Effiziente Prozessmodellanalyse mit Algorithmen der Subgraphisomorphie

Dominic Breuker
Hanns-Alexander Dietrich
Matthias Steinhorst
Jörg Becker
Patrick Delfmann

Veröffentlicht in:
Multikonferenz Wirtschaftsinformatik 2012
Tagungsband der MKWI 2012
Hrsg.: Dirk Christian Mattfeld; Susanne Robra-Bissantz

Braunschweig: Institut für Wirtschaftsinformatik, 2012

Effiziente Prozessmodellanalyse mit Algorithmen der Subgraphisomorphie

Dominic Breuker, Hanns-Alexander Dietrich, Matthias Steinhorst, Jörg Becker, Patrick Delfmann

Westfälische Wilhelms-Universität Münster,
European Research Center for Information Systems, Leonardo-Campus 3, 48149 Münster,
E-Mail: <Nachname>@ercis.uni-muenster.de

Abstract

In der Literatur existiert eine Vielzahl verschiedener Ansätze, um Prozessmodelle strukturell zu analysieren. Ein Unterproblem, das oft in vielen dieser Ansätze auftritt, ist die Identifikation von (häufig auftretenden) Subgraphen innerhalb der Modellgraphen. Um diese Problemstellung zu lösen, können graphentheoretische Algorithmen genutzt werden. Der vorliegende Artikel demonstriert, dass derartige Algorithmen in der Lage sind, große Mengen von Prozessmodellen innerhalb von (Milli-)Sekunden zu analysieren. Sie können folglich als Unterkomponente in bestehende Analyseansätze integriert werden, um (potenziell aufwändigere) Eigenentwicklungen zu ersetzen. Der Vorteil dieser Algorithmen liegt in ihrer breiten, nicht auf konkrete Modellierungssprachen oder Analysezwecke beschränkten Anwendbarkeit.

1 Einleitung

Prozessmodelle dienen im Rahmen des Geschäftsprozessmanagements als Ausgangspunkt für die Analyse der realwirtschaftlichen Abläufe, die sie beschreiben. Da Prozessmodelle eine erhebliche Größe erreichen können und gleichzeitig eine große Menge an Modellen als Analysegrundlage vorliegen kann (vgl. [31]), werden in der Literatur der Wirtschaftsinformatik und Informatik verschiedene automatisierte Analyseverfahren diskutiert. Derartige Ansätze identifizieren beispielsweise Schwachstellen innerhalb der Prozessmodelle oder überprüfen die Berücksichtigung gesetzlicher Anforderungen. Weitere Analyseverfahren bestimmen die Ähnlichkeit von Prozessmodellen oder spezifizieren Übersetzungsregeln, die ein gegebenes Modell in ein Modell einer anderen Modellierungssprache überführen. Ein Unterproblem, das in vielen dieser Verfahren zu lösen ist, bildet die Identifikation (häufig auftretender) Subgraphen innerhalb der Prozessmodellgraphen. So können Schwachstellen beispielsweise als Teilgraphen definiert werden, die auf eine ineffiziente Bearbeitung des dargestellten Sachverhalts schließen lassen. Auch die Ähnlichkeit mehrerer Prozessmodelle kann über die Anzahl häufig auftretender Subgraphen definiert werden.

Da Prozessmodelle oft als gerichtete, getypte und benannte Graphen dargestellt werden, können aus der Graphentheorie bekannte Algorithmen verwendet werden, um das Problem der Identifikation (häufig auftretender) Subgraphen zu lösen. Hier ist dieses als Problem der Subgraphisomorphie bzw. als Problem der Identifikation häufig auftretender Subgraphen bekannt. Sie bezeichnen die Suche eines Subgraphen, der isomorph, also strukturgleich, zu einem vorgegebenen (kleinen) Graphen ist. Im allgemeinen Fall sind diese Probleme nur mit exponentiellem Aufwand zu lösen. Es existieren jedoch einige Algorithmen, die in praktischen Anwendungsszenarien akzeptable Laufzeitergebnisse aufweisen. Im Rahmen des vorliegenden Artikels wurden diese Algorithmen auf mehr als 3000 Prozessmodelle angewendet. Die Ergebnisse verdeutlichen, dass Subgraphen innerhalb dieser Modelle in wenigen (Milli-)Sekunden gefunden werden können.

Die Laufzeitergebnisse legen nahe, aus der Graphentheorie bekannte Algorithmen in bestehende Analyseansätze zu integrieren, um das Teilproblem der Subgrapherkennung zu lösen. Die Anwendung dieser Algorithmen birgt den Vorteil, dass sie nicht für spezielle Analysezwecke (Schwachstellenidentifikation, Modellvergleich, etc.) entwickelt wurden und daher ein breites Spektrum verschiedener Analyseziele unterstützen können. Da sich die vorliegende Laufzeitanalyse ausschließlich auf frei verfügbare Implementierungen der Algorithmen bezieht, werden Entwickler konkreter Spezialansätze von der technisch anspruchsvollen Aufgabe befreit, einen eigenen Ansatz zur Subgrapherkennung zu entwickeln. Sie können sich stattdessen auf die analysezweckspezifische Konfiguration und Erweiterung der hier vorgeschlagenen Algorithmen konzentrieren.

Der Aufbau des Artikels stellt sich wie folgt dar. Im folgenden Abschnitt werden beispielhaft Analyseverfahren vorgestellt, die die Identifikation von (häufig auftretenden) Subgraphen in Prozessmodellen als Teilkomponente enthalten. Im Anschluss werden graphentheoretische Algorithmen für diese Probleme identifiziert (Abschnitt 3) und empirisch auf ihre Anwendbarkeit im Rahmen der Prozessmodellanalyse untersucht (Abschnitt 4). Hierzu werden sie auf realen Modellkollektionen ausgeführt. Der Beitrag schließt mit einer Bewertung der Ergebnisse und einem Ausblick auf weiteren Forschungsbedarf (Abschnitt 5).

2 Verwandte Arbeiten und Forschungslücke

Das Problem der Subgrapherkennung tritt in vielen Modellanalyseverfahren auf. Eine exemplarische Auswahl, gegliedert in die Bereiche *Compliance-Überprüfung*, *Identifikation von Schwachstellen in Geschäftsprozessen*, *Komplexitätsmanagement*, *Modellvergleich* und *Modelltransformation,* wird im Folgenden kurz umrissen.

Ziel der *Compliance-Überprüfung* ist es, zu kontrollieren, ob ein Prozess alle für die jeweilige Geschäftstätigkeit relevanten Gesetze und Regularien erfüllt. Gesetzliche Anforderungen schlagen sich dabei in Teilgraphen nieder, die entweder in einem Prozessmodell enthalten sein müssen oder nicht enthalten sein dürften. Zum Beispiel erweitern GHOSE und KOLIADIS [11] BPMN um verschiedene Konstrukte, um solche Subgraphen zu definieren und zum Teil automatisch zu suchen. WEIDLICH ET AL. [28] entwickeln einen auf sogenannten „behavioral profiles“ basierenden Ansatz, mit dem Log-Daten aus IT-Systemen auf Kompatibilität mit einfachen Mustern geprüft werden können. Durch Beschränkung auf zwei Aktivitäten pro Muster wird die rechnerische Komplexität des generellen Problems umgangen.

Die *Identifikation von Schwachstellen* dient der allgemeinen Verbesserung von Prozessabläufen. Hierfür sind Modellfragmente zu identifizieren, die entweder typische Schwachstellen oder Best Practices repräsentieren. SMIRNOV ET AL. [22] stellen dazu einen Algorithmus vor, der sogenannte *Action Patterns* in Prozessmodellen identifiziert. Ein Action Pattern identifiziert Mengen von Aktivitäten, die häufig gemeinsam auftreten. VAN DONGEN, MENDLING und VAN DER AALST [9] entwickeln einen Ansatz zur Überprüfung von Prozessmodellen auf mögliche reibungslose (bspw. deadlockfreie) Ausführung. Dies wird durch das Suchen typischer Teilgraphen, die auf eine Verletzung des Kriteriums hindeuten, realisiert. Ein ähnlicher Algorithmus findet sich bei TOURÉ, BAÏNA und BENALI [24].

Das Management großer Modellkollektionen ist Aufgabe des *Komplexitätsmanagements*. Hier sind z. B. immer wiederkehrende Aktivitätsmuster zu identifizieren, die durch aggregierte Aktivitäten kompakter dargestellt werden können. Ziel ist es, einen Prozess auf einem höheren Abstraktionsniveau zu beschreiben. WEBER ET AL. [27] entwickeln dazu einen subgraphbasierten Refactoring-Algorithmus. Ähnliche Ansätze verfolgen REIJERS, MENDLING und DIJKMAN [21] sowie UBA ET AL. [25].

Um Prozessmodelle zu *vergleichen*, entwickeln YAN, DIJKMAN und GREFEN [31] einen Algorithmus, der mithilfe von Modellfragmenten die Ähnlichkeit zweier Prozessmodelle schätzt. Dem Verfahren liegt die Annahme zugrunde, dass in ähnlichen Modellen bestimmte Fragmente besonders häufig auftreten.

Bei der *Modelltransformation* werden Verfahren benötigt, die ein gegebenes Modell in ein Modell einer anderen Modellierungssprache überführen. Hierfür sind Teilgraphen innerhalb des Modells zu identifizieren, die in vordefinierte Fragmente übersetzt werden können. Entsprechende Algorithmen stellen z. B. GARCÍA-BAÑUELOS [10] und OUYANG [20] vor.

Daneben existieren auch Ansätze, die sich ganz explizit damit beschäftigen, Subgraphen in Modellen bestimmter Modellierungssprachen zu identifizieren. So wird beispielsweise von AWAD und SAKR [1] ein Algorithmus entwickelt, der beliebige Fragmente in BPMN-Modellen sucht. Ähnliche Algorithmen präsentieren BEERI ET AL. [3] sowie MOMOTKO und SUBIETA [17].

Die Untersuchung der vorgestellten Literatur verdeutlicht, dass in der Informatik und Wirtschaftsinformatik eine große Anzahl verschiedener Analyseverfahren existiert, die spezielle Zielsetzungen verfolgen und das Grundproblem der Subgrapherkennung als Teilproblem beinhalten. Im Folgenden werden bekannte Lösungsansätze zur Subgraphisomorphie und zur Identifikation häufig auftretender Subgraphen vorgestellt und gezeigt, dass sie sich zur Analyse konzeptioneller Modell eignen. Dazu werden sie auf große Modellgraph- und Subgraphmengen angewendet und hinsichtlich ihrer Effektivität und Effizienz validiert.

3 Algorithmen

3.1 Subgraphisomorphie

Formal heißt ein Graph H subgraphisomorph zu einem Modellgraphen G, wenn eine Bijektion Φ zwischen den Knoten von H und einer Teilmenge der Knoten von G existiert, so dass für jede Kante (v, w) zwischen zwei Knoten v und w des Graphen H eine korrespondierende Kante $\{\Phi(v), \Phi(w)\}$ im Modellgraphen G existiert. Algorithmen für Subgraphisomorphie befassen sich folglich mit der Identifikation von Subgraphen, die exakt

mit dem gegebenen Suchgraphen übereinstimmen. Die bestehende Forschung auf diesem Gebiet lässt sich in zwei Bereiche aufgliedern [16]: Einerseits wird durch regelbasierte Suchverfahren der Suchraum eingeschränkt, so dass das Auffinden von Subgraphen im Mittel nur wenig Zeit in Anspruch nimmt. Andererseits kann man sich, statt Lösungen für beliebige Graphen zu suchen, auf spezielle Unterklassen von Graphen beschränken und durch Ausnutzung ihrer Eigenschaften effizientere Lösungen formulieren (z. B. [15]). In dieser Arbeit werden keine Einschränkungen bezüglich der zu untersuchenden Graphen unterstellt. Entsprechende Verfahren werden folglich aus der Analyse ausgeklammert.

Der wohl bekannteste Algorithmus zur Subgraphisomorphie wurde von ULLMANN [26] eingeführt. Er konstruiert einen Suchbaum und traversiert diesen auf Basis eines Backtracking-Verfahrens. Spezielle Regeln erlauben es, den Suchbaum an geeigneten Stellen abzuschneiden und so den Aufwand für die Identifikation der Fragmente gering zu halten. Ähnlich ist der Algorithmus *VF2* von CORDELLA ET AL. [6], welcher rekursiv Mengen von immer größer werdenden Teilgraphen erstellt, so lange bis entweder die passenden Subgraphen gefunden oder deren Existenz ausgeschlossen werden kann. Auch hier kommen Regeln zur Anwendung, die ungeeignete Zwischenergebnisse verwerfen.

Für die beiden Algorithmen von ULLMANN und CORDELLA ET AL. sind ausgereifte Implementierungen frei verfügbar, beispielsweise von CORDELLA ET AL. selbst [5]. Sie erlauben es, in beliebigen Graphen, deren Knoten und Kanten beschriftet sein können, Subgraphen zu suchen. Aufgrund der freien Verfügbarkeit und der Tatsache, dass diese Algorithmen vielfach die Performancereferenz für andere Ansätze bilden, werden sie im Rahmen dieses Beitrags auf ihre Anwendbarkeit für die Modellanalyse untersucht.

3.2 Auffinden häufiger Subgraphen

Ein auf der Subgraphisomorphie aufbauendes Problem ist das Auffinden häufig auftretender Subgraphen in einer Kollektion von Modellgraphen. Im Allgemeinen funktioniert ein Algorithmus für dieses Problem nach folgendem Prinzip [29]: Er beginnt mit einem leeren Graph und generiert rekursiv Erweiterungen, indem neue Knoten hinzugefügt werden. Jede Erweiterung muss auf Subgraphisomorphie innerhalb der Modellkollektion getestet werden. Dadurch wird der Support, also die Häufigkeit, mit der der Subgraph auftritt, bestimmt. Subgraphen, deren Support den gewünschten Minimalwert unterschreitet, werden verworfen.

Aufgrund der potenziell sehr großen Suchbäume ist nicht nur der zeitliche Aufwand, sondern auch der Speicherbedarf kritisch. Auch hier kommt es darauf an, den Suchraum möglichst so zu traversieren, dass sich Aufwand und Speicherbedarf im Regelfall in Grenzen halten. Auch für dieses Problem existieren Algorithmen (z. B. [4,12,14]). Für die vorliegende Analyse wurden wiederum zwei Algorithmen ausgewählt, für die eine freie Implementierung verfügbar ist und die in früheren Evaluationen gute Ergebnisse erzielt haben [19,29]. Im Speziellen handelt es sich dabei um *gSpan* [30] und *Gaston* [18], wobei *Gaston* im Unterschied zu *gSpan* nur auf ungerichteten Graphen mit beschrifteten Knoten und Kanten anwendbar ist. Auf gerichtete Graphen kann daher nur *gSpan* angewendet werden. Für eine genaue Beschreibung ihrer Funktionsweise sei an dieser Stelle auf die entsprechenden Arbeiten verwiesen.

4 Evaluation

4.1 Versuchsaufbau

Ziel dieses Beitrags ist es, zu ermitteln, inwiefern die in Abschnitt 3 identifizierten Algorithmen die Probleme Subgraphisomorphie und häufig auftretender Subgraph auf typischen Modellkollektionen bewältigen können. Die dafür benötigten Implementierungen stammen im Falle des ULLMANN- bzw. *VF2*-Algorithmus aus der in C geschriebenen *VFLib graph matching library*[1], im Falle von *Gaston* und *gSpan* aus *ParSeMiS*[2], einer in Java geschriebenen Bibliothek. Zur Laufzeitmessung wurde ein Intel® Core™ 2 Duo CPU E8400 3,0 GHZ mit 3,25 GB RAM und Windows 7 (32-Bit) verwendet. Zusätzlich wurden die Energiesparmechanismen deaktiviert und der Prozess als Real-Time Prozess ausgeführt. Für ParSeMiS wurde die JVM von Oracle in Version 6.0.26 mit einer Heap-Größe von 1,2 GB verwendet. Dies ist unter der vorgenommenen Konfiguration die maximale Heap-Größe.

Im Rahmen der vorliegenden Analyse wurden sowohl frei verfügbare Modellkollektionen als auch Modelle aus Beratungsprojekten verwendet. Eine Übersicht über die Kollektionen mit der Anzahl enthaltener Modelle sowie Mittelwert (Avg), Standardabweichung (Std), Minimum (Min) und Maximum (Max) der Knoten- und Kantenzahl findet sich in Tabelle 1.

Kollektion	Anzahl Modelle	Knoten				Kanten			
		Avg	Std	Min	Max	Avg	Std	Min	Max
SAP	604	20,74	18,74	3	130	20,80	20,84	2	138
AdB	2200	100,95	109,33	2	912	102,19	112,95	1	935
ÖV	604	45,69	34,70	4	227	47,35	38,05	2	246
HH	54	39,11	17,91	10	100	42,28	20,56	10	110

Tabelle 1: Modellkollektionen

Die erste Kollektion (SAP) ist das SAP-Referenzmodell [7,13], eine Sammlung von 604 EPKs, die das SAP R/3-System beschreiben. Bei der zweiten (AdB) und dritten (ÖV) Kollektion handelt es sich um Prozesslandschaften einer öffentlichen Verwaltung. Die Kollektionen enthalten 2200 (AdB) bzw. 604 (ÖV) EPKs. Die letzte Kollektion (HH) besteht aus 54 EPKs, die einem Referenzmodell für Informationssysteme im Handel [2] entstammen.

Für die Beschriftung der Knoten können für Analysezwecke zwei Fälle unterschieden werden. Im ersten Fall wird nur der Typ des Modellelements, beispielsweise „Ereignis“ oder „XOR“, herangezogen. In diesem Fall werden in einem vollständig getypten und beschrifteten Modellgraphen Subgraphen gesucht, für die zusätzlich zur Isomorphismusdefinition nur die Übereinstimmung der *Knotentypen* gefordert wird. Dies ist in Praxisanwendungen häufig der Fall. Bspw. wird ein Medienbruch in einem Prozessmodell dadurch angezeigt, dass eine Aktivität, an die ein Anwendungssystem sowie ein Dokument als Output annotiert ist, auf eine andere Aktivität folgt, die ein Anwendungssystem und ein Dokument als Input nutzt – unabhängig von den Bezeichnern der beteiligten Modellelemente. Im zweiten Fall werden der Typ und der volle Bezeichner eines Modellelements, also etwa „Ereignis“ und „Rechnung ist eingegangen“, verwendet. Es wird also für eine Entsprechung

[1] http://www.cs.sunysb.edu/~algorith/implement/vflib/implement.shtml
[2] http://www2.informatik.uni-erlangen.de/EN/research/ParSeMiS/download/index.html

zusätzlich die Übereinstimmung der Modellelementbezeichner gefordert. Da die Algorithmen den Suchraum anhand der Beschriftung einschränken können, hängt die Laufzeit direkt von deren Granularität ab. Es ist zu erwarten, dass im Falle der Berücksichtigung von Typ und Bezeichner die bestmögliche Laufzeit erreicht wird, da nur sehr wenige Knoten miteinander assoziierbar sein werden. Für die Zwecke der vorliegenden Analyse wird davon ausgegangen, dass die Bezeichner der Modellelemente semantisch standardisiert sind. Verfahren, die eine einheitliche Semantik der Bezeichner garantieren, werden zum Beispiel von THOMAS und FELLMANN [23] sowie DELFMANN, HERWIG und LIS [8] vorgestellt.

Um *VF2* und den Algorithmus von ULLMANN auswerten zu können, werden zu den Modellkollektionen passende Subgraphen benötigt. Diese zu suchenden Modellfragmente wurden mit Hilfe des *gSpan*-Algorithmus erzeugt, der verwendet wurde, um häufig vorkommende Subgraphen in allen vier Modellkollektionen zu identifizieren. Dieses Vorgehen garantiert, dass die zu suchenden Subgraphen in einigen, aber nicht allen Graphen gefunden werden können. Aufgrund des hohen Speicherbedarfs wurde dazu eine virtuelle Maschine mit 68 GB RAM in der Amazon-Cloud verwendet. So konnten deutlich größere Kollektionen von Subgraphen erstellt werden, als dies bei einem konventionellen PC der Fall gewesen wäre. Tabelle 2 enthält eine Übersicht über alle Subgraphen, die aus den jeweiligen Modellkollektionen entwickelt wurden. Die Tabelle unterscheidet dabei zwischen solchen Modellfragmenten, die nur den Knotentyp als Bezeichner enthalten und solchen, die sowohl Knotentyp als auch Bezeichner berücksichtigen.

Kollektion		Berücksichtigung von Knotentyp					Berücksichtigung von Kontentyp und -bezeichner				
		Avg	Std	Min	Max	Anzahl	Avg	Std	Min	Max	Anzahl
SAP	Knoten	4,30	1,23	1	7	399	6,82	4,11	1	17	2023
	Kanten	3,31	1,23	0	6		5,82	4,11	0	16	
AdB	Knoten	5,16	1,38	1	9	1066	3,03	1,74	1	8	298
	Kanten	4,17	1,38	0	8		2,03	1,74	0	7	
ÖV	Knoten	11,20	2,84	1	20	28184	4,98	2,62	1	10	363
	Kanten	10,23	2,83	0	19		4,07	2,70	0	10	
HH	Knoten	12,16	2,97	1	21	27580	2,18	1,31	1	7	283
	Kanten	11,21	2,98	0	20		1,19	1,34	0	6	

Tabelle 2: Subgraphkollektionen

4.2 Durchführung

Die Analyse wurde auf den Prozessmodellgraphen durchgeführt. Bei der Überführung der Modelle in das verwendete Graphformat wurden sämtliche Elementtypen des Modells als Knoten des Graphen repräsentiert. Jeder Graphknoten hat darüber hinaus Attribute für Typ und Bezeichner, so dass keine semantischen Informationen verloren gehen. Die gerichteten Kanten der EPK-Modelle wurden in gerichtete Kanten im Graphen überführt. Die Algorithmen ULLMANN und *VF2* wurden so konfiguriert, dass sie alle möglichen Ausprägungen des Subgraphen innerhalb eines Modellgraphen zurückgeben. Jeder Subgraph wurde daraufhin mit jedem Algorithmus in jeder Modellkollektion gesucht. Die Analyse beinhaltet die Gesamtlaufzeit für jeden einzelnen Subgraphen.

Die Implementierung des *Gaston* Algorithmus arbeitet nur auf ungerichteten Graphen, weshalb für Vergleichszwecke *gSpan* sowohl auf ungerichteten (gSpan U) als auch auf

gerichteten Graphen (gSpan G) ausgeführt wurde. Die gerichteten Graphen werden hierfür in ungerichtete Graphen überführt, indem die Richtung der Kante ignoriert wird. Zur Konfiguration der Algorithmen *gSpan* und *Gaston* kann die prozentuale Mindesthäufigkeit (Support) als Parameter übergeben werden. Der Support beschreibt den prozentualen Anteil der Modelle innerhalb einer Kollektion, in denen ein Subgraph vorhanden sein muss, um ausgegeben zu werden. Beide Algorithmen wurden jeweils mit den Mindesthäufigkeiten 70%, 50%, 30%, 20%, 10%, 5% und 1% ausgeführt. In zwei unabhängigen Durchläufen wurden die Laufzeit und der Speicherbedarf für jeden Algorithmus einzeln gemessen.

4.3 Ergebnisse zur Suche gegebener Subgraphen

Tabelle 3 enthält die Suchzeiten für Typberücksichtigung, Tabelle 4 die für die Berücksichtigung des Typs und des Bezeichners. Es werden jeweils der Mittelwert (Avg), die Standardabweichung (Std), das Minimum (Min) und Maximum (Max) der Suchzeit für einen Subgraphen in allen Modellgraphen einer Kollektion angegeben. Neben den absoluten Zeiten werden auch bezüglich der Größe der Modellkollektion normalisierte Größen dargestellt. Beispielsweise dauerte das Suchen eines der 399 die Typen berücksichtigenden Subgraphen der SAP-Kollektion mit *VF2* durchschnittlich 952,35 Millisekunden. Normalisiert bedeutet dies, dass eine Suche im Durchschnitt 952,35 ms / 604 Modelle = 1,58 ms/Modell dauerte.

Kollektion und Algorithmus		Zeit [ms]				Zeit [ms/Modell]				R^2
		Avg	Std	Min	Max	Avg	Std	Min	Max	
SAP	VF2	952,35	552,60	806	11751	1,58	0,91	1,33	19,46	0,01
	Ull	931,73	554,88	799	11845	1,54	0,92	1,32	19,61	0,00
AdB	VF2	6882,18	8905,68	5832	207620	3,13	4,05	2,65	94,37	0,00
	Ull	9905,28	15439,66	5861	279302	4,50	7,02	2,66	126,96	0,02
ÖV	VF2	1020,23	67,62	931	6802	1,69	0,11	1,54	11,26	0,00
	Ull	41799,38	534211,87	960	21479600	69,20	884,46	1,59	35562,25	0,02
HH	VF2	87,26	19,69	82	585	1,62	0,36	1,52	10,83	0,00
	Ull	524,70	1505,32	83	31551	9,72	27,88	1,54	584,28	0,09

Tabelle 3: Subgraphisomorphie (Berücksichtigung von Knotentypen)

Im Allgemeinen konnte die Suche der Subgraphen mit den verwendeten Algorithmen in einer Größenordnung von Millisekunden bis Sekunden bewältigt werden. Nur in seltenen Fällen benötigt der Algorithmus von ULLMANN Laufzeiten von bis zu 21479600 Millisekunden (~6 Stunden). Generell ist bei den durchschnittlichen Zeiten eine Überlegenheit des *VF2* festzustellen. Da *VF2* jedoch als Verbesserung des ULLMANN-Algorithmus entwickelt wurde, ist dies nicht verwunderlich. Eine Ausnahme bildet hier das SAP-Referenzmodell, bei dem sich *VF2* und der ULLMANN-Algorithmus kaum unterscheiden. Die höchste gemessene Laufzeit des *VF2* betrug 207620 Millisekunden (~3,5 Minuten). Im Durchschnitt benötigte VF2 in keiner Kollektion länger als 7 Sekunden. *VF2* ist daher für umfangreiche Modellanalysen im Rahmen der untersuchten Kollektionen sehr gut verwendbar.

Ein Vergleich der Ergebnisse aus Tabelle 3 und 4 verdeutlicht die Unterschiede, die durch verschiedene Granularitäten bei der Beschriftung der Knoten entstehen. Während die durchschnittlichen Laufzeiten sich nur leicht verändern, sind bei Berücksichtigung von Bezeichnern keine extrem hohen Laufzeiten mehr zu beobachten. Aufgrund des höheren

Potentials für regelbasierte Performanceverbesserungen bei dieser feingranularen Beschriftung, war dieses Ergebnis jedoch ebenfalls zu erwarten.

Um den Einfluss der Größe des Subgraphen auf dessen Suchzeit zu untersuchen, wurde für jeden Suchlauf neben der Laufzeit auch die entsprechende Anzahl Knoten und Kanten protokolliert. Im Anschluss wurde für jede Kollektion von Subgraphen eine multiple Regression durchgeführt, um die Suchzeit durch die Größe des Subgraphen zu erklären. Bestimmtheitsmaße sind jeweils in der letzten Spalte der Tabellen 3 und 4 abgetragen. Die durchweg sehr niedrigen Werte deuten darauf hin, dass die Größe des Subgraphen nur wenig Einfluss auf die Suchzeit hat. In der Tat gab es sowohl sehr große als auch sehr kleine Subgraphen mit geringen Laufzeiten. Um diesen Aspekt weiter zu untersuchen, wurde in einem nächsten Schritt eine manuelle Detailbetrachtung der Subgraphen mit extrem hoher Laufzeit durchgeführt. Dabei handelte es sich um die 17 Subgraphen, deren Suche mit dem ULLMANN-Algorithmus mehr als 1000000 Millisekunden (ca. 16,5 Minuten) dauerte. Sie stammten alle aus der getypten ÖV-Kollektion und hatten 13-18 Knoten sowie 12-17 Kanten, waren also verhältnismäßig groß. Dies deutet darauf hin, dass lange Suchzeiten nur bei großen Subgraphen auftreten, aber eben nicht notwendigerweise.

Kollektion und Algorithmus		Zeit [ms]				Zeit [ms/Modell]				R^2
		Avg	Std	Min	Max	Avg	Std	Min	Max	
SAP	VF2	770,63	62,91	678	1681	1,28	0,10	1,12	2,78	0,00
	Ull	741,48	73,29	667	1793	1,23	0,12	1,10	2,97	0,00
AdB	VF2	5989,91	221,72	5706	7181	2,72	0,10	2,59	3,26	0,00
	Ull	6020,04	319,79	5672	8057	2,74	0,15	2,58	3,66	0,04
ÖV	VF2	928,19	44,19	884	1297	1,54	0,07	1,46	2,15	0,02
	Ull	957,46	54,82	890	1431	1,59	0,09	1,47	2,37	0,00
HH	VF2	75,87	2,84	74	108	1,40	0,05	1,37	2,00	0,03
	Ull	76,64	11,03	74	260	1,42	0,20	1,37	4,81	0,00

Tabelle 4: Subgraphisomorphie (Berücksichtigung von Knotentyp und -bezeichner)

4.4 Ergebnisse zur Suche häufiger Subgraphen

Tabelle 5 stellt die durchschnittlich gemessenen Laufzeiten (Avg) in Millisekunden und die Anzahl gefundener Subgraphen (#S) in getypten und bezeichneten Kollektionen in Abhängigkeit vom Support dar. Fehlende Werte in der Tabelle können auf Grund von mangelndem Heap-Speicherplatz der JVM und dem dadurch bedingten Abbruch des Algorithmus zustande kommen (—) oder aber durch die Tatsache, dass der absolute Support die Anzahl von zwei Modellen (#M<2) unterschreitet. Die Ergebnisse von *Gaston* sind mit den Ergebnissen von *gSpan* auf ungerichteten Graphen (gSpan U) zu vergleichen, da *gSpan* auf gerichteten Graphen (gSpan G) den Suchraum schneller einschränken kann. Wie erwartet wächst die Laufzeit mit der Größe der Modellkollektion. Die Anzahl der gefundenen Subgraphen ist für den Fall von gerichteten Graphen (gSpan G) meist kleiner, da die Richtung der Kante berücksichtigt wird und Subgraphen mit gleichen Knoten, aber unterschiedlich gerichteten Kanten nicht mehr miteinander assoziiert werden können.

Überraschend ist, dass *gSpan* in den Experimenten mit dem jeweils niedrigsten Support schneller als *Gaston* Ergebnisse liefert. Dies steht im Gegensatz zu anderen empirischen Auswertungen [19,29]. Eine mögliche Erklärung ist, dass beide Algorithmen die potenziellen

Kandidaten für Subgraphen im Verlauf der Suche regelmäßig auf Subgraphisomorphie testen müssen. *gSpan* führt diesen Test immer wieder neu durch, wohingegen *Gaston* alte Bijektionen vorhält, um neue effizienter berechnen zu können. Dies lässt vermuten, dass bei den vielen nur wenige Knoten und Kanten umfassenden Subgraphen in den Kollektionen bei *Gaston* der Effizienzgewinn den Mehraufwand durch das Speichern nicht kompensieren kann. Der Geschwindigkeitsvorsprung von *Gaston* bei hohem Support liegt hingegen an der effizienten Speicherung der Bijektionen, da dort weniger Graphen durchsucht werden müssen und somit weniger gespeichert werden muss. Für den Fall von gerichteten Graphen ist *gSpan* G meist schneller als im ungerichteten Fall. Insbesondere verstärkt sich dieser Effekt für niedrigen Support, bei dem *gSpan* G um bis zu 57% schneller ist als die ungerichtete Variante. Wie bereits erwartet, spiegelt dies den positiven Effekt auf die Laufzeit durch die Einbeziehung der Kantenrichtung wider.

Kollektion und Algorithmus		70%		50%		30%		20%		10%		5%		1%	
		Avg	#S	Avg	#S	Avg	#S	Avg	#S	Avg	#S	Avg	#S	Avg	#S
SAP	Gaston	31	0	42	2	42	4	41	5	43	6	50	15	1.033	2.042
	gSpan U	47	0	32	2	47	4	46	5	47	6	55	15	468	2.042
	gSpan G	41	0	41	2	38	3	47	5	51	8	56	14	443	2.023
AdB	Gaston	144	2	157	3	188	4	240	7	358	13	691	105	—	—
	gSpan U	149	2	148	3	203	4	250	7	352	13	616	105	—	—
	gSpan G	149	2	152	3	185	4	239	7	350	15	587	87	—	—
ÖV	Gaston	44	1	42	1	48	3	53	5	58	5	75	11	10.375	30.948
	gSpan U	47	1	47	1	47	3	55	5	63	5	70	11	5.039	30.948
	gSpan G	45	1	46	1	46	3	55	4	58	5	73	12	5.428	30.337
HH	Gaston	27	2	22	3	26	5	25	6	27	11	102	82	#M<2	#M<2
	gSpan U	24	2	24	3	32	5	24	6	31	11	71	82	#M<2	#M<2
	gSpan G	19	2	20	2	24	5	26	7	27	11	60	76	#M<2	#M<2

Tabelle 5: Auffinden häufiger Subgraphen (Berücksichtigung von Knotentyp und -bezeichner)

Kollektion und Algorithmus		70%		50%		30%		20%		10%		5%		1%	
		Avg	#S	Avg	#S	Avg	#S	Avg	#S	Avg	#S	Avg	#S	Avg	#S
SAP	Gaston	58	3	100	13	149	30	303	52	2.027	144	—	—	—	—
	gSpan U	63	3	101	13	156	30	343	52	1.864	144	—	—	—	—
	gSpan G	72	4	75	7	123	30	150	42	1.137	137	11.710	399	—	—
AdB	Gaston	655	13	1.783	42	—	—	—	—	—	—	—	—	—	—
	gSpan U	733	13	2.239	42	—	—	—	—	—	—	—	—	—	—
	gSpan G	506	8	1.405	40	19.410	275	—	—	—	—	—	—	—	—
ÖV	Gaston	245	18	410	36	894	154	2.064	423	6.190	1.945	—	—	—	—
	gSpan U	226	18	352	36	1.037	154	2.637	423	9.501	1.945	31.964	10.004	—	—
	gSpan G	160	17	283	41	672	146	1.519	401	4.796	1.733	16.464	8.467	—	—
HH	Gaston	55	35	76	79	132	310	210	849	506	4.004	2.795	33.868	#M<2	#M<2
	gSpan U	86	35	93	79	188	310	328	849	858	4.004	3.893	33.868	#M<2	#M<2
	gSpan G	67	39	85	93	148	318	242	780	591	3.570	2.479	27.580	#M<2	#M<2

Tabelle 6: Auffinden häufiger Subgraphen (Berücksichtigung von Knotentypen)

Ein anderes Bild zeigt sich hingegen bei den Laufzeiten in getypten Kollektionen, dargestellt in Tabelle 6. *Gaston* ist in den Experimenten mit dem jeweils niedrigsten Support meist schneller als *gSpan* U. Dies liegt daran, dass im Gegensatz zu den getypten und

bezeichneten Kollektionen das Testen auf Subgraphisomorphie durch *gSpan* mehr Rechenzeit in Anspruch nimmt als die Datenhaltung der Bijektionen und deren Erweiterung durch *Gaston*. Die höhere Anzahl an gleichen Beschriftungen erschwert es in diesem Fall, den Suchraum für den Test auf Subgraphisomorphie einzuschränken. Im Allgemeinen liegen die Laufzeiten jedoch über denen der getypten und bezeichneten Kollektionen. Dies erklärt sich zum einen dadurch, dass mehr häufig auftretende Subgraphen vorliegen, zum anderen durch teurere Subgraphisomorphie-Tests während der Suche. Mit einer maximalen Laufzeit von weniger als 32 Sekunden sind jedoch sowohl *gSpan* als auch *Gaston* in der Lage, große Modellkollektionen schnell zu analysieren. Da *Gaston* jedoch nur auf ungerichteten Graphen arbeitet, ist *gSpan* für die Prozessmodellanalyse besser geeignet.

Neben der Laufzeit ist auch der Speicherbedarf der Algorithmen für ihren Einsatz von Bedeutung. Sollen bspw. Subgraphen mit sehr geringem Support gefunden werden, ist eine JVM mit einer Heap-Größe von 1,2 GB nicht mehr ausreichend. Besonders deutlich zeigt sich die Einschränkung in der getypten Kollektion AdB, in der bereits bei einem Support von 20% der Speicher überläuft. Der Speicherbedarf ist bei *Gaston* strikt höher als bei *gSpan.* Dies hängt mit der oben erwähnten Vorhaltung von Bijektionen zusammen.

5 Zusammenfassung und Ausblick

Im Rahmen der strukturellen Analyse von Modellen treten immer wiederkehrende Fragestellungen auf, die auf graphentheoretischen Problemen basieren. Beispiele für derartige Probleme sind die Suche vordefinierter bzw. häufig auftretender Subgraphen. Durch Anwendung aus der Graphentheorie bekannter Algorithmen auf Prozessmodellkollektionen konnte gezeigt werden, dass effiziente Lösungen für diese Probleme implementiert und frei verfügbar vorliegen. Mit Laufzeiten im Millisekunden- bis Sekundenbereich berechnen diese Algorithmen Subgraphen auch auf großen Modellmengen sehr schnell. Methoden zur Modellanalyse können von diesen Ergebnissen profitieren, da die Grundkomplexität der untersuchten Probleme nicht durch komplizierte Eigenentwicklungen abgefangen werden muss. Dies kann sowohl die Entwicklung neuer Ansätze erleichtern als auch die Effizienz bestehender Ansätze verbessern.

In dieser Arbeit wurden ausschließlich Algorithmen betrachtet, die exakte Entsprechungen in Form von Subgraphen berechnen. In vielen Analyseszenarien kann es unter Umständen aber notwendig sein, zu einem gegebenen, zu suchenden Subgraphen ähnliche Strukturen innerhalb der Modelle zu identifizieren. Diese Aufgabe ist in der Graphentheorie als das generellere Problem der Subgraphhomöomorphie bekannt. Zukünftige Forschung wird sich daher einer Laufzeitanalyse entsprechender Algorithmen widmen. Darüber hinaus wird auch eine Untersuchung von Algorithmen angestrebt, die spezielle Eigenschaften von Graphen für eine effiziente Lösung des Iso- bzw. Homöomorphieproblems ausnutzen (vgl. Abschnitt 3). In diesem Zusammenhang ist ebenfalls zu untersuchen, inwiefern Prozessmodelle diese Eigenschaften aufweisen. Darüber hinaus ist zu klären, inwiefern sich die Laufzeiten der Algorithmen auf Modellen anderer Modellierungssprachen ähneln. Vorübergehende Experimente legen die Vermutung nahe, dass die hier vorgestellten Algorithmen auch auf Datenmodellen und Prozessmodellen anderer Modellierungssprachen Subgraphen im (Milli)Sekundenbereich identifizieren können.

6 Literatur

[1] Awad, A; Sakr, S (2010): Querying Graph-Based Repositories of Business Process Models. In Yoshikawa, M; Meng, X; Yumoto, T; Ma, Q; Sun, L; Watanabe, C (Hrsg.), Database Systems for Advanced Applications. Springer, Berlin / Heidelberg, 33-44.

[2] Becker, J; Schütte, R (2004): Handelsinformationssysteme. 2. Auflage. Redline Wirtschaft bei Verlag Moderne Industrie, Landsberg.

[3] Beeri, C; Eyal, A; Kamenkovich, S; Milo, T (2008): Querying business processes with BP-QL. Information Systems Journal 33(6):477-507.

[4] Borgelt, C; Berthold, MR (2002): Mining molecular fragments: finding relevant substructures of molecules. In: Proc. of the IEEE Int.Conf. on Data Mining.

[5] Cordella, LP; Foggia, P; Sansone, C; Vento, M (1999): Performance Evaluation of the VF Graph Matching Algorithm. In: Proc. of the 10th Int. Conf. on Image Analysis and Processing. Venice , Italy.

[6] Cordella, LP; Foggia, P; Sansone, C; Vento, M (2004): A (sub)graph isomorphism algorithm for matching large graphs. IEEE transactions on pattern analysis and machine intelligence 26(10):1367-1372.

[7] Curran, T; Keller, G; Ladd, A (1997): SAP R/3 Business Blueprint: Understanding the Business Process Reference Model. Prentice Hall PTR, Upper Saddle River.

[8] Delfmann, P; Herwig, S; Lis, Ł (2009): Unified Enterprise Knowledge Representation with Conceptual Models - Capturing Corporate Language in Naming Conventions. In: Proc. of the 30th Int. Conf. on Information Systems. .

[9] Dongen, BF van; Mendling, J; Aalst, WMP van der (2006): Structural Patterns for Soundness of Business Process Models. Proc. of the 10th IEEE International Enterprise Distributed Object Computing Conference :116-128.

[10] García-Bañuelos, L (2008): Pattern Identification and Classification in the Translation from BPMN to BPEL. In Meersman, R; Tari, Z (Hrsg.), On the Move to Meaningful Internet Systems: OTM 2008. Springer-Verlag, Berlin / Heidelberg, 436-444.

[11] Ghose, A; Koliadis, G (2007): Auditing Business Process Compliance. In Krämer, B; Lin, K-J; Narasimhan, P (Hrsg.), Service-Oriented Computing – ICSOC 2007. Springer, Berlin / Heidelberg, 169-180.

[12] Huan, J; Wang, W; Prins, J (2003): Efficient mining of frequent subgraphs in the presence of isomorphism. Proc. of the 3rd IEEE Int. Conf. on Data Mining :549-552.

[13] Keller, G; Teufel, T (1998): SAP R/3 process-oriented implementation: Iterative process prototyping. Addison Wesley Longman, Harlow, England.

[14] Kuramochi, M; Karypis, G (2001): Frequent Subgraph Discovery. In: Proc.of the IEEE Int. Conf. on Data Mining. San Jose, CA , USA.

[15] Lingas, A; Sysło, MM (1988): A polynomial-time algorithm for subgraph isomorphism of two-connected series-parallel graphs. In Lepistö, T; Salomaa, A (Hrsg), Automata, Languages and Programming. Springer, Berlin / Heidelberg, 394-409.

[16] Lipets, V; Vanetik, N; Gudes, E (2009): Subsea: an efficient heuristic algorithm for subgraph isomorphism. Data Mining and Knowledge Discovery 19(3):320-350.

[17] Momotko, M; Subieta, K (2004): Process Query Language: A Way to Make Workflow Processes More Flexible. Advances in Databases and Information Systems 54:306-321.

[18] Nijssen, S; Kok, JN (2004): Frequent graph mining and its application to molecular databases. In: Proc. of the IEEE Int. Conf. on Systems, Man and Cybernetics.

[19] Nijssen, S; Kok, JN (2006): Frequent Subgraph Miners: Runtimes Don't Say Everything. In: Gärtner, T; Garriga, GC; Meinl, T (Hrsg.), Proc. of the Int.Workshop on Mining and Learning with Graphs. Berlin, Germany.

[20] Ouyang, C; Dumas, M; Hofstede, AHMT; Aalst, WMP van der (2008): Pattern-based Translation of BPMN Process Models to BPEL Web Services. International Journal of Web Services Research 5(1):1-21.

[21] Reijers, HA; Mendling, J; Dijkman, RM (2011): Human and automatic modularizations of process models to enhance their comprehension. Information Systems Journal 36(5):881-897.

[22] Smirnov, S; Weidlich, M; Mendling, J; Weske, M (2009): Action Patterns in Business Process Models. In Baresi, L; Chi, C-H; Suzuki, J (Hrsg.), Service-Oriented Computing. Springer, Berlin / Heidelberg, 115-129.

[23] Thomas, O.; Fellmann, M. (2009): Semantic Process Modeling – Design and Implementation of an Ontology-Based Representation of Business Processes. Business & Information Systems Engineering 1(6), 438-451.

[24] Touré, F; Baïna, K; Benali, K (2008): An Efficient Algorithm for Workflow Graph Structural Verification. In Meersmann, R; Tari, Z (Hrsg.), On the Move to Meaningful Internet Systems: OTM 2008. Springer, Berlin / Heidelberg, 392-408.

[25] Uba, R; Dumas, M; Garcia-Banuelos, L; Rosa, M La (2011): Clone Detection in Repositories of Business Process Models. Brisbane.

[26] Ullmann, JR (1976): An Algorithm for Subgraph Isomorphism. Journal of the ACM 23(1):31-42.

[27] Weber, B; Reichert, M; Mendling, J; Reijers, HA (2011): Refactoring large process model repositories. Computers in Industry 62(5):467-486.

[28] Weidlich, M; Polyvyanyy, A; Desai, N; Mendling, J (2010): Process Compliance Measurement Based on Behavioural Profiles. In Pernici, B (Hrsg.), Advanced Information Systems Engineering. Springer, Berlin / Heidelberg, 499-514.

[29] Wörlein, M; Meinl, T; Fischer, I; Philippsen, M (2005): A Quantitative Comparison of the Subgraph Miners MoFa, gSpan, FFSM, and Gaston. In Jorge, AM; Torgo, L; Brazdil, P; Camacho, R; Gama, J (Hrsg.), Knowledge Discovery in Databases: PKDD 2005. Springer, Berlin / Heidelberg, 392-403.

[30] Yan, X; Han, J (2002): gSpan: Graph-based substructure pattern mining. In: Proc. of the IEEE Int. Conf. on Data Mining. Maebashi City, Japan.

[31] Yan, Z; Dijkman, R; Grefen, P (2010): Fast Business Process Similarity Search with Feature-Based Similarity Estimation. In Meersmann, R; Dillon, T; Herrero, P (Hrsg.), On the Move to Meaningful Internet Systems: OTM 2010. Springer, Berlin, 60-77.