technische universität
dortmund

Effiziente Enumerationsalgorithmen für Common Subtree Probleme

Andre Droschinsky

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

Fakultät für Informatik
Algorithm Engineering (LS 11)
44221 Dortmund / Germany
http://ls11-www.cs.uni-dortmund.de/

technische universität
dortmund

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
http://ls11-www.cs.tu-dortmund.de

Inhaltsverzeichnis

Kapitel 1

Einleitung

Diese Diplomarbeit befasst sich mit der Enumeration von Common Subtrees und Common Subtree Isomorphismen. Cuissart und Hébrard schreiben: „Graphs are widely used to represent objects in various domains such as chemical information, computer imaging etc.“ [11]. Koch führt weiter aus: „In many applications it is important to find maximal common subgraphs in two graphs. Because the problem is NP-complete it cannot be solved for arbitrarily large graphs.“ [17]. Die Suche nach gemeinsamen Teilgraphen ist folglich eine wichtige und schwierige Aufgabe in vielen Anwendungsgebieten. Für dieses Problem existieren zahlreiche Veröffentlichungen [6, 7, 11, 17, 19, 25], auf die in Kapitel 3 teilweise eingegangen wird. Die Datenobjekte können in Form von Bäumen, einer Teilmenge der Graphen, vorliegen. Dies ist beispielsweise bei sogenannten Feature Trees [24] gegeben. Feature Trees sind auf eine Baumstruktur reduzierte Moleküle. Diese Arbeit befasst sich mit dem spezialisierten Problem, in dem Bäume als Eingabe dienen. Es wird unter anderem gezeigt, dass die Enumeration von maximalen Common Subtree Isomorphismen für diese Problemstellung wesentlich schneller als mit einem Algorithmus für allgemeine Graphen erfolgen kann, indem die speziellen Eigenschaften von Bäumen ausgenutzt werden. Das Vergleichsverfahren von Cazals und Karande [7] wird in Kapitel 3 vorgestellt.

Die in dieser Arbeit entwickelten Enumerationsverfahren basieren auf dem Algorithmus von Edmonds und Matula [21] zur Bestimmung der Größe eines Maximum Common Subtree. Dieses Problem ist in polynomieller Zeit lösbar [21]. In Kapitel 2 wird dieses Verfahren vorgestellt und schrittweise modifiziert, so dass zunächst ein konkreter Maximum Common Subtree Isomorphismus berechnet wird. Darauf aufbauend erfolgt in Kapitel 3 eine Erweiterung um die Enumeration dieser Isomorphismen. Dazu bedarf es der Hilfe von Enumerationsalgorithmen für verschiedene Typen von Matchings, die ebenfalls in Kapitel 3 vorgestellt werden. Als Alternative zu Isomorphismen wird die Enumeration von Maximum Common Subtrees behandelt. Des Weiteren werden zahlreiche Modifikationen der Probleme betrachtet. Dazu gehören Bäume mit Bezeichnern und die Aufzählung von maximalen anstelle von Maximum Common Subtree Isomorphismen bzw. Maximum Common Sub-

trees. Zu den vorgestellten Algorithmen erfolgen theoretische Untersuchungen in Bezug auf Laufzeit und Speicherplatzverbrauch. Die Enumeration von maximalen Common Subtree Isomorphismen ermöglicht einen direkten Vergleich mit dem Enumerationsverfahren für allgemeine Graphen bei Eingabe von Bäumen.

Die vorgestellten Algorithmen wurden mit Hilfe der Programmiersprache C++ in ein Programm übersetzt, das Teil dieser Diplomarbeit ist. Die Algorithmen werden in experimentellen Untersuchungen in Kapitel 4 auf ihre praktische Effizienz geprüft. Neben Zufallsbäumen und speziellen worst-case-Beispielen dienen dem Programm auch Feature Trees als Eingabe.

Kapitel 2

Das Maximum Common Subtree Problem

Dieses Kapitel beschreibt das Maximum Common Subtree Problem, welches in Abschnitt 2.1 formalisiert wird. In Abschnitt 2.2 wird der Algorithmus von Edmonds und Matula zur Bestimmung der Größe eines Maximum Common Subtrees vorgestellt. Im Abschnitt 2.3 wird beschrieben, wie sich dieser Algorithmus modifizieren lässt, so dass ein Maximum Common Subtree Isomorphismus bzw. Maximum Common Subtree bestimmt werden kann. Abschnitt 2.4 behandelt das Teilproblem, ein Maximum Weight Bipartite Matching zu bestimmen.

2.1 Theoretische Grundlagen

In diesem Abschnitt wird das Maximum Common Subtree Problem formal definiert. Dazu müssen zunächst einige Grundbegriffe aus der Graphentheorie eingeführt werden. Die folgenden Definitionen basieren überwiegend auf dem Buch „Graph Theory“ von R. Diestel [12].

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 *Kanten* 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 $G_1 \subset G_2$ 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.

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

Offenbar gilt, dass k der Länge des Pfades entspricht. Insbesondere sind auch Pfade der Länge 0 erlaubt. Ein Kreis besteht aus mindestens 3 Kanten, wobei der Anfangs- und Endknoten identisch sind. Der oben definierte Pfad wird in [12] auch mit *Pfad von* v_0 *nach* v_k bezeichnet.

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.
Ein zusammenhängender Teilgraph G' eines Graphen G wird *Zusammenhangskomponente* von G genannt, wenn kein zusammenhängender Teilgraph G'' von G mit $G' \subset G''$ existiert.

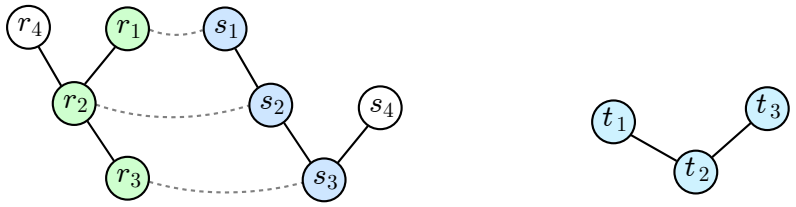
Mit diesen Begriffen lassen sich Bäume definieren.

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

Ein induzierter zusammenhängender Teilgraph eines Baumes soll *Teilbaum* genannt werden. Dieser ist insbesondere wieder ein Baum. Für einen Baum $T = (V, E)$ gilt, dass $|V| = |E| + 1$ ist. Außerdem gilt, dass durch Wegnahme einer beliebigen Kante der Zusammenhang verloren geht. Es gilt sogar, dass ein Baum durch Wegnahme einer Kante in genau zwei Zusammenhangskomponenten zerfällt. Diese offensichtliche Eigenschaft spielt in den später vorgestellten Algorithmen eine wichtige Rolle. Anzumerken ist noch, dass für zwei beliebige Knoten $u, v \in V$ eines Baumes $T = (V, E)$ genau ein Pfad von u nach v in T enthalten ist [12].

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 $\varphi : V_1 \to V_2$ 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 φ wird *Isomorphismus* genannt.
Die *Größe* $|\varphi|$ eines Isomorphismus $\varphi : V_1 \to V_2$ 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 ist in G_2 enthalten bezeichnet. Für das in der Klasse NP enthaltene Entscheidungsproblem, ob zwei Graphen isomorph sind, ist zum Zeitpunkt dieser Arbeit nicht bekannt, ob es in P enthalten ist. Es ist ebenfalls unbekannt, ob es NP-vollständig ist [14]. Die Komplexitätsklasse P beinhaltet alle Probleme, die in polynomieller Zeit deterministisch entschieden werden können. In der Klasse 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.



(a) Maximum CSTI (b) Maximum CST

Abbildung 2.1: Zusammenhang zwischen Maximum CSTI und Maximum CST

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.

Edmonds hat gezeigt, dass sich die Größe eines Maximum Common Subtree zweier Bäume in Polynomialzeit bestimmen lässt [21]. Dieser Algorithmus wird im folgenden Abschnitt 2.2 vorgestellt und spielt in vielen der in dieser Arbeit vorgestellten Enumerationsalgorithmen eine wichtige Rolle. In natürlicher Weise lässt sich aus den Begriffen Common Subtree und Isomorphismus ein *Common Subtree Isomorphismus* ableiten.

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 $\varphi : V'_R \to V'_S$ (vgl. Def. 2.8) als *Common Subtree Isomorphismus* (CSTI) bezeichnet.
Falls φ größtmöglich ist, wird φ *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 φ 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 φ 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 CSTI, die wie folgt definiert sind.

Definition 2.11 (Maximaler CST, Maximaler CSTI [17]). Sei φ 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 *maximaler CST* genannt, wenn es keinen anderen Common Subtree T' von T_1 und T_2 gibt mit $T \subset T'$.

Maximale CST und CSTI können als lokal beste Lösungen aufgefasst werden. Es ist unmittelbar klar, dass jeder Maximum CST bzw. Maximum CSTI auch ein maximaler CST bzw. CSTI ist. Umgekehrt gilt dies jedoch nicht.

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

Definition 2.12 (Gelabelter Graph [9]). Ein *gelabelter Graph* ist ein Graph $G = (V, E)$ zusammen mit einer Abbildung $l : V \cup E \to \Sigma$.

Die Elemente der endlichen Menge Σ werden *Label* oder *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 *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 [6] vollständig sein. In [2] und vielen anderen Quellen haben nur Knoten Bezeichner. Sollen nur die Knoten oder nur die Kanten unterscheidbar sein, so kann jeweils allen Kanten oder allen Knoten das gleiche Label gegeben werden. Ein Beispiel für die Menge Σ 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.

Definition 2.13 (Isomorphismus auf gelabelten Graphen [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 *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 [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.

2.2 Der Algorithmus von Edmonds

In diesem Abschnitt wird der Algorithmus von Edmonds und Matula [21] vorgestellt. Bei Eingabe zweier Bäume $R = (V_R, E_R)$ und $S = (V_S, E_S)$ bestimmt dieser Algorithmus die

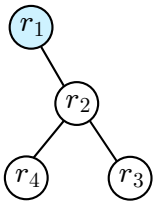


Abbildung 2.2: Gewurzelter Baum mit Wurzel r_1

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[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

Sei $e = uv$ eine beliebige Kante eines Baumes T. Wird diese entfernt, entstehen zwei Zusammenhangskomponenten, wovon eine den Knoten u und die andere den Knoten v enthält. Diese Komponenten können als *gewurzelte Teilbäume* mit Knoten u bzw. v als Wurzel aufgefasst werden.

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_v^u bzw. T_u^v bezeichnet. Die Komponente T_v^u besitzt den Knoten v als Wurzel, T_u^v den Knoten u.

[1] laut Matulas persönlicher Kommunikation mit Edmonds [21]

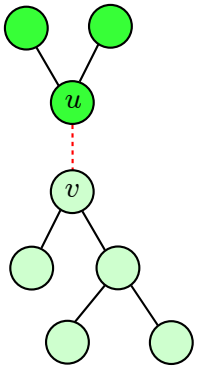


Abbildung 2.3: Der *gewurzelte Teilbaum* T_v^u ist hellgrün dargestellt. T_u^v ist dunkelgrün dargestellt. Die gestrichelte Linie in rot ist die Trennungskante.

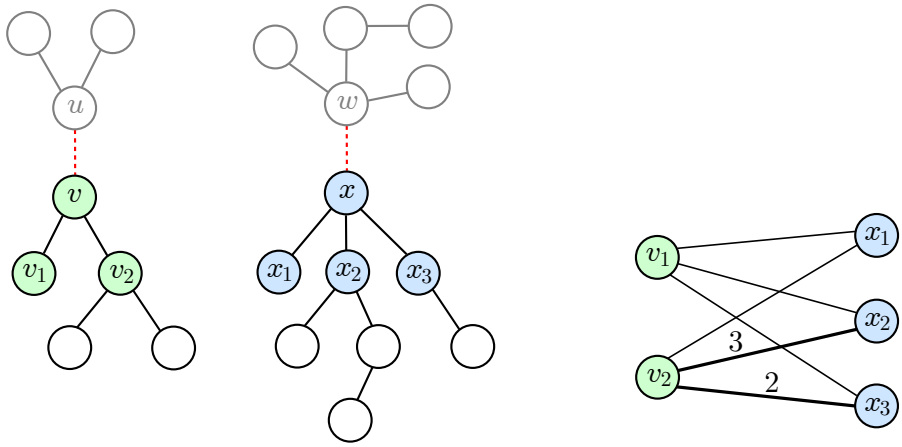
In Abbildung 2.3 ist die Zerlegung eines Baumes T in die zwei gewurzelten Teilbäume T_u^v und T_v^u dargestellt. Ein CSTI auf zwei gewurzelten Teilbäumen wird im Folgenden manchmal abkürzend Isomorphismus genannt. Dabei ist aber stets klar, dass ein maximaler CSTI bzw. MCSTI gemeint ist.

2.2.2 Dynamische Programmierung

Im zweiten Schritt des Algorithmus von Edmonds wird für alle Paare (R_v^u, S_x^w) von gewurzelten Teilbäumen die Größe $D(R_v^u, S_x^w)$ 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 $D(R_v^u, S_x^w)$ 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 $w(v_i x_j) = D(R_{v_i}^v, S_{x_j}^x)$ 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 $D(R_v^u, S_x^w) = 1$. Ansonsten gilt $D(R_v^u, S_x^w) = 1 + W(M)$ für das oben genannte Matching. Jeder gewurzelte Teilbaum $R_{v_i}^v$ besitzt mindestens einen Knoten weniger als R_v^u und jeder Teilbaum $S_{x_j}^x$ mindestens einen weniger als S_x^w. D ist somit wohldefiniert.

In Abbildung 2.4 sind zwei gewurzelte Teilbäume R_v^u und S_x^w 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 $D(R_v^u, S_x^w) = 5$. Dies wird durch $\varphi(v) = x$ und beispielsweise $\varphi(v_1) = x_1$ und $\varphi(v_2) = x_2$ realisiert.



(a) R_v^u und S_x^w

(b) Zugehöriger gewichteter bipartiter Graph

Abbildung 2.4: Der Wert $D(R_v^u, S_x^w)$ wird mit Hilfe eines MaxWBM bestimmt.

Eingabe: R_v^u, S_x^w
Ausgabe: $D(R_v^u, S_x^w)$

if $D(R_v^u, S_x^w)$ noch nicht berechnet **then**
  if R_v^u oder S_x^w ist ein Blatt **then**
    $D(R_v^u, S_x^w) \leftarrow 1$
  else
    for all $i \in \{1, \ldots, k\},\ j \in \{1, \ldots, l\}$ // k, l wie in Abschnitt 2.2.2 **do**
      $w(v_i x_j) \leftarrow \mathrm{GetD}(R_{v_i}^v, S_{x_j}^x)$
    end for
    $D(R_v^u, S_x^w) \leftarrow 1 + W(M)$ // $W(M)$ wie in Abschnitt 2.2.2
  end if
end if
return $D(R_v^u, S_x^w)$

Algorithmus 2.1: $\mathrm{GetD}(R_v^u, S_x^w)$

Die Werte von D lassen sich berechnen, indem die Teilbäume der Größe nach sortiert werden, wie von Matula in [21] beschrieben. Dann sind in jedem Schritt alle Kantengewichte bekannt. Die Sortierung ist aber nicht nötig, alternativ lassen sich die Werte von D mit dem vom Verfasser dieser Arbeit entwickelten Algorithmus 2.1 (GetD) berechnen.

Falls der Wert $D(R_v^u, S_x^w)$ 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. Da

die gewurzelten Bäume, bezogen auf festes S und R, statisch sind, müssen beim Aufruf von GetD nicht zwei gewurzelte Bäume übergeben werden, sondern lediglich Referenzen auf diese. Die Funktionsaufrufe selbst kosten folglich jeweils nur O(1) Zeit.

Dass bei dieser Implementierung die Sortierung entfällt, ist nicht der einzige Vorteil. GetD kann jederzeit für beliebige Teilbäume aufgerufen werden. Bei der Berechnung des Wertes wird sichergestellt, dass kein Matching berechnet wird, das für die Berechnung des gesuchten Wertes nicht notwendig ist. Im Algorithmus von Edmonds werden alle Werte benötigt, es gibt mit dieser Implementierung folglich keine Ersparnis. Allerdings werden später Algorithmen vorgestellt, die andere Werte nach dem gleichen Schema berechnen. Dort werden teilweise, abhängig von der Eingabe, nicht alle Werte benötigt. Unnötige Berechnungen entfallen dabei automatisch.

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_x^w zwei gewurzelte Teilbäume. Dann liefert, wie in Abschnitt 2.2.2 beschrieben, $\mathrm{GetD}(R_v^u, S_x^w)$ die Größe eines MCSTI der beiden Teilbäume R_v^u und S_x^w mit der Einschränkung, dass v auf x abgebildet wird. $\mathrm{GetD}(R_u^v, S_w^x)$ ist die Größe eines MCSTI der beiden Teilbäume R_u^v 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 $D(R_v^u, S_x^w) = 2$, dargestellt durch die hellgrünen und hellblauen Knoten. Weiterhin gilt $D(R_u^v, S_w^x) = 2$. Das ist durch die dunkel gefärbten Knoten dargestellt. Ein MCSTI mit $\varphi(v) = x$ und $\varphi(u) = w$ 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_x^w) gewählt wurde, muss natürlich das Paar (R_u^v, S_w^x) nicht mehr gewählt werden, da der Lösungswert offensichtlich identisch ist.

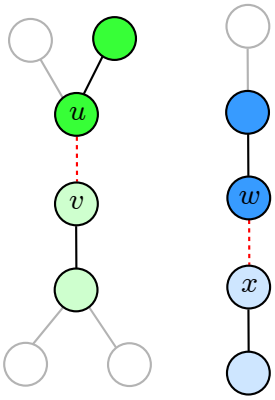


Abbildung 2.5: Ein Maximum CSTI mit der Einschränkung $\varphi(u) = w$ und $\varphi(v) = x$

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 ← 0 // Bisher gefundenes Maximum
7:      for all Paare (R_v^u, S_x^w) von gewurzelten Teilbäumen do
8:          s ← GetD(R_v^u, S_x^w)+GetD(R_u^v, S_w^x)
9:          if s > m then
10:             m ← 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.

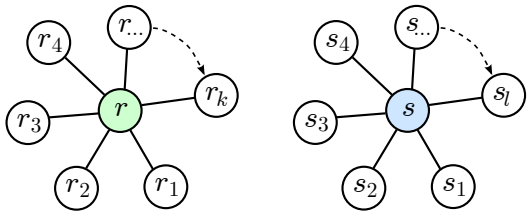


Abbildung 2.6: Bild zur Laufzeitabschätzung in Satz 2.16

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* $O(|E_R| \cdot |E_S| \cdot \max\{|E_R|, |E_S|\}^3)$ *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 $(R_r^{r_i}, S_s^{s_j})$, $i \in \{1, \ldots, k\}$, $j \in \{1, \ldots, l\}$ zugehörigen bipartiten Graphen.

2.3 Bestimmung eines MCST oder MCSTI

In diesem Abschnitt werden Modifikationen des Algorithmus von Edmonds [21] vorgestellt. Statt der Größe eines Maximum Common Subtrees soll ein Maximum Common Subtree Isomorphismus oder ein Maximum Common Subtree ausgegeben werden. Im Folgenden wird dabei nur der nichttriviale Fall behandelt, in dem beide Bäume mindestens zwei Knoten besitzen. Die genannten Modifikationen stammen vom Verfasser dieser Arbeit.

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* definiert.

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.

Wenn für ein Matching M von G kein anderes Matching M' mit $M \subset M'$ existiert, wird M *maximal* genannt. Zu einem maximalen Matching lässt sich also keine weitere Kante hinzufügen. Wenn ein Knoten v zu einer Matchingkante aus M inzident ist, sagt man M *matched* v, ansonsten M *matched* v *nicht* [12]. Eine andere Bezeichnung für letztere Aussage ist v *ist* M*-exponiert*. Der Verweis auf M wird weggelassen, wenn klar ist, welches Matching gemeint ist.

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 $w : E \to \mathbb{Q}$ 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 $W(M') > W(M)$.

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 systematisches 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 $\{v_1 x_{z_1}, v_2 x_{z_2}, \ldots, v_k x_{z_k}\}$. 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 injektiven Folgen (z_i). Diese soll lexikographisch 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* $O(\frac{l!}{(l-k)!})$ *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'_i) lexikographisch auf (z_i) folgend. Wenn dann z_j durch z'_j für ein oder mehrere j ersetzt wird, wird zum aktuellen Gewicht $w(v_j x_{z'_j}) - w(v_j x_{z_j})$ 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 $\frac{l!}{(l-k)!}$. 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_i) zu (z'_i) 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 $\frac{l!}{(l-k)!} \cdot (1 + 1 + \frac{1}{2} + \frac{1}{6} + \frac{1}{24} + \ldots) \leq \frac{l!}{(l-k)!} \cdot (2 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \ldots) \leq \frac{l!}{(l-k)!} \cdot 3$ beschränkt. Mit der konstanten Zeit $O(1)$ pro Änderung (vgl. den folgenden Absatz) ergibt dies eine Zeitschranke von $O(\frac{l!}{(l-k)!})$. 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 exponierten Knoten aus X festzuhalten. $N[0]$ steht dabei für den ersten (kleinster Index) exponierten Knoten. $N[m] = n$ bedeutet dabei, dass der nächste exponierte Knoten nach x_m der Knoten x_n ist. Falls $N[m] = 0$, gibt es keinen exponierten 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 exponierte auf x_m folgende Knoten dann nicht mehr der Knoten $x_n = x_{z'_j}$ sondern $x_{N[n]}$ ist. Andererseits ist durch die Änderung von z_j zu z'_j der Knoten x_m exponiert. 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 $N(r) = m$ galt, 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. □

Um statt des Gewichts eines MaxWBM' ein konkretes Matching zu erhalten, kann der oben beschriebene Algorithmus insofern modifiziert werden, dass während der Aufzählung immer das Matching mit dem bisher größten gefundenen Gewicht gespeichert wird. Dabei ist darauf zu achten, dass nicht jedes mal das komplette Matching gespeichert wird, sondern nur die Kanten bzw. Folgenglieder, die sich im Vergleich zum vorherigen gespeicherten Matching unterscheiden. Am Ende des Durchlaufs wird dann das zuletzt gespeicherte Matching ausgegeben. Dieses ist ein MaxWBM'. Alternativ kann der Algorithmus auch zweimal durchlaufen werden. Im ersten Durchlauf wird das Gewicht eines MaxWBM' bestimmt, im zweiten Durchlauf wird das lexikographisch erste Matching mit diesem Gewicht ausgegeben. In beiden Fällen bleibt die Laufzeitschranke aus Satz 2.22 gültig.

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. Beispielweise gilt für ein Straßennetz 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 $\frac{l!}{(l-k)!}$ 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 $w : E \to \mathbb{Q}$. 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.

Ein MaxWBPM lässt sich laut Fukuda [13] mit der ungarischen Methode von Kuhn [18] bestimmen. Die ungarische Methode bestimmt eine optimale Lösung für ein Zuordnungsproblem [18]. Fukuda schreibt: „The Hungarian method is an efficient algorithm for finding a minimal cost perfect matching in a weighted bipartite graph. [...] It is well-known that an optimal solution to assignment problems can be computed efficiently by the Hungarian method [18].“ Zu beachten ist, dass Fukuda in [13] ein Minimum Weight Bipartite Perfect Matching (MinWBPM) auf ein Zuordnungsproblem zurückführt. Auch in vielen anderen Literaturquellen wird ein MinWBPM statt einem MaxWBPM gesucht. Dies stellt aber kein Problem dar. Zum einen können alle Gewichte negiert werden. Dann entspricht ein MaxWBPM in dem gegebenen Graphen einem MinWBPM in dem Graphen mit den negativen Gewichten. Zum anderen kann auch die Transformation auf das Zuordnungsproblem modifiziert werden. Diese zweite Möglichkeit wird im Folgenden beschrieben.

Die Bestimmung eines MaxWBPM lässt sich nach [13] als lineares Programm **(P)** formulieren. Die Menge $\delta(v)$ ist durch alle zu v inzidenten Kanten definiert. Zu beachten ist, dass im Folgenden maximiert wird, während in [13] minimiert wird.

$$\begin{array}{lll} \text{Maximiere} & \sum\limits_{e\in E} w(e)\ x(e) & \\ \text{Nebenbedingungen} & \sum\limits_{e\in\delta(v)} x(e) = 1 & \forall v \in V \cup X \\ & x(e) \in \{0,1\} & \forall e \in E \end{array}$$

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

$$\begin{array}{lll} \text{Minimiere} & \sum\limits_{v\in V} y(v) + \sum\limits_{x\in X} y(x) & \\ \text{Nebenbedingung} & y(v) + y(x) \geq w(e) & \forall e = vx \in E \end{array}$$

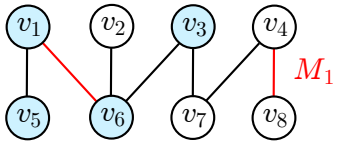
Für eine Lösungsvektor $y \in \mathbb{R}^{V\cup X}$ des dualen Programms wird die *zulässige Kantenmenge* definiert durch $E(y) := \{e = vx \in E \mid y(v) + y(x) = w(e)\}$ und der *zulässige Teilgraph* durch $G(y) := (V \cup X, E(y))$ [13]. Zu beachten ist, dass auf den Kanten von $G(y)$ keine Gewichtsfunktion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

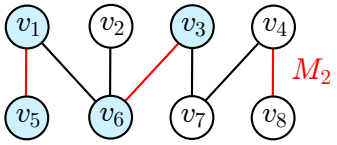
Eine optimale Lösung y^* von **(D)** lässt sich in Zeit $O(|V|^3)$ finden. Dabei ergibt sich gleichzeitig ein Matching größten Gewichts sowie der Graph $G(y^*)$ [13]. Auf diesen wird in Abschnitt 3.2 zurückgegriffen. Die Idee, eine optimale Lösung zu finden, wird im Folgenden beschrieben [18]. Dabei 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.

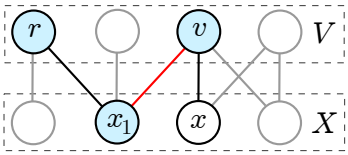


(a) Matching vor Augmentierung

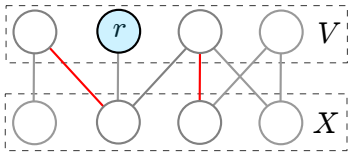


(b) Matching nach Augmentierung

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



(a) T (blaue Knoten) vor Wahl der Kante vx



(b) T nach Augmentierung

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 Schlupfs 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 Baumerweiterungsschritt 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 verschiedenfarbigen 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 $y(v) \leftarrow y(v) + \varepsilon$ für alle $v \in B(T)$ und $y(x) \leftarrow y(x) - \varepsilon$ für alle $x \in A(T)$ modifiziert. Dabei wird $\epsilon > 0$ 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 $O(|V|)$ möglich. Insgesamt ergibt sich daher eine Zeitschranke von $O(|V|^3)$.

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

Dieses Kapitel behandelt die Enumeration von Common Subtrees und Common Subtree Isomorphismen. In Abschnitt 3.1 werden Enumeration und Enumerationsalgorithmen zunächst allgemein besprochen. Verwandte Arbeiten zum Thema Enumeration in Hinblick auf Common Subtree Isomorphismen werden in Abschnitt 3.2 behandelt. Der grundlegende Algorithmus zur Enumeration von Maximum Common Subtree Isomorphismen wird in Abschnitt 3.3 vorgestellt. Im folgenden Abschnitt 3.4 werden die Enumeration von maximalen Common Subtree Isomorphismen und weitere Kriterien vorgestellt, nach denen enumeriert werden kann. Um die in der Einführung erwähnten Feature Trees als Eingabe in Betracht ziehen zu können, sind Knotenbezeichner erforderlich. Knoten- und Kantenbezeichner werden in Abschnitt 3.5 behandelt. In Abschnitt 3.6 werden Algorithmen zur Enumeration von Common Subtrees und Teilbäumen, auf denen Isomorphismen existieren, vorgestellt. Im folgenden Abschnitt 3.7 wird diskutiert, wie durch Parallelisierung die Berechnungszeit verringert werden kann.

3.1 Grundlagen der Enumeration

Viele Probleme in der Informatik oder auch in anderen Wissenschaften haben eine eindeutige Lösung. Beispielsweise könnte eine Frage für solch ein Problem lauten: „Gibt es für gegebene Bäume ein Maximum Common Subtree mindestens der Größe 10?“. Die Antwort darauf ist „ja“ oder „nein“. Die zweite Stufe ist die Frage nach einer konkreten Größe. Eine passende Fragestellung ist: „Wie groß ist ein Maximum Common Subtree?“. Darauf aufbauend wird dann in der dritten Stufe eine konkrete Lösung gesucht: „Berechne einen Maximum Common Subtree und gebe diesen aus!“. Dies ist die übliche Gliederung in der theoretischen Informatik [3]. Diese Varianten wurden in Kapitel 2 behandelt.

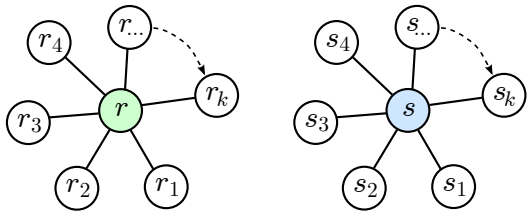


Abbildung 3.1: Anzahl Maximum CSTI nicht polynomiell beschränkt.

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) Laufzeitschranke 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 polynomiell. Aus diesem Grund werden Enumerationsalgorithmen anders klassifiziert. Im Folgenden werden einige Klassen, basierend auf [15], vorgestellt.

(a) *Polynomial total time*. Die Gesamtzeit der Berechnung ist beschränkt durch ein Polynom in der Größe der Eingabe sowie der tatsächlichen Anzahl der Lösungen. Ein Beispiel nach [15] ist ein Algorithmus zur Aufzählung aller maximalen unabhängigen Teilmengen von Paull und Unger mit einer Laufzeit von $O(n^2C)$. Dabei ist n die Größe der Eingabe und C die Anzahl der Lösungen.

(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-

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 MaxWBM' 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.

(d) *Polynomial space*. Zur Berechnung der Ausgabe darf nur polynomiell viel Speicherplatz, bezogen auf die Eingabelänge, verwenden werden. Für den MaxWBM'-Algorithmus in der Enumerationsvariante trifft dies beispielsweise zu. Die Algorithmen aus Abschnitt 3.6 brauchen allerdings mehr als polynomiell viel Platz.

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

In diesem Abschnitt werden verschiedene Ansätze zur Enumeration von *Common Subgraph Isomorphismen* beschrieben. Zunächst lässt sich festhalten, dass es zahlreiche Varianten dieses Problems gibt [4, 7, 11, 17, 19]. Die Notationen und Abkürzungen sind in den genannten Quellen nicht einheitlich. Die im Folgenden angegebene Definitionen orientieren sich an den Notationen aus Kapitel 2 dieser Arbeit.

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 höchstmöglicher 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 $\varphi : V(G_1') \to V(G_2')$ 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.

Ein Verfahren, in dem Maximalität nicht vorausgesetzt wird, wird von Levi [19] beschrieben. Dort werden CISGI der Größe k enumeriert, indem Paare von Teilgraphen der Größe k bestimmt und auf diesen dann Isomorphismen gesucht werden. Dort wird außerdem vorgeschlagen, maximale CISGI mit Hilfe einer Reduzierung auf das *Cliquen-Problem* zu finden. Von Koch [17] wird diese Reduzierung für die Enumeration von maximalen CSGI, CISGI, CCSGI und CCISGI aufgegriffen. Die Transformation findet statt, indem ein *Vertex Product Graph* (Definition 3.3) bzw. *Edge Product Graph* gebildet wird, auf dem dann maximale *Cliquen* (Definition 3.2) bzw. maximale *c-zusammenhängende Cliquen* (Definition 3.4) gesucht werden. Zur Enumeration von Cliquen gibt es viele Algorithmen, der schnellste und meist genutzte ist der Bron-Kerbosch-Algorithmus [17]. Auf diesem basiert auch die im Folgenden vorgestellte Enumeration.

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 $u_H v_H \in E_H$ 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 $u_1 v_1$ und $u_2 v_2$ 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 CCISGI 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

in [7] beschriebene Algorithmus wurde im Rahmen dieser Arbeit implementiert. In Abschnitt 4.1 erfolgt ein Vergleich der Laufzeiten zwischen dem allgemeineren Algorithmus zur Enumeration von maximalen CCISGI, mit Bäumen als Eingabe, und einem auf Bäume spezialisierten Algorithmus zur Enumeration aller maximalen CSTI, der in Abschnitt 3.4.1 vorgestellt wird.

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[1] 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-zusammenhängende Cliquen beschrieben.

Der Algorithmus startet jeweils nacheinander in jedem Knoten u_i des Graphen. Dieser wird in der Menge R gespeichert. In der Menge R sind die Knoten der Clique, die gerade erweitert wird. In der Menge P sind die Knoten enthalten, durch die R erweitert werden kann. Für jeden Knoten aus P gilt, dass dieser mit allen Knoten aus R über eine c- oder eine d-Kante verbunden ist. Dabei muss mindestens eine dieser Verbindungen eine c-Kante sein. Dies stellt sicher, dass R nach der Hinzunahme eines Knotens aus P weiterhin eine c-zusammenhängende Clique ist. Zu Beginn sind folglich in P alle Knoten gespeichert, die mit u_i über eine c-Kante verbunden sind. Die Knoten aus P werden dann nacheinander zur Menge R hinzugefügt. Für jede Hinzunahme beginnt eine neue Rekursion, die die folgenden Knoten, wieder rekursiv, hinzunimmt. Auf diese Weise werden alle möglichen Kombinationen von Knoten aus P zu R hinzugenommen. Somit wird keine maximale c-zusammenhängende Clique vergessen und dementsprechend alle aufgezählt.

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-Kanten verbunden sind. Wenn R um einen Knoten u_i erweitert wird, können alle Knoten

[1] Dies hat keinen Einfluss auf den Algorithmus und betrifft nur die Konstruktion des VPG.

$u_j \in Q$, 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 Cliquen 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 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'.

Eine Möglichkeit, MaxWBM (in einem nicht notwendig vollständigen bipartiten Graphen) zu enumerieren, wird im Folgenden beschrieben. Dabei werden Ergebnisse von Fukuda und Matsui [13], Kuhn [18] und Uno [27] miteinander kombiniert, um einen polynomial-delay-Algorithmus für die Enumeration aller MaxWBM zu erhalten.

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.

Anstelle von perfekten Matchings maximalen Gewichts in G können perfekte Matchings in $G(y^*)$ enumeriert werden [13]. Der Enumerationsalgorithmus von Uno [27] für perfekte Matchings wird im Folgenden vorgestellt. Anschließend wird noch die nötige Modifikation beschrieben, um MaxWBM im Graphen G', der ein Teilgraph von G ist, zu erhalten.

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.

Jeder alternierender Kreis K besteht abwechselnd aus zum Matching gehörenden Kanten und nicht zum Matching gehörenden Kanten. Außerdem sind alle Knoten aus V_K paarweise verschieden. Beides folgt daraus, dass M ein Matching ist. Wenn zu einem gegebenen perfekten Matching die Kanten entlang eines alternierenden Kreises ausgetauscht werden, entsteht ein anderes perfektes Matching. In einem bipartiten Graphen besteht die symmetrische Differenz zwischen zwei perfekten Matchings aus alternierenden Kreisen. In einem bipartiten Graphen mit perfektem Matching gibt es also genau dann ein anderes perfektes Matching, wenn es einen alternierenden Kreis gibt [27]. Genau diese Eigenschaft wird im rekursiven Algorithmus von Uno genutzt.

Zunächst sucht der Algorithmus von Uno ein perfektes Matching im gegebenen Graphen. Dieser Schritt kann im Rahmen dieser Arbeit entfallen, denn nach Abschnitt 2.4.1 liefert die Konstruktion des zulässigen Teilgraphen $G(y)$ ein perfektes Matching M. Dieses wird als erstes Matching ausgegeben. Dann wird die Rekursion auf $G(y)$ und M begonnen.

In jedem Rekursionsaufruf wird ein alternierender Kreis gesucht. Dazu wird der Graph gerichtet, indem die Matchingkanten von V zu X zeigen und die anderen Kanten von X zu V. Mit Tiefensuche kann dann ein alternierender Kreis gefunden werden, falls dieser existiert. Falls kein Kreis gefunden wird, gibt es kein weiteres perfektes Matching und die Rekursion stoppt. Ansonsten werden die Kanten entlang des alternierenden Kreises ausgetauscht und es entsteht ein neues perfektes Matching M'. Dieses wird ausgegeben. Anschließend wird eine Kante $e = vx$ gewählt, die zu M und zum alternierenden Kreis gehört. Diese Kante gehört somit nicht zu M'. Daraus werden zwei Teilprobleme zur Enumeration aller perfekten Matchings auf dem gegebenen Graphen generiert. Das eine Teilproblem umfasst die Matchings, in denen e enthalten ist. Das andere umfasst die Matchings, in denen e nicht enthalten ist.

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

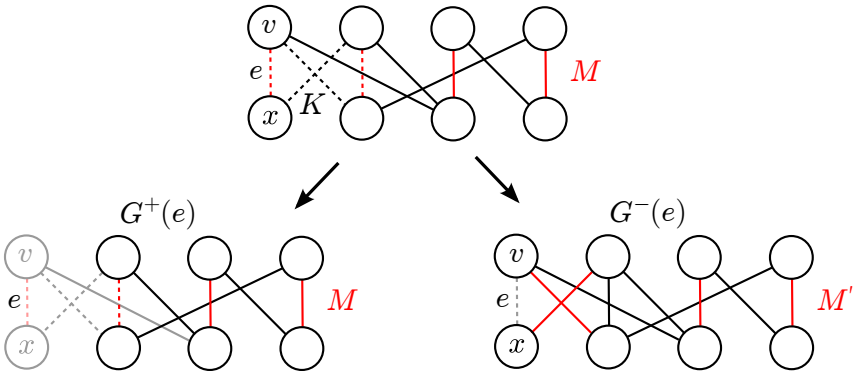


Abbildung 3.2: Teilprobleme im Algorithmus von Uno [27]

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 helleren Farbwerten dargestellt.

Die Angaben in [27] in Bezug auf $G^+(e)$ sind widersprüchlich. Im dort angegebenen Pseudocode erfolgt der Aufruf auf dem Graphen $G^+(e)$ mit dem Matching M. Die Kante e soll einerseits in allen perfekten Matchings in $G^+(e)$ enthalten sein. Andererseits ist die Kante e nicht Teil des Graphen $G^+(e)$. In der dieser Arbeit beiliegenden Implementierung wird vermerkt, dass die Kante e fest zu allen weiteren Matchings in der Rekursion auf $G^+(e)$ gehört. Bei der Enumeration der Matchings werden diese vermerkten Kanten dann hinzugefügt. Auf diese Weise werden alle perfekten Matchings korrekt aufgezählt.

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.

Die Geschwindigkeit des Algorithmus wird auf zwei Arten gesteigert. Vor der Bestimmung eines Kreises werden alle Kanten entfernt, die zu keinem Kreis gehören. Diese lassen sich mit Hilfe einer Zerlegung des gerichteten Graphen in starke Zusammenhangskomponenten finden. Kanten, die zu keiner starken Zusammenhangskomponente gehören, werden gelöscht, da sie keinem Kreis angehören. Matchingkanten, die auf diese Weise gelöscht werden, werden entsprechend vermerkt, dass sie Teil aller weiteren Matchings in dieser Rekursion sind. Die andere Art betrifft die Wahl der Kante e. Wenn diese geschickt gewählt wird, wird eine Gesamtlaufzeit von $O(nN_m)$ bewiesen, wenn das erste Matching gegeben ist. Dabei ist n durch die Zahl der Knoten und N_m durch die Anzahl der perfekten Matchings definiert. Falls das erste Matching nicht gegeben ist, kann es in polynomieller Zeit gefunden werden. Damit fällt der Algorithmus in die Klasse der polynomial-total-time-Algorithmen. Details zur Wahl von e sowie der Pseudocode des Enumerationsalgorithmus finden sich in [27]. Der folgende Satz wurde vom Verfasser dieser Arbeit bewiesen.

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

Beweis. Im Verlauf des Algorithmus werden immer dann perfekte Matchings ausgegeben, wenn das Problem in zwei Teilprobleme zerlegt wird. Dies ist genau dann der Fall, wenn ein alternierender Kreis gefunden wird. Zu jeder Zeit liegen nicht mehr als m Rekursionen auf dem Stack, wobei m der Anzahl der Kanten von $G(y^*)$ entspricht, denn bei den Graphen der Teilprobleme wird mindestens eine Kante entfernt. Im schlimmsten Fall wird in keiner dieser Rekursionen ein Kreis gefunden. Dann allerdings beendet sich der Algorithmus, da es keine weiteren perfekten Matchings gibt. Jede einzelne der maximal m Rekursionen ist offensichtlich in polynomieller Zeit möglich. □

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 enumeriertes 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.

Problematisch ist, dass mit dieser Modifikation einige MaxWBM mehrfach aufgezählt werden, wie Abbildung 3.3 zu entnehmen ist. Die in blau dargestellten Knoten sind die Hilfsknoten aus V''. Es gilt $M \setminus E' = M' \setminus E'$, dargestellt durch die roten Kanten zwischen den schwarzen Knoten. Dieses Problem tritt genau dann auf, wenn der Kreis nur entlang solcher Matchingkanten verläuft, die zu Hilfsknoten inzident sind. Solche Kreise werden verboten. Wenn in der aktuellen Rekursion nur noch Matchingkanten vx mit $v \in V''$ existieren, existieren nur noch solche Kreise. In dem Fall kann die Rekursion abgebrochen werden. Dies lässt sich in Zeit $O(1)$ feststellen, wenn die Anzahl dieser Kanten in einer Variablen gespeichert wird. Ansonsten gibt es mindestens einen alternierenden Kreis mit einer Matchingkante, die zu einem Knoten $v \in V'$ inzident ist, denn alle Kanten des aktuell gegebenen Graphen gehören zu einem alternierenden Kreis. Wenn die Suche nach einem alternierenden Kreis K bei einem Knoten $v \in V'$ beginnt, ist die mit v verbundene Matchingkante Teil des Kreises. Das Matching M', das nach Austausch der Kanten entlang des Kreises berechnet wurde, unterscheidet sich damit bezüglich G' von dem vorherigen Matching M. Insgesamt ergibt sich daraus, dass kein Matching mehrmals enumeriert wird.

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

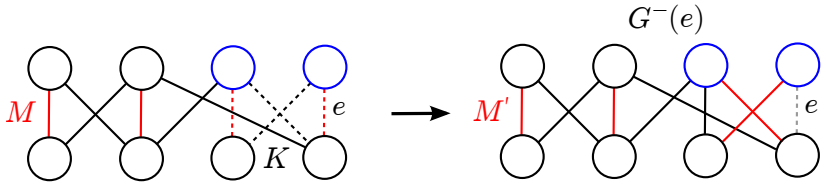


Abbildung 3.3: Identische Matchings in Bezug auf G'

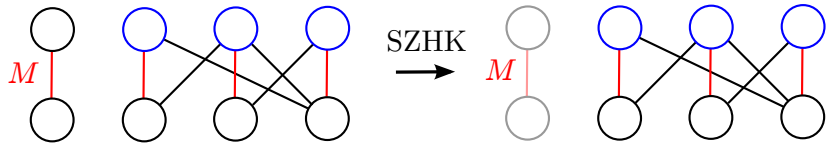


Abbildung 3.4: Abbruch der Rekursion, Matchingkanten sind nur noch zu Hilfsknoten inzident

und des Abbruchs, wenn nur noch zu Hilfsknoten inzidente Matchingkanten existieren, können die Kosten somit nicht mehr auf die Rekursionen verteilt werden, die wegen des Abbruchs nicht mehr stattfinden. In Abbildung 3.4 existieren nach der Entfernung der Knoten und Kanten, die zu keinem Kreis gehören, nur noch Matchingkanten, die mit Hilfsknoten verbunden sind. Es gibt folglich keine weitere Rekursion und insgesamt nur ein perfektes Matching in Bezug auf G'. Ein allgemeines Beispiel ist ein zulässiger Teilgraph $G(y^*) = (((\{v\} \cup V'') \cup (\{x\} \cup X), \{vx\} \cup V'' \times X)$ Dort existiert nur ein perfektes Matching in Bezug auf G', die Laufzeit beträgt dennoch $O(n^2)$, wobei n der Knotenanzahl entspricht.

Satz 3.7. *Der auf dem Algorithmus von Uno [27] basierende und in diesem Abschnitt vorgestellte Algorithmus zur Enumeration von MaxWBM ist ein polynomial-delay-Algorithmus. Die Laufzeit ist bei gegebenem ersten perfekten Matching M durch $O(n^2 N_m)$ beschränkt. Der zusätzliche Speicherverbrauch ist durch $O(m)$ beschränkt.*

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 $\varphi : \emptyset \to \emptyset$, 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

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

```
1: m ← SizeMCSTI(R, S) // Algorithmus 2.2
2: for all r ∈ RST_R mit I(r) < I(r̄) do
3:   for all s ∈ RST_S do
4:     if D(r, s) + D(r̄, s̄) = m // alle Werte für D wurden in Zeile 1 berechnet then
5:       for all Maximum CSTI φ1 von (r, s) // Wurzelknoten aufeinander abbilden do
6:         for all Maximum CSTI φ2 von (r̄, s̄) // Wurzelknoten aufeinander abb. do
7:           Verbinde die Isomorphismen φ1 und φ2 der gewurzelten Teilbäume zu einem
             Maximum CSTI φ von R und S.
8:           Gebe φ aus, falls φ bisher nicht gefunden wurde.
9:         end for
10:      end for
11:    end if
12:  end for
13: 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 φ hinzuzufügen, systematisch so zu durchlaufen, dass alle Maximum CSTI genau einmal gefunden werden. Die nötigen Formalisierungen werden im Folgenden beschrieben.

Zur Kantenmengen 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 $RST_R := \{R^u_v \mid (u, v) \in E'_R\}$ definiert, entsprechendes gilt für RST_S. Für einen gewurzelten Teilbaum $t = T^u_v$ mit $T \in \{R, S\}$ sei $\bar{t} := T^v_u$ 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 *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) $m = \text{GetD}(R^u_v, S^w_x) + \text{GetD}(R^v_u, S^x_w)$ gilt. Die Voraussetzung $I(r) <$

$I(\overline{r})$ wird in Abschnitt 3.3.2 besprochen. Anschließend werden alle Maximum CSTI von (r, s) enumeriert und diese nacheinander mit allen Maximum CSTI von $(\overline{r}, \overline{s})$ verbunden (Zeile 5 bis 7). Verbinden bedeutet dabei, dass in φ genau die Zuordnungen aus φ_1 und φ_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 $(\overline{r}, \overline{s})$ aus Algorithmus 3.1.

Sei $(r, s) = (R_v^u, S_x^w)$ 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 $\mathsf{M}_1, \ldots, \mathsf{M}_m$ zu zerlegen, in denen jeweils weitere Vorgaben für φ 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 $M_i = \{v_1 x_1^{(i)}, v_2 x_2^{(i)}, \ldots, v_k x_k^{(i)}\}$ das i-te enumerierte 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

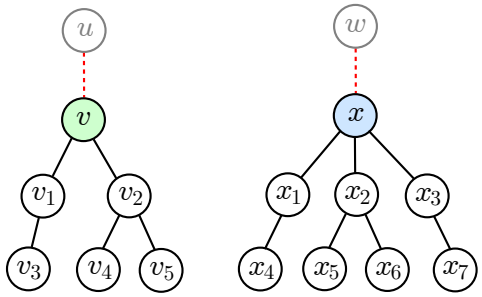


Abbildung 3.5: Beispiel zur Enumeration auf zwei gewurzelten Teilbäumen

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 φ in M_i. Sei M'_j die Menge aller Maximum CSTI auf dem Paar $(R^v_{v_j}, S^x_{x_j})$ von gewurzelten Teilbäumen. In M_i sind genau $\prod_{j=1,2,\ldots,k} |\mathsf{M}'_j|$ verschiedene Maximum CSTI für das Paar (R^u_v, S^x_w) von gewurzelten Teilbäumen enthalten. Das sind genau die Isomorphismen, die die Vorgaben von M_i an φ erfüllen. Diese lassen sich enumerieren, indem alle Kombinationen von Isomorphismen aus M'_1 bis M'_k miteinander und mit φ 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 $(R^v_{v_j}, S^x_{x_j})$ erfolgt allerdings nur dann, wenn beide Teilbäume $R^v_{v_j}$ und $S^x_{x_j}$ 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 $(R^v_{v_1}, S^x_{x_1})$ bis $(R^v_{v_k}, S^x_{x_k})$ 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 polynomiell 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 $(r, s) = (R^u_v, S^w_x)$ 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 $(R^v_{v_1}, S^x_{x_1})$ und M'_2 als die Menge aller Maximum CSTI auf $(R^v_{v_2}, S^x_{x_2})$. Alle Kombinationen von Isomorphismen aus diesen beiden Mengen werden nun miteinander und mit φ 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 $\{v_4x_5, v_5x_6\}$, das zur Festlegung von $\varphi(v_4) = x_5$ 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 φ. Dabei ergibt sich $\varphi(v_4) = x_6$ und $\varphi(v_5) = x_5$, die anderen Zuordnungen von φ 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 φ_1 bzw. φ_2 ausgehend von dem Paar $(r, s) = (R^u_v, S^w_x)$ bzw. $(\overline{r}, \overline{s})$, 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, φ_1) (Algorithmus 3.2) aufgerufen. In Zeile 6 wird entsprechend $\varphi(u) = w$ gesetzt und GetMaximumCSTI$(\overline{r}, \overline{s}, \varphi_2)$ 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^u_v, S^w_x) 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 φ entsprechend festgelegt. Ansonsten wurde

Eingabe: Referenzen auf R_v^u, S_x^w und φ
Ausgabe: 1, falls ein weiterer Isomorphismus aufgezählt werden konnte; 0, sonst

```
1: if f = false // Es wird nicht der erste MCSTI auf R_v^u, S_x^w berechnet then
2:    Berechne die nächste Kombination von MCSTI auf M'_1 bis M'_k
3:    if es gibt eine weitere Kombination then
4:       return 1
5:    end if
6: end if
7: if f = true // Bisher noch kein MCSTI auf R_v^u, S_x^w bestimmt then
8:    f ← false
9:    Berechne erstes MaxWBM und erweitere φ um die durch das Matching bestimmte
      Zuordnung der Kinder
10: else
11:   // Alle Kombinationen auf letztem Matching wurden aufgezählt
      Berechne nächstes MaxWBM, verändere φ entsprechend
12:   if es gab kein nächstes MaxWBM, da bereits alle enumeriert wurden then
13:      f ← true // Es gibt keinen weiteren MCSTI auf R_v^u, S_x^w
14:      return 0
15:   end if
16: end if
17: Berechne die erste Kombination von MCSTI auf M'_1 bis M'_k
18: return 1
```

Algorithmus 3.2: GetMaximumCSTI(R_v^u, S_x^w, φ)

in Zeile 2 keine weitere Kombination gefunden. Das bedeutet, auf dem Paar $(R_{v_1}^v, S_{x_1}^x)$ konnte kein weiterer Isomorphismus generiert werden. Einer von zwei möglichen Gründen ist, dass v_1 oder x_1 ein Blatt ist. Der andere mögliche Grund ist, dass der rekursive Aufruf von GetMaximumCSTI($R_{v_1}^v, S_{x_1}^x, \varphi$), 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 φ 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 $(R_{v_i}^v, S_{x_i}^x)$, in denen v_i und x_i keine Blätter sind, und die den Mengen M'_1 bis M'_k entsprechen.

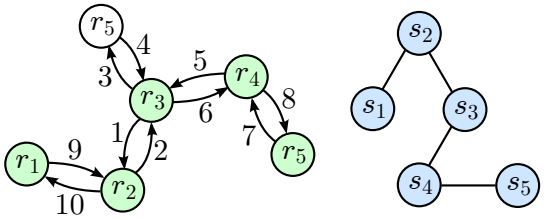


Abbildung 3.6: Festlegung der Ordnungsfunktion I in Algorithmus 3.1

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 $(\overline{r}, \overline{s})$ enumeriert und mit den Maximum CSTI von (r, s) (Zeile 5) verbunden. Ohne die Einschränkung $I(r) < I(\overline{r})$ in Zeile 2 würde die Auswahl von $\overline{r}$ und $\overline{s}$ in Zeile 2 und 3 dazu führen, dass in Zeile 6 die Maximum CSTI von $(\overline{\overline{r}}, \overline{\overline{s}}) = (r, s)$ aufgezählt würden, so dass in dem Fall genau die gleichen Isomorphismen bestimmt werden, mit vertauschten Rollen von φ_1 und φ_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 $I(R^{r_i}_{r_j})$. Die Festlegung der Ordnungsfunktion I wird im nächsten Abschnitt behandelt. Wird im Algorithmus $(r, s) = (R^{r_4}_{r_3}, S^{s_4}_{s_3})$ 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 $\varphi(r_i) = s_i$ für alle $i \in \{1, \ldots, 5\}$. Die Wahl von $(r, s) = (R^{r_1}_{r_2}, S^{s_1}_{s_2})$ führt zu genau dem gleichen Isomorphismus. Gleiches gilt für $(r, s) = (R^{r_3}_{r_2}, S^{s_3}_{s_2})$ und $(r, s) = (R^{r_5}_{r_4}, S^{s_5}_{s_4})$. 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 φ und $\mathsf{R} := \{r_{i_1}, \ldots, r_{i_n}\}$ die Menge der gewurzelten Teilbäume, durch die φ in Algorithmus 3.1, zusammen mit jeweils einem gewurzelten Baum aus S, enumeriert wird. Da die Ordnungsfunktion I bijektiv ist, gibt es genau ein $r \in \mathsf{R}$ mit $I(r) = \min\{I(r') \mid r' \in \mathsf{R}\}$. 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

M das zuletzt ausgewählte Matching in irgendeiner Rekursionsebene und $V' := \{v' \in V \mid v'$ ist durch M gematched$\}$. Falls $I(r) > I(r')$ für ein $r' \in \{R^v_{v'}, R^{v'}_v \mid v' \in V'\}$ 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 $I(R^{r_i}_{r_j}) = I(R^{r_j}_{r_i}) \pm 1$ für benachbarte Knoten r_i, r_j ist, wie dies in Abbildung 3.6 dargestellt ist, ist es ausreichend, $I(r) < I(r')$ für $r' \in \{R^v_{v'} \mid v' \in V'\}$ 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 $r = R^{r_3}_{r_2}$ beginnend enumeriert wird. Falls $r \neq R^{r_3}_{r_2}$ 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 $r = R^u_v$ 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 φ abgebildet werden, enumeriert wurden. Folglich kann die Kante uv aus R gelöscht werden. Der so modifizierte Graph wird im folgenden mit R' bezeichnet. Die Löschung 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 φ 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 Kantenlöschung 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.

Nach Satz 2.2.4 liegt die Laufzeit des Algorithmus von Edmonds in der Größenordnung $O(|E_R|^5)$ für $|E_R| \approx |E_S|$. Die Gesamtrechenzeit der Aufrufe des Algorithmus von Edmonds wird minimiert, wenn die gewurzelten Bäume r in einer Reihenfolge gewählt werden, die zu möglichst wenigen Aufrufen des Algorithmus von Edmonds führen. Sei Z im Folgenden irgendeine größte Zusammenhangskomponente aus R'. Es ist klar, dass die Größe eines Maximum CSTI von R' und S nicht größer als $|Z|$ sein kann. Es bietet sich somit an, r aus Z zu wählen. Weiterhin ist es hilfreich, r so zu wählen, dass Z nach der Kantenlöschung in zwei Zusammenhangskomponenten zerfällt, die möglichst gleich groß sind. Dieses Vorgehen ist natürlich nur eine Heuristik. Beispielsweise sind in dem Baum R aus Abbildung 3.1 alle Kanten gleichwertig, eine gezielte Auswahl bringt keinen Vorteil gegenüber einer zufälligen Auswahl. In Abbildung 3.7 ist ein Beispiel dargestellt, wo diese Heuristik sogar nachteilig wirkt. Die farbig dargestellten Knoten gehören zu jedem Maximum CSTI. Die vorgestellte Heuristik würde zuerst einen gewurzelten Baum r entlang einer der in rot dargestellten Kanten wählen. Mit dieser Auswahl werden allerdings nur die Hälfte aller Maximum CSTI enumeriert und der Algorithmus von Edmonds insgesamt drei mal aufgerufen. Wenn die Enumeration entlang einer Kante zwischen zwei grünen Knoten startet, führt dies nur zu einem weiteren Aufruf des Algorithmus von Edmonds. Eine zufällige Auswahl ist in diesem Beispiel in $\frac{1}{3}$ aller Fälle besser.

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

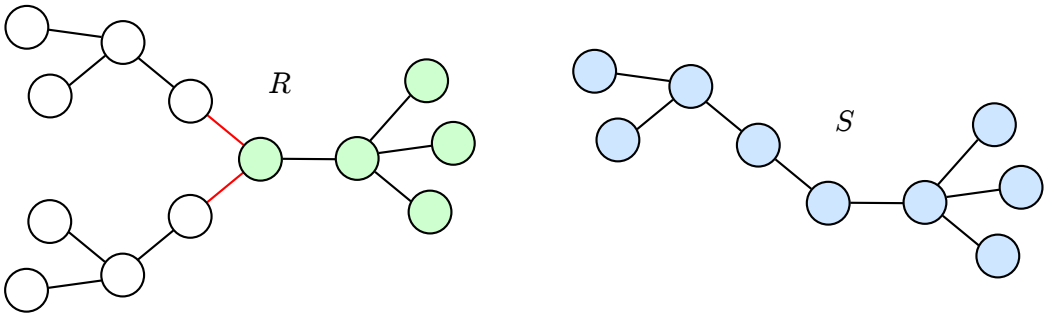


Abbildung 3.7: Wahl des gewurzelten Teilbaums r in Algorithmus 3.1 in der Variante „Kanten löschen“

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 Löschung 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.

Das Verfahren mit Kantenlöschungen ist zu den Varianten und den Bezeichnern nur bedingt kompatibel. Die dabei auftretenden Probleme werden in den entsprechenden Abschnitten besprochen. Ein wesentlicher Vorteil ist allerdings, dass der Enumerationsalgorithmus 3.1 mit diesem Verfahren in die Klasse der polynomial-delay-Algorithmen fällt.

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 Kantenlöschung 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 polynomieller Zeit bestimmt werden. Der Algorithmus von Edmonds benötigt ebenfalls nur polynomiell 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-*

space-Algorithmus. 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 $(R^{r_3}_{r_2}, S^{s_3}_{s_2})$ 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.

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

1: **for all** $r \in RST_R$ mit $I(r) < I(\overline{r})$ **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$(\overline{r}, \overline{s}, \varphi_2) = 1$ **do**
5: Verbinde die Isomorphismen φ_1 und φ_2 der gewurzelten Teilbäume zu einem maximalen CSTI φ von R und S.
6: Gebe φ aus.
7: **end while**
8: **end while**
9: **end for**
10: **end for**

Algorithmus 3.3: EnumMaximalCSTI(R, S)

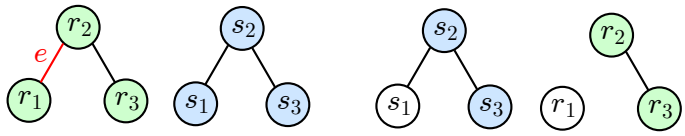


Abbildung 3.8: Kantenlöschung in R führt zu nicht maximalen Isomorphismen

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 φ 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 φ_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 polynomiell 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 aufgezählt (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 φ_1 so groß ist, dass es mindestens einen Isomorphismus φ_2 auf den Teilbäumen $(\overline{r}, \overline{s})$ gibt, so dass $|\varphi_1| + |\varphi_2| \geq m$ ist. In Zeile 8 werden dann alle Isomorphismen φ_2 aufgezählt, 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 ∈ RST_R mit I(r) < I(r̄) do
5:   for all s ∈ RST_S do
6:     if D(r, s) + D(r̄, s̄) ≥ m then
7:       for all Maximalen CSTI φ1 von (r, s) mit |φ1| ≥ m − D(r̄, s̄) do
8:         for all Maximalen CSTI φ2 von (r̄, s̄) mit |φ2| ≥ m − |φ1| do
9:           Verbinde die Isomorphismen φ1 und φ2 der gewurzelten Teilbäume zu einem
             maximalen CSTI φ von R und S.
10:          Gebe φ aus, falls φ 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 (Zeile 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[2] p festzuhalten, wie groß der aktuell generierte Isomorphismus höchstmöglich 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 φ 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.

[2]Es gibt je eine globale Variable für φ_1 und φ_2.

Sei $(r, s) = (R_v^u, S_x^w)$ 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 $M = \{v_1x_1, v_2x_2, \ldots, v_kx_k\}$ das letzte enumerierte Matching und M' das vorherige Matching. Falls M das erste Matching ist, gelte $M' = \emptyset$. Zu p wird der Wert $p(M) := \sum_{v_jx_j \in M} D(R_{v_j}^v, S_{x_j}^x)$ addiert, anschließend wird p um $p(M')$ verringert. Diese Änderungen an p haben zur Folge, dass p dann der höchstmöglichen 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 $(R_{v_j}^v, S_{x_j}^x)$ 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 $(R_{v_j}^v, S_{x_j}^x)$ für ein j berechnet wurde, sich der Wert in der globalen Variablen 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 φ_1 bzw. φ_2 bezogen auf Zeile 7 bzw. 8 in Algorithmus 3.4. Die Aufzählung aller Kombinationen von Isomorphismen auf den gewurzelten Teilbäumen $(R_{v_j}^v, S_{x_j}^x)$ erfolgt wie in Abschnitt 3.3.1 beschrieben. Der Wert p entspricht dabei während der gesamtem Enumeration der Größe des Isomorphismus φ_1 bzw. φ_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 φ_1 bzw. φ_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 $(r, s) = (R_v^u, S_x^w)$ gilt dort $D(R_v^u, S_x^w) = 6$. 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

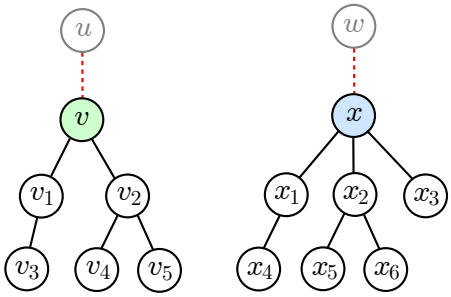


Abbildung 3.9: Beispiel zur Aktualisierung von p

nach der obigen Beschreibung um $D(R_v^u, S_x^w) - 1$ verringert, so dass $p = 1$ gilt. Sei $M_1 = \{v_1x_1, v_2x_2\}$ das erste enumerierte Matching. Für dieses wird p um $p(M_1) = D(R_{v_1}^v, S_{x_1}^x) + D(R_{v_2}^v, S_{x_2}^x) = 2 + 3$ 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 $(R_{v_1}^v, S_{x_1}^x)$ und $(R_{v_2}^v, S_{x_2}^x)$ 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 $p(M_2) = D(R_{v_1}^v, S_{x_1}^x) + D(R_{v_2}^v, S_{x_3}^x) = 2 + 1 = 3$ erhöht, so dass anschließend $p = 4$ gilt. Das entspricht genau der höchstmöglichen Größe, auf die der Isomorphismus φ 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 verringt und um $D(R_v^u, S_x^w) - 1 = 5$ 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 verworfener 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 $O((|V_R| + |V_S|) \cdot |V_R| \cdot |V_S|)$ 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

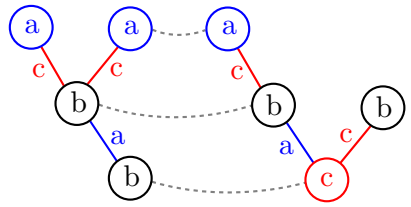
Der Benutzer könnte eine Anforderung stellen, dass Knoten nur dann aufeinander abgebildet werden, wenn der Knotengrad übereinstimmt oder einem bestimmten Wert oder Wertebereich entspricht. Entsprechend werden dann nur diese Matchings berücksichtigt. Die anderen werden verworfen, ähnlich dem Vorgehen bei den Isomorphismen mit Mindestgröße.

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 Matchings enumeriert 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 beschrieben, dass Knoten oder Kanten auch gewisse Bedeutungen haben können. Beispielsweise gilt das für die in der Einleitung erwähnten Feature Trees



(a) Zwei gelabelte Bäume

$w(\sigma_1, \sigma_2)$	a	b	c
a	2	0	-1
b	0	3	-1
c	-1	-1	4

(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 $l_1 : V_R \cup E_R \to \Sigma$ und $l_2 : V_R \cup E_R \to \Sigma$ Labelfunktionen.
Wenn R' und S' isomorph bezüglich l_1 und l_2 sind, wird der zugehörige Isomorphismus $\varphi : V'_R \to V'_S$ als *Common Labeled Subtree Isomorphismus* (CLSTI) bezeichnet.
Falls φ größtmöglich ist, wird φ *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 $l_1 : V_R \cup E_R \to \Sigma$ und $l_2 : V_R \cup E_R \to \Sigma$ Labelfunktionen und $w : \Sigma \times \Sigma \to \mathbb{Q}$ eine symmetrische Gewichtsfunktion.
Wenn R' und S' isomorph sind[3], wird der zugehörige Isomorphismus $\varphi : V'_R \to V'_S$ 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 φ *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.

[3] 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 Molekül. 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 entsprechendes Fragments entsprechen. Übertragen auf CWSTI stehen die Label für die Anzahl dieser Atome, mit $\Sigma \subset \mathbb{N}$. 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

Die Enumeration von CWSTI wurde im Rahmen dieser Arbeit durch Modifikationen am Algorithmus von Edmonds, sowie den Enumerationsalgorithmen aus Abschnitt 3.3 und 3.4 realisiert. Diese Modifikationen werden im Folgenden, getrennt nach Bezeichnern für Knoten und Kanten, beschrieben. Beide Modifikationen können einzeln oder auch zusammen hinzugenommen werden.

Knotenbezeichner

Seien R und S zwei Bäume, sowie φ 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 dargestellen Zahlen repräsentieren den Wert $w(l_1(r_i), l_2(s_i))$. So ist der Isomorphismus φ_1, definiert durch $\varphi_1(r_2) = s_2$, ein nicht-maximaler MCWSTI. Das gleiche gilt für φ_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

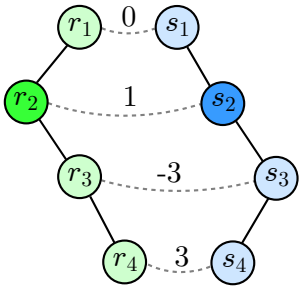


Abbildung 3.11: Der Isomorphismus $\varphi(r_2) = s_2$ ist ein nicht-maximaler Maximum CWSTI.

zum einen, dass MCWSTI φ 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 φ mit $|\varphi| = 1$ erweitert werden. Zum anderen führen nicht-maximale MCWSTI in Verbindung mit Kantenlöschungen (vgl. Abschnitt 3.3.2) dazu, dass diese mehrfach gefunden werden können. Beispielsweise wird φ_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 φ 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 $(r, s) = (R_v^u, S_x^w)$ 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 $w(l_1(uv), l_2(wx))$ hinzuaddiert. 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 $D(R_{v_i}^v, S_{x_j}^x)$ enthalten, so dass an dieser Stelle dann nichts addiert werden muss.

Zu beachten ist, dass in der Summe $D(r,s)+D(\overline{r},\overline{s})$ das Gewicht $w(l_1(uv),l_2(wx))$ zwei mal 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 Kantenlöschungen 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 $(r,s)=(R_v^u,S_x^w)$ 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 $l_1(u)\neq l_2(w)\vee l_1(v)\neq l_2(x)\vee l_1(uv)\neq l_2(wx)$, wird $D(R_v^u,S_x^w)$ auf -1 gesetzt. Dieser Wert signalisiert, dass ein CLSTI φ 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_ix_j aus dem bipartiten Graphen entfernt, falls $D(R_{v_i}^v,S_{x_j}^x)=-1$ ist, denn es gilt dann $l_1(v_i)\neq l_2(x_j)$ oder $l_1(vv_i)\neq l_2(xx_j)$. 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-

ständigen bipartiten Graphen wird von Uno in [27] beschrieben. Dabei handelt es sich um ein rekursives Verfahren, dass dem Verfahren aus Abschnitt 3.2.2 nicht unähnlich ist. Auch dort erfolgt eine Zerlegung in jeweils zwei Teilprobleme. Damit sind alle Voraussetzungen gegeben, um MCLSTI und maximale CLSTI, nach Wunsch mit Mindestgröße, aufzuzählen.

Laufzeit und Speicherverbrauch

Die Laufzeitschranke zur Enumeration von Maximum CLSTI stimmt mit der Laufzeit zur Enumeration von Maximum CSTI überein. Der zulässige Teilgraph $G(y^*)$ lässt sich auch aus einem nicht vollständigen bipartiten Graphen mit Gewichtsfunktion konstruieren. Die nötigen Änderungen zur Bestimmung von D verändern die Zeitschranke nicht. Somit bleibt die Zeitschranke insgesamt identisch. Für die Platzschranke gilt die gleiche Aussage.

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

In diesem Abschnitt wird ein im Rahmen dieser Arbeit entwickelter Algorithmus zur Enumeration von Common Subtrees (vgl. Definition 2.9) vorgestellt. Anhand Abbildung 2.1 wurde gezeigt, wie aus einem CSTI ein CST gewonnen werden kann. Die naheliegende Idee, maximale bzw. Maximum CSTI aufzuzählen und die zugehörigen maximalen bzw. Maximum CST auszugeben, ist nicht besonders zielführend. Denn zu einem CST kann es, wie aus Abschnitt 2.1 bekannt, mehrere CSTI geben. Dies würde dazu führen, dass gleiche CST mehrfach ausgegeben werden. Analog zur Unterdrückung der Ausgabe identischer CSTI aus Abschnitt 3.3.2 könnte möglicherweise eine Ordnung auf den generierten CST definiert werden, so dass dann jeder CST nur genau einmal ausgegeben wird. Problematisch ist, dass zu einem CST mehr als polynomiell viele CSTI existieren können, wie Abbildung 3.1 zu entnehmen ist. Wegen der zu erwartenden hohen Laufzeit wurde diese Idee verworfen.

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

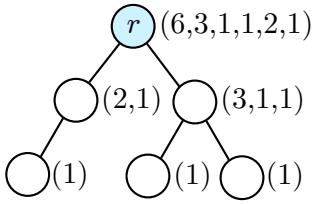


Abbildung 3.12: Baumkanonisierung

Paaren von gewurzelten Teilbäumen enumeriert. Anschließend werden die CST miteinander verbunden. Falls ein CST zum ersten Mal auf diese Art berechnet wurde, wird er ausgegeben und gespeichert. Die Speicherung erfolgt, um zu erkennen, ob ein CST bereits einmal berechnet wurde. In Abschnitt 3.6.1 wird eine Kanonisierung von gewurzelten Bäumen beschrieben. Diese erlaubt eine effiziente Speicherung und einen schnellen Isomorphietest, wie in Abschnitt 3.6.2 beschrieben wird. Im folgenden Abschnitt 3.6.3 wird das genaue Vorgehen, Maximum CST auf Paaren von gewurzelten Teilbäumen zu enumerieren, beschrieben. Wie daraus Maximum CST für die Bäume der Eingabe berechnet werden und identische Maximum CST nur einmal ausgegeben werden, wird in Abschnitt 3.6.4 beschrieben. Darauf folgt in Abschnitt 3.6.5 die Enumeration von maximalen CST und maximalen CST mit Mindestgröße. Laufzeit- und Speicherplatzanalysen erfolgen in Abschnitt 3.6.6. Abschnitt 3.6.7 beschäftigt sich mit einer Variante, in der statt Common Subtrees Tupel von Teilbäumen der gegebenen Bäume enumeriert werden, auf denen ein Isomorphismus existiert. Abschließend werden in Abschnitt 3.6.8 einige Beschleunigungstechniken für die Enumeration von maximalen CST und den Tupeln von Teilbäumen aus Abschnitt 3.6.7 vorgestellt.

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[4] lexikographischer Ordnung bezogen auf ihre Baumkanonisierung sortiert.

Zu einer Kanonisierung K sei $T(K) := T$ ein gewurzelter Baum T mit $K(T) = K$. In Abbildung 3.12 ist ein gewurzelter Baum und die jeweilige Kanonisierung für alle Knoten angegeben. Die kanonische Darstellung erlaubt einen schnellen und einfachen Isomorphietest für gewurzelte Bäume. Die folgende Aussage ist [28] entnommen.

[4]Valiente definiert die Ordnung als nicht absteigend [28].

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.*

Zu beachten ist, dass der Beweis zu Satz 3.13 für eine nicht absteigende Ordnung angegeben ist [28]. Der Beweis für eine nicht aufsteigende Ordnung verläuft analog. Dass im Rahmen dieser Arbeit eine nicht aufsteigende Ordnung gewählt wurde, hat technische Implementierungsgründe. So fügt sich die erste Zahl, die der Baumgröße entspricht, lexikographisch in die nicht aufsteigende Ordnung ein und erlaubt somit eine einfachere Sortierung. Zum Zeit- und Platzbedarf der Bestimmung von $K(T)$ für einen gewurzelten Baum T gilt nach [28] Folgendes.

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_v^u, 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_v^u, S_x^w
Ausgabe: Verweis auf $B(R_v^u, S_x^w)$

1: **if** $B(R_v^u, S_x^w) = \emptyset$ // $B(R_v^u, S_x^w)$ noch nicht berechnet **then**
2: **if** R_v^u oder S_x^w ist ein Blatt **then**
3: $B(R_v^u, S_x^w) \leftarrow \{(1)\}$
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** $(K_1, \ldots, K_k) \in \mathrm{GetB}(R_{v_1}^v, S_{x_1}^x) \times \ldots \times \mathrm{GetB}(R_{v_k}^v, S_{x_k}^x)$ **do**
7: Sortiere die Kanonisierungen K_1 bis K_k lexikographisch nicht aufsteigend. Die sortierte Reihenfolge sei K_1' bis K_k'.
8: $K \leftarrow (\mathrm{GetD}(R_v^u, S_x^w)); K_1'; \ldots; K_k'$
9: Füge K in $B(R_v^u, S_x^w)$ ein, falls noch nicht vorhanden.
10: **end for**
11: **end for**
12: **end if**
13: **end if**
14: **return** Verweis auf $B(R_v^u, S_x^w)$

Algorithmus 3.5: $\mathrm{GetB}(R_v^u, S_x^w)$

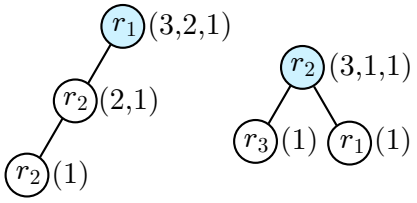


Abbildung 3.13: Die Baumkanonisierung hängt von der Wahl der Wurzel ab.

Unterschied ist, dass nicht Maximum CSTI von (r, s) bzw. $(\overline{r}, \overline{s})$ miteinander verbunden werden, sondern die in $B(r, s)$ bzw. $B(\overline{r}, \overline{s})$ 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.

Beim Verbinden der Maximum CST aus $B(r, s)$ und $B(\overline{r}, \overline{s})$ ist darauf zu achten, dass die Kanonisierung eines Baumes von der Wahl der Wurzel abhängt, wie Abbildung 3.13 zu entnehmen ist. Die zugehörigen Kanonisierungen nach Valiente [28] sind neben den Knoten angegeben. Nach Jovanović und Danilović [16] hat ein Baum ein *Zentrum*, dass aus genau einem oder zwei Knoten besteht.

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

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

Das Zentrum eines Baumes T kann in Zeit $O(|T|)$ gefunden werden. Beispielsweise lässt sich mit dem wohlbekannten Algorithmus Tiefensuche die Exzentrizität bestimmen. Daraus kann dann recht einfach das Zentrum bestimmt werden. Falls das Zentrum aus genau einem Knoten besteht, wird dieser als Wurzel definiert. Von dieser Wurzel ausgehend wird dann die Kanonisierung berechnet. Falls es zwei zentrale Knoten gibt, werden die Kanonisierungen für diese beiden Knoten als Wurzel berechnet. Als Wurzel wird dann derjenige Knoten behalten, der zu einer lexikographisch kleineren Kanonisierung führt. Bezogen auf Abbildung 3.13 ist r_2 Zentrum des dort dargestellten Baumes.

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 $B(R_v^u, S_x^w)$. Dadurch, dass maximale Matchings enumeriert werden, können in $B(R_v^u, S_x^w)$ Kanonisierungen unterschiedlicher Länge gespeichert werden. Da nur maximale CST enumeriert werden sollen, dürfen Kanonisierungen K nur dann eingefügt werden, wenn es keine Kanonisierung K' in $B(R_v^u, S_x^w)$ 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 $B(R_v^u, S_x^w)$ vorhanden sein und kein solches K' existieren, kann K in $B(R_v^u, S_x^w)$ 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

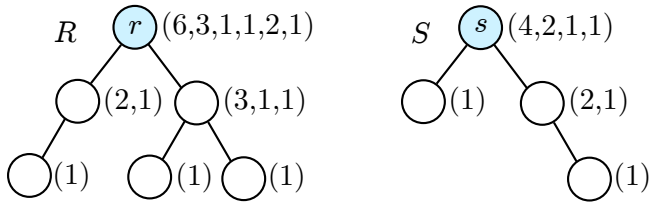


Abbildung 3.14: Der Baum S ist in R enthalten.

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 nächstkleinere Kanonisierung gewählt. Das Vorgehen ist analog zu den genannten Abschnitten. Wenn für ein i die lexikographisch nächstkleinere Kanonisierung K_i bestimmt wird, und diese zusammen mit den anderen bestimmten Kanonisierungen nicht zur Mindestgröße m führen kann, können alle lexikographisch kleineren Kanonisierungen aus $B(R^v_{v_i}, S^x_{x_i})$ übersprungen werden, da diese höchsten genau so groß wie K_i sind. Für K_i 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 $T(K_1)$ isomorph 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 isomorph 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 Isomorphismus $\varphi : R' \to S$ als *Subtree Isomorphismus* (STI) bezeichnet.

Das *Subtree Isomorphismus Problem* stellt die Frage, ob ein Teilbaum von R in S enthalten ist [21]. Dieses Problem wurde 1968 unabhängig von Edmonds und Matula in Polynomialzeit gelöst [21]. Seitdem gab es einige Verbesserungen in Bezug auf die Laufzeit. 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$(K, B(R^u_v, S^w_x))$“ (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

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**
9: Füge K in B ein.
10: **for all** Kanonisierungen K' aus B mit $|K'| < |K|$ **do**
11: **if** $T(K')$ ist isomorph zu einem Teilbaum von $T(K)$ **then**
12: Entferne K' aus B.
13: **end if**
14: **end for**
15: **return**

Algorithmus 3.7: InsertKintoB(K, B)

zu dem anderen Baum. Die Ausgabe der maximalen CST mit Mindestgröße kann, im Gegensatz zu den Maximum CST, erst ganz am Ende erfolgen, da aus dem Binärbaum B Kanonisierungen gelöscht werden können.

3.6.6 Laufzeit und Speicherverbrauch

In Zeile 5 von Algorithmus 3.5 werden alle MaxWBM bzw. maximalen Matchings aufgezählt. Mit den Bäumen aus Abbildung 3.1 ist diese Anzahl nicht durch ein Polynom beschränkt. Andererseits existiert für isomorphe Bäume R und S nur ein maximaler CST. Deshalb sind die hier vorgestellten Algorithmus zur Aufzählung von maximalen und Maximum CST keine polynomial-total-time-Algorithmen.

Der nötige Speicherverbrauch ist ebenfalls nicht durch ein Polynom beschränkt. Wenn in Algorithmus 3.5, Zeile 6, die in den Binärbäumen GetB$(R^v_{v_1}, S^x_{x_1})$ bis GetB$(R^v_{v_k}, S^x_{x_k})$ gespeicherten Kanonisierungen jeweils alle verschieden sind, gibt es mehr als polynomiell viele verschiedene Kanonisierungen K, die in den Binärbaum $B(R^u_v, S^w_x)$ 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 ← ∅
2: if SizeMCSTI(R, S) < m then
3:   return Es gibt keinen CST mit einer Mindestgröße von m.
4: end if
5: for all r ∈ RST_R mit I(r) < I(r̄) do
6:   for all s ∈ RST_S do
7:     if D(r, s) + D(r̄, s̄) ≥ m then
8:       for all Kanonisierungen K_1 aus GetB(r, s) mit |K_1| ≥ m − D(r̄, s̄) do
9:         for all Kanonisierungen K_2 aus GetB(r̄, s̄) mit |K_2| ≥ m − |K_1| do
10:          K ← 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 $V(R'_1) \neq V(R'_2) \vee V(S'_1) \neq V(S'_2)$. 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-

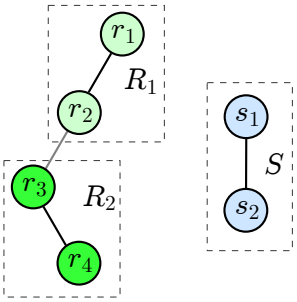


Abbildung 3.15: Enumeration von Teilbäumen, auf denen ein Isomorphismus existiert

phismen für das Tupel (V_R, V_S) sichergestellt werden. Dies kann mit einem Teilmengentest überprüft werden. Wenn bereits ein anderes Tupel (V'_R, V'_S) mit $V_R \subsetneq V'_R$ und $V_S \subsetneq V'_S$ existiert, so wird (V_R, V_S) nicht eingefügt, da (V_R, V_S) nicht maximal ist. Wenn im Laufe der Enumeration ein anderes Tupel (V'_R, V'_S) mit $V_R \subsetneq V'_R$ und $V_S \subsetneq V'_S$ gefunden wird, ist das Tupel (V_R, V_S) nicht gültig und muss nachträglich gelöscht werden. Weil die Knoten nach ihren Nummern sortiert gespeichert sind, gelingt der Teilmengentest in Linearzeit. Einen entsprechenden Algorithmus anzugeben ist nicht schwierig. Diese Teilmengentests erfolgen nur zwischen Tupeln unterschiedlicher Größe.

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 polynomiell 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 dynamisches 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 $S^x_{x'}$ und $S^x_{x''}$ 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 $B(R^v_{v'}, S^x_{x'})$ und $B(R^v_{v'}, S^x_{x''})$ 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 $R^v_{v'}$ und $R^v_{v''}$ 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

In diesem Abschnitt wird beschrieben, wie Algorithmus 3.3 (EnumMaximalCSTI) parallelisiert werden kann. Dabei stehen die praktischen Laufzeiten auf handelsüblichen Desktop-PCs im Vordergrund. Die Enumeration maximaler CSTI mit Mindestwert und Maximum CSTI lässt sich analog parallelisieren. Dabei muss dann zusätzlich die Mindestgröße der CSTI sichergestellt werden bzw. es müssen Maximum anstelle von maximalen Matchings enumeriert werden.

Erste Testläufe des Programms ergaben, dass die Gesamtzeit zur Enumeration vor allem von der Anzahl der Isomorphismen abhängt und nur zu einem geringen Anteil von der Vorabberechnung des Algorithmus von Edmonds. In Tabelle 4.2 sind diese Zeiten für Bäume verschiedener Größen angegeben. Falls nur maximale CSTI enumeriert werden sollen, wird der Algorithmus von Edmonds gar nicht benötigt, vgl. Algorithmus 3.3.

Die Idee zur Parallelisierung besteht darin, die Schleifen in Zeile 1 und 2 aus Algorithmus 3.3 aufzulösen und stattdessen eine Queue zu generieren, in der alle Paare (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 φ_1 von (r, s) bzw. φ_2 von $(\overline{r}, \overline{s})$ 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

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 $I(r) < I(\overline{r})$ 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$(\overline{r}, \overline{s}, \varphi_2^{(i)}) = 1$ **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)

als unproblematisch nachgewiesen werden. Diese haben keinen messbare Auswirkung auf die Laufzeit. Sehr problematisch ist der Unterschied zwischen Release- und Debugmodus. Während im Debugmodus relativ gesehen sehr gute Resultate erzielt werden, ist die Berechnungsgeschwindigkeit im Releasemodus nur teilweise linear mit der Anzahl der Prozesse skalierend. Die genaue Ursache dafür konnte während der Arbeit nicht ermittelt werden. Aus diesem Grund wurde die Parallelisierung der Algorithmen nicht weiter verfolgt. In Abschnitt 4.1.1 erfolgen Laufzeitvergleiche unter Nebenläufigkeit.

Kapitel 4

Experimentelle Resultate

In diesem Kapitel werden die experimentellen Resultate des Programms für verschiedene Eingaben und die verschiedenen in Kapitel 3 vorgestellten Verfahren analysiert. Sämtliche Berechnungen in diesem Kapitel wurden auf einer Intel Core I5 3570K 4-Kern CPU durchgeführt. Als Eingabe für das Programm dient die Standardkonfigurationsdatei, die im Anhang A.2 beschrieben ist. In dieser ist die Speicherung von bis zu zehntausend MaxWBM pro Paar von gewurzelten Teilbäumen aktiviert. Die Kanten aus R werden möglichst mittig gewählt. Das Standardverfahren zur Vermeidung gleicher CSTI ist die Erkennung über die Ordnungsfunktion. Der Algorithmus von Uno, sowie Knoten- und Kantenbezeichner sind deaktiviert. Davon abweichende Werte, soweit nicht offensichtlich, sind an entsprechender Stelle angegeben. Falls in den Tabellen unter Baumkanten nur eine Zahl steht, haben beide Bäume diese Anzahl an Kanten. Bei den Bäumen handelt es sich um zufällig mit OGDF [23] erzeugte Bäume, wenn nichts anderes angegeben ist. Die Erzeugung beginnt mit einem Baum, der aus genau einem Knoten besteht. Dann erfolgt eine zufällige Auswahl eines Knotens. Dieser wird mit einem neu erzeugten Knoten durch eine Kante verbunden. Das Vorgehen wird so oft wiederholt, bis die gewünschte Größe erreicht ist. Bei Größenangaben ist T die Abkürzung für Tausend und M für Millionen. Die angegebenen Werte sind gerundet. In Abschnitt 4.1 wird die Enumeration von Common Subtree Isomorphismen untersucht. Abschnitte 4.2 und 4.3 evaluieren die Enumeration von Common Subtrees und den Teilbaumpaaren aus Abschnitt 3.6.7. In allen Abschnitten werden zunächst die verschiedenen Ergebnisse vorgestellt und im Anschluss daran erfolgt eine Analyse und Bewertung. Dabei wird Bezug auf die theoretischen Überlegungen genommen.

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.

Kanten	Zeit	CSTI	CSTI pro s	verworfen
25	0,19 s	2,51 M	13,3 M/s	4,1 %
30	1,36 s	17,00 M	12,6 M/s	2,2 %
35	8,06 s	90,00 M	11,2 M/s	3,5 %
40	71,50 s	831,00 M	11,6 M/s	2,8 %
45	544,00 s	6344,00 M	11,7 M/s	1,1 %

Tabelle 4.1: Maximale CSTI - Jeweils 10 Durchläufe

4.1.1 Ergebnisse der Programms

Maximale Common Subtree Isomorphismen

In Tabelle 4.1 ist das starke Wachstum der Anzahl der Lösungen zu beobachten. Nach Satz 3.9 ist die worst case Laufzeit quadratisch von der Größe eines MCSTI abhängig. Die praktische Laufzeit pro Isomorphismus sinkt jedoch nur langsam mit steigender Kantenzahl. Außerdem ist noch das Verhältnis der von der Ordnungsfunktion I verworfenen Matchings zur Gesamtzahl der berechneten Matchings angegeben (verworfen).

Maximum Common Subtree Isomorphismen

In Tabelle 4.2 sind die Laufzeiten zur Berechnung von Maximum CSTI dargestellt. Zur Enumeration der MaxWBM wurde der modifizierte Algorithmus von Uno genutzt. Die Tabelle ist um die Laufzeit des Algorithmus von Edmonds ergänzt. Dies schließt die Neuberechnungen bei den Kantenlöschungen mit ein. Der Anteil an der Gesamtlaufzeit ist nur bei kleinen Eingabegrößen signifikant. Insbesondere die nah beieinander liegenden Zeiten bei der Berechnung mit Hilfe der Ordnungsfunktion und durch Kantenlöschung sind zu beachten. Außerdem ist in der Spalte |MCSTI| die durchschnittliche Größe eines MCSTI angegeben.

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-

Kanten	Zeit	MCSTI	MCSTI pro s	Edmonds	verworfen	|MCSTI|
Vermeidung gleicher Isomorphismen mit Ordnungsfunktion						
40	0,23 s	1 M	3,55 M/s	121 ms	2,28 %	27,9
50	13,83 s	118 M	8,52 M/s	180 ms	0,01 %	34,1
60	13,55 s	68 M	5,00 M/s	271 ms	0,01 %	40,1
70	494,12 s	4412 M	8,93 M/s	372 ms	<0,001 %	45,7
Vermeidung gleicher Isomorphismen durch Kantenlöschung						
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

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

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' (Aufzählung aller maximalen Matchings, nur Maximum Matchings behalten) und

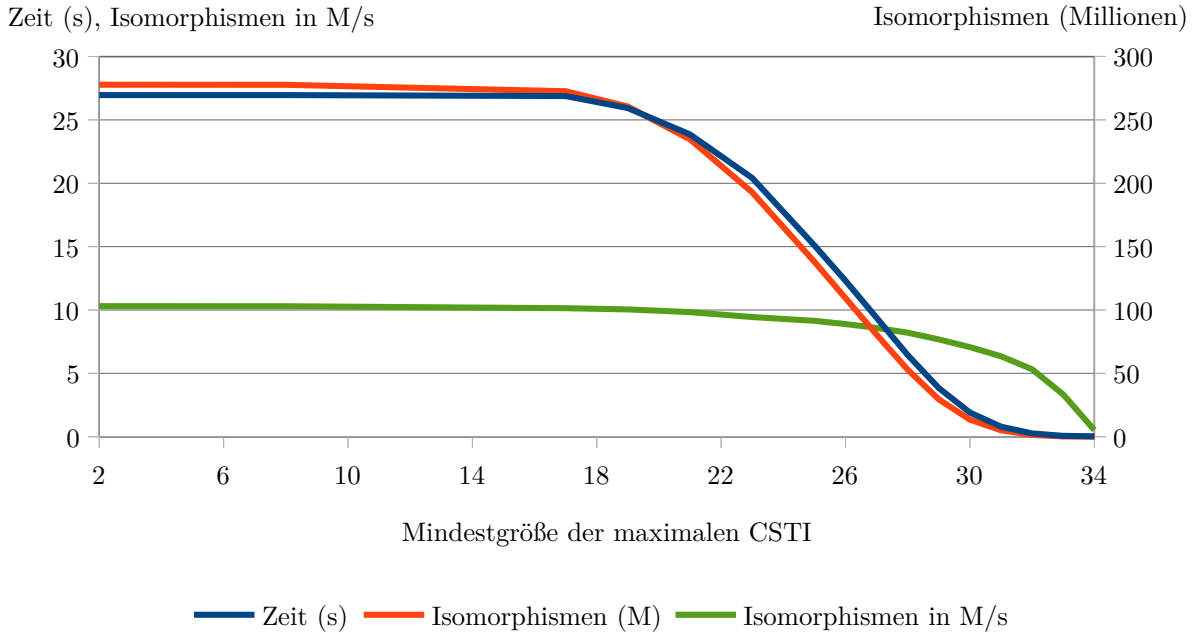


Abbildung 4.1: Anzahl der maximalen CSTI mit Mindestgröße (x-Achse) - Bäume mit 50 Kanten

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

Tabelle 4.4: MCSTI mit verschiedener Kantenauswahl - Jeweils 10 Durchläufe

MaxWBM (Modifizierter Algorithmus von Uno [27]) verglichen. Bei den MCWSTI sind 4 verschiedene Knoten- und Kantenbezeichner mit insgesamt 10 verschiedenen Gewichten vorhanden. Die Zeiten stellen die Enumerationszeit ohne den Algorithmus von Edmonds dar. Bei der Enumeration wurde die Speicherung von Matchings deaktiviert. Zusammen mit Bezeichnern und erhöhter Kantenzahl ist die Enumeration mit dem modifizierten Algorithmus von Uno besser, ansonsten die Enumeration mit den MaxWBM'. Die zufälligen Bäume mit 60 Kanten hatten weniger MCSTI als die Bäume mit 50 und 55 Kanten. Damit lässt sich der lokale Tiefpunkt der dunkelroten Kurve begründen.

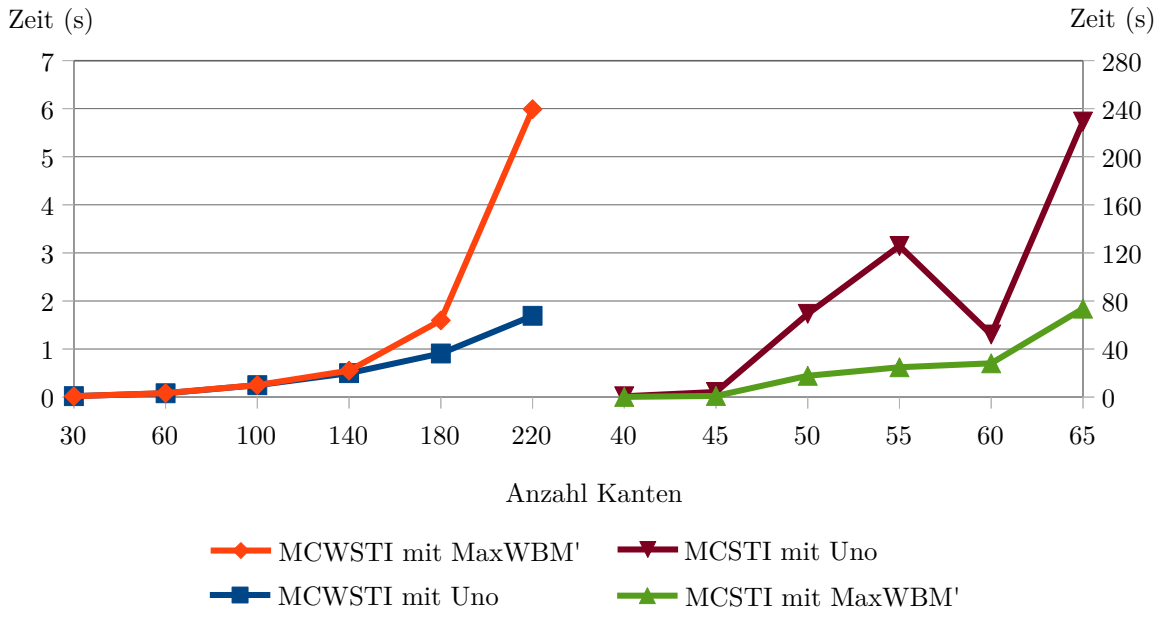


Abbildung 4.2: Enumerationszeiten für MCSTI und MCWSTI, jeweils 10 Durchläufe

Feature Trees

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 $Q(F,G) := \frac{\text{SizeMCSTI(F,G)}}{\max\{|V_F|,|E_F|\}}$ 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(.)C(.)==C(.)C(.)=C1(.)_C(.)=2CCCCC2(.)_N=1C(.)=C(.)C=CC1_O(.)_N(.)(.)_C(.)(.)_C(.)(.)_O=C(.)(.)_C(.)_C(.)_` (54695776)
- `O=C1N(.)C(.)==C(.)C(.)=C1(.)_C(.)=2CCCCC2(.)_N=1C(.)=C(.)C=CC1_O(.)_O(.)_N(.)(.)_C(.)(.)_C(.)(.)_O=C(.)(.)_C(.)_` (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

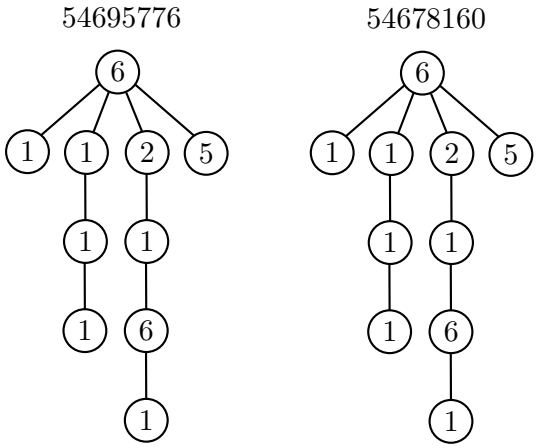


Abbildung 4.3: Feature Trees „54695776“ und „54678160“

	$\|R\| < \|S\|$		$\|S\| < \|R\|$	
Kanten	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

das dieser Arbeit beiliegende Programm nicht alle der gegebenen Informationen. Dazu wären weitergehende Modifikationen, nicht nur an der Berechnung der D-Werte, nötig. Die genauen Berechnungen sind in [24] beschrieben. Der dazu nötige Aufwand wurde im Rahmen dieser Arbeit als zu groß eingeschätzt. Prinzipiell sollte es aber möglich sein, die in dieser Arbeit vorgestellten Algorithmen so zu erweitern, dass bessere Vergleiche zwischen den Feature Trees möglich sind.

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

Verfahren	Kantenanzahl	Zeit	verworfen
Ordnungsfunktion	11,10	11,49 s	90,000 %
Kantenentfernung	11,10	1,41 s	-
Ordnungsfunktion	10,11	1,36 s	<0,001 %
Kantenentfernung	10,11	1,36 s	-

Tabelle 4.6: MCSTI - Ordnungsfunktion gegen Kantenlöschung mit Sterngraphen

Modus und Kantenanzahl	Anzahl Prozesse			
	1	2	3	4
Debug, 49	157,3 s (100 %)	78,5 s (49,9 %)	52,6 s (33,4 %)	40,3 s (25,6 %)
Release, 49	20,8 s (100 %)	10,5 s (50,3 %)	14,4 s (69,0 %)	10,4 s (50,0 %)
Debug, 51	379,0 s (100 %)	190,3 s (50,2 %)	127,5 s (33,7 %)	97,9 s (25,8 %)
Release, 51	50,7 s (100 %)	25,2 s (49,7 %)	17,0 s (33,5 %)	22,8 s (45,0 %)
Release, 53	56,0 s (100 %)	27,8 s (49,7 %)	30,6 s (54,7 %)	39,5 s (70,6 %)
Release, 55	114,2 s (100 %)	57,3 s (50,2 %)	48,4 s (42,4 %)	72,4 s (63,4 %)

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

In Tabelle 4.7 ist die benötigte Laufzeit für Bäume verschiedener Größen, jeweils im Debug- und Releasemodus, aufgeführt. Die Anzahl der nebenläufigen Prozesse wurde dabei von 1 bis 4 variiert. Die Zahl in Klammern stellt die relative Zeit bezogen auf die nicht parallelisierte Berechnung mit einem Prozess dar. Auffällig ist die hohe relative Effizienz von mehreren Prozessen im Debugmodus. Im Releasemodus schwanken die Ergebnisse, abhängig von der Anzahl der Kanten und nebenläufigen Prozesse. Für eine feste Einstellung sind, auf dem selben Rechner, recht ähnliche Ergebnisse reproduzierbar.

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 Kantenlöschung, 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.

Nach Tabelle 4.4 ist in Bezug auf die Wahl der Kantenreihenfolge, bei gleichen Eingaben, ein linearer Zusammenhang zwischen der Anzahl der Rekursionen und der benötigten Enumerationszeit zu erkennen. Zunächst überrascht es, dass der Start von einer mittleren Kante aus eher unterdurchschnittliche Ergebnisse liefert. Die Argumentation für einen Start in der Mitte war der Algorithmus von Edmonds. Dieser sollte möglichst selten ausgeführt werden. Der Tabelle ist zu entnehmen, dass das auch gelungen ist. Bei einer Reihenfolge der Eingabe entsprechend, und von der mittleren Kante beginnend, ist die Zeit für den Algorithmus von Edmonds, bis auf vernachlässigbare Unterschiede, identisch. Mit der Zufallsauswahl werden schlechtere Ergebnisse erzielt. Da die Zeit des Algorithmus, außer bei sehr kleinen Bäumen, nur einen geringen Anteil an der Gesamtzeit ausmacht, ist vor allem die Enumerationszeit entscheidend.

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 $(\overline{r}, \overline{s})$. Jeder MCSTI φ_2 von $(\overline{r}, \overline{s})$ 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 $(\overline{r}, \overline{s})$. Für eine andere Ordnungsfunktion I_2 gilt möglicherweise $I_2(r) > I_2(\overline{r})$. Dann liegt die Gesamtzahl Z_2 der Rekursionen bei l mal den Rekursionen auf (r, s) plus den Rekursionen auf $(\overline{r}, \overline{s})$. 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.

Ein weiteres interessantes Ergebnis ist die hohe Effizienz der Enumeration von maximalen CSTI mit Mindestgröße. Nach Abbildung 4.1 (grüne Linie) erfolgt die Aufzählung von maximalen CSTI ohne Mindestgröße am schnellsten. Obwohl nachgewiesen werden konnte, dass der Algorithmus zur Berechnung von maximalen CSTI mit Mindestgröße kein polynomial-total-time-Algorithmus ist, ist er bei Zufallsbäumen sehr effizient. Die Geschwindigkeit der Enumeration sinkt nur langsam und recht gleichmäßig ab. Dass sich maximale CSTI in diesem Beispiel und auch im Allgemeinen schneller als Maximum CSTI enumerieren lassen, geht aus den Sätzen zur Laufzeit der beiden Probleme hervor.

Im direkten Vergleich der Enumeration von maximalen Isomorphismen mit dem auf Bäume spezialisierten Algorithmus 3.3 (EnumMaximalCSTI) und dem Algorithmus für allgemeine Graphen über der Reduktion auf das Cliquen-Problem zeigt sich, dass mit zunehmender Anzahl der Kanten der spezialisierte Algorithmus im Verhältnis immer besser wird. Auch die absoluten Zeiten zeigen sehr deutlich die Vorteile von Algorithmus 3.3 auf. Dieser sollte deshalb unbedingt dem allgemeinen Algorithmus vorgezogen werden, wenn es sich bei der Eingabe um zwei Bäume handelt.

Der modifizierte Algorithmus von Uno benötigt im worst case, bezogen auf die Anzahl der Knoten, quadratische Zeit. Abhängig von der Anzahl der Hilfsknoten und dem zulässigen Teilgraphen kann die durchschnittliche Zeit deutlich darunter liegen. Das Verhältnis von maximalen zu Maximum Matchings sei als $Q := \frac{a}{b}$ definiert, wobei a der Anzahl der maximalen und b der Anzahl der Maximum Matchings entspricht. Die durchschnittliche Zeit, ein Maximum Matching durch Aufzählung aller maximalen Matchings zu finden, beträgt $\Theta(Q)$, wie sich aus Abschnitt 3.2.2 folgern lässt. Anhand Abbildung 4.2 lässt sich ablesen, dass bei größeren Bäumen und insbesondere Bezeichnern der modifizierte Algorithmus von Uno besser ist. Dieses Ergebnis ist nicht unerwartet. Ein hoher Knotengrad lässt Q tendenziell schneller wachsen als die durchschnittliche Laufzeit im modifizierten Algorithmus von Uno. Bezeichner führen tendenziell zu einem höheren Wert Q und weniger Kanten im zulässigen Teilgraphen. Dies begünstigt den modifizierten Algorithmus von Uno. An dieser Stelle ist eine Heuristik denkbar, die je nach Knotenzahl und dem Vorhandensein von Bezeichnern die eine oder die andere Methode zur Enumeration der Maximum Matchings auswählt. Die Speicherung der Maximum Matchings sollte dennoch ausgewählt werden. Für die untersuchten Bäume mit 60 Kanten beträgt die Berechnungszeit mit Speicherung der Matchings 13,5 Sekunden, wie Tabelle 4.2 zu entnehmen ist. Nach Abbildung 4.2 beträgt die Zeit ohne Speicherung 28,3 Sekunden (Aufzählung) bzw. 52,3 Sekunden (Uno) und liegt damit deutlich darüber.

Die Evaluierung mit Feature Trees als Eingabe führte zu einem Fund von zwei Molekülen mit sehr ähnlichen Strukturen. Die benötigte Zeit von knapp 6 Minuten für 499500 Aufrufe des Algorithmus von Edmonds lässt vermuten, dass dieser Algorithmus ein probates Mittel ist, ähnliche chemische Strukturen aufzufinden. Dazu sind aber weitere Modifizierungen am Quellcode nötig. So müssten beispielsweise die D-Werte nicht über Matchings berechnet werden, sondern über eine combine function genannte Funktion [24]. Außerdem sind Moleküle im Allgemeinen keine Baumstrukturen [24], so dass eine Analyse mit den in dieser Arbeit entwickelten Enumerationsalgorithmen nur über einen Umweg, beispielsweise der Darstellung durch Feature Trees, erfolgen kann.

Nach Tabelle 4.5 ergibt sich, dass bei Bäumen mit verschiedenen Größen der Baum R der kleinere Baum sein sollte, wenn Maximum CSTI mit Hilfe von Kantenlöschungen enumeriert werden sollen. Die Laufzeit des Algorithmus von Edmonds ist unabhängig von der Reihenfolge der Bäume. Wenn R kleiner ist, gibt es weniger Kanten, die gelöscht

werden können. Die in Theorem 3.8 angegebene Schranke von $|V_R| + 2 - m$ Aufrufen für den Algorithmus von Edmonds spiegelt sich in den praktischen Laufzeitergebnissen wieder. Für die in der Tabelle angegebenen Zufallsbäume beträgt die Größe eines MCSTI jeweils 11. Mit R als kleinerem Baum gilt $|V_R| + 2 - m = 11 + 2 - 11 = 2$. Falls R der größere Baum ist, ist dieser Wert deutlich größer. Die praktischen Resultate orientieren sich an den theoretisch vorhergesagten Ergebnissen.

In Bezug auf Nebenläufigkeit entsprechen die erzielten Ergebnisse nur teilweise den Erwartungen. Im Debugmodus verhält sich die Laufzeit antiproportional in Bezug zur Anzahl der nebenläufigen Prozesse. Das Ergebnis kann somit als sehr gut bewerten werden. Im Releasemodus sind die relativen Ergebnisse zumindest für zwei nebenläufige Prozesse genau so gut wie im Debugmodus. Mit drei und vier nebenläufigen 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

Tabelle 4.8 zeigt die Zeiten zur Enumeration aller MCST auf Bäumen verschiedener Größe an. Zu beachten ist die geringe Anzahl von MCST für ein gegebenes Paar von Zufallsbäumen. Der Algorithmus von Edmonds nimmt für alle Baumgrößen einen signifikanten Anteil an der Gesamtzeit ein.

Maximale Common Subtrees

In Tabelle 4.9 ist die Zeit zur Berechnung von maximalen CST angegeben. Da zur Prüfung auf Subtree Isomorphie der Algorithmus von Edmonds benutzt wird, steigen die Laufzeiten bereits für Bäume mit nur wenigen Kanten rasch an. Die Anzahl dieser Prüfungen ist in der Tabelle angegeben. Auffällig ist, dass die durchschnittliche Zahl an maximalen CST nicht viel größer als die der Maximum CST ist.

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)

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

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

Tabelle 4.10 gibt einen Überblick über die Anzahl an Maximum CST bei Bäumen mit verschiedenen Größen. Ein größerer relativer Unterschied führt zu einer geringeren Anzahl an Maximum CST. Dieses Ergebnis überrascht nicht. Wenn ein Baum in einem anderen enthalten ist, gibt es nur einen Maximum CST. Dieser ist isomorph zu dem Baum, der enthalten ist.

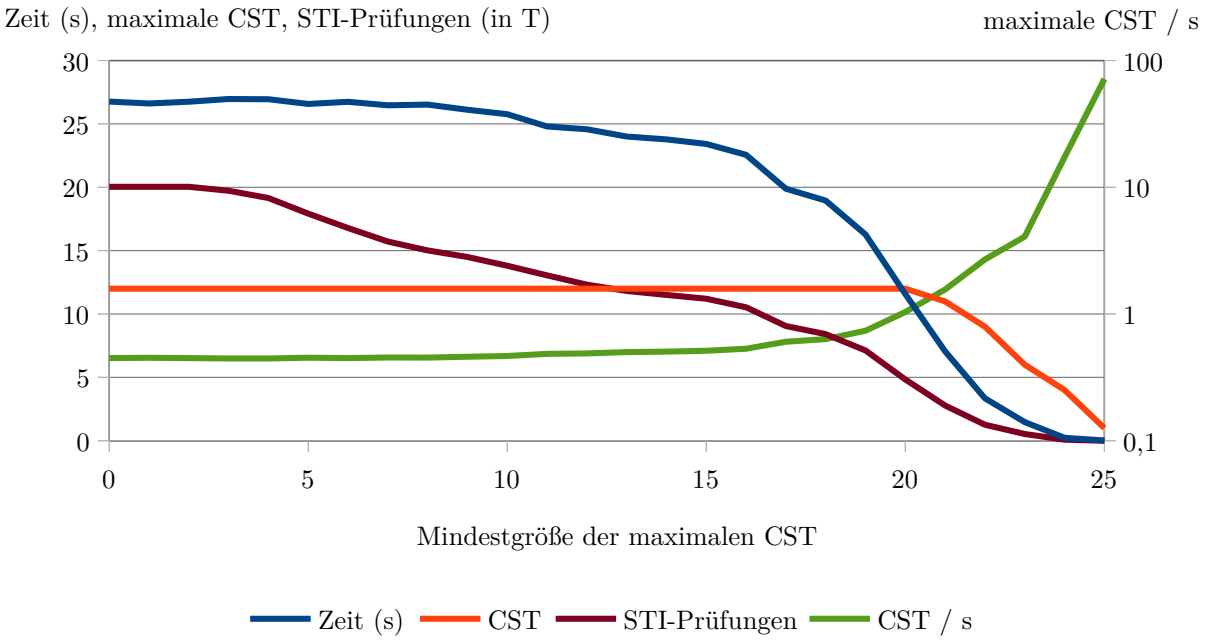


Abbildung 4.4: Anzahl der maximalen CST mit Mindestgröße (x-Achse) - Bäume mit 35 Kanten

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.

Dass die nötige Zeit zur Berechnung aller maximalen CST so hoch ist, ist vor allem der Subtree Isomorphie Prüfung über den Algorithmus von Edmonds geschuldet. Abbildung 4.4 zeigt einen zu Abbildung 4.1 umgekehrten Verlauf der grünen Kurve. Die benötigte Zeit (blaue Kurve) sinkt schneller als die Anzahl der Common Subtrees (rote Kurve). Daraus ergibt sich der steigende Verlauf der grüne Kurve (CST pro Sekunde). Der Kurvenverlauf lässt sich mit der Anzahl der nötigen STI-Prüfungen begründen (dunkelrote Kurve). So höher die Mindestgröße ist, desto weniger solcher Prüfungen erfolgen, entsprechend geringer ist die Berechnungszeit.

Das Verhältnis von MCST zu maximalen CST bei den untersuchten Zufallsbäumen mit 20 Kanten liegt nach Tabellen 4.8 und 4.9 bei ungefähr 1 zu 3. Somit ist jeder dritte maximale CST ein MCST. Im Gegensatz dazu ist das Verhältnis MCSTI zu maximalen CSTI viel größer. Bei den untersuchten Bäumen mit 40 Kanten beträgt es ungefähr 1 zu 1000, wie Tabellen 4.1 und 4.2 zu entnehmen ist.

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. Das 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

Kanten	Zeit	Teilbaumpaare	Teilbaumpaare pro s	Edmonds
20	0,12 s	816	6857	0,02 s
25	0,46 s	2910	6312	0,04 s
30	0,86 s	3774	4383	0,05 s
35	3,43 s	12794	3726	0,08 s
40	18,10 s	48568	2688	0,11 s
45	105,00 s	241826	2293	0,13 s
50	Abbruch wegen Speichermangel (Verbrauch >8GB)			

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 Teilbaumbaaren 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

Mit steigender Kantenzahl nimmt die Anzahl der Lösungen und besonders die benötigte Enumerationszeit zu, wie in Tabelle 4.12 zu sehen ist. Dies liegt an der hohen Zahl an nötigen Tests auf echte Teilmengen, um die Maximalität der Isomorphismen sicherzustellen. Bei höherer Kantenanzahl ist, wie zuvor, mit Speicherplatzproblemen zu rechnen.

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.

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

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

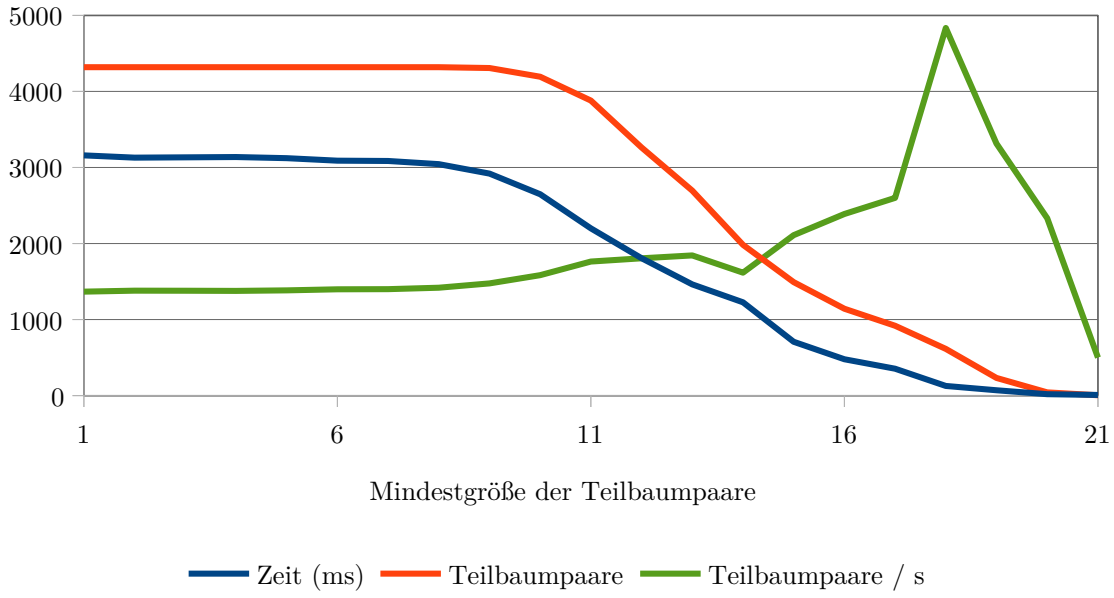


Abbildung 4.5: Teilbäume mit ausschließlich maximalen Isomorphismen - 23 Kanten pro Baum

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

Kanten	Zeit	Teilbaumpaare	Teilbaumpaare pro s	Edmonds
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

Tabelle 4.13: Größtmögliche Isomorphismen, verschiedene Baumgrößen - Jeweils 10 Durchläufe

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.

Die hohe Zahl an nötigen Tests auf echte Teilmengen, falls keine Mindestgröße gefordert ist, bedingen bereits bei kleinen Baumgrößen hohe Laufzeiten. So größer eine geforderte Mindestgröße für die Teilbaumpaare ausfällt, desto weniger Tests auf echte Teilmengen erfolgen. Das Maximum bei Mindestgröße 18 in Abbildung 4.5 für die berechneten Teilbaumpaare pro Sekunde lässt sich auf diesen Umstand zurückführen. Wegen der konstanten Zeit des Algorithmus von Edmonds für alle Mindestgrößen und der geringen Zahl an Teilbaumpaaren über Größe 18 fällt die Zahl der berechneten Teilbaumpaare pro Sekunde ab Mindestgröße 18.

Kapitel 5

Zusammenfassung und Ausblick

Basierend auf dem Algorithmus von Edmonds [21] wurde ein polynomial-delay-Algorithmus zur Enumeration von Maximum Common Subtree Isomorphismen entwickelt. Mit Hilfe der Erweiterung um Bezeichner können damit Feature Trees effizient verglichen werden. Für den Vergleich werden nicht alle zu einem Feature Tree gespeicherten Daten verarbeitet. Für eine zukünftige Erweiterung bietet sich an, auch die restlichen Daten zu berücksichtigen.

Des Weiteren wurde ein polynomial-total-time Algorithmus zur Enumeration aller maximalen Common Subtree Isomorphismen entwickelt. Die Auswertung in Kapitel 4.1 ergab, dass dieser bereits bei einer Anzahl von 25 Kanten pro Baum über 400 mal so schnell wie der Vergleichsalgorithmus [7] zur Enumeration von zusammenhängenden induzierten Teilgraphen bei Eingabe von zwei Bäumen ist. Das in der Einleitung genannte Ziel wurde somit erreicht.

Das Teilproblem der Enumeration von Matchings auf einem bipartiten Graphen mit Gewichten wurde auf zwei Arten gelöst. Eine Transformationen auf einen anderen bipartiten Graphen ohne Gewichte, auf dem dann perfekte Matchings enumeriert werden, hat sich besonders bei Graphen mit höherem Knotengrad und in Zusammenhang mit Bezeichnern als vorteilhaft erwiesen. Bei niedrigem Knotengrad und ohne Bezeichner hat das entwickelte Verfahren zur Enumeration aller maximalen Matchings in einem vollständigen bipartiten Graphen die besseren Ergebnisse erzielt. Bei beiden Methoden führt eine Speicherung berechneter Maximum Weight Bipartite Matchings, die dann in Linearzeit abgerufen werden können, zu einer niedrigeren Berechnungszeit für die Enumeration von Maximum Common Subtree Isomorphismen.

Maximum Common Subtrees können mit einem entwickelten Verfahren, dass direkt bei der dynamischen Programmierung im Algorithmus von Edmonds ansetzt, enumeriert werden. Der dazu nötige Zeitaufwand ist deutlich geringer, als die MCST aus den MCSTI zu berechnen, wie in Abschnitt 4.2.2 gezeigt wurde. Auch maximale Common Subtrees können auf diese Weise berechnet werden. Dazu ist jedoch eine Subtree Isomorphie Prüfung nötig. Diese erfolgt im Rahmen dieser Arbeit mit Hilfe des Algorithmus von Edmonds

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.

Ein Algorithmus, der Paare von Teilbäumen aufzählt, auf denen mindestens ein Isomorphismus existiert und alle darauf existierenden Isomorphismen maximal oder sogar größtmöglich sind, zeigt einen hohen Speicherverbrauch auf. Für Graphen mit mehr als etwa 50 Kanten ist der Algorithmus somit nur eingeschränkt nutzbar. Ein Ansatz, die Lösungen aufzuzählen, ohne vorherige Lösungen speichern zu müssen, ähnlich dem Verfahren zur Enumeration von c-zusammenhängenden Cliquen, das in Abschnitt 3.2.1 vorgestellt wurde, könnte Abhilfe schaffen. Mit den Bezeichnungen aus diesem Abschnitt werden in der Menge R zwei Teilmengen V'_R und V'_S der Knotenmengen der gegebenen Bäumen gespeichert, auf denen mindestens ein Isomorphismus existiert. Dies lässt sich nach Aho und Hopcroft [1] in Linearzeit überprüfen. In der Menge P sind entsprechend die Knotenpaare, bestehend aus je einem Knoten aus $V(R)$ und $V(S)$, enthalten, um die irgendein Isomorphismus $\varphi : V'_R \to V'_S$ erweitert werden kann. Wenn eine Erweiterung nicht möglich ist, dann sind alle Isomorphismen von V'_R nach V'_S maximal bezüglich der Bäume der Eingabe. Die mehrfache Aufzählung gleicher Teilbaumpaare wird analog zur Enumeration von Cliquen durch Mengen von verbotenen Knoten verhindert. Der nötige Speicherbedarf ist mit diesem Verfahren linear von der Anzahl der Knotenpaare abhängig.

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.

Um die Benutzerfreundlichkeit des der Arbeit beiliegenden Programm zu erhöhen, bietet es sich an, eine grafische Schnittstelle mit erweiterten Möglichkeiten zur Ein- und Ausgabe zu entwickeln. Das schließt weitere Möglichkeiten zur Erstellung der Bäume oder der Festlegung der Bezeichner ein. Die Bäume und Maximum Common Subtrees könnten mit Hilfe der Algorithmen aus dem Open Graph Drawing Framework [23] dargestellt werden.

Anhang A

Weitere Informationen

In Anhang A.1 wird die Struktur der dieser Arbeit beiliegende Implementierung grob beschrieben. Darauf folgen in Anhang A.2 Hinweise zur Konfigurationsdatei. In Anhang A.3 folgt eine kurze Bedienungsanleitung des Programms.

A.1 Implementierung

Das Programm wurde mit Visual Studio 2012 in der Programmiersprache C++ auf einem 64-bit Windows 7 System als Konsolenanwendung erstellt. Dazu wurde das Open Graph Drawing Framework (OGDF) [23] eingebunden. Das Programm besteht aus folgenden Klassen-/Quelldateien.

- `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`
  In der Klasse `Mcs` sind die Hauptalgorithmen, wie die Enumeration aller Maximum CSTI, implementiert. Das schließt die Erstellung der Objekte vom Typ `RstPair` und den Algorithmus von Edmonds mit ein. In den Algorithmen werden die entsprechenden Funktionen auf den `RstPair`-Objekten aufgerufen. Diese Klasse wird mit zwei Objekten vom Typ `Tree` initialisiert.

- `Solution.cpp, mcs.h`
  Die Klasse `Solution` speichert den aktuell generierten Isomorphismus. Dort wird gleichzeitig die aktuelle Größe bzw. das Gewicht, im Fall von CWSTI, gespeichert. Außerdem wird festgehalten, wie groß der Isomorphismus höchstmöglich werden kann. Diese Klasse wird auch als Hilfsklasse zur Bestimmung der Matchings genutzt, da im Rahmen dieser Arbeit Matchingkanten und die Zuordnungen von Knoten in einem Isomorphismus zueinander korrelieren.

- `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

Alle Einstellungen, wie die Wahl der Algorithmen, können über die Konfigurationsdatei `mcs.cfg` vorgenommen werden. Eine Standardkonfigurationsdatei wird automatisch erstellt, falls keine vorhanden sind. Leerzeilen und Zeilen, die mit `//` beginnen, werden überlesen. Die einzelnen Einträge sind in der Datei mit Kommentaren beschrieben.

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.

Der Eintrag `BATCHRUN` legt fest, wie oft die vorgegebenen Algorithmen wiederholt werden sollen. Dabei werden jeweils andere Zufallsbäume erzeugt. Dies ermöglicht die Berechnung von durchschnittlichen bzw. summierten Werten über verschiedene Eingaben hinweg.

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.

Mit den Optionen `NODELABELS` und `EDGELABELS` werden Knoten- bzw. Kantenbezeichner hinzugefügt. Die Bewertungsfunktion und die Label sind in der gegebenen Programmversion nicht konfigurierbar.

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 Falscheingaben abgefangen werden, wird nicht gegeben.

A.3 Bedienungsanleitung

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 n 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.

Die Auswahl der Algorithmen erfolgt ausschließlich über die Konfigurationsdatei. Je nach Einstellungen der Konfigurationsdatei werden Zwischenergebnisse angezeigt oder nur eine Zusammenfassung ausgegeben. Dort werden unter anderem die Anzahl der berechneten Lösungen, verworfenen (bad) Matchings und so weiter angezeigt. Das Programm beendet sich nach den Berechnungen automatisch. Es ist deshalb sinnvoll, es aus einem Konsolenfenster heraus zu starten und/oder die Ausgabe in eine Datei umzulenken.

Abbildungsverzeichnis

Algorithmenverzeichnis

Tabellenverzeichnis

Index

Literaturverzeichnis

[1] Aho, Alfred V. und John E. Hopcroft: *The Design and Analysis of Computer Algorithms.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st Auflage, 1974.

[2] Aoki, Kiyoko F., Atsuko Yamaguchi, Yasushi Okuno, Tatsuya Akutsu, Nobuhisa Ueda, Minoru Kanehisa und Hiroshi Mamitsuka: *Efficient Tree-Matching Methods for Accurate Carbohydrate Database Queries.* Genome Informatics, 14:134–143, 2003.

[3] Arora, Sanjeev und Boaz Barak: *Computational Complexity: A Modern Approach.* Cambridge University Press, New York, NY, USA, 1st Auflage, 2009.

[4] Bahiense, Laura, Gordana Manić, Breno Piva und Cid C. de Souza: *The maximum common edge subgraph problem: A polyhedral investigation.* Discrete Applied Mathematics, 160(18):2523 – 2541, 2012. V Latin American Algorithms, Graphs, and Optimization Symposium — Gramado, Brazil, 2009.

[5] Bang-Jensen, Jørgen und Gregory Z. Gutin: *Digraphs: Theory, Algorithms and Applications.* Springer Publishing Company, Incorporated, 2nd Auflage, 2008.

[6] Bunke, H.: *On a Relation Between Graph Edit Distance and Maximum Common Subgraph.* Pattern Recogn. Lett., 18(9):689–694, August 1997.

[7] Cazals, F. und C. Karande: *An Algorithm for Reporting Maximal C-cliques.* Theor. Comput. Sci., 349(3):484–490, Dezember 2005.

[8] Cook, William J., William H. Cunningham, William R. Pulleyblank und Alexander Schrijver: *Combinatorial Optimization.* John Wiley & Sons, Inc., New York, NY, USA, 1998.

[9] Cormen, Thomas H., Charles E. Leiserson, Ronald L. Rivest und Clifford Stein: *Introduction to Algorithms, Third Edition.* The MIT Press, 3rd Auflage, 2009.

[10] Creignou, Nadia, Arne Meier, Julian-Steffen Müller, Johannes Schmidt und Heribert Vollmer: *Paradigms for Parameterized Enumeration.* In: Chat-

terjee, Krishnendu und Jirí Sgall (Herausgeber): *Mathematical Foundations of Computer Science 2013*, Band 8087 der Reihe *Lecture Notes in Computer Science*, Seiten 290–301. Springer Berlin Heidelberg, 2013.

[11] Cuissart, Bertrand und Jean-Jacques Hébrard: *A Direct Algorithm to Find a Largest Common Connected Induced Subgraph of Two Graphs.* In: Brun, Luc und Mario Vento (Herausgeber): *GbRPR*, Band 3434 der Reihe *Lecture Notes in Computer Science*, Seiten 162–171. Springer, 2005.

[12] Diestel, Reinhard: *Graph Theory.* Springer Verlag, 2010.

[13] Fukuda, Komei und Tomomi Matsui: *Finding All Minimum Cost Perfect Matchings in Bipartite Graphs*, 1991.

[14] Garey, Michael R. und David S. Johnson: *Computers and Intractability; A Guide to the Theory of NP-Completeness.* W. H. Freeman & Co., New York, NY, USA, 1990.

[15] Johnson, David S., Mihalis Yannakakis und Christos H. Papadimitriou: *On generating all maximal independent sets.* Information Processing Letters, 27(3):119 – 123, 1988.

[16] Jovanović, A. und D. Danilović: *A new algorithm for solving the tree isomorphism problem.* Computing, 32(3):187–198, 1984.

[17] Koch, Ina: *Enumerating all connected maximal common subgraphs in two graphs.* Theoretical Computer Science, 250(1–2):1 – 30, 2001.

[18] Kuhn, Harold W.: *The Hungarian Method for the assignment problem.* Naval Research Logistics Quarterly, 2:83–97, 1955.

[19] Levi, G.: *A note on the derivation of maximal common subgraphs of two directed or undirected graphs.* CALCOLO, 9(4):341–352, 1973.

[20] Matula, David W.: *An algorithm for subtree identification.* SIAM Rev., 10:273–274, 1968.

[21] Matula, David W.: *Subtree Isomorphism in $O(n^{\frac{5}{2}})$.* In: B. Alspach, P. Hell und D.J. Miller (Herausgeber): *Algorithmic Aspects of Combinatorics*, Band 2 der Reihe *Annals of Discrete Mathematics*, Seiten 91 – 106. Elsevier, 1978.

[22] Mutzel, Petra: *Vorlesung Graphenalgorithmen, Kapitel 1.4, TU Dortmund*, WiSe 2011/2012.

[23] *Open Graph Drawing Framework (OGDF).* http://www.ogdf.net/, 2013.

[24] Rarey, Matthias und J.Scott Dixon: *Feature trees: A new molecular similarity measure based on tree matching.* Journal of Computer-Aided Molecular Design, 12(5):471–490, 1998.

[25] Raymond, John W. und Peter Willett: *Maximum common subgraph isomorphism algorithms for the matching of chemical structures.* Journal of Computer-Aided Molecular Design, 16:2002, 2002.

[26] Shamir, R. und D. Tsur: *Faster Subtree Isomorphism.* In: *Proceedings of the Fifth Israel Symposium on the Theory of Computing Systems (ISTCS '97)*, ISTCS '97, Seiten 126–, Washington, DC, USA, 1997. IEEE Computer Society.

[27] Uno, Takeaki: *Algorithms for Enumerating All Perfect, Maximum and Maximal Matchings in Bipartite Graphs.* In: Leong, Hon Wai, Hiroshi Imai und Sanjay Jain (Herausgeber): *ISAAC*, Band 1350 der Reihe *Lecture Notes in Computer Science*, Seiten 92–101. Springer, 1997.

[28] Valiente, Gabriel: *Algorithms on Trees and Graphs.* Springer-Verlag, Berlin, 2002.

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