Graphen, vgl. Abschnitt 2.2.2 und 3.3.1 do
6: for all $(K_1, \ldots, K_k) \in \text{GetB}(R^u_{v_1}, S^w_{x_1}) \times \ldots \times \text{GetB}(R^u_{v_k}, S^w_{x_k})$ do
7: Sortiere die Kanonisierungen $K_1$ bis $K_k$ lexikographisch nicht aufsteigend.
8: Die sortierte Reihenfolge sei $K'_1$ bis $K'_k$.
9: $K \leftarrow (\text{GetD}(R^u_v, S^w_x)); K'_1; \ldots; K'_k$
10: Füge $K$ in $B(R^u_v, S^w_x)$ ein, falls noch nicht vorhanden.
11: end for
12: end if
13: end if
14: return Verweis auf $B(R^u_v, S^w_x)$

Algorithmus 3.5: GetB($R^u_v, S^w_x$)


Unterschied ist, dass nicht Maximum CSTI von $(r, s)$ bzw. $(\tau, \pi)$ miteinander verbunden werden, sondern die in $B(r, s)$ bzw. $B(\tau, \pi)$ gespeicherten Kanonisierungen der Maximum CST. Diese werden in einen Binärbaum $B$ eingefügt, in dem alle bisher berechneten Kanonisierungen der Maximum CST von $R$ und $S$ gespeichert sind. Falls eine Kanonisierung noch nicht vorhanden war, wird sie gespeichert und ausgegeben. Ansonsten wird sie verworfen und die nächste Kanonisierung berechnet.

3.6. ENUMERATION VON COMMON SUBTREES

**Eingabe:** Zwei Bäume \( R \) und \( S \)

**Ausgabe:** Alle Maximum Common Subtrees von \( R \) und \( S \)

1: \( B \leftarrow \emptyset \)
2: \( m \leftarrow \text{SizeMCSTI}(R, S) \) // Algorithmus 2.2
3: \textbf{for all} \( r \in RST_R \) mit \( I(r) < I(\tau) \) \textbf{do}
4: \textbf{for all} \( s \in RST_S \) \textbf{do}
5: \textbf{if} \( D(r, s) + D(\tau, s) = m \) \textbf{then}
6: \textbf{for all} Kanonisierungen \( K_1 \) aus \( \text{GetB}(r, s) \) \textbf{do}
7: \textbf{for all} Kanonisierungen \( K_2 \) aus \( \text{GetB}(\tau, s) \) \textbf{do}
8: \( K \leftarrow K_1; K_2 \)
9: Erhöhe die erste Zahl der Zahlenfolge \( K \) um \( |K_2| \).
10: Berechne \( T' \leftarrow T(K) \) und bestimme den Wurzelknoten \( w \), wie in Abschnitt 3.6.4 beschrieben.
11: Berechne \( K(T') \).
12: \textbf{if} \( K(T') \) ist noch nicht in \( B \) vorhanden \textbf{then}
13: Gebe \( K(T') \) aus, und füge \( K(T') \) in den Binärbaum \( B \) ein.
14: \textbf{end if}
15: \textbf{end for}
16: \textbf{end for}
17: \textbf{end if}
18: \textbf{end for}
19: \textbf{end for}

**Algorithmus 3.6:** EnumMaximumCST(R, S)

**Definition 3.17 (Exzentrizität, Zentrum eines Baumes [16]).** Sei \( T = (V, E) \) ein Baum. Die **Exzentrizität** eines Knotens \( u \) ist definiert als das Maximum über die Länge der Pfade von \( u \) zu allen Knoten \( v \in V \).

Das Zentrum eines Baumes sind diejenigen Knoten aus \( V \), die eine minimale Exzentrizität aufweisen.

Der gesamte Ablauf ist in Algorithmus 3.6 festgehalten. In Zeile 6 und 7 werden die Kanonisierungen enumeriert. Zu beachten ist, dass nur die Kanonisierungen berechnet werden, die tatsächlich gebraucht werden, da GetB nur auf den Paaren von gewurzelten Teilbäumen aufgerufen wird, die zu einem Maximum CST führen können. In Zeile 8 und 9 werden zwei Kanonisierungen verbunden. In den gewurzelten Baum \( T(K_1) \) wird \( T(K_2) \) unter der Wurzel eingehangen. Die Anzahl der Knoten erhöht sich dadurch um \( |K_2| \), weshalb die erste Zahl der Zahlenfolge \( K \) um diesen Wert erhöht wird. In Zeile 10 wird ein \( K \) zugehöriger Baum \( T(K) \) berechnet. Dies ist in Linearzeit bezogen auf \( |K| \) möglich. Dazu werden rekursiv die Kind-Teilbäume von \( K \) erstellt und diese dann unter der Wurzel verbunden. Die anderen Schritte wurden in diesem Abschnitt hinreichend beschrieben.

3.6.5 Maximale CST und maximale CST mit Mindestgröße

Im Folgenden wird beschrieben, wie anstelle von Maximum CST maximale CST mit Mindestgröße \( m \) enumeriert werden. Das Vorgehen ähnelt der Enumeration von maximalen CSTI mit Mindestgröße. Dazu wird Algorithmus 3.5 (GetB) so modifiziert, dass in Zeile 5 maximale Matchings, und nicht Maximum Matchings, aufgezählt werden, falls \( m \) kleiner der Größe eines MCST ist. Falls alle maximalen CST aufgezählt werden sollen, ohne eine Mindestgröße festzulegen, können sämtliche im Folgenden beschriebenen Prüfungen auf die Größe entfallen. Das betrifft die dynamische Programmierung und das Zusammenfügen der Teillösungen.

Ein wesentlicher Unterschied betrifft Zeile 9 und das Einfügen von \( K \) in \( B(R^u_v, S^w_x) \). Dadurch, dass maximale Matchings enumeriert werden, können in \( B(R^u_v, S^w_x) \) Kanonisierungen unterschiedlicher Länge gespeichert werden. Da nur maximale CST numeriert werden sollen, dürfen Kanonisierungen \( K \) nur dann eingefügt werden, wenn es keine Kanonisierung \( K' \) in \( B(R^u_v, S^w_x) \) gibt, so dass \( T(K) \) isomorph zu einem Teilbaum von \( T(K') \) ist. Dies wäre ein Widerspruch zur Maximalität des CST \( T(K) \). Sollte \( K \) nicht in \( B(R^u_v, S^w_x) \) vorhanden sein und kein solches \( K' \) existieren, kann \( K \) in \( B(R^u_v, S^w_x) \) eingefügt werden. Allerdings sind in dem Fall alle Kanonisierungen \( K' \) aus dem Binärbaum zu löschen, für die der CST \( T(K') \) isomorph zu einem Teilbaum von \( T(K) \) ist. Insgesamt wird so sichergestellt, dass die durch die Kanonisierungen repräsentierten Bäume nicht ineinander enthalten sind.

Ein weiterer Unterschied liegt in der Auswahl der Kanonisierungen \( K_1 \) bis \( K_k \) in Zeile 6. Nur dann, wenn die Größe der Kanonisierung \( K \) mindestens \( m - D(R^u_v, S^w_x) \) beträgt, wird diese in den Binärbaum eingefügt. Denn nur in diesem Fall ist es möglich, zusammen mit einer Kanonisierung aus \( B(R^u_v, S^w_x) \), die Mindestgröße \( m \) zu erreichen. Weil die Kanonisierungen in den Binärbäumen sortiert vorliegen, können die Tupel \( (K_1, \ldots, K_k) \), die zu einer Kanonisierung \( K \) mit Mindestgröße \( m \) führen, effizient aufgezählt werden. Das Verfahren ist ähnlich der Aufzählung der Kombination von Isomorphismen auf gewurzelten Teilbäumen aus Abschnitt 3.3.1 und 3.4.2. Die erste Auswahl der Kanonisierungen \( K_1 \) bis \( K_k \) ist
3.6. ENUMERATION VON COMMON SUBTREES

jeweils die lexikographisch Größte aus den entsprechenden Binärbäumen. Wenn für ein \( i \)
und eine lexikographisch kleinste Kanonisierung \( K_i \) alle Kombinationen gebildet wurden,
wird für \( K_i \) bis \( K_k \) die lexikographisch größte und für \( K_{i-1} \) die lexikographisch nächsten
Kanonisierung gewählt. Das Vorgehen ist analog zu den genannten Abschnitten. Wenn
für ein \( i \) die lexikographisch nächstkleinere Kanonisierung \( K_i \) bestimmt wird, und diese zu-
sammen mit den anderen bestimmten Kanonisierungen nicht zur Mindestgröße \( m \) führen
cann, können alle lexikographisch kleineren Kanonisierungen aus \( B(R^n_v, S^n_x) \) übersprungen
werden, da diese höchstens genau so groß wie \( K_i \) sind. Für \( K_1 \) wird dann die lexikographisch
größtmögliche Kanonisierung und für \( K_{i-1} \) die lexikographisch nächstkleinere gewählt.

Die Überprüfung, ob für zwei Kanonisierungen mit \( |K_1| < |K_2| \) der CST \( T(K_1) \) iso-
morph zu einem Teilbaum von \( T(K_2) \) ist, lässt sich nicht über einen einfachen Vergleich
der Zahlenfolgen herausfinden, wie Abbildung 3.14 zu entnehmen ist. Der Baum \( S \) ist iso-
morph zu einem Teilbaum von \( R \), die Folge \( K(s) \) ist aber keine Teilfolge von \( K(r) \). Abhilfe
schaffen Algorithmen, die das Subtree Isomorphie Problem lösen.

**Definition 3.18 (Subtree Isomorphismus [21])**. Seien \( R \) und \( S \) Bäume, und \( R' = (V'_R, E'_R) \) ein Teilbaum von \( R \). Wenn \( R' \) und \( S \) isomorph sind, wird ein zugehöriger Iso-
morphismus \( \varphi : R' \to S \) als Subtree Isomorphismus (STI) bezeichnet.

Das Subtree Isomorphismus Problem stellt die Frage, ob ein Teilbaum von \( R \) in \( S \)
Ein Algorithmus mit Laufzeit \( O((|S|^{1.5}/\log |S||R|)) \), der dieses Problem löst, wurde von
Shamir und Tsur vorgestellt [26].

Die zusätzlichen Prüfungen auf Subtree Isomorphie lassen sich in einem Algorithmus
festhalten. Zur Enumeration von maximalen CST wird Zeile 9 in Algorithmus 3.5 (GetB)
durch den Aufruf „InsertKintoB(\( K, B(R^n_v, S^n_x) \))“ (Algorithmus 3.7) ersetzt. Algorithmus
3.8 zur Aufzählung aller maximalen CST von \( R \) und \( S \) mit Mindestgröße \( m \) orientiert
sich an Algorithmus 3.4 (EnumMinSizeMaximalCSTI). Zu beachten ist, dass in dem dieser
Arbeit beiliegenden Programm die Subtree Isomorphie Prüfung wegen des Umfangs dieser
Arbeit über die Berechnung der Größe eines MCSTI von \( R \) und \( S \) erfolgt. Falls diese Größe
dem Minimum von \( |R| \) und \( |S| \) entspricht, ist ein Teilbaum des einen Baumes isomorph

\[
\text{Abbildung 3.14: Der Baum } S \text{ ist in } R \text{ enthalten.}
\]
**Eingabe:** Baumkanonisierung $K$, Verweis auf Binärbaum $B$

1. if $K$ ist in $B$ enthalten then
2. return
3. end if
4. for all Kanonisierungen $K'$ aus $B$ mit $|K'| > |K|$ do
5. if $T(K)$ ist isomorph zu einem Teilbaum von $T(K')$ // STI Prüfung then
6. return
7. end if
8. end for
10. for all Kanonisierungen $K'$ aus $B$ mit $|K'| < |K|$ do
11. if $T(K')$ ist isomorph zu einem Teilbaum von $T(K)$ then
13. end if
14. end for
15. return

**Algorithmus 3.7:** InsertKintoB($K, B$)


### 3.6.6 Laufzeit und Speicherverbrauch


Der nötige Speicherverbrauch ist ebenfalls nicht durch ein Polynom beschränkt. Wenn in Algorithmus 3.5, Zeile 6, die in den Binärbäumen GetB($R^u_{v_1}, S^x_{x_1}$) bis GetB($R^u_{v_k}, S^x_{x_k}$) gespeicherten Kanonisierungen jeweils alle verschieden sind, gibt es mehr als polynomialiell viele verschiedene Kanonisierungen $K$, die in den Binärbaum $B(R^u_v, S^x_z)$ eingefügt werden.

### 3.6.7 Enumeration von Teilbäumen, auf denen ein Isomorphismus existiert

In diesem Abschnitt wird beschrieben, wie zu zwei Bäumen $R$ und $S$ alle Paare $(R', S')$ von Teilbäumen von $R$ und $S$ aufgezählt werden, auf denen mindestens ein Isomorphismus $\varphi : V(R') \to V(S')$ existiert. Außerdem soll für alle Isomorphismen $\varphi' : V(R') \to V(S')$
Eingabe: Zwei Bäume $R$ und $S$, Mindestgröße $m$
Ausgabe: Alle maximalen CST von $R$ und $S$ mit einer Mindestgröße von $m$.

1: $B \leftarrow \emptyset$
2: if $\text{SizeMCSTI}(R, S) < m$ then
3: return Es gibt keinen CST mit einer Mindestgröße von $m$.
4: end if
5: for all $r \in RST_R$ mit $I(r) < I(\tau)$ do
6: for all $s \in RST_S$ do
7: if $D(r, s) + D(\tau, \bar{s}) \geq m$ then
8: for all Kanonisierungen $K_1$ aus GetB($r, s$) mit $|K_1| \geq m - D(\tau, \bar{s})$ do
9: for all Kanonisierungen $K_2$ aus GetB($\tau, \bar{s}$) mit $|K_2| \geq m - |K_1|$ do
10: $K \leftarrow K_1; K_2$
11: Erhöhe die erste Zahl der Zahlenfolge $K$ um $|K_2|$.
12: Berechne $T' := T(K)$ und bestimme den Wurzelknoten $w$, wie in Abschnitt 3.6.4 beschrieben.
13: Berechne $K' := K(T')$.
14: InsertKintoB($K', B$) // Einfügen mit Subtree Isomorphie Prüfung
15: end for
16: end for
17: end if
18: end for
19: end for
20: Gebe alle Baumkanonisierungen in $B$ aus.

Algorithmus 3.8: EnumMinSizeMaximalCST($R, S, m$)

gelten, dass diese maximal bezüglich $R$ und $S$, also nicht erweiterbar, sind. Dabei gelte $(R'_1, S'_1) \neq (R'_2, S'_2)$ genau dann, wenn $V(R'_1) \neq V(R'_2) \lor V(S'_1) \neq V(S'_2)$. Die in Abbildung 3.15 dargestellten Teilbaumpaare $(R_1, S)$ und $(R_2, S)$ erfüllen diese Bedingung. Der im Folgenden beschriebene Algorithmus enumeriert diese Paare $(R', S')$ von Teilbäumen anhand der Knotenmengen $(V(R'), V(S'))$. Des Weiteren kann die Mindestgröße $m$ von $R'$ bzw. $S'$ als Parameter angegeben werden.

Der Algorithmus zur Enumeration ist mit Algorithmus 3.8 (EnumMinSizeMaximalCST) weitgehend identisch. Statt Kanonisierungen wird ein Tupel von Knotenmengen gespeichert. Jedem Knoten wird dabei eine natürliche Zahl zugeordnet, die alle voneinander verschieden sind. Die Knotenmengen der beiden Bäume werden dabei jeweils, bezüglich der Knotennummer sortiert, gespeichert. Wenn ein neues Tupel $(V_R, V_S)$ eingefügt werden soll, kann in Zeit $O(|V_R| \log |V_R|)$ überprüft werden, ob ein identisches Tupel im Binärbaum vorhanden ist. Falls es nicht vorhanden ist, muss dennoch die Maximalität aller Isomor-

Der Unterschied zu Algorithmus 3.8 (EnumMinSizeMaximalCST) besteht also im Wesentlichen darin, Knotenmengen sortiert zu speichern und zu vergleichen. Der Unterschied zu Algorithmus 3.5 (GetB) ist ähnlich. Auch dieser wird entsprechend so modifiziert, dass Tupel von Knotenmengen gespeichert werden. Die Speicherung erfolgt wie zuvor in einem sortierten Binärbaum. Die Sortierung erfolgt dabei lexikographisch in Bezug auf die aneinandergereihten Knotennummern. Falls der Parameter \(m\) der Größe eines MCST von \(R\) und \(S\) entspricht, sind keine Teilmengentest nötig.

Auf eine formale Angabe von Algorithmen kann wegen der großen Ähnlichkeit zu Algorithmen 3.5 und 3.8 verzichtet werden.

**Laufzeit und Speicherplatzverbrauch**

Bezüglich Laufzeit gilt die gleiche Aussage wie zur Enumeration von CST. Die Laufzeit zur Enumeration aller MaxWBM bzw. maximalen Matchings ist bei einem entsprechenden bipartiten Graphen nicht polynomial beschränkt. Dennoch ist die einzige Lösung für isomorphe Bäume \(R\) und \(S\) das Teilbaumpaar \((R, S)\).

Zur Speicherplatzabschätzung seien Bäume \(R\) und \(S\) wie in Abbildung 3.1 gegeben, wobei \(R\) insgesamt \(k+1\) Knoten und \(S\) insgesamt \(2k+1\) Knoten habe. Dann entspricht die Anzahl der Teilbaumpaare, die gleichzeitig im Binärbaum \(B\) gespeichert werden müssen, der Anzahl der \(k\)-elementigen Mengen einer \(2k\)-elementigen Menge. Diese Anzahl ist kombinatorisch durch \(\binom{2k}{k}\) bestimmt und somit nicht durch ein Polynom beschränkt.
Es handelt sich demnach weder um einen polynomial-total-time-Algorithmus, noch um einen polynomial-space-Algorithmus.

### 3.6.8 Beschleunigungstechniken

Im Folgenden werden einige Möglichkeiten dargestellt, die Berechnung der in Abschnitt 3.6 vorgestellten Algorithmen situativ zu beschleunigen. Bis auf letztgenannte wurden diese in dem dieser Arbeit beiliegenden Programm umgesetzt.

Wenn \( R \) in \( S \) enthalten ist, ist der einzige maximale CST ein zu \( R \) isomorpher Baum. Wegen des zuvor aufgerufenen Algorithmus von Edmonds lässt sich das in konstanter Zeit feststellen. Die Ausgabe in kanonischer Form gelingt dann in Zeit \( O(|V_R|^2) \). Falls \( S \) in \( R \) enthalten ist, gilt die Aussage analog.

Falls \( R \) und \( S \) isomorph sind, existiert nur eine Lösung für das in Abschnitt 3.6.7 vorgestellte Problem. Diese besteht aus den Bäumen \( R \) und \( S \), denn zwischen allen anderen Teilbäumen \( R' \subset R \) und \( S' \subset S \), auf denen ein Isomorphismus existiert, existiert mindestens auch ein nicht maximaler Isomorphismus. Die Knotenmenge der Bäume kann in Linearzeit \( O(|R| + |S|) \) ausgegeben werden.

Die obigen Prüfungen erfolgen nicht nur auf den Bäumen \( R \) und \( S \), sondern auch in Algorithmus 3.5 (GetB) während der dynamischen Programmierung, zwischen Zeile 1 und 2. Falls die genannten Fälle auf ein Paar \( (r, s) \) von gewurzelten Teilbäumen zutreffen, wird das einzige Element im Binärbaum \( B(r, s) \) entsprechend festgelegt. Im Fall von CST wird die Kanonisierung für den gewurzelten Teilbaum, der in dem anderen enthalten ist, berechnet und im Binärbaum \( B(r, s) \) gespeichert. Die Kanonisierungen für die Kinder, deren Kinder usw. werden dabei in den zugehörigen Binärbäumen gespeichert und stehen damit für andere Berechnungen zur Verfügung. Im Fall von Teilbäumen nach Abschnitt 3.6.7 werden analog die Knotenmengen sortiert gespeichert. Dies gilt auch für die Kinder, deren Kinder usw. Zeilen 2 bis 13 in Algorithmus 3.5 werden in den genannten Fällen nicht mehr ausgeführt.

Die Berechnungszeit zur Enumeration von MCST kann durch weitere Änderungen am modifizierten Algorithmus von Uno aus Abschnitt 3.2.2 verringert werden. Seien für Knoten \( x' \) und \( x'' \) die gewurzelten Bäume \( S_{x'}^+ \) und \( S_{x''}^+ \) isomorph. Wenn ein Knoten \( v' \) im aktuellen Matching \( M \) über eine Matchingkante mit \( x' \) verbunden ist, führt ein alternierender Kreis über die Kanten \( e = v'x' \) und \( v'x'' \) in den Teilproblemen \( G^+(e) \) und \( G^-(e) \) zu identischen Kanonisierungen, da in \( B(R_{v'}, S_{x'}^+) \) und \( B(R_{v''}, S_{x''}^+) \) genau die gleichen Kanonisierungen gespeichert sind, und alle Knoten \( v'' \), die mit \( x'' \) verbunden werden können, auch mit \( x' \) verbunden werden können. Solche Kreise werden verboten. Für isomorphe gewurzelte Teilbäume \( R_{v'}^+ \) und \( R_{v''}^+ \) gilt analoges. Desto mehr der gewurzelten Teilbäume isomorph sind, desto weniger Matchings werden enumeriert. Die Enumeration von maximalen CST lässt sich ähnlich beschleunigen, indem die Aufzählung der maximalen Matchings entspre-
chend modifiziert wird, so dass dort Zuordnungen der Kanten übersprungen werden, die zu gleichen CST führen.

Bezüglich der Laufzeit- und Speicherplatzanalysen aus Abschnitt 3.6.6 und 3.6.7 ändern diese Beschleunigungstechniken nichts. Es lassen sich jeweils Bäume $R$ und $S$ angeben, in denen diese Techniken nur selten oder gar nicht genutzt werden können und somit nur einen geringen oder gar keinen Einfluss auf Laufzeit und Speicherverbrauch haben.

### 3.7 Parallelisierung


Die Idee zur Parallelisierung besteht darin, die Schleifen in Zeile 1 und 2 aus Algorithmus 3.3 aufzulösen und statt dessen eine Queue zu generieren, in der alle Paaare $(r,s)$ von gewurzelten Teilbäumen enthalten sind. Dann werden vom Hauptprogramm $k$ nebenläufige Prozesse gestartet. Diese greifen unter Verwendung eines Mutex auf diese Queue zu und nehmen jeweils das vorderste Paar $(r,s)$ aus der Queue. Auf diesem Paar finden dann die Berechnungen aus Algorithmus 3.3, Zeile 3 bis 8, statt. Wenn die Queue leer ist, beendet sich der entsprechende nebenläufige Prozess. Sobald alle $k$ nebenläufigen Prozesse beendet wurden, sind alle CSTI bestimmt worden. Zu beachten ist, dass beim Aufruf des Algorithmus zur Bestimmung aller maximalen CSTI $\varphi_1$ von $(r,s)$ bzw. $\varphi_2$ von $(r,s)$ lokale Variablen verwendet werden, wie in Abschnitt 3.3.1 beschrieben wurde. Es ist deshalb nötig, für jeden Prozess eine eigene Kopie dieser Variablen bereitzustellen. Algorithmus 3.9 stellt den Hauptalgorithmus dar, der die nebenläufigen Prozesse, dargestellt in Algorithmus 3.10 aufruft. Bei der Ausgabe der Isomorphismen ist darauf zu achten, dass die verschiedenen Prozesse einen Mutex verwenden oder die Isomorphismen in getrennten Speicherbereichen oder Dateien abgelegt werden.

Es hat sich herausgestellt, dass nicht nur von Variablen, auf die schreibend zugegriffen wird, Kopien erstellt werden sollten, sondern auch von den Daten, auf die ausschließlich lesend zugegriffen wird. Dies betrifft insbesondere die Bäume der Eingabe. Dies führte zu einer weiteren Verringerung der Berechnungszeit. Die Verwendung von Mutexen konnte
3.7. PARALLELISIERUNG

**Eingabe:** Zwei Bäume $R$ und $S$, Anzahl an nebenläufigen Prozessen $k$

**Ausgabe:** Alle maximalen Common Subtree Isomorphismen von $R$ und $S$

1. Erstelle eine Queue $Q$ und füge alle Paare $(r, s)$ von gewurzelten Teilbäumen mit $I(r) < I(\bar{r})$ in die Queue ein.
2. **for** $i=1$ **to** $k$ **do**
3. Führe ConcurrentEnumMaximalCSTI($R, S, Q, i$) als nebenläufigen Prozess aus.
4. **end for**
5. Warte auf die Beendigung aller nebenläufigen Prozesse.

**Algorithmus 3.9:** EnumAllMaximalCSTI($R, S, k$)

**Eingabe:** Zwei Bäume $R$ und $S$, Prozessnummer $i$, Queue $Q$

**Ausgabe:** Ein Teil der maximalen Common Subtree Isomorphismen von $R$ und $S$

1. **if** $Q$ nicht leer **then**
2. Nimm das nächste Element $(r, s)$ mit Hilfe eines Mutex aus der Queue $Q$.
3. **while** GetMaximalCSTI($r, s, \varphi_1^{(i)}$) = 1 // vgl. Abschnitt 3.3.1 **do**
4. **while** GetMaximalCSTI($\bar{r}, \bar{s}, \varphi_2^{(i)}$) = 1 **do**
5. Verbinde die Isomorphismen $\varphi_1^{(i)}$ und $\varphi_2^{(i)}$ der gewurzelten Teilbäume zu einem maximalen CSTI $\varphi^{(i)}$ von $R$ und $S$.
6. Gebe $\varphi^{(i)}$ unter Beachtung der Nebenläufigkeit aus.
7. **end while**
8. **end while**
9. **end if**

**Algorithmus 3.10:** ConcurrentEnumMaximalCSTI($R, S, Q, i$)

KAPITEL 3. ENUMERATION VON COMMON SUBTREES UND CSTI
Kapitel 4

Experimentelle Resultate


4.1 Common Subtree Isomorphismen

In diesem Abschnitt werden die verschiedenen Varianten von Common Subtree Isomorphismen aus Abschnitt 3.3 und 3.4 untersucht. Die in Abschnitt 4.1.1 vorgestellten Ergebnisse werden in Abschnitt 4.1.2 bewertet.
Kapitel 4. Experimentelle Resultate

<table>
<thead>
<tr>
<th>Kanten</th>
<th>Zeit (s)</th>
<th>CSTI (M)</th>
<th>CSTI pro s (M/s)</th>
<th>verworfen (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>0,19</td>
<td>2,51</td>
<td>13,3</td>
<td>4,1</td>
</tr>
<tr>
<td>30</td>
<td>1,36</td>
<td>17,00</td>
<td>12,6</td>
<td>2,2</td>
</tr>
<tr>
<td>35</td>
<td>8,06</td>
<td>90,00</td>
<td>11,2</td>
<td>3,5</td>
</tr>
<tr>
<td>40</td>
<td>71,50</td>
<td>831,00</td>
<td>11,6</td>
<td>2,8</td>
</tr>
<tr>
<td>45</td>
<td>544,00</td>
<td>6344,00</td>
<td>11,7</td>
<td>1,1</td>
</tr>
</tbody>
</table>

Tabelle 4.1: Maximale CSTI - Jeweils 10 Durchläufe

4.1.1 Ergebnisse der Programms

Maximale Common Subtree Isomorphismen


Maximum Common Subtree Isomorphismen


Maximale Common Subtree Isomorphismen mit Mindestwert

In Abbildung 4.1 sind für zwei zufällige Bäume mit 50 Kanten die Anzahl der maximalen CSTI der Mindestgröße 2, das entspricht allen maximalen CSTI, bis 34, das entspricht den Maximum CSTI, die benötigte Laufzeit und die berechneten Isomorphismen pro Sekunde angegeben.

Vergleich Maximale CSTI - Baumalgorithmus gegen Reduktion auf VPG und Auffinden von $c$-zusammenhängenden Cliquen

In Tabelle 4.3 wird die Laufzeit von Algorithmus 3.3 (EnumMaximalCSTI) mit der Laufzeit der Reduktion auf den VPG und das Finden von $c$-zusammenhängenden Cliquen vergli-


### 4.1. COMMON SUBTREE ISOMORPHISMEN

<table>
<thead>
<tr>
<th>Kanten</th>
<th>Zeit</th>
<th>MCSTI</th>
<th>MCSTI pro s</th>
<th>Edmonds</th>
<th>verworfen</th>
<th>[MCSTI]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vermeidung gleicher Isomorphismen mit Ordnungsfunktion</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>0,23 s</td>
<td>1 M</td>
<td>3,55 M/s</td>
<td>121 ms</td>
<td>2,28 %</td>
<td>27,9</td>
</tr>
<tr>
<td>50</td>
<td>13,83 s</td>
<td>118 M</td>
<td>8,52 M/s</td>
<td>180 ms</td>
<td>0,01 %</td>
<td>34,1</td>
</tr>
<tr>
<td>60</td>
<td>13,55 s</td>
<td>68 M</td>
<td>5,00 M/s</td>
<td>271 ms</td>
<td>0,01 %</td>
<td>40,1</td>
</tr>
<tr>
<td>70</td>
<td>494,12 s</td>
<td>4412 M</td>
<td>8,93 M/s</td>
<td>372 ms</td>
<td>&lt;0,001 %</td>
<td>45,7</td>
</tr>
</tbody>
</table>

| Vermeidung gleicher Isomorphismen durch Kantenlösung |
| 40     | 0,28 s | 1 M   | 2,97 M/s    | 182 ms  | -         | 27,9    |
| 50     | 14,10 s| 118 M | 8,37 M/s    | 289 ms  | -         | 34,1    |
| 60     | 13,56 s| 68 M  | 5,00 M/s    | 422 ms  | -         | 40,1    |
| 70     | 468,09 s| 4412 M| 9,43 M/s    | 594 ms  | -         | 45,7    |

**Tabelle 4.2:** Maximum CSTI - Jeweils 10 Durchläufe

<table>
<thead>
<tr>
<th>Kanten</th>
<th>Isomorphismen</th>
<th>Zeit CSTI</th>
<th>Zeit CCISGI</th>
<th>Zeit CCISGI/CSTI</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>45 T</td>
<td>8 ms</td>
<td>0,28 s</td>
<td>35</td>
</tr>
<tr>
<td>18</td>
<td>118 T</td>
<td>16 ms</td>
<td>1,19 s</td>
<td>74</td>
</tr>
<tr>
<td>20</td>
<td>290 T</td>
<td>29 ms</td>
<td>4,35 s</td>
<td>150</td>
</tr>
<tr>
<td>22</td>
<td>556 T</td>
<td>49 ms</td>
<td>12,83 s</td>
<td>262</td>
</tr>
<tr>
<td>25</td>
<td>2506 T</td>
<td>189 ms</td>
<td>86,09 s</td>
<td>456</td>
</tr>
</tbody>
</table>

**Tabelle 4.3:** Maximale CSTI bzw. CCISGI - Jeweils 10 Durchläufe

... chen, um daraus die maximalen CCISGI zu erhalten. Insbesondere die letzte Spalte, die den Zeitfaktor zwischen dem Reduktions-Algorithmus und dem Baumalgorithmus angibt, ist zu beachten.

**MCSTI mit verschiedener Wahl der Kantenreihenfolge in R**

In Tabelle 4.4 sind die benötigte Zeit und die benötigten Rekursionen zum Auffinden aller MCSTI in Bäumen mit 50 bzw. 55 Kanten angegeben. Dabei erfolgte die Auswahl der Kanten in R in der Reihenfolge, wie die Kanten im Baum der Eingabe eingelesen werden (Eingabe), in einer Reihenfolge, die eine Kante möglichst in der Mitte wählt (Mitte), und einer zufälligen Reihenfolge (Zufall). Identische Isomorphismen wurden durch Kantenlöschung vermieden. Die Auswahl der Reihenfolge hat einen nicht zu vernachlässigenden Einfluss auf die Zahl der Rekursionen und auf die Laufzeit.

**Vergleich MaxWBM und MaxWBM’**

In Abbildung 4.2 wird die Enumeration von MCSTI und MCWSTI mit Hilfe von MaxWBM’ (Auflistung aller maximalen Matchings, nur Maximum Matchings behalten) und
Abbildung 4.1: Anzahl der maximalen CSTI mit Mindestgröße (x-Achse) - Bäume mit 50 Kanten

<table>
<thead>
<tr>
<th>Kanten</th>
<th>Kantenwahl</th>
<th>Zeit</th>
<th>Rekursionen</th>
<th>Rekursionen pro s</th>
<th>Edmonds</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>Eingabe</td>
<td>12,1 s</td>
<td>547 M</td>
<td>45,2 M/s</td>
<td>285 ms</td>
</tr>
<tr>
<td>50</td>
<td>Mitte</td>
<td>14,0 s</td>
<td>675 M</td>
<td>48,2 M/s</td>
<td>273 ms</td>
</tr>
<tr>
<td>50</td>
<td>Zufall</td>
<td>15,0 s</td>
<td>705 M</td>
<td>47,0 M/s</td>
<td>404 ms</td>
</tr>
<tr>
<td>55</td>
<td>Eingabe</td>
<td>17,6 s</td>
<td>982 M</td>
<td>55,8 M/s</td>
<td>330 ms</td>
</tr>
<tr>
<td>55</td>
<td>Mitte</td>
<td>23,3 s</td>
<td>1243 M</td>
<td>53,3 M/s</td>
<td>340 ms</td>
</tr>
<tr>
<td>55</td>
<td>Zufall</td>
<td>24,1 s</td>
<td>1280 M</td>
<td>51,3 M/s</td>
<td>507 ms</td>
</tr>
</tbody>
</table>

Tabelle 4.4: MCSTI mit verschiedener Kantenauswahl - Jeweils 10 Durchläufe

4.1. COMMON SUBTREE ISOMORPHISMEN

Die Bäume in der FTree-Datenbank besitzen 3 bis etwa 15 Knoten. Das Einlesen und ein paarweiser Vergleich der ersten 1000 Feature Trees benötigte 5 Minuten und 37 Sekunden. Dabei wurden die 50 Paare gespeichert, die die größte Ähnlichkeit haben. Die Ähnlichkeit von 2 Feature Trees \( F = (V_F, E_F), G = (V_G, E_G) \) wurde in dieser Arbeit als Quotient \( Q(F, G) := \frac{\text{Size}_{\text{MCSTI}}(F, G)}{\max(|V_F|, |E_F|)} \) definiert. In Abschnitt 3.5 wurde die Vergleichsfunktion \( c \) durch \( c(0, 0) = 1 \) und \( c(a, b) = \frac{2\min(a, b)}{a+b} \) für \( a + b > 0 \) definiert. Es gilt folglich \( c(a, b) \in [0, 1] \).

Das ist genau der Wert, der 2 Knoten, die im MCWSTI einander zugeordnet werden, zum Gewicht beitragen. Es gilt folglich \( Q(F, G) \in [0, 1] \) und \( Q(F, F) = 1 \). Für die Feature Trees „54695776“ und „54678160“ beträgt dieser Quotient 1. Die Feature Trees sind in Abbildung 4.3 dargestellt. Die Zahl gibt die Anzahl der Nicht-Wasserstoff-Atome an, die durch den Knoten repräsentiert werden. Der Quotient überrascht nicht, wie ein Vergleich der chemischen Struktur zeigt.

\[ O=C1N(\ldots)C(\ldots)C(\ldots)=C1(\ldots)C(\ldots)\ldots N=1C(\ldots)=C(\ldots)C=CC1\ldots (54695776) \]
\[ O=C1N(\ldots)C(\ldots)=C1(\ldots)C(\ldots)\ldots N=1C(\ldots)=C(\ldots)C=CC1\ldots (54678160) \]

Nur anhand der Knotenverbindungen und der Anzahl der Atome lassen sich diese Moleküle nicht unterscheiden. Die Datensätze in der FTree-Datenbank enthalten noch weitere Informationen, die bei diesen Strukturen weitgehend übereinstimmen. Allerdings berücksichtigt
Abbildung 4.3: Feature Trees „54695776“ und „54678160“

| Kanten | \(|R| < |S|\) | \(|S| < |R|\) |
|--------|--------------|--------------|
|        | Enumeration  | Edmonds      | Enumeration  | Edmonds      |
| 10, 100| 71 ms        | 93 ms        | 77 ms        | 201 ms       |
| 10, 150| 212 ms       | 147 ms       | 224 ms       | 359 ms       |
| 10, 250| 672 ms       | 248 ms       | 733 ms       | 729 ms       |
| 10, 500| 2725 ms      | 521 ms       | 2947 ms      | 1895 ms      |

Tabelle 4.5: Maximum CSTI mit Bäumen verschiedener Größen - Jeweils 10 Durchläufe


MCSTI - Bäume \(R\) und \(S\) mit verschiedener Kantenanzahl

In diesem Abschnitt wird die Berechnungszeit für Bäume \(R\) und \(S\) mit verschiedenen Größen untersucht. Dabei wurde das Verfahren der Kantenlöschung eingesetzt. Die Startwerte für den Zufallszahlengenerator wurden im Fall \(|S| < |R|\) vertauscht, so dass jeweils auf genau den gleichen Bäumen enumeriert wurde. Nicht unerwartet ist der Algorithmus schneller, wenn \(R\) kleiner als \(S\) ist. In diesem wird der Algorithmus von Edmonds nicht so häufig aufgerufen wie im umgekehrten Fall. Ergebnisse finden sich in Tabelle 4.5...
4.1. COMMON SUBTREE ISOMORPHISMEN

<table>
<thead>
<tr>
<th>Verfahren</th>
<th>Kantenanzahl</th>
<th>Zeit</th>
<th>verworfen</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ordnungsfunktion</td>
<td>11,10</td>
<td>11,49 s</td>
<td>90,000 %</td>
</tr>
<tr>
<td>Kantenentfernung</td>
<td>11,10</td>
<td>1,41 s</td>
<td>-</td>
</tr>
<tr>
<td>Ordnungsfunktion</td>
<td>10,11</td>
<td>1,36 s</td>
<td>&lt;0,001 %</td>
</tr>
<tr>
<td>Kantenentfernung</td>
<td>10,11</td>
<td>1,36 s</td>
<td>-</td>
</tr>
</tbody>
</table>

Tabelle 4.6: MCSTI - Ordnungsfunktion gegen Kantenlöschung mit Sterngraphen

<table>
<thead>
<tr>
<th>Modus und Kantenanzahl</th>
<th>Anzahl Prozesse</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Debug, 49</td>
<td>157,3 s (100 %)</td>
</tr>
<tr>
<td>Release, 49</td>
<td>20,8 s (100 %)</td>
</tr>
<tr>
<td>Debug, 51</td>
<td>379,0 s (100 %)</td>
</tr>
<tr>
<td>Release, 51</td>
<td>50,7 s (100 %)</td>
</tr>
<tr>
<td>Release, 53</td>
<td>56,0 s (100 %)</td>
</tr>
<tr>
<td>Release, 55</td>
<td>114,2 s (100 %)</td>
</tr>
</tbody>
</table>

Tabelle 4.7: Maximale CSTI unter Nebenläufigkeit - Jeweils ein Durchlauf

Ordnungsfunktion gegen Kantenlöschung

In Abschnitt 3.4.1 wurde ein zusätzlicher Zeitfaktor von $O(m)$ für die Ordnungsfunktion angegeben, wenn $m$ die Größe eines CSTI ist. Dieser soll praktisch nachgewiesen werden. $R$ und $S$ wurden wie in Abbildung 3.1 mit mit $|R| = |S| + 1$ gewählt. Das Speichern von Matchings wurde deaktiviert. Die Ergebnisse sind in Tabelle 4.6 dargestellt.

Parallelisierung der Enumeration von maximalen CSTI


4.1.2 Analyse und Bewertung

Aus den Tabellen und Diagrammen aus Abschnitt 4.1 lassen sich mehrere interessante Ergebnisse ableiten. Ein sehr wichtiges Ergebnis ist die hohe Effizienz der Ordnungsfunktion $I$ auf Zufallsbäumen. Bei den maximalen CSTI in Tabelle 4.1 werden im Schnitt
2,7 % der Matchings von der Ordnungsfunktion verworfen, bei den Maximum CSTI im Schnitt sogar unter 1 %. Für speziell konstruierte Beispiele erhöht sich mit der Ordnungsfunktion die Laufzeit um einen Faktor \( O(m) \) gegenüber der Kantenerlösung, wenn \( m \) die Größe eines MCSTI ist. In Tabelle 4.6 ist dazu ein Beispiel angegeben. Dieser Verhalten wurde in Abschnitt 3.4.1 vorhergesagt. Dass Algorithmus 3.3 (EnumMaximalCSTI) kein polynomial-delay-Algorithmus ist, hat wegen der geringen Zahl der verworfenen Matchings auf Zufallsbäumen fast keinerlei Auswirkung.


Für die unterschiedliche Anzahl an Rekursionen gibt es eine Begründung. Dazu wird Algorithmus 3.1 (EnumMaximumCSTI), Zeile 5 und 6, genauer untersucht. Sei dazu \( k \) die Anzahl der MCSTI auf dem Paar gewurzelter Teilbäume \((r, s)\) und \( l \) die Anzahl der MCSTI auf dem Paar \((\bar{r}, \bar{s})\). Jeder MCSTI \( \varphi_2 \) von \((\bar{r}, \bar{s})\) wird \( k \) mal berechnet. Die Gesamtzahl \( Z_1 \) der Rekursionen für diese Wahl von \( r \) und \( s \) ergibt sich somit aus der Summe der Rekursionen auf \((r, s)\) plus \( k \) mal den Rekursionen auf \((\bar{r}, \bar{s})\). Für eine andere Ordnungsfunktion \( I_2 \) gilt möglicherweise \( I_2(r) > I_2(\bar{r}) \). Dann liegt die Gesamtzahl \( Z_2 \) der Rekursionen bei \( l \) mal den Rekursionen auf \((r, s)\) plus den Rekursionen auf \((\bar{r}, \bar{s})\). Diese beiden Anzahlen sind im Allgemeinen nicht identisch. \( I_2 \) kann auch eine ganz andere Reihenfolge festlegen, so dass dann die Anzahl ebenso abweichen kann. Als Ergebnis lässt sich festhalten, dass die Reihenfolge der Kantenauswahl insbesondere von einer möglichst günstigen Wahl für \( Z_1 \) und \( Z_2 \) abhängig sein sollte.


In Bezug auf Nebenläufigkeit entsprechen die erzielten Ergebnisse nur teilweise den Erwartungen. Im Debugmodus verhält sich die Laufzeit antiproportional in Bezug zur Anzahl der nebeneinläufigen Prozesse. Das Ergebnis kann somit als sehr gut bewerten werden. Im Releasemodus sind die relativen Ergebnisse zumindest für zwei nebeneinläufigen Prozesse genau so gut wie im Debugmodus. Mit drei und vier nebeneinläufigen Prozessen im Releasemodus sind die Ergebnisse zuweilen schlechter als mit zwei Prozessen. Die Ursache dieses Verhaltens blieb während der Bearbeitung dieser Arbeit unbekannt. Das Ergebnis zeigt, dass theoretische Überlegungen in Bezug auf Parallelisierung nicht ohne Weiteres in die Praxis übertragbar sind und die konkrete Prozessorarchitektur vermutlich einen nicht zu vernachlässigenden Einfluss hat.

4.2 Common Subtrees

In diesem Abschnitt werden die verschiedenen Varianten von Common Subtrees aus Abschnitt 3.6 untersucht. Die in Abschnitt 4.2.1 vorgestellten Ergebnisse werden in Abschnitt 4.2.2 bewertet.

4.2.1 Ergebnisse der Programms

Maximum Common Subtrees


Maximale Common Subtrees

4.2. COMMON SUBTREES

| Kanten | Zeit  | MCST | MCST pro s | Edmonds | |MCST|
|--------|-------|------|------------|---------|------|
| 20     | 0,4 s | 157  | 402,0      | 0,3 s   | 15,6 |
| 50*    | 0,2 s | 19   | 79,8       | 0,2 s   | 34,1 |
| 50     | 2,7 s | 221  | 82,4       | 1,8 s   | 33,6 |
| 100    | 16,6 s| 337  | 20,3       | 7,7 s   | 61,17|
| 200    | 141,1 s| 486  | 3,4        | 32,3 s  | 112,29|
| 500*   | 725,0 s| 177  | 0,2        | 20,7 s  | 256,80|
| 750*   | 2235,7 s| 1728 | 0,8        | 47,2 s  | 370,9 |

Tabelle 4.8: Maximum CST - Jeweils 100 Durchläufe (*: 10 Durchläufe)

<table>
<thead>
<tr>
<th>Kanten</th>
<th>Zeit</th>
<th>CST</th>
<th>CST pro s</th>
<th>STI Prüfungen</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>4,3 s</td>
<td>26</td>
<td>6,1</td>
<td>20 T</td>
</tr>
<tr>
<td>17</td>
<td>6,7 s</td>
<td>36</td>
<td>5,4</td>
<td>27 T</td>
</tr>
<tr>
<td>20</td>
<td>19,1 s</td>
<td>59</td>
<td>3,1</td>
<td>60 T</td>
</tr>
<tr>
<td>20*</td>
<td>1225,0 s</td>
<td>478</td>
<td>0,4</td>
<td>2213 T</td>
</tr>
<tr>
<td>22</td>
<td>31,1 s</td>
<td>73</td>
<td>2,4</td>
<td>76 T</td>
</tr>
<tr>
<td>24</td>
<td>61,0 s</td>
<td>74</td>
<td>1,2</td>
<td>132 T</td>
</tr>
<tr>
<td>25</td>
<td>76,2 s</td>
<td>80</td>
<td>1,0</td>
<td>144 T</td>
</tr>
</tbody>
</table>

Tabelle 4.9: Maximale CST - Jeweils 10 Durchläufe (*: 100 Durchläufe)

Maximale Common Subtrees mit Mindestwert

Abbildung 4.4 zeigt für zwei zufällige Bäume mit 35 Kanten die Anzahl der maximalen CST der Mindestgröße 0, das entspricht allen maximalen CST, bis 25, das entspricht den Maximum CST, die benötigte Laufzeit, die Anzahl der Prüfungen auf Subtree Isomorphie und die berechneten Common Subtrees pro Sekunde. Für die berechneten CST pro Sekunde ist die Skala logarithmisch dargestellt.

MCST - Bäume $R$ und $S$ mit verschiedener Kantenanzahl

KAPITEL 4. EXPERIMENTELLE RESULTATE

Zeit (s), maximale CST, STI-Prüfungen (in T)  maximale CST / s

0 5 10 15 20 25
0
5
10
15
20
25
30
0,1
1
10
100
Zeitm(s), mmaximalemCST, mSTI-Prüfungenm(inmT) maximalemCSTm/ms

Abbildung 4.4: Anzahl der maximalen CST mit Mindestgröße (x-Achse) - Bäume mit 35 Kanten

| Kanten | Zeit | MCST | MCS pro s | Edmonds | |MCST| |
|--------|-----|------|-----------|---------|--------|------|
| 25, 5  | 0,1 s| 100  | 1333      | 0,1 s   | 6,00   |
| 25, 10 | 0,2 s| 113  | 565       | 0,2 s   | 10,71  |
| 25, 25 | 0,6 s| 183  | 288       | 0,4 s   | 18,59  |
| 25, 100 | 2,3 s| 153  | 66        | 1,8 s   | 25,35  |
| 25, 200 | 4,0 s| 107  | 27        | 3,7 s   | 25,97  |
| 25, 400 | 8,1 s| 100  | 3         | 7,6 s   | 26,00  |
| 25, 100* | 12,8 s| 100  | 7,8       | 12,6 s  | 26,00  |

Tabelle 4.10: Maximum CST - Jeweils 100 Durchläufe (*: Sterngraphen)

4.2.2 Analyse und Bewertung

Zunächst lässt sich festhalten, dass die Anzahl der MCST und auch der maximalen CST wesentlich geringer als die Anzahl der MCSTI bzw. maximalen CSTI ist. Die durchschnittliche Zeit zur Bestimmung eines MCST ist allerdings höher als zur Bestimmung eines MCSTI. Die Gesamtzeit zur Enumeration auf 10 zufälligen Baumpaaren mit 50 Kanten betrug 0,2 s für alle MCST. Auf den selben Bäumen dauerte es 14 Sekunden alle MCSTI zu enumerieren. Diese Ergebnisse lassen sich den Tabellen 4.2 und 4.8 entnehmen. Das Ziel, einen Algorithmus anzugeben, der zur Enumeration aller MCST weniger Zeit benötigt als ein Algorithmus, der die MCST aus den MCSTI konstruiert, wurde klar erreicht. Solch ein
Algorithmus benötigte die Zeit zur Enumeration der MCSTI und zusätzlich noch die Zeit, daraus die MCST, inklusive Isomorphieprüfung, zu berechnen.

Wegen der geringen Anzahl von MCST für kleine Bäume erscheint die Wahl für die Baumkanonisierung nach Valiente [28], deren Bestimmung nach Satz 3.14 für n Knoten $O(n^2)$ Zeit benötigt, nicht mehr so gut. Dass die Anzahl so gering ausfällt, war zu Beginn der Arbeit nicht abzusehen. Bei kleinen Bäumen ist vermutlich ein Linearzeitalgorithmus zur Isomorphieprüfung von Bäumen die bessere Wahl, auch dann, wenn sich aus dem Algorithmus keine Zwischenergebnisse, wie eine eindeutige Kanonisierung, wiederverwerten lassen.


Der Speicherverbrauch zur Berechnung der MCST ist bei großen Zufallsbäumen etwa zur Hälfte durch die Speicherung der MCST bedingt. Die andere Hälfte wird im Wesentlichen für die Enumeration von Matchings und den Algorithmus von Edmonds benötigt. Bei 10 Baumpaaren mit jeweils 750 Kanten betrug der initiale Speicherverbrauch etwa 1 GB und wuchs auf 1,5 bis 2 GB an, bevor dann die Berechnungen für das nächste Baumpaar begannen. Obwohl Algorithmus 3.6 zur Enumeration von MCST kein polynomial-total-space-Algorithmus ist, lässt er sich auch für Zufallsbäume mit vielen hundert und auch über tausend Knoten nutzen.

In Tabelle 4.10 ist die Wirkung der Beschleunigungstechniken aus Abschnitt 3.6.8 gut zu erkennen. Wenn ein Baum in einem anderen enthalten ist, ist die Laufzeit fast ausschließlich vom Algorithmus von Edmonds bestimmt. Dies trifft insbesondere auch für die Sterngraphen aus Abbildung 3.1 zu. In der Tabelle ist weiterhin zu erkennen, dass die Laufzeit des Algorithmus von Edmonds nicht nur von der Kantenzahl, sondern auch der Struktur der Bäume abhängt. Die Sterngraphen führen zu bipartiten Graphen mit vielen Knoten, so dass die Berechnung der zulässigen Teilgraphen entsprechend länger dauert. In der Tabelle ist der Zeitunterschied mit 1,8 s bzw. 12,6 s bei Bäumen mit 25 und 100
KAPITEL 4. EXPERIMENTELLE RESULTATE

<table>
<thead>
<tr>
<th>Kanten</th>
<th>Zeit</th>
<th>Teilbaumpaare</th>
<th>Teilbaumpaare pro s</th>
<th>Edmonds</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>0,12 s</td>
<td>816</td>
<td>6857</td>
<td>0,02 s</td>
</tr>
<tr>
<td>25</td>
<td>0,46 s</td>
<td>2910</td>
<td>6312</td>
<td>0,04 s</td>
</tr>
<tr>
<td>30</td>
<td>0,86 s</td>
<td>3774</td>
<td>4383</td>
<td>0,05 s</td>
</tr>
<tr>
<td>35</td>
<td>3,43 s</td>
<td>12794</td>
<td>3726</td>
<td>0,08 s</td>
</tr>
<tr>
<td>40</td>
<td>18,10 s</td>
<td>24568</td>
<td>2688</td>
<td>0,11 s</td>
</tr>
<tr>
<td>45</td>
<td>105,00 s</td>
<td>241826</td>
<td>2293</td>
<td>0,13 s</td>
</tr>
<tr>
<td>50</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Tabelle 4.11: Teilbaumpaare mit ausschließlich grösstmöglichen Isomorphismen - 10 Durchläufe

Kanten mehr als Faktor 10. Solch ein wort case wurde in Zusammenhang mit der Laufzeitabschätzung in Satz 2.16 beschrieben.

4.3 Teilbaumpaare, auf denen ein Isomorphismus existiert

In diesem Abschnitt werden die verschiedenen Varianten zur Enumeration von Teilbaumpaaren nach Abschnitt 3.6.7 untersucht. Wie zuvor werden die in Abschnitt 4.3.1 vorgestellten Ergebnisse in Abschnitt 4.3.2 bewertet.

4.3.1 Ergebnisse der Programms

Ausschließlich grösstmögliche Isomorphismen

In Tabelle 4.11 ist die Anzahl der Teilbaumpaare angegeben, auf denen ausschließlich grösstmögliche maximale Isomorphismen bezüglich der gegebenen Bäume existieren. Nach Abschnitt 3.6.7 ist der Algorithmus zur Berechnung kein polynomial-space-Algorithmus. Die experimentellen Ergebnisse in der Tabelle sind dazu konform.

Ausschließlich maximale Isomorphismen


Ausschließlich maximale Isomorphismen mit Mindestgröße

Abbildung 4.5 zeigt für zwei zufällige Bäume mit 23 Kanten, und 21 als Größe eines MCSTI, die Anzahl der Teilbaumpaare mit Mindestgröße 1 bis 21, auf denen ausschließlich maximale Isomorphismen bezüglich der gegebenen Bäume existieren. Alle Teilbäume aus den enumerierten Teilbaumpaaren haben Mindestgröße 8.
4.3. TEILBAUMPAARE, AUF DENEN EIN ISOMORPHISMUS EXISTIERT

<table>
<thead>
<tr>
<th>Kanten</th>
<th>Zeit</th>
<th>Teilbaumpaare</th>
<th>Teilbaumpaare pro s</th>
<th>Teilmengen-Tests</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>0,7 s</td>
<td>5853</td>
<td>7898</td>
<td>8 M</td>
</tr>
<tr>
<td>18</td>
<td>3,6 s</td>
<td>18124</td>
<td>5009</td>
<td>64 M</td>
</tr>
<tr>
<td>20</td>
<td>13,0 s</td>
<td>35032</td>
<td>2694</td>
<td>278 M</td>
</tr>
<tr>
<td>22</td>
<td>45,9 s</td>
<td>65143</td>
<td>1419</td>
<td>1137 M</td>
</tr>
<tr>
<td>25</td>
<td>969,2 s</td>
<td>261691</td>
<td>269</td>
<td>23309 M</td>
</tr>
</tbody>
</table>

Tabelle 4.12: Teilbaumpaare mit ausschließlich maximalen Isomorphismen - Jeweils 10 Durchläufe

Abbildung 4.5: Teilbäume mit ausschließlich maximalen Isomorphismen - 23 Kanten pro Baum

Größtmögliche Isomorphismen auf Bäumen verschiedener Größe

In Tabelle 4.13 erfolgt die Berechnung der Teilbaumpaare auf Bäumen unterschiedlicher Größe. Zu beachten ist, dass die durchschnittliche Geschwindigkeit nahezu konstant ist.

4.3.2 Analyse und Bewertung

Es hat sich herausgestellt, dass die Anzahl der Lösungen sehr groß werden kann und bei entsprechender Eingabegröße mehr Speicherplatz benötigt wird, als auf dem Testrechner zur Verfügung stand. Da es mit der gegenwärtigen Implementierung nötig ist, alle bereits berechneten Lösungen zu speichern, ist es nicht praktikabel, dieses Verfahren auf größeren Bäumen anzuwenden. Das Verhältnis der Teilbaumpaare mit ausschließlich größtmöglichen Isomorphismen zu Teilbaumpaaren mit ausschließlich maximalen Isomorphismen beträgt für die Zufallsbäume mit 20 Knoten etwa 1 zu 45 und für Zufallsbäume mit 25 Knoten
etwa 1 zu 90, und liegt damit zwischen dem Verhältnis von MCST zu maximalen CST und MCSTI zu maximalen CSTI. Während es deutlich mehr maximale bzw. Maximum CSTI als Teilbaumpaare mit den genannten Eigenschaften gibt, gibt es deutlich weniger maximale CST bzw. Maximum CST.


\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|c|c|}
\hline
Kanten & Zeit & Teilbaumpaare & Teilbaumpaare pro s & Edmonds \\
\hline
10, 25 & 0,1 s & 1 T & 11398 & 0,01 s \\
10, 50 & 2,2 s & 20 T & 8957 & 0,03 s \\
10, 75 & 8,3 s & 70 T & 8479 & 0,04 s \\
10, 100 & 21,2 s & 197 T & 9300 & 0,06 s \\
10, 125 & 43,3 s & 403 T & 9318 & 0,08 s \\
10, 150 & 62,4 s & 580 T & 9285 & 0,09 s \\
10, 175 & 96,3 s & 883 T & 9166 & 0,11 s \\
10, 200 & 128,4 s & 1222 T & 9517 & 0,12 s \\
\hline
\end{tabular}
\caption{Tabelle 4.13: Größtmögliche Isomorphismen, verschiedene Baumgrößen - Jeweils 10 Durchläufe}
\end{table}
Kapitel 5

Zusammenfassung und Ausblick


anstelle eines auf Subtree Isomorphie spezialisierten Algorithmus, beispielsweise aus [26]. Das wirkt sich negativ auf die Laufzeit zur Enumeration aller maximalen CST aus. Für eine künftige Programmversion stellt eine effiziente Subtree Isomorphie Prüfung eine wesentliche Verbesserung für dieses Problem dar.


Für alle Verfahren ist es möglich, eine Mindestgröße der Lösungen anzugeben. Die praktischen Resultate zeigen, dass auch mit dieser Bedingung eine effiziente Enumeration möglich ist, auch wenn sich worst case-Beispiele konstruieren lassen, die beweisen, dass es sich jeweils um keine polynomial-total-time-Enumeration handelt.

Der Speicherverbrauch zur Berechnung von maximalen und Maximum Common Subtrees lässt sich verringern. Falls zwei gewurzelte Teilbäume $r_1$ und $r_2$ isomorph sind, sind für alle gewurzelten Teilbäume $s$ von $S$ die in den Binärbäumen $B(r_1, s)$ und $B(r_2, s)$ gespeicherten Kanonisierungen identisch. Es reicht daher, diese nur einmal zu speichern und die Zeiger in den entsprechenden Datenstrukturen auf die selben Binärbäume zeigen zu lassen. Für isomorphe gewurzelte Teilbäume von $S$ gilt die Aussage analog.

Eine weitere interessante Option ist die Bestimmung der Anzahl der Maximum Common Subtree Isomorphismen, ohne diese zu enumerieren. Das gelingt, indem im Algorithmus von Edmonds diese Anzahl für alle Paare von gewurzelten Teilbäumen berechnet wird und daraus dann die gesamte Anzahl. Hinweise zur Berechnung dieser Anzahl finden sich in Abschnitt 3.3.1. So lässt sich vorab einschätzen, wie groß der Aufwand ist, alle Lösungen zu enumerieren.

Eine Ausweitung der Enumeration auf andere Graphklassen, wie Serien-Parallele Graphen, scheint nicht ausgeschlossen. Graphklassen, auf denen die Berechnung eines Maximum Common Subgraph Isomorphismus über eine Transformation in eine Baumstruktur
erfolgen kann, lassen möglicherweise eine Enumeration aller Maximum Common Subgraphs mit dem in dieser Arbeit entwickelten Algorithmus zu. Dies zu überprüfen ist eine spannende Aufgabe für zukünftige Untersuchungen.

Anhang A

Weitere Informationen


A.1 Implementierung


- **tree.cpp, tree.h**
  In der Klasse Tree wird ein Baum verwaltet. Es kann ein Zufallsbaum oder ein Sterngraph erstellt werden, ein Datensatz aus der FTree-Datenbank eingelesen und daraus der entsprechende Baum generiert werden, sowie die kanonische Darstellung des Baumes berechnet werden. Diese Klasse enthält außerdem für jede Kante des Baumes zwei Objekte vom Typ RootedSubTree.

- **RootedSubTree.cpp, RootedSubTree.h**
  Die Klasse RootedSubTree verwaltet die gewurzelten Teilbäume, vgl. Definition 2.15. Diese Klasse bietet vorwiegend Zugriffsmöglichkeiten, beispielsweise auf die über Zeiger verlinkten Kindteilbäume oder die kanonische Darstellung des gewurzelten Teilbaums.

- **RstPair.cpp, mcs.h**
  Mit Hilfe der Klasse RstPair werden Paare von gewurzelten Teilbäumen verwaltet. In dieser Klasse wurden alle Berechnungen implementiert, die auf einem Paar von gewurzelten Teilbäumen basieren. Das schließt die Erstellung des zulässigen Teilgraphen, die Bestimmung der D-Werte, die Rekursionen für die Bestimmung der CSTI.
und CST auf dem Paar von gewurzelten Teilbäumen und die Enumeration von Matchings ein.

- **mcs.cpp, mcs.h**

- **Solution.cpp, mcs.h**

- **clique.cpp, mcs.h**
  In der Klasse `Clique` ist das in Abschnitt 3.2.1 vorgestellte Verfahren zur Enumeration von maximalen CCISGI implementiert.

- **Hauptprogramm.cpp**
  Hier findet sich die Funktion `main()`. Dort wird unter anderem die Konfigurationsdatei eingelesen und die verschiedenen Algorithmen aufgerufen. Von hier findet auch, mit Hilfe der Klasse `mcs`, ein Vergleich der in der FTree-Datenbank gespeicherten Feature Trees statt.

### A.2 Konfigurationsdatei


Die Einträge `TREESIZE1`, `TREESIZE2` und `MINSIZE` sind optional. Bei Programmstart wird die Größe der Bäume und die Mindestgröße der maximalen CSTI und CST abgefragt. Diese können alternativ über die Konfigurationsdatei festgelegt werden, so dass das Programm ohne Benutzereingabe die Algorithmen ausführt.

A.3. Bedienungsanleitung

Der Zufallszahlengenerator kann über `RNGSTART` und `RNGSTART2` initialisiert werden. Auf diese Weise sind die Ergebnisse reproduzierbar. Falls die Werte übereinstimmen, sind die generierten Bäume identisch. Eine Ausnahme ist, wenn beide Werte auf 0 gesetzt sind. Dann werden beide Bäume unabhängig zufällig erzeugt.


Für Laufzeitvergleiche sollte die Option `ONLY_TOTAL` auf 1 gesetzt werden, denn Textausgaben während der Berechnung erhöhen die Gesamtzeit und verfälschen das Ergebnis.

Ungültige Eingaben werden beim Programmstart überprüft. Dabei erfolgt ein Hinweis, wie diese Werte zu korrigieren sind. Eine Garantie, dass alle möglichen Falschteingaben abgefangen werden, wird nicht gegeben.

Bei Programmstart müssen bis zu drei Zahlen eingegeben werden, die Kantenanzahl der beiden Bäume und gegebenenfalls die Mindestgröße der maximalen CSTI bzw. CST. Die Eingabe entfällt, falls diese Zahlen über die Konfigurationsdatei voreingestellt wurden.

Eine positive Zahl für die Baumgröße bestimmt die Anzahl der Kanten des Baumes. Eine negative Zahl liest den \(|n|\)-ten Eintrag aus der beiliegenden FTree-Datenbank `1252427226359312397.fdf` und generiert daraus einen Baum. Falls die erste eingegebene Zahl 0 und die zweite Zahl \(p\) positiv ist, werden die ersten \(p\) Bäume aus der FTree-Datenbank paarweise miteinander verglichen und nach Ähnlichkeit sortiert angezeigt.

# Abbildungsverzeichnis

2.1 Common Subtree und Common Subtree Isomorphismus .......................... 6
2.2 Gewurzelter Baum ............................................................................. 8
2.3 Gewurzelter Teilbaum ........................................................................ 9
2.4 Algorithmus von Edmonds - Dynamische Programmierung .................... 10
2.5 Algorithmus von Edmonds - Zusammenfügen der Teillösungen .................. 12
2.6 Matchings im Algorithmus von Edmonds .............................................. 13
2.7 Augmentierung eines Matchings ........................................................... 20
2.8 Erweiterung und Augmentierung in MaxWBPM ...................................... 20

3.1 Anzahl Maximum CSTI nicht polynomiell beschränkt ............................. 24
3.2 Teilprobleme im Algorithmus von Uno ................................................ 30
3.3 Identische Matchings im partiell modifizierten Algorithmus von Uno ......... 32
3.4 Rekursionsabbruch im modifizierten Algorithmus von Uno ..................... 32
3.5 Enumeration auf zwei gewurzelten Teilbäumen ..................................... 35
3.6 Festlegung der Ordnungsfunktion $I$ in Algorithmus 3.1 ......................... 38
3.7 Wahl des gewurzelten Teilbaums $r$ in Algorithmus 3.1 ......................... 41
3.8 Kantenlöschung bei maximalen Common Subtree Isomorphismen ............. 44
3.9 Beispiel zur Aktualisierung von $p$ in Abschnitt 3.4.2 .............................. 48
3.10 Common Weighted Subtree Isomorphismus ........................................... 50
3.11 Nicht-maximaler MCWSTI ................................................................. 52
3.12 Baumkanonisierung ........................................................................... 55
3.13 Baumkanonisierung, verschiedene Wurzeln ........................................ 58
3.14 Baumkanonisierung, ineinander enthaltene Bäume ................................. 61
3.15 Enumeration von Teilbäumen, auf denen ein Isomorphismus existiert ........ 64

4.1 Auswertung - Maximale CSTI mit Mindestgröße .................................. 72
4.2 Auswertung - Vergleich MaxWBM und MaxWBM' ................................ 73
4.3 Auswertung - Feature Trees .................................................................. 74
4.4 Auswertung - Maximale CST mit Mindestgröße .................................... 80
4.5 Auswertung - Teilbaumpaare mit ausschließlich maximalen Isomorphismen mit Mindestgröße ................................................................. 83
## Algorithmenverzeichnis

<table>
<thead>
<tr>
<th>Kapitel</th>
<th>Titel</th>
<th>Seite</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Berechnung von $D(R^u_v, S^w_x)$ für ein Paar von gewurzelten Bäumen</td>
<td>10</td>
</tr>
<tr>
<td>2.2</td>
<td>Berechnung der Größe eines Maximum Common Subtree Isomorphismus</td>
<td>12</td>
</tr>
<tr>
<td>3.1</td>
<td>Enumeration aller Maximum Common Subtree Isomorphismen</td>
<td>33</td>
</tr>
<tr>
<td>3.2</td>
<td>Aufzählung des nächsten Maximum CSTI auf einem Paar von gewurzelten Bäumen</td>
<td>37</td>
</tr>
<tr>
<td>3.3</td>
<td>Enumeration aller maximalen Common Subtree Isomorphismen</td>
<td>44</td>
</tr>
<tr>
<td>3.4</td>
<td>Enumeration aller maximalen CSTI mit einer Mindestgröße $m$</td>
<td>46</td>
</tr>
<tr>
<td>3.5</td>
<td>Berechnung von $B(R^u_v, S^w_x)$ für ein Paar von gewurzelten Bäumen</td>
<td>58</td>
</tr>
<tr>
<td>3.6</td>
<td>Enumeration aller Maximum Common Subtrees</td>
<td>59</td>
</tr>
<tr>
<td>3.7</td>
<td>Einfügen einer Kanonisierung in einen Binärbaum mit Subtree Isomorphieprüfung</td>
<td>62</td>
</tr>
<tr>
<td>3.8</td>
<td>Enumeration aller maximalen CST mit einer Mindestgröße $m$</td>
<td>63</td>
</tr>
<tr>
<td>3.9</td>
<td>Nebenläufige Enumeration aller maximalen Common Subtree Isomorphismen</td>
<td>67</td>
</tr>
<tr>
<td>3.10</td>
<td>Nebenläufige Enumeration von maximalen CSTI - Hilfsalgorithmus</td>
<td>67</td>
</tr>
</tbody>
</table>
Tabellenverzeichnis

4.1 Auswertung - Maximale CSTI .............................................. 70
4.2 Auswertung - Maximum CSTI ............................................ 71
4.3 Auswertung - Vergleich CSTI und CCISGI ............................. 71
4.4 Auswertung - Wahl der Kantenreihenfolge ........................... 72
4.5 Auswertung - Maximum CSTI mit Bäumen verschiedener Größe .... 74
4.6 Auswertung - MSTI Ordnungsfunktion gegen Kantenlöschung .... 75
4.7 Auswertung - Nebenläufigkeit .......................................... 75
4.8 Auswertung - Maximum CST ............................................. 79
4.9 Auswertung - Maximale CST ............................................ 79
4.10 Auswertung - Maximum CST mit Bäumen verschiedener Größe .... 80
4.11 Auswertung - Teilbaumpaare mit ausschließlich größtmöglichen Isomorphismen ........................................... 82
4.12 Auswertung - Teilbaumpaare mit ausschließlich maximalen Isomorphismen ........................................ 83
4.13 Auswertung - Teilbaumpaare mit ausschließlich größtmöglichen Isomorphismen, verschiedene Baumgrößen .......................... 84
Index

$B(R_u^w, S_x^w)$, siehe Binärbaum
$D(R_u^w, S_x^w)$, 9
$R_u^w$, 8
$S_x^w$, 8
c-Kante, 27

Algorithmus von Edmonds, 7
Algorithmus von Uno, 29

Baum, 5
gewurzelt, 8
Kanonisierung, 55
Teil-, 5
gewurzelt, 8
Zentrum eines, 58
Binärbaum, 56

CCISG, siehe Connected Common Induced Subgraph
Clique, 26
c-zusammenhängend, 27
CLSTI, siehe Common Labeled Subtree Isomorphismus
Common Labeled Subtree Isomorphismus, 50
Common Subgraph, 25
Common Subgraph Isomorphismus, 25
Common Subtree, 6
Common Subtree Isomorphismus, 6
Common Weighted Subtree Isomorphismus, 50
Connected Common Induced Subgraph, 25
CST, siehe Common Subtree maximal, 6

CSTI, siehe Common Subtree Isomorphismus maximal, 6
CWSTI, siehe Common Weighted Subtree Isomorphismus

Enumeration, 24
Enumerationsalgorithmus, 24
Exzentrizität, 58

G(y), siehe Zulässiger Teilgraph
Graph, 3
azyklich, 4
bipartit, 14
gelabelt, 7
gerichtet, 4
zusammenhängend, 5
Isomorphismus, 5
auf gelabelten Graphen, 7

Kante, siehe Graph
Knoten, siehe Graph
M-exponiert, siehe Matching
M-gematched, siehe Matching
Kreis, 4
Alternierender, 29

Matching, 14
Maximum Weight Bipartite, 14
Maximum Weight Bipartite Perfect, 17
Perfekt, 17

MaxWBM, 14
MaxWBM', 15
MaxWBPM, 17
MCLSTI, *siehe* Common Labeled Subtree Isomorphismus
MCST, *siehe* Common Subtree
MCSTI, *siehe* Common Subtree Isomorphismus
MCWSTI, *siehe* Common Weighted Subtree Isomorphismus

Ordnungsfunktion, 38

Pfad, 4
  *M*-augmentierend, 19
Polynomial delay, 24
Polynomial space, 25
Polynomial total time, 24

STI, *siehe* Subtree Isomorphismus
Subtree Isomorphismus, 61

Teilgraph, 4
Teilgraph, induziert, 4

Vertex Product Graph, 26
VPG, *siehe* Vertex Product Graph

Wald, 5

Zulässiger Teilgraph, 19
Zusammenhangskomponente, 5
Literaturverzeichnis


[10] CREIGNOU, NADIA, ARNE MEIER, JULIAN-STEFFEN MÜLLER, JOHANNES SCHMIDT und HERIBERT VOLLMER: Paradigms for Parameterized Enumeration. In: Chat-


Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst habe und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet sowie Zitate kenntlich gemacht habe.

Dortmund, den 24. Februar 2014

Andre Droschinsky