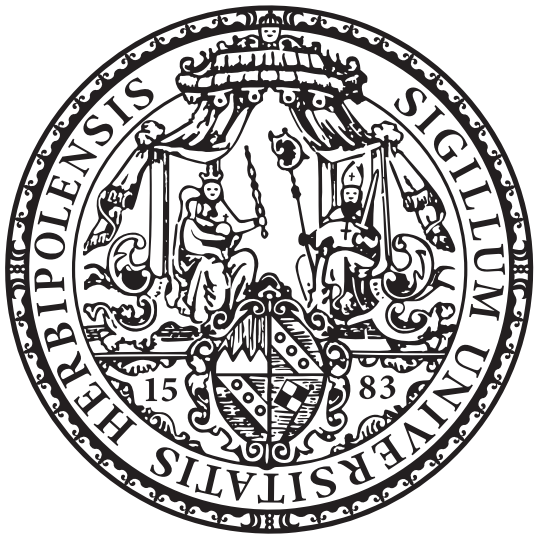
Bachelorarbeit

Visualisierung von Prozessgraphen im Rahmen des Process Mining

Nina Kamphowe

Abgabedatum: 17. Juli 2024
Überarbeitet: 21. August 2024
Betreuer: Prof. Dr. Alexander Wolff
Tim Hegemann, M. Sc.



Julius-Maximilians-Universität Würzburg
Lehrstuhl für Informatik I
Algorithmen und Komplexität

Zusammenfassung

Die datenbasierte Optimierung von Geschäftsprozessen gewinnt in modernen Unternehmen zunehmend an Bedeutung. Die Technologie des Process Mining spielt hierbei eine Schlüsselrolle, indem sie die Analyse, Visualisierung und Optimierung realer Geschäftsprozesse anhand digitaler Spuren in IT-Systemen ermöglicht. Ein wesentlicher Aspekt des Process Mining ist die hierarchische Darstellung der Geschäftsprozesse in Form von Prozessgraphen. In dieser Bachelorarbeit geht es um die Visualisierung von Prozessgraphen mithilfe des Sugiyama-Frameworks für das Zeichnen von gerichteten Graphen. In Prozessgraphen entspricht die Kantenrichtung meistens der zeitlichen Abfolge.

Ein besonderer Fokus liegt auf dem ersten Schritt des Sugiyama-Frameworks: dem Entfernen von Kreisen in gerichteten Graphen. Hierfür werden drei verschiedene, aus der Literatur bekannte Algorithmen vorgestellt. Darüber hinaus werden verschiedene Kantenformen, einschließlich orthogonaler Kanten und Bézierkurven, analysiert und die Bestimmung geeigneter Kontrollpunkte für Bézierkurven erläutert. Zur Verbesserung der Lesbarkeit werden die Anzahl der Kantenkreuzungen, die Größe des Zeichenbereichs und die Anzahl der Kantenknicke heuristisch minimiert. Der gesamte Zeichenalgorithmus wurde in Java implementiert und anhand verschiedener möglichst realer Prozesse getestet, um Effizienz und Qualität der Darstellungen zu bewerten.

Abstract

The data-driven optimization of business processes is gaining increasing importance in modern companies. Process Mining technology plays a key role in this context by enabling the analysis, visualization, and optimization of real business processes based on digital traces in IT systems. A significant aspect of Process Mining is the hierarchical representation of business processes in the form of process graphs. This bachelor's thesis focuses on the visualization of process graphs using the Sugiyama framework for drawing directed graphs. In process graphs, the direction of the edges usually corresponds to the chronological sequence.

A particular focus is placed on the first step of the Sugiyama framework: removing cycles in directed graphs. Three different algorithms, known from the literature, are presented for this purpose. Furthermore, various edge shapes, including orthogonal edges and Bézier curves, are analyzed, and the determination of suitable control points for Bézier curves is explained. To improve readability, the number of edge crossings, the size of the drawing area, and the number of edge bends are heuristically minimized. The entire drawing algorithm was implemented in Java and tested using various as realistic as possible processes to evaluate the efficiency and quality of the representations.

Inhaltsverzeichnis

1. Einleitung

Die datenbasierte Optimierung von Geschäftsprozessen in Unternehmen gewinnt aufgrund von steigender Komplexität der Geschäftstätigkeiten und zunehmender Datenverarbeitungskapazitäten laufend an Bedeutung. Eine Schlüsseltechnologie ist dabei das Process Mining, welches die Analyse, Visualisierung und Optimierung realer Geschäftsprozesse anhand von digitalen Spuren in IT-Systemen ermöglicht. Ein zentraler Aspekt des Process Mining ist die hierarchische Darstellung der Geschäftsprozesse in Form von Prozessgraphen.

Diese Bachelorarbeit widmet sich der Thematik der Visualisierung von Prozessgraphen im Rahmen des Process Mining unter Anwendung des Sugiyama-Frameworks. Ziel ist es, verschiedene Algorithmen zu untersuchen und deren Nutzen hinsichtlich Laufzeit und Übersichtlichkeit der erzeugten Zeichnungen zu bewerten. Dabei wird besonders auf die Herausforderungen eingegangen, die bei der Darstellung großer und komplexer Prozesse auftreten.

Im Verlauf dieser Arbeit werden zunächst die Grundlagen des Process Mining und der Prozessvisualisierung sowie die Transformation der Aktivitätsdaten erläutert. Danach folgt eine detaillierte Betrachtung der einzelnen Schritte des Sugiyama-Frameworks, welches als grundlegendes Vorgehensmodell für die Erstellung des Graph-Layouts dient. Anschließend werden verschiedene Darstellungsaspekte, wie die Form der gezeichneten Kanten, behandelt. Zuletzt werden die verschiedenen Algorithmen und Darstellungsoptionen anhand von Laufzeit und Optik der Zeichnungen bewertet und gegeneinander abgewogen. Ziel dieser Arbeit ist die Entwicklung einer Methode zur übersichtlichen Darstellung von Prozessgraphen auf Basis des Sugiyama-Framworks. Ein besonderer Fokus wird dabei auf das Brechen von Kreisen durch Lösung des Minimum Feedback Arc Set Problem gelegt, welches für die einheitliche Flussrichtung des Graphen von Bedeutung ist. Bei der Darstellung des Graphen sollen anschließend Textfelder für die Knoten, Beschriftungen der Kanten und Pfeilspitzen zur Richtungsanzeige adäquat integriert werden.

Ein zentrales Anliegen ist dabei, die Kantenkreuzungen zu minimieren, da diese nach Purchase [Pur00] die Lesbarkeit und Verständlichkeit der Graphen erheblich beeinträchtigen können. Ebenso wird darauf geachtet, eine möglichst einheitliche Flussrichtung der Kanten zu erhalten und den Zeichenbereich möglichst klein zu halten, um eine kompakte und übersichtliche Darstellung zu gewährleisten. Der Zeichenalgorithmus wird in Java [JSGB00] umgesetzt und anhand verschiedener Prozesse von unterschiedlichem Umfang getestet.

1.1. Process Mining

Process Mining ist ein innovatives Forschungs- und Anwendungsgebiet (siehe [vdAAdM+11, vdA12]), das sich an der Schnittstelle von Datenanalyse, Geschäftsprozessmanagement und Informationstechnologie befindet. Es zielt darauf ab, reale Geschäftsprozesse auf Basis digitaler Spuren, die in IT-Systemen hinterlassen werden, zu analysieren, zu visualisieren und zu optimieren. Diese digitalen Spuren, oft als „Event Logs“ bezeichnet, enthalten Informationen über die verschiedenen Aktivitäten, die in einem Geschäftsprozess stattfinden.

Durch die Anwendung von Process Mining können Unternehmen wertvolle Einblicke in ihre tatsächlichen Prozessabläufe gewinnen, die oft von den dokumentierten Prozessen abweichen. Dies ermöglicht es, ineffiziente Prozessschritte, Engpässe und Abweichungen zu identifizieren und gezielte Maßnahmen zur Prozessverbesserung zu ergreifen. Process Mining verbindet somit die Stärken der klassischen Prozessmodellierung mit den Möglichkeiten der Big Data-Analyse.

Ein wichtiges Instrument ist dabei die Visualisierung der Prozesse in Form von Prozessgraphen, denn hier können Umwege, Schleifen oder unerwartete Abbrüche des Prozesses direkt erkannt werden. Dies erleichtert die Prozessanalyse erheblich. Da die Prozesse oft sehr lang und komplex sind und sich laufend verändern, wäre eine manuelle Zeichnung der Prozessgraphen sehr aufwendig und ineffizient, weshalb Algorithmen zur automatischen Zeichnung komplexer Prozessgraphen nötig sind.

Im Folgenden wird zunächst die Vorgehensweise bei der Erfassung der Daten im Rahmen des Process Mining erläutert. Im Anschluss werden die grundlegenden Begriffe und Datenstrukturen des Process Mining eingeführt und deren Zusammenhang untereinander dargestellt.

1.1.1. Aktivitäts- und Falltabelle

Die benötigten Daten liegen in den Quellsystemen der Unternehmen vor. Bevor sie verwendet werden können, muss allerdings eine Vorverarbeitung vorgenommen werden. In diesem Schritt werden alle nötigen Daten extrahiert, um daraus eine Aktivitäts- und eine Falltabelle („Event Log“) zu erstellen. Die Vorgehensweise ist dabei wie folgt:

1. Extraktion der Daten aus IT-Systemen (i.d.R. ERP[1]-Systeme wie SAP, CRM[2]-Systeme oder spezielle Workflow-Management-Systeme)

2. Transformation der Daten in das notwendige Format (Entfernung von Duplikaten, Ergänzung fehlender Werte, Vereinheitlichung von Zeitstempeln, etc.)

[1] Enterprise-Resource-Planning
[2] Customer-Relationship-Management

3. Erstellung und Laden eines Datenmodells aus Aktivitäts- und Falltabelle sowie assoziierten Metadaten

Die *Aktivitätstabelle* enthält Daten über einzelne Prozessaktivitäten. Jede Zeile repräsentiert eine ausgeführte Aktivität und enthält mehrere Spalten, die verschiedene Attribute der Aktivität beschreiben. Die notwendigen Spalten sind Fall-ID, Aktivitätsbezeichnung und Zeitstempel, weitere mögliche Attribute sind assoziierte Personen, Kosten, etc.

Fall ID	Aktivitätsbezeichnung	Zeitstempel
1	Erste Aktivität	2024-05-19 11:15:53
1	Weitere Aktivität	2024-05-19 11:23:41
1	Letzte Aktivität	2024-05-21 18:32:23
2	Eine Aktivität	2024-06-01 08:13:51
2	Weitere Aktivität	2024-06-02 13:09:02
...	...	...
1000	Noch eine Aktivität	2024-08-25 09:50:07
1000	Prozessschritt 42	2024-08-25 09:55:24
1000	Letzte Aktivität	2024-08-25 12:23:18

Tab. 1.1.: Eine beispielhafte Aktivitätstabelle

Eine *Falltabelle* enthält aggregierte Daten der einzelnen Fälle (Prozessinstanzen). Jede Zeile repräsentiert einen Fall und enthält Informationen über die Gesamtausführung dieses Falls. Die Falltabelle besteht im minimalen Fall nur aus der Fall-ID-Spalte, es werden aber typischerweise weitere Attribute wie Ergebnis, zugehöriger Kunde, oder andere Metadaten gespeichert.

Fall ID	Erfolg	...	Kunde ID
1	1	...	024581
2	1	...	536888
3	0	...	191241
...	...	...	...
1000	1	...	151632

Tab. 1.2.: Eine beispielhafte Falltabelle

1.1.2. Prozessvarianten

Ein Geschäftsprozess kann in der Praxis auf verschiedene Weise ablaufen. Diese Ablaufwege werden als Prozessvarianten bezeichnet. Eine *Variante* ist dabei definiert als eine bestimmte Abfolge von Prozessschritten. Zwei Fälle entsprechen genau dann derselben Variante, wenn sie die gleichen Aktivitäten in der gleichen Reihenfolge enthalten, unabhängig von ihrer jeweiligen Durchlaufzeit.

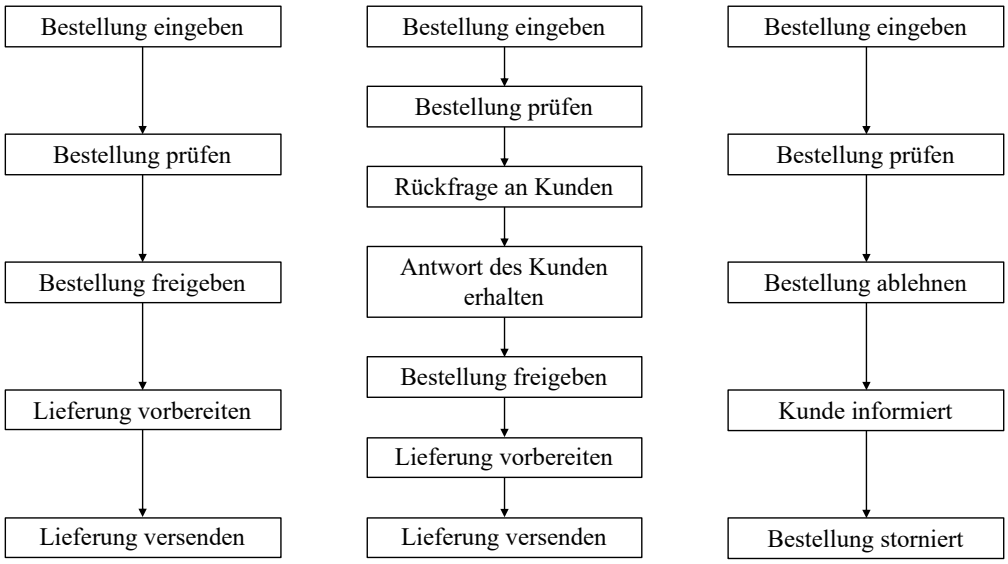


Abb. 1.1.: Drei mögliche Varianten eines einfachen Bestellprozesses

1.2. Prozessgraphen

Prozessgraphen sind eine besondere Form von Graphen. Ein Graph ist eine mathematische Struktur, die aus einer Menge von Knoten und einer Menge von Kanten besteht. Knoten repräsentieren Entitäten, während Kanten die Verbindungen oder Beziehungen zwischen diesen Entitäten darstellen. Graphen werden verwendet, um eine Vielzahl von Problemen in Informatik, Mathematik und vielen anderen Disziplinen zu modellieren.

Im Folgenden werden zunächst die grundlegenden Begriffe der mathematischen Graphentheorie (siehe [Die17]) eingeführt. Im Anschluss werden die Besonderheiten von Prozessgraphen dargestellt und die damit zusammenhängenden Herausforderungen bei der algorithmischen Zeichnung erläutert.

1.2.1. Grundlegende Begriffe der Graphentheorie

Ein *Graph* G ist ein Paar $G = (V, E)$, wobei $V(G)$ bzw. V die Menge der *Knoten* und $E(G)$ bzw. E die Menge der *Kanten* ist. Knoten sind die grundlegenden Einheiten eines Graphen und repräsentieren eine Entität, im Falle von Prozessgraphen eine Aktivität im Prozess. Kanten stellen Verbindungen zwischen Knoten dar und können gerichtet ($E(G) \subseteq V(G) \times V(G)$) oder ungerichtet ($E(G) \subseteq \binom{V(G)}{2}$) sein. Ein Graph ist *gerichtet*, falls alle Kanten eine Richtung haben und wird in diesem Fall auch als Digraph bezeichnet (siehe Abb. 1.2). Sofern nicht anders definiert, stellen im Folgenden $n := |V(G)|$ und $m := |E(G)|$ die Anzahl der Knoten und Kanten eines Graphen G dar.

Zwei Knoten $u, v \in V(G)$ sind *adjazent*, wenn sie durch eine Kante verbunden sind. Im Falle eines gerichteten Graphen sind $u, v \in V(G)$ adjazent, falls $(u, v) \in E(G)$ oder $(v, u) \in E(G)$, im Falle eines ungerichteten Graphen falls $\{u, v\} \in E(G)$. Der *Grad* $d(v)$ eines Knotens $v \in V(G)$ ist die Anzahl der Kanten, die diesen Knoten mit anderen verbindet. Für einen gerichteten Graphen ist dies die Summe aus *Eingangs- und Ausgangsgrad* eines Knotens

$$d(v) = d^+(v) + d^-(v)$$

wobei der Eingangs- und Ausgangsgrad eines Knotens der Anzahl ein- und ausgehender Kanten entsprechen, d.h.

$$d^+(v) = |\{(u, v) \in E(G) : u \in V(G)\}|,\ d^-(v) = |\{(v, u) \in E(G) : u \in V(G)\}|$$

Ein *Weg* ist eine Folge von Knoten, welche jeweils durch eine Kante verbunden sind. In einem gerichteten Graphen mit $(v_i, v_{i+1}) \in E$ für alle $i = 1, \ldots, n-1$ wären mögliche Wege beispielsweise

$$((v_1, v_2), (v_2, v_3), \ldots, (v_{n-1}, v_n)) \text{ oder } ((v_1, v_2), (v_2, v_3), \ldots, (v_{k-1}, v_k)) \text{ mit } k < n$$

Ein *Kreis* ist ein Weg, der am gleichen Knoten beginnt und endet. Ein Graph ist *zusammenhängend*, wenn es für jedes Knotenpaar (u, v) einen Weg von u nach v oder von v nach u gibt.

Ein *Teilgraph* $\tilde{G} = (\tilde{V}, \tilde{E})$ von G ist ein Graph, dessen Knoten und Kanten eine Teilmenge eines größeren Graphen bilden, d.h. $\tilde{V} \subset V(G)$ und $\tilde{E} \subset E(G)$ und zusätzlich im Falle eines ungerichteten Graphen $\tilde{E} \subset \binom{\tilde{V}}{2}$ oder im Falle eines gerichteten Graphen $\tilde{E} \subset \tilde{V} \times \tilde{V}$.

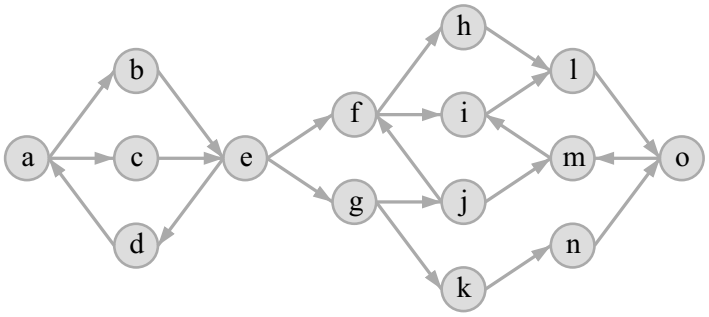


Abb. 1.2.: Beispiel eines gerichteten, zusammenhängenden und nicht kreisfreien Graphen.

1.2.2. Besonderheiten von Prozessgraphen

Die Darstellung von Prozessgraphen stellt eine besondere Herausforderung dar, da sie über die Anforderungen normaler Graphen hinausgeht. Bei normalen Graphen sind ty-

pischerweise nur die Knoten und Kanten zu berücksichtigen. Prozessgraphen hingegen erfordern zusätzlich die Einbindung von Textfeldern für die Knoten, welche Informationen über die jeweiligen Prozessschritte liefern.

Ein weiteres wichtiges Element sind die Beschriftungen der Kanten, welche Auskunft über die Verbindungen zwischen den Knoten geben. Dies können beispielsweise die Häufigkeit oder die Durchlaufzeit eines Übergangs sein. Diese Beschriftungen müssen präzise platziert werden, um Verwirrung zu vermeiden und die Lesbarkeit zu gewährleisten.

Die Richtungsanzeige durch Pfeilspitzen ist ebenfalls essentiell, um die Flussrichtung der Prozesse zu verdeutlichen. Dies hilft, den Ablauf und die Sequenz der Aktivitäten verständlich darzustellen. Dabei ist besonders zu beachten, dass möglichst viele Kanten in dieselbe Richtung zeigen und trotz Zyklen im Prozess nur wenige Pfeile entgegen der allgemeinen Prozessrichtung verlaufen, um die Lesbarkeit und Verständlichkeit zu garantieren.

Auch die Form der Kanten spielt eine Rolle, da sie gerade, abgerundet oder gebogen verlaufen können und damit die Übersichtlichkeit verbessern können. Vor allem bei gebogenen Kanten in Form von Bézierkurven stellt aber die Überschneidungsfreiheit mit Textfeldern und anderen Kanten eine Herausforderung dar.

Insgesamt erfordert die automatische Visualisierung von Prozessgraphen einen umfangreichen Algorithmus, welcher alle genannten Aspekte berücksichtigt, um eine intuitive und präzise Darstellung komplexer Prozessabläufe zu gewährleisten. Dies ist entscheidend für die Analyse und Optimierung der Geschäftsprozesse, da es den Betrachtern ermöglicht, die Prozessstruktur schnell zu erfassen und mögliche Verbesserungsbereiche zu identifizieren.

In Abb. 1.3 wird ein Prozessgraph dargestellt. Weitere exemplarische Zeichnungen befinden sich im Anhang unter A.2.

1.3. Verwandte Arbeiten

Im Bereich der Visualisierung von Prozessgraphen gibt es bereits eine Reihe von Arbeiten, die sich mit verschiedenen Aspekten der Erstellung und Optimierung von Graph-Layouts beschäftigen. Im Folgenden werden einige wichtige Arbeiten vorgestellt und ihre Schwerpunkte sowie wesentlichen Erkenntnisse zusammengefasst, wobei auch berücksichtigt wird, inwieweit das Sugiyama-Framework verwendet wurde.

Die Arbeit „An automatic layout algorithm for BPEL processes“ von Albrecht et al. [AEHK10] stellt einen automatischen Layout-Algorithmus für BPEL[3]-Prozesse vor. BPEL ist eine standardisierte Sprache zur Beschreibung von Geschäftsprozessen, die auf Web

[3]Business Process Execution Language

Services basieren. Die Autoren adressieren die Herausforderung, komplexe BPEL-Prozessmodelle übersichtlich und verständlich zu visualisieren.

Albrecht et al. entwickeln einen Algorithmus, der auf heuristischen Methoden basiert und verschiedene Layout-Kriterien wie Minimierung von Kantenkreuzungen und gleichmäßige Verteilung der Knoten berücksichtigt. In der Arbeit wird das Sugiyama-Framework teilweise verwendet. Während der Algorithmus einige Prinzipien des Sugiyama-Frameworks wie die Schichtzuweisung und die Minimierung von Kantenkreuzungen übernimmt, weicht er in anderen Bereichen ab, um spezifische Anforderungen von BPEL-Prozessen zu erfüllen.

Gschwind et al. präsentieren in „A Linear Time Layout Algorithm for Business Process Models“ [GPZ+14] einen linearen Layout-Algorithmus für Geschäftsprozessmodelle. Die Arbeit fokussiert sich auf die Effizienz des Algorithmus, um große Prozessmodelle in akzeptabler Zeit zu visualisieren.

Der von Gschwind et al. entwickelte Algorithmus setzt auf eine lineare Zeitkomplexität und nutzt spezielle Techniken zur Reduktion von Kantenkreuzungen und zur Verbesserung der Symmetrie der Zeichnungen. Der Algorithmus legt besonderen Wert auf eine performante Umsetzung, um auch bei sehr großen Modellen schnelle Ergebnisse zu liefern. Das Sugiyama-Framework wird in dieser Arbeit nur teilweise verwendet. Einige Kernkonzepte wie die Minimierung von Kantenkreuzungen und die hierarchische Anordnung der Knoten werden integriert. Jedoch wird das Framework nicht vollständig implementiert, um die lineare Zeitkomplexität zu gewährleisten.

Diese beiden Ansätze dienen primär der Prozessmodellierung, welche vor der Überführung eines Prozesses in das reale Geschäftsumfeld stattfindet. Die Modellierung der Prozesse ist darauf ausgerichtet, neue Prozesse zu planen und dabei vorhandene Ressourcen zu berücksichtigen. Dies geschieht mittels BPEL- oder BPMN[4]-Prozessmodellen, die nicht nur die Prozessschritte, sondern auch zusätzliche Elemente wie Entscheidungsknoten oder Ressourcen in die Zeichnungen integrieren.

Der Ansatz des Process Mining hingegen verfolgt ein anderes Ziel: Hierbei werden die Abläufe bestehender Prozesse analysiert, um deren Effizienz und Konformität zu bewerten, insbesondere nachdem die Prozesse bereits über einen längeren Zeitraum durchgeführt wurden. Elemente aus den Modellierungssprachen, wie Entscheidungsknoten, sind in diesem Kontext nicht erforderlich. Stattdessen konzentriert sich die Analyse ausschließlich auf die Prozessaktivitäten und deren Verbindungen. Dazu werden wichtige Prozesskennzahlen, wie die Durchlaufzeit des Prozesses oder die Anzahl der Aktivitäten und Varianten in die Darstellung integriert, was in den Modellierungssprachen nicht vorgesehen ist. Aus diesem Grund sind die beiden oben genannten Ansätze für die vorliegende Arbeit nicht anwendbar.

Eine weitere Arbeit zur automatischen Prozessvisualisierung ist die Arbeit „A Stable

[4]Business Process Model and Notation

Graph Layout Algorithm for Processes“ [MSW19] von Mennens et al., welche die Erstellung eines stabilen Layout-Algorithmus für Prozessgraphen beschreibt. Diese erzeugt Prozessgraphen, wie sie im Rahmen des Process Mining benötigt werden. Stabilität bedeutet in diesem Kontext, dass ähnliche Eingabeprozesse zu ähnlichen Layouts führen, was die Wiedererkennbarkeit und Nachvollziehbarkeit der Prozesszeichnungen erhöht. Die häufigste Prozessvariante wird gerade in der Mitte der Zeichnung platziert und seltenere Varianten daneben angeordnet.

Der Algorithmus von Mennens et al. berücksichtigt verschiedene Optimierungskriterien wie die Minimierung von Kantenkreuzungen, die gleichmäßige Verteilung der Knoten und die Symmetrie der Zeichnungen. Ein besonderes Merkmal ist die Beibehaltung der relativen Positionen von Knoten bei minimalen Änderungen im Prozessgraphen. Das Sugiyama-Framework wird in dieser Arbeit vollständig verwendet. Die Stabilität und die Symmetrie der Zeichnungen werden durch zusätzliche Heuristiken und Optimierungen erreicht, die auf dem Sugiyama-Framework aufbauen.

Durch die zusätzlichen Anforderungen der Symmetrie und Stabilität ist der Algorithmus von Mennens et al. jedoch komplexer und aufwendiger als die Visualisierung mittels des klassischen Sugiyama-Frameworks. Im Folgenden sollen diese Anforderungen daher vernachlässigt werden, das Layout wird ohne Berücksichtigung der Wichtigkeit verschiedener Varianten erzeugt. Wichtige Varianten sollen stattdessen farblich hervorgehoben werden, und mittels verschiedener Kantenformen soll die Nachverfolgbarkeit des Prozessablaufs verbessert werden. Ein Ansatz dieser Art ist in der Literatur bisher noch nicht zu finden.

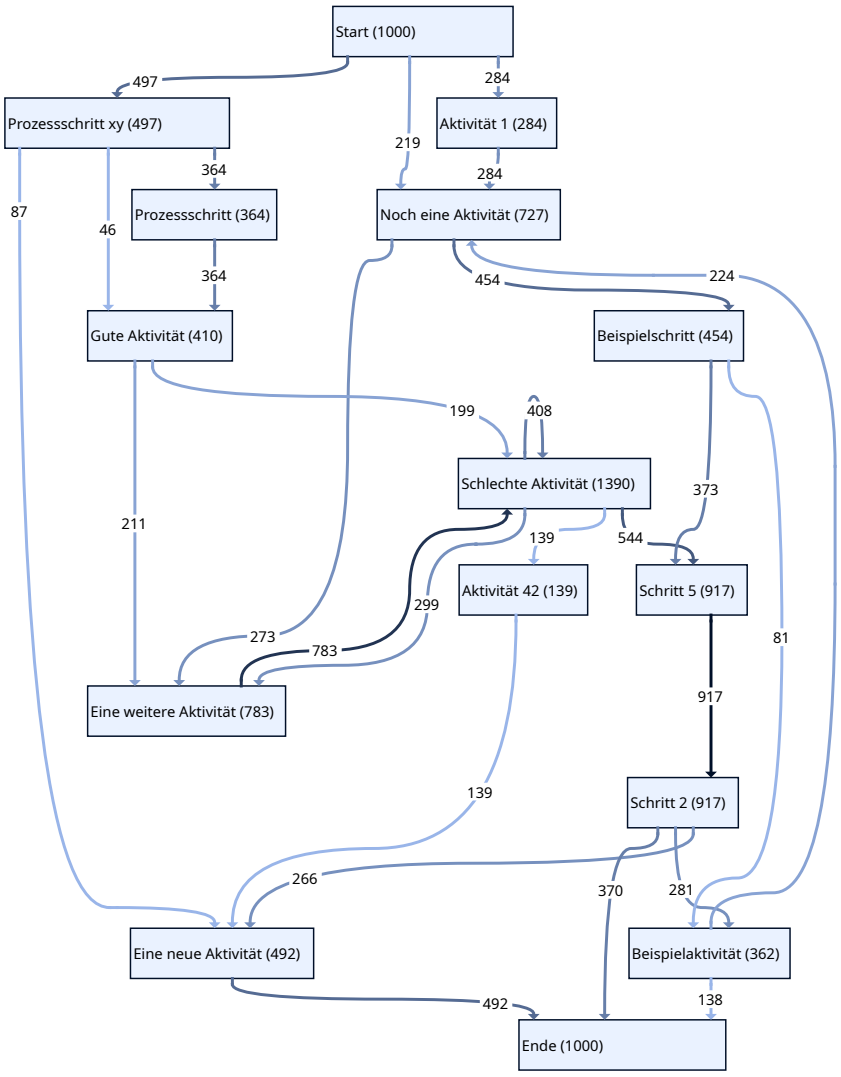


Abb. 1.3.: Prozessgraph eines Beispiel-Prozesses. Zeichnungen dieser Art werden im Rahmen der Arbeit automatisch erstellt. Diese Zeichnung wurde bereits mithilfe des entwickelten Zeichen-Algorithmus erzeugt.

2. Algorithmus

Im folgenden Kapitel werden die einzelnen Schritte des Zeichenalgorithmus erläutert. Dazu erfolgt zunächst eine Transformation der Prozessdaten in die Form eines Graphen mit Knoten und Kanten. Im Anschluss werden die Schritte des Sugiyama-Frameworks sowie die verwendeten Algorithmen dargestellt. Dabei wird jeweils auf die Besonderheiten von Prozessgraphen und deren Handhabung eingegangen.

2.1. Vorverarbeitung der Prozessdaten

Bevor die Zeichnung des Prozessgraphen erfolgen kann, müssen die Daten zunächst in einen Graphen mit Knoten und Kanten umgewandelt werden, denn zunächst liegen die Prozessdaten nur in Form von Aktivitätstabellen vor. Diese enthalten für jede Aktivität die ID des zugehörigen Falles, eine Aktivitätsbezeichnung sowie einen Zeitstempel. Daher wird zunächst eine Aggregation der Fälle nach dem Folgenden Schema vorgenommen:

1. Sammlung aller Fälle aus der Aktivitätstabelle
2. Sortierung der Aktivitäten nach Fällen
3. Zeitliche Sortierung der Aktivitäten je Fall
4. Hinzufügen der Pseudoaktivitäten „Start“ und „Ende“

Der letze Schritt ist notwenig, damit im späteren Prozessgraphen alle Pfade am selben Punkt starten und enden und trägt wesentlich zu Übersichtlichkeit des Graphen bei. So werden Fallobjekte erstellt, welchen ihre jeweiligen Aktivitäten in der richtigen Reihenfolge enthalten.

Zusätzlich wird auf Ebene der Aktivitäten für jeden Aktivitätstyp ein Aktivitätsobjekt erstellt und diesem die ein- und ausgehenden Kanten aus allen betracheten Fällen zugeordnet. So kann im Graph später bestimmt werden, wie häufig eine bestimmte Aktivität in allen Fällen auftritt und welche Aktivitäten darauf folgen. Außerdem kann für zwei bestimmte, aufeinanderfolgende Aktivitäten A und B bestimmt werden, wie lange es im Mittel nach Abschluss von A dauert, bis B beginnt. Dies entspricht der mittleren Durchlaufzeit einer Kante. Diese Informationen sind wichtig für die spätere Untersuchung der Prozesse und die Prozessoptimierung.

Zuletzt werden aus den Fällen die verschiedenen Prozessvarianten extrahiert und deren Häufigkeit bestimmt. Dies ist ebenfalls wichtig für die Darstellung, da im Rahmen des Process Mining oftmals aus Übersichtlichkeitsgründen nicht alle Prozessvarianten, sondern z.B. nur die häufigsten 20% der Varianten betrachtet werden, denn Prozessoptimierungen für diese Varianten versprechen größere Erfolge. An dieser Stelle können die zu betrachtete Fälle anhand ihrer Zugehörigkeit zu einer Variante gefiltert werden.

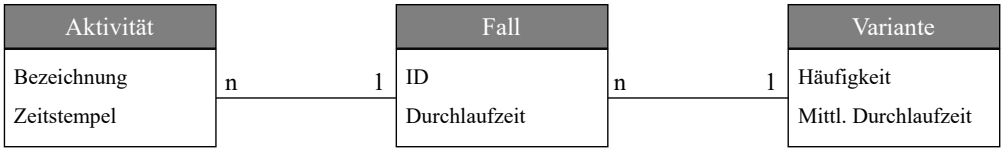


Abb. 2.1.: Zusammenhang zwischen Aktivitäten, Fällen und Varianten

2.2. Sugiyama-Framework

Das Sugiyama-Framework, benannt nach Kozo Sugiyama [STT81], ist ein weit verbreitetes Verfahren zur hierarchischen Darstellung gerichteter Graphen, das auch in der Visualisierung von Prozessgraphen Anwendung findet. Es wurde entwickelt, um eine klare und ästhetische Anordnung komplexer Graphen zu ermöglichen. Durch die systematische Reduzierung von Kantenkreuzungen und die Optimierung der Kantenlängen trägt es dazu bei, dass umfangreiche und vielschichtige Prozessmodelle verständlich dargestellt werden. Das Framework verfolgt einen mehrstufigen Ansatz, der folgende Schritte umfasst (siehe Abb. 2.2):

1. Kreise brechen (Cycle Breaking): Bei einigen der gerichteten Kanten wird die Richtung umgekehrt, sodass der Graph insgesamt kreisfrei ist. Dies ist nur notwendig für die Erstellung des Layouts, die Kanten werden in der Zeichnung wieder in der ursprünglichen Richtung dargestellt. Um eine möglichst kleine Menge solcher Kanten zu identifizieren, muss das Minimum Feedback Arc Set Problem [You63] gelöst werden.

2. Ebenen zuweisen (Layer Assignment): Die Knoten des Graphen werden auf verschiedene Ebenen verteilt, um eine hierarchische Struktur zu schaffen. Dies erleichtert das Verständnis der über- und untergeordneten Beziehungen zwischen den Knoten.

3. Dummy-Knoten einfügen (Dummy Vertices Insertion): Für Kanten, welche über eine Ebene hinausgehen, werden Knoten auf den dazwischenliegenden Ebenen eingefügt.

4. Knoten sortieren (Node Ordering): Innerhalb jeder Schicht werden die Knoten so angeordnet, dass die Anzahl der Kantenkreuzungen minimiert wird.

5. Knotenplatzierung (Node Placement): Schließlich werden die Knoten auf den einzelnen Schichten so platziert, dass die Gesamtstruktur des Graphen ästhetisch und leicht verständlich ist. Dies umfasst auch die Anpassung der Kantenform und die Integration von Pfeilspitzen zur Richtungsanzeige.

Für jeden Schritt stehen verschiedene Algorithmen zur Verfügung. Der hier entwickelte Ansatz orientiert sich stark an dem von Zink et al. [ZWBW22] vorgestellten Algorithmus, welcher die Zeichnung ungerichteter Graphen mit Ports, zur Bündelung von Kanten an den Knoten, ermöglicht und ebenfalls auf dem Sugiyama-Framework basiert. Da bei [ZWBW22] aufgrund der ungerichteten Kanten der erste Schritt des Kreise-Brechens nicht nötig war, wurde dieser im Rahmen dieser Arbeit entwickelt. Es werden drei aus der Literatur bekannte Algorithmen zur Entfernung der Kreise vorgestellt, implementiert und später gegeneinander abgewogen. Weiterhin sind die von [ZWBW22] erzeugten Zeichnungen sehr breit, während bei Prozessgraphen wegen der besseren Lesbarkeit schmalere Zeichnungen von Vorteil sind. Daher wird im zweiten Schritt eine Anpassung vorgenommen, welche zu schmaleren Zeichnungen führt. Die letzten drei Schritte werden analog zu [ZWBW22] durchgeführt.

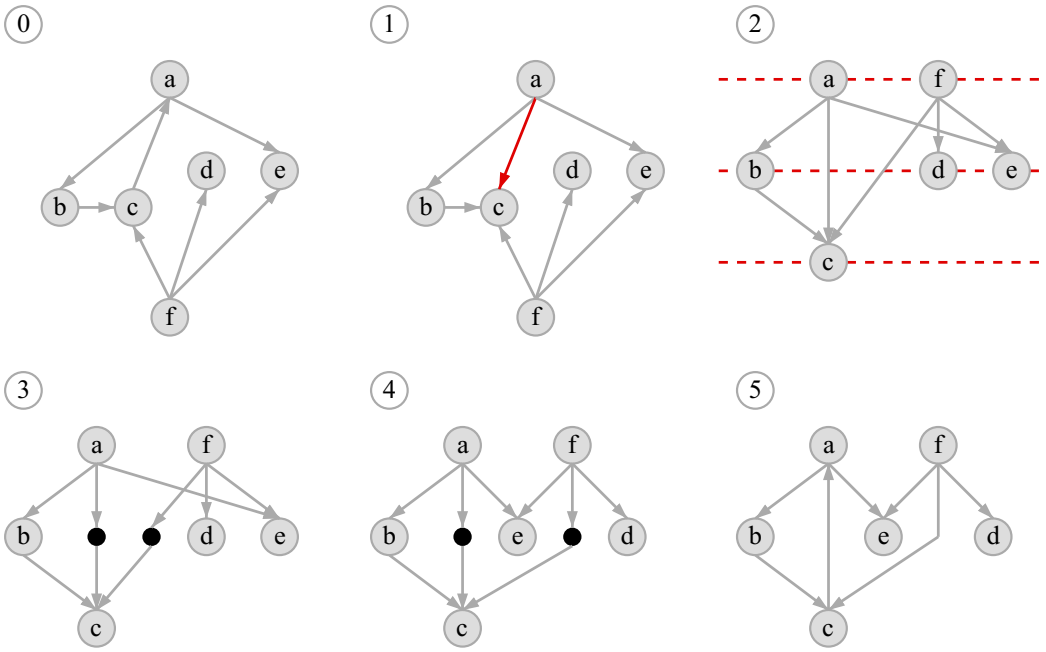


Abb. 2.2.: Schritte des Sugiyama-Frameworks zur automatischen Zeichnung von Graphen

2.2.1. Kreise brechen

Um eine hierarchische Darstellung des Graphen zu ermöglichen, müssen zunächst alle Kreise entfernt werden. Dies geschieht durch übergangsweise Umkehr der Richtung einer bestimmten Menge von Kanten, dem sogenannten feedback arc set (FAS). In der späteren Zeichnung werden die Kanten in ihrer ursprünglichen Richtung dargestellt. Dabei ist wichtig, eine möglichst kleine Menge von Kanten umzukehren, damit in der späteren Darstellung möglichst wenig Kanten entgegen der allgemeinen Flussrichtung verlaufen.

Definition 2.1. *Ein* feedback arc set $R \subset E(G)$ *ist eine Menge von Kanten, deren Umkehr dazu führt, dass G kreisfrei ist. Ein minimum feedback arc set* $R^* \subset E(G)$ *ist eine minimale Menge solcher Kanten.*

Die Bestimmung eines minimalen FAS stellt ein NP-schweres Problem dar (siehe [Kar72]), daher werden im Folgenden einige heuristische Lösungsansätze vorgestellt. Ein erster Ansatz stammt von Eades, Lin und Smyth [ELS93] und wurde bereits 1993 vorgestellt. Dazu sind zunächst einige Vorüberlegungen notwendig.

Definition 2.2. *Sei G ein Graph mit den Knoten* $V(G) = \{v_1, \ldots, v_n\}$ *und* $s = v_{i_1} v_{i_2} \ldots v_{i_n}$ *eine beliebige lineare Anordnung aller Knoten. Eine* optimale lineare Anordnung *ist eine Knotensequenz* $\tilde{s} = v_{i_1} \ldots v_j \ldots v_k \ldots v_{i_n}$*, für welche die Menge rückwärts gerichteter Kanten* $\{(v_k, v_j) \in E(G) : k > j\}$ *minimal ist. (siehe Abb. 2.3)*

Jede lineare Anordnung s von Knoten induziert ein FAS

$$R_s = \{(v_j, v_k) \in E : j > k\}$$

Dementsprechend induziert eine optimale lineare Anordnung ein minimales FAS. Dies gilt auch umgekehrt. Daher sind die beiden Probleme, die Bestimmung einer optimalen linearen Anordnung der Knoten sowie das Finden eines minimalen FAS, äquivalent. Im Folgenden wird ersteres Problem betrachtet.

Falls $d^+(u) = 0$ für einen Knoten $u \in V(G)$ gilt, handelt es sich bei u um eine *Senke*, mit $d^-(u) = 0$ ist u eine *Quelle*. Der vorgeschlagene Algorithmus GreedyFAS entfernt zunächst alle Quellen bzw. Senken und fügt sie den zu Beginn leeren Sequenzen s_1 bzw. s_2 hinzu. Im Anschluss werden sukzessive die Knoten mit maximalem Gewicht $\delta(u) = d^+(u) - d^-(u)$ entfernt und ebenfalls zu s_1 hinzugefügt, bis keine Knoten mehr vorhanden sind. Eine lineare Sortierung entsteht durch Zusammenfügen der Sequenzen $s_1 s_2$.

Mittels dieses Ansatzes lässt sich für die meisten Anwendungsfälle ein hinreichend kleines Feedback Arc Set bestimmen, welches aber nicht zwingend minimal ist. Für das induzierte Feedback Arc set gilt $|R_s| < m/2 - n/6$. Der Algorithmus benötigt eine Laufzeit von $O(m)$ und dabei einen Speicherplatz von $O(m)$, was ihn zu einer sehr effizienten

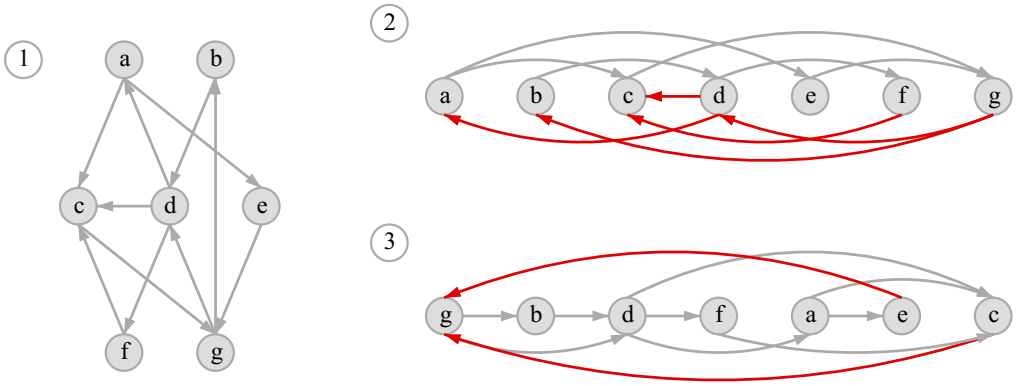


Abb. 2.3.: Zwei mögliche lineare Anordnungen des Graphen (1). (2) zeigt eine beliebige Reihenfolge und (3) die optimale Reihenfolge der Knoten. Das induzierte FAS besteht aus den rückwärts gerichteten Kanten und ist rot gekennzeichnet.

Lösung macht. Es gibt jedoch weitere Ansätze, durch welche ein kleineres Feedback arc set bestimmt werden kann. Daher wird im Folgenden ein Ansatz zur Optimierung einer linearen Sortierung vorgestellt, mit welchem sich die Größe des resultierenden Feedback Arc Set noch verringern lässt.

Dieser Ansatz stammt von von Simpson, Srinivasan und Thome [SST16] und basiert auf der Sortierung einer beliebigen linearen Anordnung der Knoten. Alg. SortFAS nimmt eine Optimierung der Reihenfolge der Knoten vor, wobei die Menge rückwärts gerichteter Kanten reduziert werden soll. Dazu werden die Knoten der Reihe nach durchlaufen und jeweils zwischen den bereits positionierten Knoten eingefügt, sodass möglichst wenige rückwärts gerichtete Kanten entstehen. Der Algorithmus benötigt eine Laufzeit von $O(n^2)$.

Ein dritter Ansatz stammt von Geladaris, Lionakis und Tollis [GLT22] und basiert auf dem oft in Suchmaschinen verwendeten PageRank-Algorithmus. Bei diesem Ansatz wird zunächst aus einem gerichteten Graphen ein gerichteter Kantengraph erstellt, welcher die Bewertung der Verbundenheit der Kanten aus $E(G)$ mit dem restlichen Graphen ermöglicht. Im Anschluss wird ein PageRank-Score für jeden Knoten des Kantengraphen berechnet. Dieser Score soll die Wichtigkeit der jeweiligen Knoten im Kantengraph darstellen. Die Knoten mit hoher Wichtigkeit werden dann nach und nach zum FAS hinzugefügt, bis der zugrundeliegende Graph kreisfrei ist.

Definition 2.3. *Der* gerichtete Kantengraph $L(G)$ *eines gerichteten Graphen* G *enthält einen Knoten für jede Kante* $(u, v) \in E(G)$ *und eine Kante für jeden Pfad der Länge zwei* $((w, u), (u, v)) \in E(G)^2$.

Für einen gerichteten Kantengraphen $L(G)$ gilt

$$|V(L(G))| = |E(G)| \quad \text{und} \quad |E(L(G))| = \sum_{u \in V} (d^+(u) \cdot d^-(u))$$

Insgesamt folgt

$$L(G) \in O\left(|E(G)| + \sum_{u \in V} (d^+(u) \cdot d^-(u))\right)$$

Abb. 2.4 zeigt einen beispielhaften gerichteten Kantengraphen. Der Algorithmus Get-

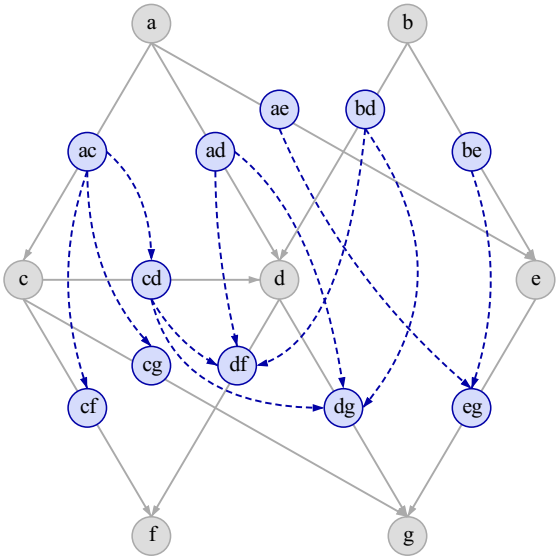


Abb. 2.4.: Ein beliebiger Graph mit zugehörigem Kantengraphen. Die Kanten werden als Knoten dargestellt und Wege der Länge zwei bilden die Kanten des Kantengraphen.

LineGraph bestimmt den Kantengraphen eines gegebenen Graphen G mittels eines auf Tiefensuche basierenden Ansatzes und benötigt eine Laufzeit von $O(n + m + |L(G)|)$. Dabei wird für jede Kante $(u, v) \in E$ ein Knoten in $L(G)$ erstellt und die dazugehörigen Kanten rekursiv hinzugefügt.

Der PageRank-Algorithmus stammt von Brin & Page [BP98] aus dem Jahr 1998 und wird oft in Suchmaschinen verwendet, um die Wichtigkeit verschiedener Webseiten zu bestimmen. Dabei erhalten die Seiten einen Score basierend auf der Anzahl ein- und ausgehender Links von und zu anderen Seiten.

Dieser Ansatz wird hier genutzt, um besonders wichtige Knoten des Graphen zu identifizieren. Dazu wird PageRank verwendet, um einen PageRank-Score für jeden Knoten des Kantengraphen zu berechnen, welcher die Verbundenheit mit anderen Knoten aus $V(L(G))$ darstellen soll. Ein höherer Score eines Knotens aus $V(L(G))$ bedeutet dabei eine hohe Relevanz der dazugehörigen Kante aus $E(G)$. Die Idee besteht nun darin, Kanten mit hoher Wichtigkeit sukzessive zu entfernen, da die Entfernung solcher Kanten aufgrund ihrer starken Verbundenheit zu anderen Knoten mit hoher Wahrscheinlichkeit Kreise bricht.

Der Algorithmus weist zunächst jedem Knoten aus $V(L(G))$ die Wichtigkeit $1/n$ zu und passt diese anschließend über eine feste Anzahl an Iterationen an. Die Anzahl der Iterationen kann dabei frei gewählt werden, wobei eine größere Anzahl an Iterationen bessere Ergebnisse erzielt. Nach den Autoren reichen fünf Iterationen aus, um hinreichend gute Ergebnisse zu erzielen. PageRank benötigt dabei eine Rechenzeit von $O(k(n+m))$, wobei k die Anzahl der Iterationen darstellt.

Der Algorithmus PageRankFAS ist eine Kombination aus den bereits vorgestellten Algorithmen GetLineGraph und PageRank und berechnet ein FAS zu einem Eingabegraphen, indem zunächst der Kantengraph bestimmt wird und anschließend wichtige Kanten aus $E(G)$ entfernt werden, bis G kreisfrei ist. Die Autoren von [GLT22] wenden diesen Algorithmus jeweils einzeln nur auf die stark verbundenen Komponenten des Graphen an. Von dieser Variante wird hier Abstand genommen, da Prozessgraphen oftmals nur kleine stark verbundene Komponenten aufweisen und der Algorithmus auch auf dem gesamten Graphen gute Ergebnisse erzielt.

2.2.2. Level zuweisen

In diesem Schritt wird den Knoten des Graphen G jeweils eine Ebene l zugewiesen, sodass für zwei Knoten $v, w \in V(G)$ mit $(v, w) \in E(G)$ gilt: $l(v) < l(w)$. Hier wird dem Algorithmus von Zink et al. [ZWBW22] gefolgt und ebenfalls ein Network-Simplex-Ansatz, wie von Gansner et al. [GKNV93] beschrieben, verwendet. Dieser minimiert die Summe der Ebenen, über welche alle Kanten gespannt werden und führt daher tendenziell zu breiteren Zeichnungen.

Für Prozessgraphen eignen sich allerdings schmalere und höhere Zeichnungen (d.h. viele Ebenen mit jeweils wenigen Knoten) besser. Die Abfolge der Aktivitäten ist besser verständlich, wenn diese vertikal zueinander angeordnet sind und sich höchstens eine bestimmte Anzahl von Knoten auf einer Ebene befindet. Daher wird die Ebenenzuweisung im Anschluss angepasst.

Gesucht ist an dieser Stelle eine Zuweisung der Knoten zu den Ebenen, sodass alle Kanten vorwärts gerichtet sind, jede Ebene höchstens k Knoten enthält und die Anzahl der benötigten Ebenen minimal wird. Dieses Problem ist äquivalent zum Problem des Precedence-

constrained Multi-processor Scheduling (PCMPS), welches n partiell geordnete Jobs mit einheitlicher Dauer auf W gleichartige Maschinen verteilt, sodass die Gesamtdauer zur Abfertigung aller Jobs minimiert wird.

Die Lösung des Problems erfolgt nach [Gra69] durch Zuweisung der Jobs zu den Maschinen unter Berücksichtigung der jeweiligen Reihenfolge und Maschinenkapazität. Die Jobs werden dazu zunächst in topologischer Sortierung in einer Liste gespeichert. Ein Job ist verfügbar, falls all seine Vorgänger bereits abgearbeitet wurden. In jeder Produktionsperiode können maximal W Jobs auf die Maschinen verteilt werden. Für jede Periode werden nun verfügbare Jobs auf freie Maschinen verteilt, bis alle Maschinen belegt oder keine Jobs mehr verfügbar sind.

Dieses Prinzip lässt sich nun auf die Zuweisung der Knoten eines Graphen zu den Ebenen anwenden. Hier werden die Knoten respektive ihrer partiellen Sortierung zu den Ebenen zugeweisen, bis die Kapazität der Ebene ausgeschöpft ist, oder keine Knoten mehr verfügbar sind. (siehe Abb. 2.5)

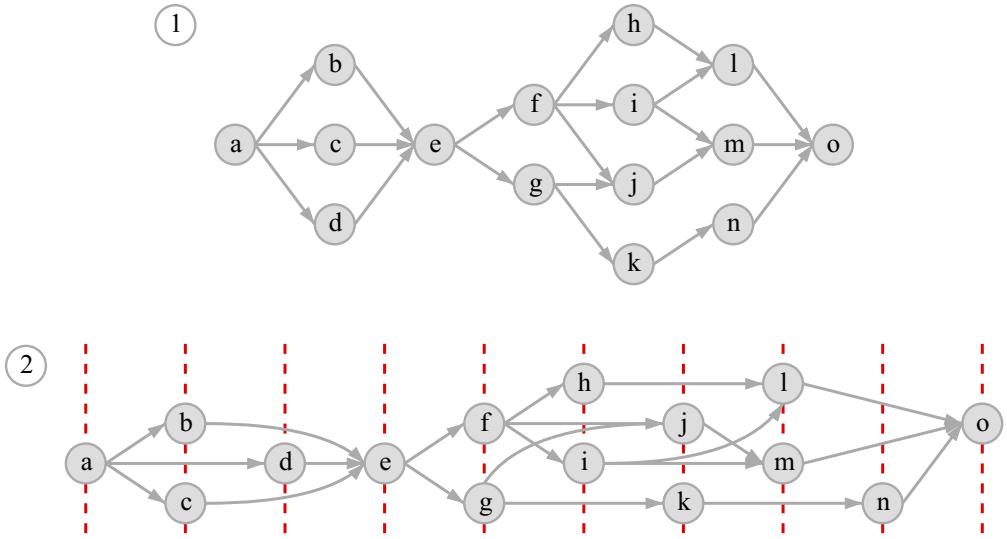


Abb. 2.5.: (1) zeigt einen Graphen, bei welchem den Knoten bereits Ebenen zugewisen wurden. (2) zeigt die Ebenenzuweisung nach Adjustierung mittels PCMPS-Vorgehensweise. Die maximale Anzahl der Knoten pro Ebene beträgt hier zwei.

So wird erreicht, dass die Flussrichtung der Kanten im Graphen klar sichtbar ist und Betrachter die zeitliche Abfolge verschiedener Prozessschritte schnell erkennen können. Die Anzahl der Knoten pro Ebene kann je nach Bedarf frei gewählt werden, für einen gut verständlichen Prozessgraphen eignen sich zwei bis fünf Knoten pro Ebene sehr gut.

2.2.3. Dummy-Knoten einfügen und Ports platzieren

In diesem Schritt werden Kanten, deren Höhe größer als eine Ebene ist, in Segmente unterteilt und zwischen diesen jeweils Dummy-Knoten eingefügt. Die Höhe h einer Kante (v, w) ist dabei definiert als $h(v, w) = |l(v) - l(w)|$. Durchquert eine Kante eine Ebene, wird in dieser Ebene ein Dummy-Knoten hinzugefügt und die Kante wird in einen oberen und unteren Teil unterteilt.

Analog zur Vorlage [ZWBW22] wird an den Stellen ein- und ausgehender Kanten mit Ports gearbeitet. Ein Port gehört zu genau einer Kante und dient als Verbindung zwischen Kante und zugehörigem Knoten. Die Form des Ports wird abhängig davon gewählt, ob es sich um eine ein- oder ausgehende Kante handelt. Im ersten Fall wird der Port als Pfeilspitze, im zweiten Fall als gerade Linie dargestellt (siehe Abb. 2.6). Verbindet eine Kante einen Knoten mit einem Darüberliegenden, wird der Port an der Oberseite des Knotens platziert, andernfalls befindet er sich an der Unterseite des Knotens. Ports an den Seiten des Knotens kommen nicht vor, da diese das Verständnis der Flussrichtung des Prozesses beeinträchtigen würden.

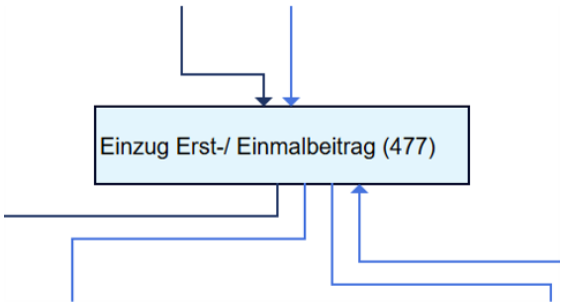


Abb. 2.6.: Einzelne Aktivität mit ein- und ausgehenden Kanten. Die Ports eingehender Kanten erhalten die Form einer Pfeilspitze, Ports ausgehender Kanten die Form einer Linie.

Der Ansatz [ZWBW22] berücksichtigt an dieser Stelle noch Port-Gruppen und Port-Paarungen, welche dazu verwendet werden, Kanten eines Knoten optisch zu gruppieren. Hier können Schwierigkeiten auftreten, wenn beispielsweise eine nach unten und eine nach oben verlaufende Kante an derselben Stelle in den Knoten einmünden sollen. In Prozessgraphen sind solche komplexeren Port-Strukturen aber nicht nötig, da ein Port jeweils zu genau einer Kante gehört. Weiterhin liegen zwei Knoten, welche durch eine Kante verbunden sind, niemals auf derselben Ebene, sodass es eine eindeutige Zuweisung eines Ports zur Ober- oder Unterseite des zugehörigen Knotens gibt.

2.2.4. Anzahl der Kantenkreuzungen minimieren

Für die Minimierung der Kreuzungen wird der von Sugiyama et al. [STT81] vorgestellte Schwerpunkt-basierte Ansatz zur Sortierung der Knoten innerhalb der Ebenen verwen-

det. Dies ist ein heuristischer Ansatz, mittels welchem die Anzahl der Kreuzungen bestmöglich minimiert werden soll. Das Problem der Minimierung von Kantenkreuzungen ist NP-schwer [EW94], weshalb sich vermutlich mit keinem effizienten Algorithmus eine garantiert optimale Lösung bestimmen lässt. Die Ebenen werden nacheinander durchlaufen und jeweils die Sortierung der folgenden Ebene angepasst, um eine hinreichend gute Lösung zu erhalten. Diese Prozedur kann mehrfach durchgeführt werden, um die Ergebnisse noch zu verbessern.

Für die Implementierung gibt es hier verschiedene Möglichkeiten, bei welchen entweder die Reihenfolge der Ports oder die Reihenfolge der Knoten zuerst betrachtet wird. Analog zu [ZWBW22] wird hier ein Ansatz gewählt, bei welchem zuerst die Reihenfolge der Ports optimiert wird. Dazu wird eine beliebige Permutation der Knoten und Ports gewählt und ein Durchlauf der schwerpunktbasierten Umsortierung durchgeführt.

2.2.5. Horizontale Positionen der Knoten berechnen

Im letzten Schritt müssen die tatsächlichen Koordinaten der Knoten, dargestellt als Rechtecke, bestimmt werden. Hier wird die Zuweisung der Knoten zu den Ebenen verwendet, denn alle Knoten auf einer Ebene erhalten dieselbe y-Koordinate. Die x-Koordinate eines Knoten hängt von der Breite des Rechteckes sowie den anderen Knoten auf der Ebene ab. Die Knoten sollen horizontal möglichst gleichmäßig auf der Ebene verteilt werden, wobei ein bestimmter Mindestabstand zwischen zwei benachbarten Knoten eingehalten werden muss. Um diesen Mindestabstand zu gewährleisten, werden zwischen den Knoten Dummy-Kanten eingefügt.

Die Breite eines Rechteckes hängt dabei einerseits von den ein- und ausgehenden Kanten (Ports) ab, denn diese haben eine bestimmte Breite und einen festgelegten Mindestabstand zum nächsten Port. Andererseits ist auch die Breite des Textes (Aktivitätsbeschreibung) entscheidend. Um die Breite eines Feldes zu adjustieren, werden Dummy-Ports eingefügt. Die Höhe der Rechtecke soll für jeden Knoten gleich sein, daher werden für die Konzeption der Zeichnung auf jeder Ebene eine obere - und untere Hilfsebene eingeführt, welche genau der Ober- und Unterseite der darauf liegenden Rechtecke entsprechen.

Für die genaue Bestimmung der x-Koordinaten der Knoten wird der heuristische Algorithmus von Brandes und Köpf aus [BK02] bzw. [BWZ20] verwendet, welcher lange Kanten vertikal möglichst gerade darstellt. Dazu werden nach Bedarf die Breiten der Rechtecke angepasst. Um zu vermeiden, dass Knoten zu breit werden, wird die Breite analog zu [ZWBW22] beschränkt.

Im Anschluss werden die Kanten zunächst orthogonal, d.h. vertikal und horizontal gezeichnet. Dazu werden die Dummy-Knoten ausgeblendet und die Kanten ggf. in Liniensegmente unterteilt. Um die Überschneidung horizontaler Liniensegmente zwischen zwei Ebenen zu vermeiden, werden Zwischenebenen eingefügt und der Abstand zwischen den

Ebenen dementsprechend erhöht, auch dies erfolgt analog zu [ZWBW22]. Letzendlich starten und enden alle Kanten mit einem vertikalen Segment und enthalten dazwischen ggf. weitere horizontale und vertikale Segmente, welche jeweils abwechselnd auftreten. Die visuellen Darstellungsaspekte werden detailliert im folgenden Kapitel behandelt.

3. Visuelle Aspekte der Darstellung

Das folgende Kapitel behandelt die visuelle Darstellung der Kanten und Knoten in den Zeichnungen. Es beschreibt die Anzeige der Häufigkeiten von Kanten und Knoten mittels Textfeldern, die Hervorhebung wichtiger Kanten durch Farbgebung sowie die Darstellung der Kanten als orthogonale Linien oder Bézierkurven. Für die Bézierkurven werden Algorithmen zur Vermeidung von Kreuzungen mit Textfeldern entwickelt und erläutert.

3.1. Darstellung der Knoten

Die Knoten in Prozessgraphen repräsentieren die einzelnen Aktivitäten innerhalb eines Prozesses. Damit die als gerahmte Textfelder dargestellten Aktivitäten klar und verständlich sind, benötigen die Knoten Textlabels, die den Namen der jeweiligen Aktivität anzeigen. Dass die Namen der Aktivitäten unterschiedliche Längen und die Textfelder damit unterschiedliche Größen haben, muss bei der Zeichnung berücksichtigt werden, denn es sollen keine Kanten durch die Textfelder verlaufen.

Wie bereits beschrieben, wird die passende Breite der Textfelder unter Berücksichtigung von ein- und ausgehenden Kanten sowie Textlänge des Labels bereits in der fünften Phase des Sugiyama-Frameworks bestimmt und die Textfelder anhand dessen passend platziert.

Die einzige Anpassung, die im Kontext von Prozessgraphen an dieser Stelle noch vorgenommen werden sollte, ist die Anzeige der Häufigkeit der einzelnen Aktivitäten. Diese sind für die spätere Prozessanalyse von Bedeutung, da sie anzeigen, wie wichtig die einzelnen Aktivitäten im Prozess sind. Hier bietet es sich an, die Anzahl jeder Aktivität hinter der Aktivitätsbezeichnung im Textlabel zu vermerken. Dadurch wird das Textlabel etwas breiter, ansonsten müssen aber keine weiteren Anpassungen vorgenommen werden.

Abb. 3.1.: Ausschnitt einer einzelnen Aktivität, welche im betrachteten Zeitraum 53 mal erfasst wurde. Die Breite des Knotens passt zur Breite des darin liegenden Textfeldes, welches die Aktivitätsbeschreibung sowie die Häufigkeit enthält.

3.2. Darstellung der Kanten

Grundsätzlich sieht der Algorithmus aus [ZWBW22] die Kanten als orthogonale Verbindungslinien, bestehend aus horizontalen und vertikalen Segmenten, zwischen den Knoten vor. Eine Abrundung der Ecken ist in diesem Rahmen natürlich problemlos möglich. Für die Darstellung eines Prozessflusses ist es allerdings visuell ansprechender, die Kanten noch stärker zu runden und so den Anschein einer Gitterdarstellung vollständig zu vermeiden, weshalb auch Bézierkurven als Darstellungsmethode in Betracht gezogen werden sollen. Die drei möglichen Kantenformen werden in der folgenden Abb. 3.2 dargestellt.

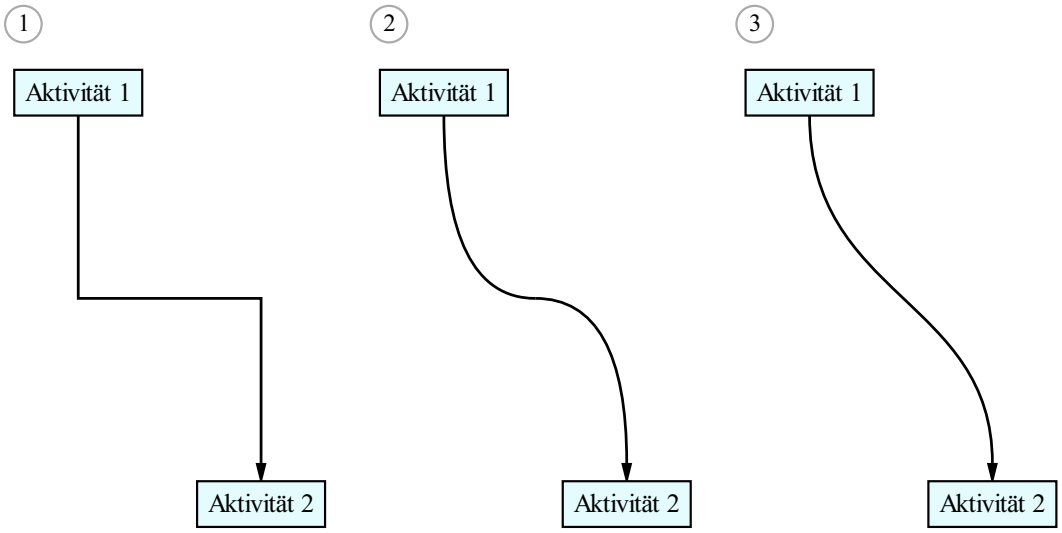


Abb. 3.2.: Orthogonale Kantendarstellung (1), Bézierkurve zweiten Grades (2) und Bézierkurve dritten Grades (3)

Bézierkurven sind mathematische Kurven, die häufig in der Computergrafik und im Design verwendet werden und durch eine Reihe von Kontrollpunkten definiert werden, die die Form der Kurve beeinflussen [Far02]. Die Kurven sind parametrisch und basieren auf Bernstein-Polynomen. Im Folgenden werden Bézierkurven zweiten und dritten Grades verwendet, welche jeweils zwei bzw. drei Kantenknicke abrunden und drei bzw. vier Kontrollpunkte benötigen.

Hierbei besteht allerdings die Gefahr, dass sich die Bézierkurven mit bereits platzierten Knoten überschneiden, da lange Kanten teilweise stark abgerundet werden. Der Zeichenalgorithmus sollte dies entsprechend berücksichtigen, indem für jede Kante mögliche Überschneidungen geprüft - und die Kanten ggf. entsprechend adjustiert werden.

Im Folgenden werden zunächst Bézierkurven und ihre grundlegenden Eigenschaften sowie

Berechnungsvorschriften dargestellt. Im Anschluss wird die Integration von Bézierkurven zweiten oder dritten Grades in die Prozessgraphen sowie die Wahl der Kontrollpunkte erläutert. Dabei werden jeweils Ansätze zur Vermeidung von Kreuzungen der Bézierkurven mit umliegenden Textfeldern vorgestellt, welche auf der geeigneten Adjustierung von Kontrollpunkten basieren.

3.2.1. Bézierkurven zweiten Grades

Eine Bézierkurve zweiten Grades (quadratische Bézierkurve) wird durch drei Kontrollpunkte P_0, P_1 und P_2 definiert (siehe Abb. 3.3). Die Parametergleichung für eine quadratische Bézierkurve für $t \in (0, 1)$ ist:

$$B : [0, 1] \to \mathbb{R}^2,\ B(t) = (1-t)^2 P_0 + 2t(1-t)P_1 + t^2 P_2 \tag{3.1}$$

Hierbei beschreibt t den Parameter, der von 0 bis 1 variiert, wobei $t = 0$ den Anfangspunkt P_0 und $t = 1$ den Endpunkt P_2 darstellt.

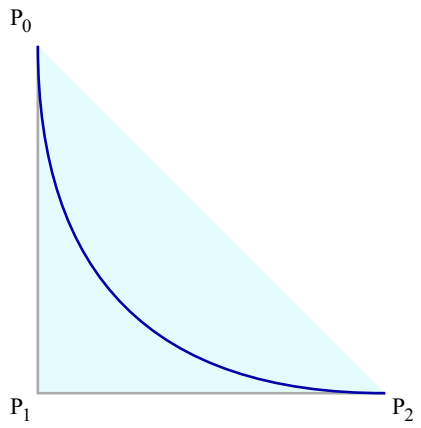


Abb. 3.3.: Bézierkurve zweiten Grades und die durch ihre Kontrollpunkte induzierte konvexe Hülle

Die hier vorliegenden Kanten bestehen zunächst aus einer ungeraden Anzahl an geraden Segmenten, wobei sie mit einem vertikalen Segment starten und enden und dazwischen jeweils abwechsenld horizontale und vertikale Segmente enthalten. Da die Anzahl der Segmente nicht einheitlich ist, werden die Kanten im Folgenden abschnittsweise durch Bézierkurven zweiten Gerades dargestellt. Dazu werden abhängig von der Anzahl der Segmente die Eckpunkte der Kanten $E_0, E_1, \ldots$ aus dem orthogonalem Layout sowie Mittelpunkte zwischen Eckpunkten als Kontrollpunkte der Bézierkurven verwendet. Im Fol-

genden ist $M_{i,i+1}$ definiert als der Mittelpunkt zwischen den beiden Eckpunkten E_i, E_{i+1} und $\overline{E_i E_{i+1}}$ stellt den Kantenabschnitt zwischen den Eckpunkten E_i und E_{i+1} aus der orthogonalen Zeichnung dar.

Die Unterteilung der Kanten in Abschnitte und die Wahl der Kontrollpunkte der Bézierkurven ist abhängig von der Anzahl an Kantensegmenten. Kanten mit nur einem Segment werden als gerade Linien gezeichnet. Für Kanten mit drei Segmenten werden zwei Bézierabschnitte gezeichnet, welche jeweils durch die Kontrollpunkte (E_0, E_1, M_{12}) und (M_{12}, E_2, E_3) definiert werden (siehe Abb. 3.4). Bei Kanten mit mehr als drei Segmenten werden die mittleren vertikalen Segmente analog mittig unterteilt, um die Kurve abschnittsweise durch Bézierkurven zweiten Grades darstellen zu können.

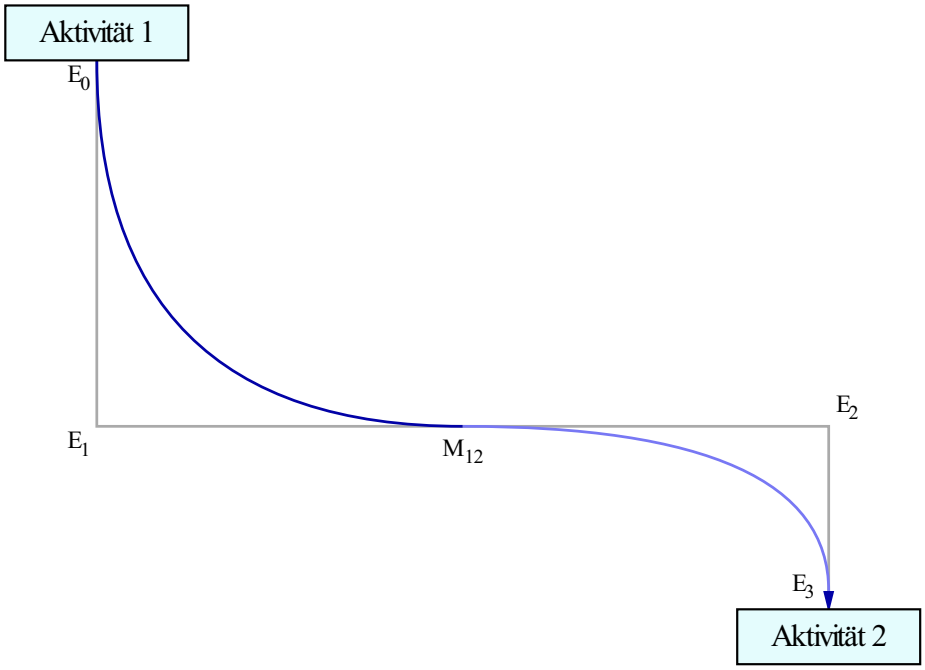


Abb. 3.4.: Die Eckpunkte und Mittelpunkte der orthogonalen Kanten dienen als Kontrollpunkte der Bézierkurve. Die Kurve setzt sich aus Bézierkurven zweiten Grades zusammen, die beiden Abschnitte sind farbig gekennzeichnet.

Zur Vermeidung von Überschneidungen mit Textfeldern wird ein mehrschrittiges Ver-

fahren angewandt. Da die Bézierkurve nach [Sed12] immer innerhalb ihrer umgebenden *konvexen Hülle*, also des durch die Kontrollpunkte erzeugten Dreiecks liegt, muss dieses entsprechend angepasst werden. Dazu werden zunächst alle Knoten innerhalb der konvexen Hülle bestimmt. Von diesen wird derjenige Knoten gewählt, welcher am nächsten am Kontrollpunkt P_1 liegt. Dessen nächstgelegene Ecke an P_1 wird als P_N bezeichnet und dient als Berechnungsbasis für die neuen Kontrollpunkte. Zur Distanzmessung wird hier die Manhattan-Metrik verwendet.

Dazu wird eine Gerade g zwischen die Punkte P_0 und P_2 gelegt und anschließend so verschoben, dass sie durch P_N verläuft. Im Anschluss werden ein neuer Start- und Enpunkt P_0' und P_2' für die Kurve bestimmt, welche genau der Schnittpunkte der verschobenen Gerade mit $\overline{P_0P_1}$ und $\overline{P_1P_2}$ entsprechen (siehe Abb. 3.5). So wird erreicht, dass innerhalb der konvexen Hülle der Bézierkurve kein Knoten mehr liegt und eine Überschneidung garantiert vermieden wird.

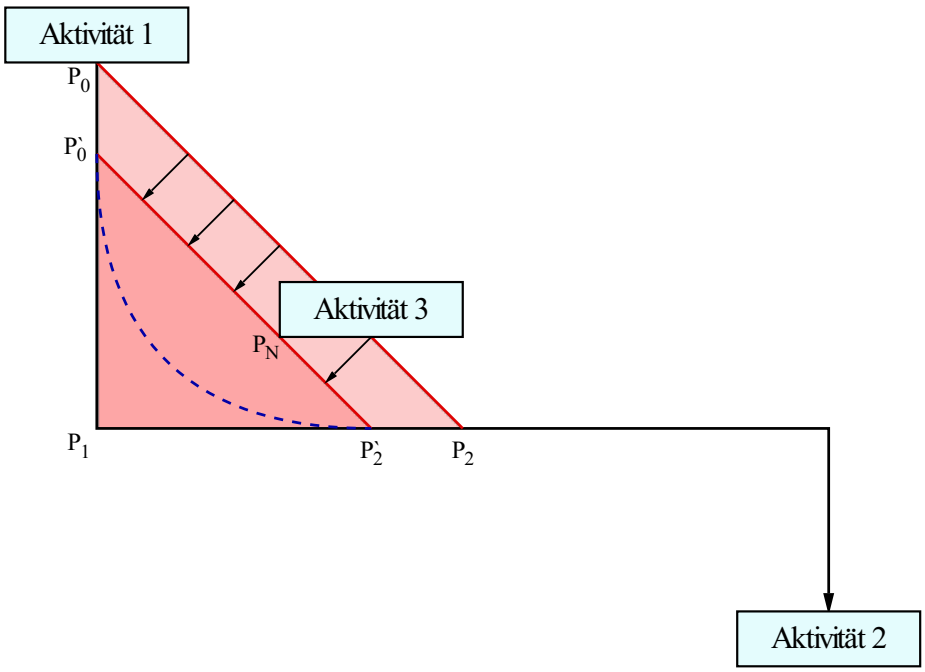


Abb. 3.5.: Anpassung der Kontrollpunkte einer Bézierkurve zweiten Grades im Falle einer Überlappung der zugehörigen konvexen Hülle mit einem Textfeld

3.2.2. Bézierkurven dritten Grades

Eine Bézierkurve dritten Grades (kubische Bézierkurve) wird durch vier Kontrollpunkte P_0, P_1, P_2 und P_3 definiert (siehe Abb. 3.6). Die Parametergleichung für eine kubische Bézierkurve für $t \in (0, 1)$ lautet:

$$B : [0, 1] \to \mathbb{R}^2,\ B(t) = (1-t)^3 P_0 + 3t(1-t)^2 P_1 + 3t^2(1-t)P_2 + t^3 P_3 \tag{3.2}$$

In diesem Fall beschreibt $t = 0$ den Anfangspunkt P_0 und $t = 1$ den Endpunkt P_3, während P_1 und P_2 die Krümmung und Form der Kurve beeinflussen.

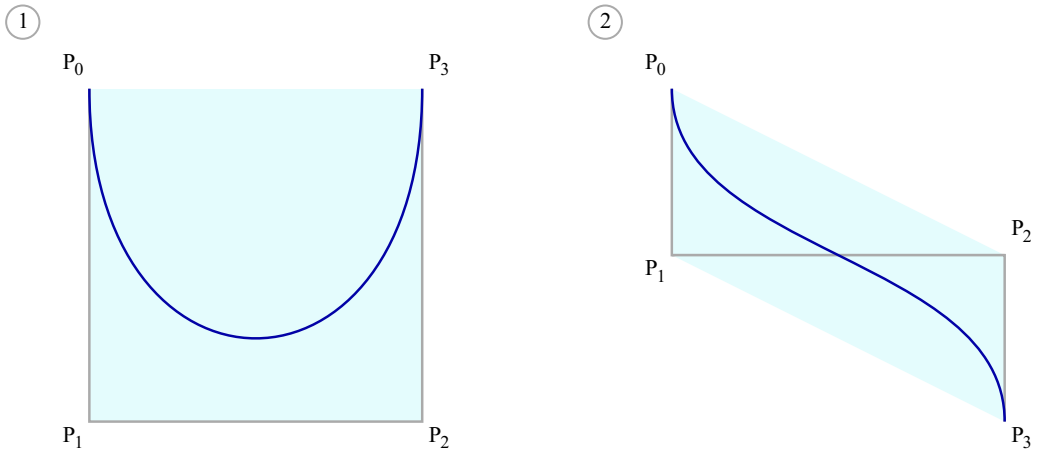


Abb. 3.6.: Bézierkurve dritten Grades und die durch ihre Kontrollpunkte induzierte konvexe Hülle

Die Unterteilung der Kanten basierend auf der Anzahl der Segmente sowie die abschnittsweise Darstellung der Kurven als Bézierkurven dritten Grades erfolgt ähnlich wie im vorherigen Fall. Hier muss allerdings nur bei Kurven mit fünf oder mehr Segmenten eine Unterteilung der mittleren vertikalen Segmente an ihrem Mittelpunkt vorgenommen werden. Ansonsten dienen auch hier die Eckpunkte $E_0, E_1, \ldots$ der orthogonalen Kanten als Kontrollpunkte (siehe Abb. 3.7).

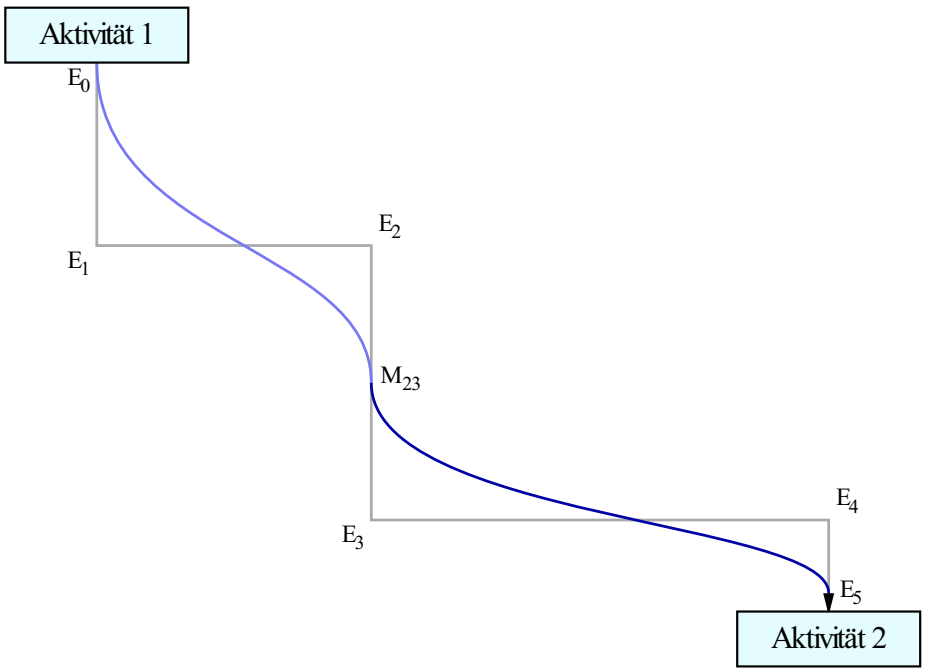


Abb. 3.7.: Die Eckpunkte und Mittelpunkte der orthogonalen Kanten dienen auch hier als Kontrollpunkte der beiden Bézierkurven. Die Unterteilung in Segmente ist hier nicht zwingend, sondern erst ab fünf Eckpunkten notwendig.

Bei der Vermeidung der Kantenkreuzungen wird hier ähnlich zum vorherigen Fall vorgegangen. Hier können allerdings nur die Punkte P_0 und P_3 verschoben werden, da P_1 und P_2 nicht durchlaufen werden, sondern nur die Form der Kurve determinieren. Zunächst muss die Form der konvexen Hülle genauer betrachtet werden.

Falls sich die konvexe Hülle wie in Fall (2) aus Abb. 3.6 aus den beiden Dreiecken $P_0P_1P_2$ und $P_1P_2P_3$ zusammensetzen lässt, können die beiden Dreiecke separat betrachtet werden. In diesem Fall wäre zunächst P_N zu P_1 zu bestimmen und P_0' dort zu platzieren, wo sich eine Gerade durch die Punkte P_2 und P_N mit der Strecke $\overline{P_0P_1}$ schneidet. Mit dem zweiten Dreieck kann dann analog verfahren werden, nur dass hier P_N zu P_2 gesucht wird und P_3' dem Schnittpunkt einer Gerade durch P_1 und P_N mit der Strecke $\overline{P_2P_3}$ entspricht (siehe Abb. 3.8).

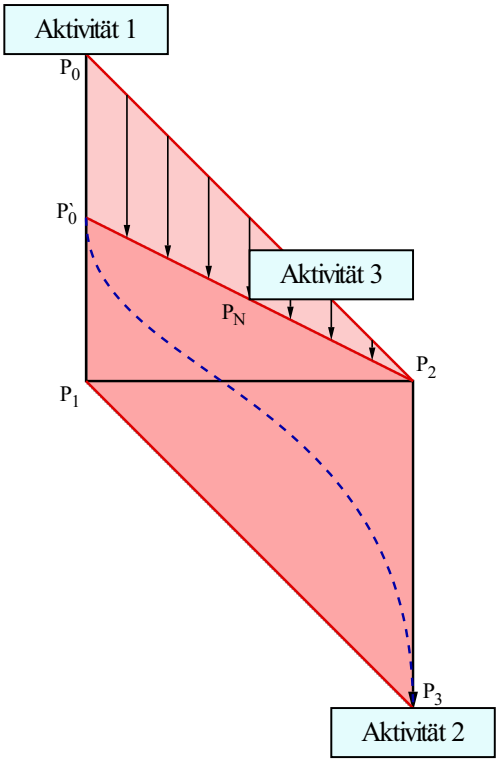


Abb. 3.8.: Anpassung der Kontrollpunkte einer Bézierkurve dritten Grades im Falle einer Überlappung der zugehörigen konvexen Hülle mit einem Textfeld

Falls die konvexe Hülle der Form (1) aus Abb. 3.6 entspricht und sich nicht durch die beiden umgebenden Dreiecke darstellen lässt, ist die zugehörige Kante eine Schleife. Sie startet und endet dementsprechend bei derselben Aktivität. In diesem Fall ist keine Anpassung der Kontrollpunkte nötig, da diese bereits im Rahmen des Sugiyama-Algorithmus mit einem hinreichend großen Abstand zum zugehörigen Knoten platziert wurden und keine Kreuzungen mit Textfeldern entstehen.

3.2.3. Bestimmung von Kantenkreuzungen bei Bézierkurven

Zeichnet man die Kanten des Graphen als Bézierkurven, kann es zu zusätzlichen Kreuzungen benachbarter Kanten kommen (siehe Abb. 3.9). Das liegt daran, dass sich die Abstände der Kontrollpunkte zueinander für verschiedene Kurven teilweise stark unterscheiden und einge Ecken weniger stark abgerundet werden als andere. Besonders bei großen Graphen treten diese doppelten Kantenkreuzungen häufig auf.

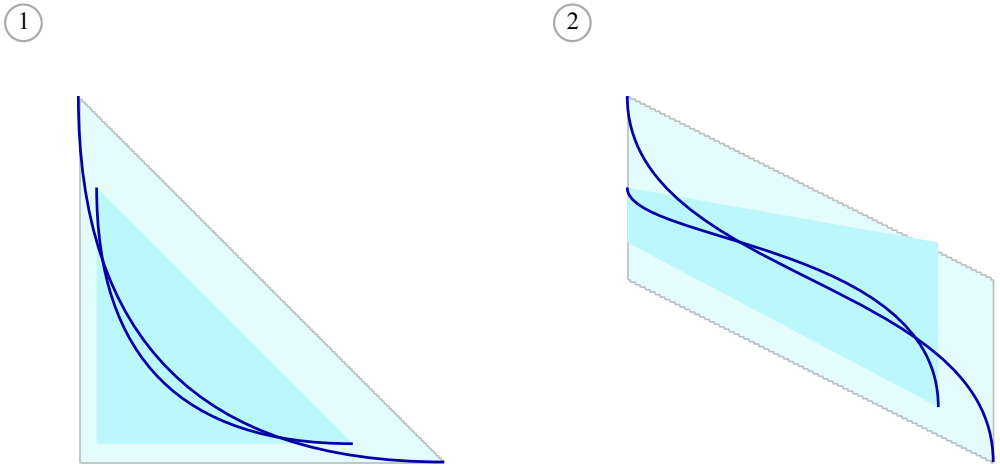


Abb. 3.9.: Doppelte Kantenkreuzungen bei Bézierkurven zweiten (1) und dritten Grades (2), welche im Falle überlappender konvexer Hüllen auftreten können.

Für die Bewertung der Zeichnungen muss dieser Effekt berücksichtigt werden, da doppelte Kreuzungen die Lesbarkeit des Graphen beeinträchtigen. Zur Berechnung der Kreuzungen werden die einzelnen Béziersegmente der Kurven stückweise linearisiert und alle Kreuzungen zwischen linearen Segmenten aufaddiert. So kann später untersucht werden, ab welcher Kanten- und Knotenanzahl eines Graphen die Integration von Bézierkurven derart viele zusätzliche Kreuzungen verursacht, dass sie nicht mehr sinnvoll ist.

Die Vermeidung solcher Kantenkreuzungen wäre im Rahmen einer Nachverarbeitung möglich, ist aber nicht Teil dieser Arbeit und wird daher nicht weitergehend untersucht. Die zusätzlichen Kantenkreuzungen werden akzeptiert und dienen im Folgenden eher als Maß für die Lesbarkeit von Zeichnungen mit Bézierkurven.

3.3. Kantenfärbung und Labels

Die Farbe der Kanten in Prozessgraphen kann dazu verwendet werden, die Wichtigkeit oder Bedeutung der einzelnen Kanten hervorzuheben. Da bei den hier erzeugten Zeichnungen ein helles Layout verwendet wird, werden die häufig durchlaufenen Kanten dunkelblau und die seltener durchlaufenen hellblau gefärbt. So können die wichtigen Prozesspfade unmittelbar erkannt werden.

Da die Färbung der Kanten allein aber als Information nicht immer ausreicht, werden zusätzlich noch Textfelder mit der genauen Häufigkeit der Kanten eingefügt. Diese werden in der Mitte der Kante und auf der Linie platziert. Möglich ist dabei, dass Textfelder genau an der Kreuzung von zwei Kanten positioniert werden. Dies wird aktuell nicht berücksichtigt, da in keinem beobachteten Fall eine solche Überlappung störend aufgefallen ist. Die Berücksichtigung von Überlappungen wäre aber eine mögliche zukünftige Anpassung des Algorithmus.

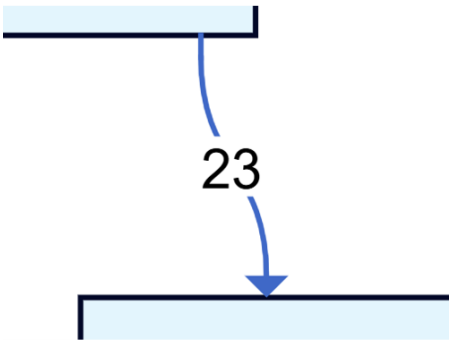


Abb. 3.10.: Kante mit Textlabel, welche im betrachteten Zeitraum 23 mal durchlaufen wurde. Die mittelblaue Färbung der Kante deutet an, dass sie zu den mittelmäßig wichtigen Verbindungen des Prozesses gehört.

4. Bewertung des Algorithmus

Im folgenden Kapitel werden die vorgestellten Algorithmen zur Auflösung von Kreisen getestet und anhand der erzeugten Zeichnungen bewertet. Auch die verschiedenen Kantenformen werden mittels verschiedener Kriterien gegeneinander abgewogen.

Dazu werden zunächst die Bewertungskriterien genannt und deren Interpretation erläutert. Anschließend wird dargestellt, welche Prozessdaten zur Evaluation verwendet werden und wie diese generiert werden. Zuletzt erfolgt der Test der verschiedenen Algorithmen zur Auflösung von Kreisen sowie zur Darstellung der Kanten anhand der vorgestellten Kriterien. Die Testergebnisse werden dargestellt, interpretiert und die verschiedenen Algorithmen anhand der Ergebnisse bewertet.

4.1. Bewertungskriterien

Bei der Bewertung von Prozessgraphen spielen sowohl objektive, als auch subjektive Kriterien eine zentrale Rolle, um die Qualität der Visualisierung zu beurteilen. Unter die objektiven Kriterien fallen vor allem Aspekte wie Rechenzeit, die Anzahl der Kantenkreuzungen und -knicke oder die Größe des Zeichenbereichs. Diese entscheiden nicht allein über die wahrgenommene Ästhetik einer Zeichnung, welche jeder Betrachter unterschiedlich beurteilt, weshalb auch subjektive Aspekte wie Übersichtlichtlichkeit und Lesbarkeit eine Rolle spielen. Hier werden allerdings nur objektive Kriterien zur Evaluation herangezogen, da die subjektiven Aspekte schwer messbar sind. Die folgenden objektiven Bewertungskriterien werden bei der späteren Evaluation der Zeichnungen berücksichtigt:

- Rechenzeit: Die Zeit, die benötigt wird, um den Prozessgraphen zu generieren. Da hier verschiedene Algorithmen für die erste Phase des Sugiyama-Frameworks gegeneinander abgewogen werden sollen, werden primär die Rechenzeiten dieser Abschnitte für die verschiedenen Methoden verglichen.

- Größe des Zeichenbereichs: Die umrahmende Fläche, auf welcher der Prozessgraph dargestellt wird. Ein kleinerer Zeichenbereich trägt tendenziell dazu bei, dass der Graph übersichtlich bleibt.

- Kantenkreuzungen: Eine hohe Anzahl von Kantenkreuzungen erschwert die Interpretation des Graphen.

- Kantenknicke: Viele Knicke erschweren die Nachverfolgung der Kanten und erhöhen die visualle Komplexität. Im Falle von Bezirkurven wird hier die Anzahl der Knicke der darunterliegenden orthogonalen Zeichnung betrachtet.
- Größe des FAS: Wird die Richtung vieler Kanten vor dem Zeichnen umgekehrt, ist die allgemeine Flussrichtung weniger klar erkennbar, weshalb das FAS möglichst klein sein sollte.

4.2. Betrachtete Prozesse

Um die Funktionweise des Algorithmus möglichst umfangreich zu testen, werden verschiedene Prozessgraphen mit unterschiedlicher Knoten- und Kantenanzahl gezeichnet und die Resultate verglichen. Die gezeichneten Graphen enthalten zwischen 10 und 50 Knoten und zwischen 15 und 90 Kanten. Dies entspricht auch der Größe von Prozessgraphen im realen Umfeld und passt daher als Testgröße. Die zugrundeliegenden Prozessdaten wurden mithilfe eines selbst entwickelten Simulationsprogrammes in Python [Fou24] erzeugt.

4.2.1. Prozessdatensimulation

Da die Simulation realistischer Prozessdaten einige Herausforderungen beinhaltet, wird im Folgenden zunächst die Vorgehensweise dargestellt. Zu Beginn wird eine Menge von Aktivitäten, welche innerhalb des Prozesses auftreten, benötigt. Diese enthält auch eine zentrale Start- und Endaktivität, mit welcher jeder Fall beginnt und endet. Den Aktivitäten werden anschließend Übergangswahrscheinlichkeiten und mittlere Übergangszeiten zugewiesen.

	Start	A	B	C	Ende
Start	0.0	0.1	0.7	0.2	0.0
A	0.0	0.1	0.6	0.3	0.0
B	0.0	0.0	0.4	0.4	0.2
C	0.0	0.4	0.0	0.2	0.4
Ende	0.0	0.0	0.0	0.0	0.0

	Start	A	B	C	Ende
Start	-	1m	1m	1m	-
A	-	10m	20m	1h	-
B	-	-	2h	1d	1m
C	-	15m	-	1h	1m
Ende	-	-	-	-	-

Tab. 4.1.: Beispielhafte Übergangswahrscheinlichkeiten (links) und -zeiten (rechts) eines Prozesses mit drei Aktivitäten

Die Tabellen 4.1 zeigen mögliche Übergangswahrscheinlichkeiten und Übergangszeiten für einen Prozess mit den drei Aktivitäten A, B und C. Bei den Übergangszeiten wird jeweils noch die Zeiteinheit (m für Minute, h für Stunde oder d für Tag) vermerkt.

Auf Basis dieser Werte wird jeder Fall einzeln simuliert. Der Fall beginnt bei der Aktivi-

tät START, diese wird als Pseudoaktivität aber nicht der Aktivitätstabelle hinzugefügt. Die Übergangswahrscheinlichkeiten der möglichen nächsten Aktivitäten werden kumuliert, und mithilfe einer gleichverteilten Zufallszahl $z \in [0, 1]$ wird die nächste Aktivität bestimmt. Ausgehendend von der gewählten Aktivität wird analog verfahren. Die gewählten Aktivitäten werden dabei in der Aktivitätstabelle vermerkt. Sobald der Fall mit diesem Verfahren die Pseudeaktivität ENDE erreicht, wird zum nächsten Fall gewechselt, wieder mit dem START begonnen und analog verfahren.

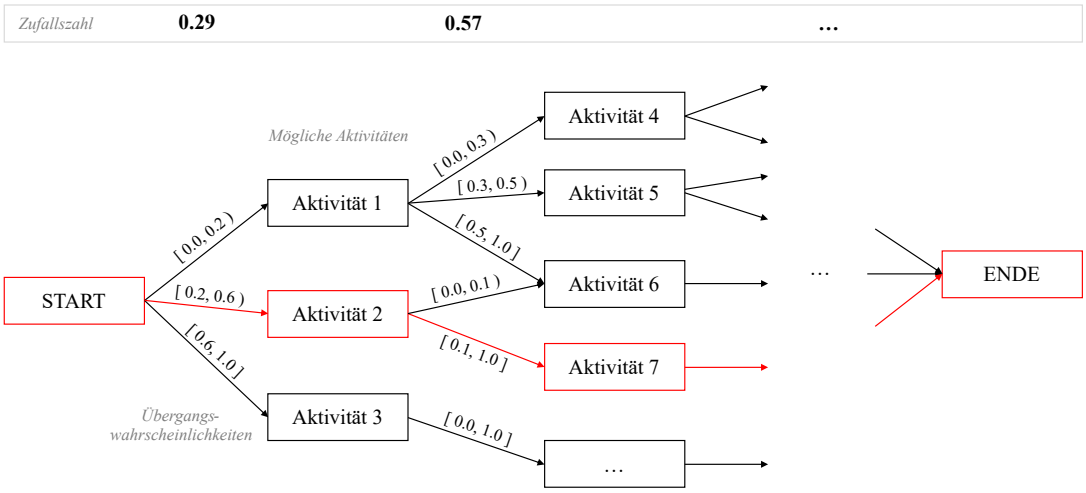


Abb. 4.1.: Vorgehensweise bei der stochastischen Simulation von Prozessdaten mittels uniform verteilter Zufallszahlen

Während der Wahl der jeweils nächsten Aktivität werden auch zufällige Übergangszeiten für die jeweiligen Prozessschritte festgelegt. Der Zeitstempel der Pseudoaktivität START wird innerhalb eines festgelegten Zeitraums zufällig gewählt. Die Bestimmung des Zeitstempels der folgenden Aktivitäten basiert auf der mittleren Übergangsdauer des jeweiligen Prozessschrittes, welche mittels eines normalverteilten Zufallsfaktors verkürzt oder verlängert wird. Hierbei dient die definierte Übergangszeit als Mittelwert, die Standardabweichung wird auf 10% festgelegt. So werden realistische Zeitstempel erzeugt, welche im Mittel der gewählten Übergangsdauer entsprechen.

4.2.2. Simulierte Prozesse

Mithilfe des Simulationsprogramms wurden Prozessdaten für vier verschiedene Prozesse mit jeweils 1000 Fällen erzeugt, welche im Folgenden kurz vorgestellt werden.

- Simpler Prozess (ca. 10 Knoten und ca. 15 Kanten): Ein kleiner Prozess, welcher bereits kreisfrei ist.

- Beispielprozess (ca. 15 Knoten und ca. 30 Kanten): Ein mittelgroßer Prozess mit Kreisen, welcher aber immer noch hinreichend klein ist, dass er in jedem Falle übersichtlich dargestellt werden kann.
- Schadensprozess (ca. 50 Knoten und ca. 90 Kanten): Ein realistischer Kfz-Schadensprozess eines Versicherers, welcher den Weg von der Schadensmeldung des Kunden über die Bewertung des Falles durch einen Sachbearbeiter bis hin zur Regulierung des Schadens enthält. Der Prozess ist sehr groß und enthält viele Kreise und Umwege, weshalb er schnell unübersichtlich wird und eine durchdachte Visualisierung benötigt. Bei der Anordnung der Kanten als Bézierkurven muss besonders auf die Vermeidung zusätzlicher Kreuzungen geachtet werden.
- Antragsprozess (ca. 45 Knoten und ca. 80 Kanten): Ein ebenfalls realistischer Antragsprozess eines Versicherers in der Sparte Lebensversicherung, welcher die Schritte von Beratungsgespräch über die Antragseinreichung und Bearbeitung durch einen Sachbearbeiter bis hin zur Annahme oder Ablehnung enthält. Auch dieser Prozess ist sehr groß und bedarf besonderer Beachtung bei der Visualisierung.

4.3. Evaluation

Für den Test wurden von jedem Prozess 10 Testdatensätze erzeugt und jeweils die 20%, 40%, ..., 100% häufigsten Varianten gezeichnet. Dabei wurden jeweils die drei Kreisbrecher-Algorithmen verwendet und alle drei Kantenformen gezeichnet, somit ergeben sich 1800 Zeichnungen als Testbasis. Die Tests wurden auf einem Computer mit Betriebssystem Windows 11, Intel Core i5-Prozessor der 11. Generation und 8 GB RAM durchgeführt.

Im Folgenden werden zunächst die drei Kreisbrecher-Algorithmen anhand der Größe des erzeugten FAS sowie der zur Berechnung des FAS benötigten Dauer bewertet. Im Anschluss wird untersucht, wie sich die Größe des FAS auf andere Aspekte wie Kantenkreuzungen und -knicke sowie die Größe des Zeichenbereiches auswirkt. Zuletzt werden noch die verschiedenen Linienformen betrachtet, die Bewertungskriterien sind hier die Anzahl der Kantenkreuzungen sowie die Dauer des Kantenrouting-Prozesses.

4.3.1. Bewertung der Kreisbrecher-Algorithmen anhand des FAS

Die Ergebnisse zeigen klar, dass der Greedy-Algorithmus das kleinste FAS erzeugt, während das durch den Pagerank-Algorithmus erzeugte FAS etwas größer und das durch den Sort-Algorithmus bestimmte FAS nochmal tendenziell größer ist (siehe Abb. 4.2).

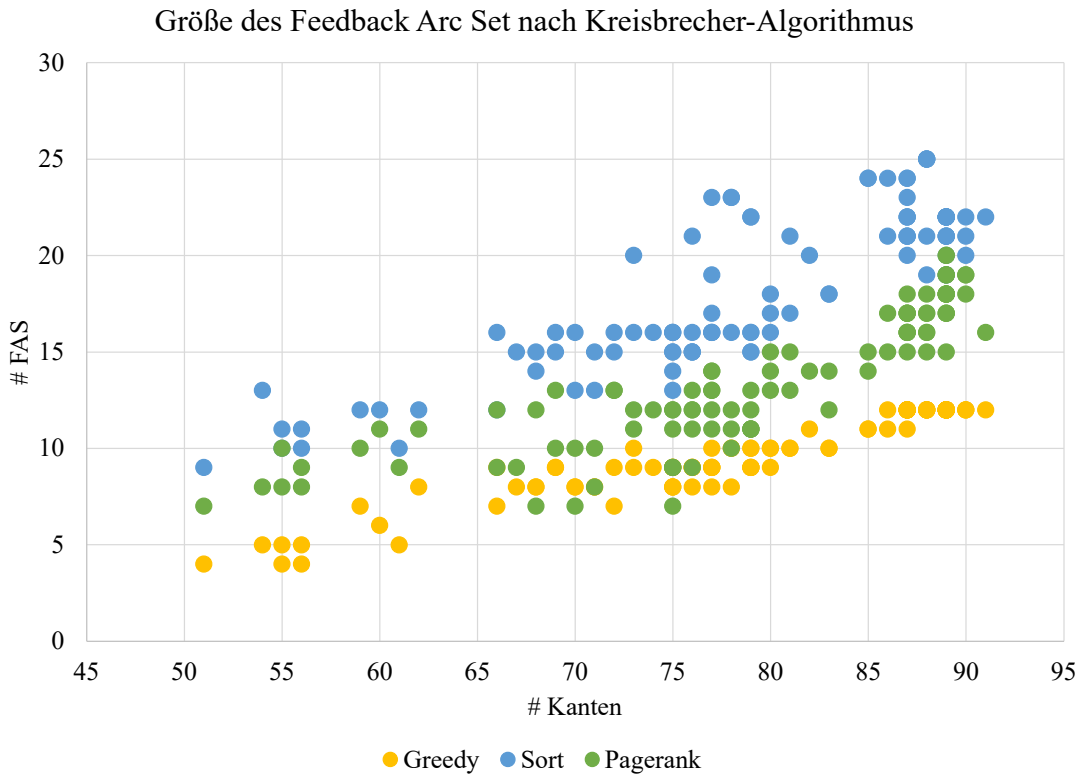


Abb. 4.2.: Größe des erzeugten FAS durch die verschiedenen Kreisbrecher-Algorithmen

Tabelle 4.2 zeigt die mittlere Größe des FAS der verschiedenen Algorithmen jeweils absolut und im Verhältnis zu den Kanten. Dabei zeigt sich, dass das durch den Pagerank-Algorithmus erzeugte FAS im Schnitt um 2.10 Knoten und damit 40.8% größer ist als das vom Greedy-Algorithmus erzeugte FAS, während es beim Sort-Algorithmus 4.92 und damit etwa doppelt so viele umgekehrte Kanten sind. Die Standardabweichung der Größe des FAS bei den drei Algorithmen unterstreicht diesen Zusammenhang.

	Greedy	Sort	Pagerank
Größe FAS	5.37	10.28	7.46
Größe FAS / Anz. Kanten (in %)	7.97	15.90	11.22
Standardabw. FAS	4.86	8.58	6.54
Rechenzeit (in ms)	0.99	1.39	5.29

Tab. 4.2.: Auswertung der Kreisbrecher-Algorithmen anhand von Mittelwerten

Auch bei der Berechnungsdauer (siehe Abb. 4.3) das FAS schneidet der Greedy-Algorithmus besser ab als die beiden anderen Varianten. Hier schlägt der Sort-Algorithmus mit einer

zusätzlichen Dauer von 0.4 ms aber den Pagerank-Algorithmus, welcher mit zusätzlichen 4.3 ms mehr als die fünffache Zeit das Greedy-Algorithmus benötigt.

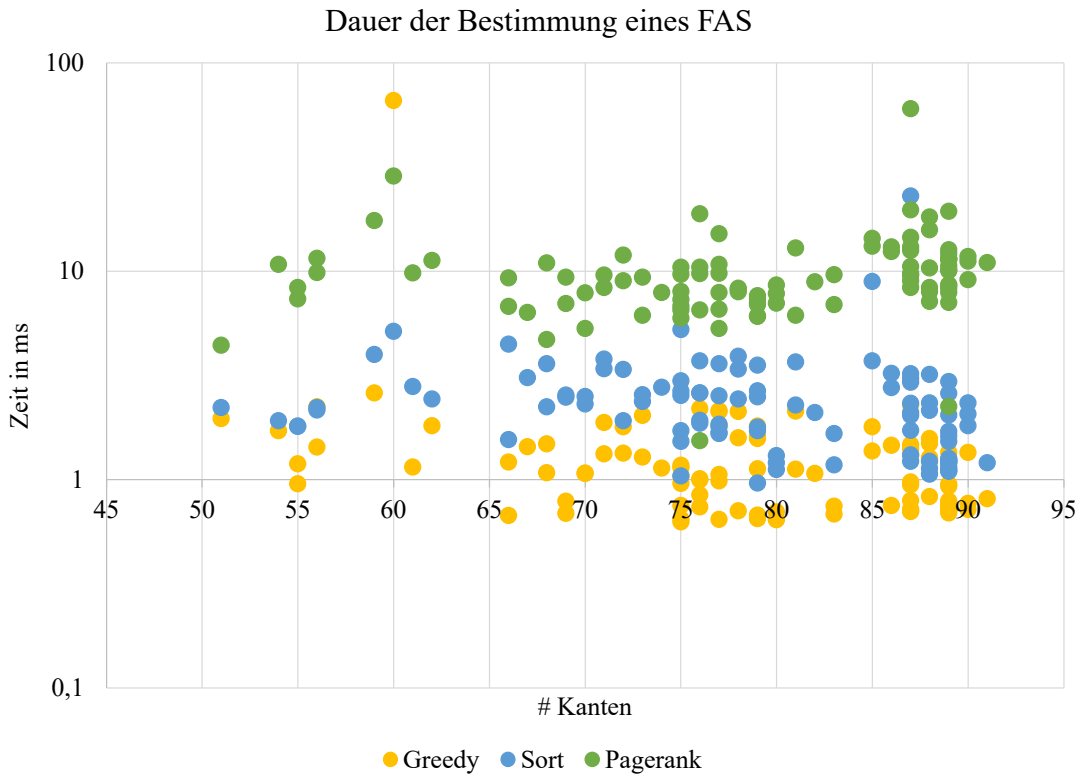


Abb. 4.3.: Benötigte Zeit für die Bestimmung eines FAS bei den verschiedenen Kreisbrecher-Algorithmen

Hier muss allerdings erwähnt werden, dass in dieser Arbeit der Pagerank-Algorithmus bewusst nicht auf die stark zusammenhängenden Komponenten des Graphen angewandt wurde, wie es von den Autoren vorgeschlagen wurde. Als Basis wurde hier der gesamte Graph verwendet, da es wegen der vergleichsweise geringen Kantenzahl kaum stark zusammenhängende Komponenten in den hier verwendeten Graphen gibt. Wendet man den Algorithmus auf die Zusammenhangskomponenten eines Graphen mit höherer Kantenzahl an, ergeben sich eventuell andere Resultate.

Insgesamt unterscheiden sich die Resultate der Algorithmen sehr stark. Der Greedy-Algorithmus ist für die Zeichnung von Prozessgraphen zweifelsfrei am besten geeignet, da hier sowohl die Laufzeit, als auch die Größe des FAS am geringsten sind. Wegen des vergleichsweise geringen durchschnittlichen Grades der Knoten in Prozessgraphen reicht an dieser Stelle die einfachste Option aus, es sind keine komplexeren Ansätze wie der Pagerank-Algorithmus nötig. Falls aber der Greedy-Ansatz nicht zur Verfügung steht, wäre letzterer aber vermutlich dem Sort-Ansatz vorzuziehen. Die hohe Laufzeit kann in

Kauf genommen werden, da die Laufzeiten insgesamt auf einem sehr niedrigen Niveau liegen. Der Fokus sollte auf ein möglichst kleines FAS gelegt werden.

Insgesamt lässt sich festhalten, dass der Greedy-Algorithmus den besten und schnellsten Ansatz bietet. Die anderen Algorithmen eignen sich aber grundsätzlich auch, da sie ebenso akzeptable Ergebnisse liefern.

4.3.2. Weitere Aspekte der Kreisbrecher-Algorithmen

Die Bestimmung des FAS ist die wichtigste Aufgabe der Kreisbrecher-Algorithmen und wurde daher im Rahmen der Evaluation zuerst betrachtet. Das FAS wirkt sich aber auf andere visuelle Aspekte, wie die Anzahl der Kantenkreuzungen und -knicke sowie die Größe des Zeichenbereiches aus. Daher werden diese Aspekte im Folgenden genauer betrachtet.

		Greedy	Sort	Pagerank
Kreuzungen	Anzahl	16.11	31.54	31.43
	Standardabw.	15.37	29.89	32.64
	pro Kante	0.27	0.49	0.48
Knicke	Anzahl	109.69	115.57	114.40
	Standardabw.	81.79	90.34	83.42
	pro Kante	1.92	1.96	2.02
Zeichenfläche	Größe in Mpx	3.052	3.333	3.112

Tab. 4.3.: Weitere Auswertung der Kreisbrecher-Algorithmen anhand von Mittelwerten

Tabelle 4.3 zeigt die Kennzahlen der drei Algorithmen. Dabei wird deutlich, dass sich die Größe des FAS direkt auf die Anzahl der Kantenkreuzungen auswirkt, denn diese ist beim Greedy-Algorithmus nur halb so groß wie bei den beiden anderen Algorithmen. Dass die beiden anderen Algorithmen aber ähnlich viele Kreuzungen produzieren ist verwunderlich, da das durch den Sort-Algorithmus induzierte FAS tendenziell größer ist als das durch den Pagerank-Algorithmus induzierte FAS.

Auf die Anzahl der Kantenknicke sowie die Größe des Zeichenbereichs hat das FAS aber offenbar nur wenig Einfluss, denn diese beiden Kennzahlen sind für alle Algorithmen ähnlich. Zwar zeigt sich, dass auch hier der Greedy-Algorithmus etwas besser abschneidet, die Unterschiede sind aber durchaus vernachlässigbar.

Insgesamt wird deutlich, dass die Bestimmung des FAS eine sehr wichtige Rolle einnimmt, da sich die Anzahl der umgekehrten Kanten direkt auf die Kreuzungen und damit auf die Qualität der Zeichnung auswirkt. Der Kreisbrecher-Algorithmus sollte daher mit Bedacht gewählt werden. Im Anhang unter A.2 befinden sich exemplarische Zeichnungen eines Prozesses unter Verwendung aller drei Algorithmen, in welchen die Unterschiede sichtbar

werden. Trotz der Differenzen sind aber alle Graphen gut verständlich, was verdeutlicht, dass sich grundsätzlich alle Algorithmen eignen.

4.3.3. Bewertung der Kantenformen

Wie bereits in Abschnitt 3.2.3 dargestellt, kann es bei der Zeichnung der Kanten als Bézierkurven zu zusätzlichen Kreuzungen von Kanten kommen. Weniger Kreuzungen durch Bézierkurven sind aber nicht möglich, da die Zeichnung der Bézierkurven auf dem orthogonalen Layout basiert. Fraglich ist daher, wie viele zusätzliche Kreuzungen durch Bézierkurven zweiten und dritten Grades entstehen.

Tabelle 4.4 zeigt die mittlere Anzahl der Kreuzungen bei Graphen mit und ohne Bézierkurven, sowie deren Standardabweichung und die für Berechnung und Zeichnung benötigte Zeit. Dabei wird deutlich, dass durch die Verwendung von Bézierkurven durchaus mehr Kantenkreuzungen entstehen. Im Schnitt sind es allerdings nur 2.61 und damit 8.84% zusätzliche Kreuzungen bei Bézierkurven zweiten Grades und 2.39 zusätzliche Kreuzungen bei Bézierkurven dritten Grades, was 8.89% mehr Kreuzungen entspricht. Dies zeigt einerseits, dass die beiden Bézierkurven-Formen sehr ähnliche Ergebnisse liefern und dass sie andererseits im Schnitt nur wenige zusätzliche Kreuzungen erzeugen.

	Orth.	Bez. 2.Gr.	Bez. 3.Gr.
Anz. Kreuzungen	26.36	28.97	28.75
Anz. Kreuzungen pro Kante	0.412	0.449	0.449
Standardabw. Kreuzungen	28.01	30.80	30.08
Zeichendauer im ms	15.49	20.61	20.57

Tab. 4.4.: Auswertung der Kantenformen anhand von Mittelwerten

Die Untersuchung der mittleren Zeichendauer zeigt ähnliche Ergebnisse. Klar ist hier, dass die orthogonale Zeichnung weniger Zeit benötigt als die Erstellung der orthogonalen Zeichnung mit anschließender Berechnung der Bézierkurven. Auch dass sich die benötigten Zeiten für Bézierkurven zweiten oder dritten Grades nur wenig unterscheiden, ist nicht verwunderlich, da die Berechnung der Kurven dem selben Schema folgt. Interessant ist aber, dass die Berechnung der Bézierkurven nur etwa 30% mehr Zeit benötigt als die orthogonale Zeichnung.

Dies zeigt, dass alle Kurvenformen technisch effizient umsetzbar sind. Die zusätzlichen Kantenkreuzungen durch Bézierkurven sind im Schnitt sehr gering, können aber vor allem bei größeren Prozessgraphen die Lesbarkeit beeinträchtigen und die Qualität der Zeichnungen deutlich mindern. Bei größeren Graphen muss daher im Einzelfall entschieden werden, ob die leichtere Verfolgbarkeit der Kanten durch Abrundung oder die Vermeidung zusätzlicher Kreuzungen höhere Priorität hat.

5. Fazit

Im Rahmen dieser Bachelorarbeit wurde ein vollständiger Algorithmus zur übersichtlichen Darstellung von Prozessgraphen auf Basis einer Aktivitätstabelle entwickelt. Dieser Algorithmus integriert nicht nur die Aktivitätsbeschreibungen, sondern auch die Häufigkeiten der Knoten und Kanten in Form von Textlabels. Der Prozess wird hierarchisch dargestellt, wobei auf eine möglichst einheitliche Flussrichtung der Kanten geachtet wird.

Besonders hervorzuheben sind die Implementierung mehrerer Varianten zur Auflösung von Kreisen sowie deren Evaluation. Hier wurden ganz unterschiedliche Ansätze betrachtet, welche in unterschiedlichen Situationen sinnvoll sein können. Insgesamt zeigte sich, dass der vorgestellte Greedy-Algorithmus der einfachste und effizienteste Ansatz für das Auflösen von Kreisen in Prozessgraphen ist. Dieser bestimmt nicht nur das kleinste FAS, sondern produziert darüber hinaus auch bedeutend weniger Kantenkreuzungen als die beiden anderen vorgestellten Algorithmen.

Weiterhin wurden im Rahmen dieser Arbeit mit orthogonalen Kanten und Bézierkurven zweiten und dritten Grades verschiedene Darstellungsoptionen für die Kanten bereitgestellt. Dabei wurde besonders darauf geachtet, dass keine Kreuzungen von Bézierkurven und Textfeldern entstehen, indem Algorithmen zur Vermeidung solcher Überschneidungen entwickelt und implementiert wurden. Die anschließende Evaluation der Kantenformen zeigte, dass die Berechnung von Bézierkurven effizient möglich ist und nur wenige zusätzliche Kreuzungen entstehen, wodurch sie für kleine bis mittelgroße Graphen eine ästhetische Alternative zu orthogonalen Kanten darstellen.

Trotz dieser Fortschritte gibt es noch offene Punkte, die weiter bearbeitet werden können. Dazu gehört die Vermeidung doppelter Kantenkreuzungen der Bézierkurven, was beispielsweise durch eine Bestimmung der betroffenen Kanten mit anschließender Nachverarbeitung der Kontrollpunkte realisiert werden kann. Eine weitere Anpassung der Kanten betrifft die Abrundung von Ecken, bei welchen die Kontrollpunkte der Bézierkurve derart nah beeinander liegen, dass die Bézierkurve kaum noch erkennbar ist. Dies könnte ebenfalls über eine Nachverarbeitung der Kontrollpunkte gelöst werden.

In den Zeichnungen im Anhang A.2 wird außerdem deutlich, dass die Sortierung der Ports an einigen Stellen noch nicht optimal ist. Hier entstehen teilweise Kreuzungen, welche sich durch einfache Umordnung der Ports auflösen ließen. Die Positionen der Ports könnten im Anschluss an die Erstellung des Layouts nochmals angepasst werden, um solche Kreuzungen zu vermeiden.

Darüber hinaus könnte das Layout der Graphen noch etwas kompakter gestaltet werden. Die Knoten und Kanten werden aktuell mit relativ großem Abstand zueinander platziert, wodurch große Graphen vor allem auf kleineren Bildschirmen nur schwer vollständig darstellbar sind. Wird der freie Platz zwischen den Objekten in der Zeichnung effizienter genutzt, kann dies vermieden werden. Ein möglicher Ansatz hierfür wäre der von Hegemann und Wolff [HW23] vorgestellte Algorithmus, welcher nach Platzierung der Knoten und Kanten eine Nachverarbeitung vornimmt und das Layout kompaktifiziert. Dieser hebt allerdings an einigen Stellen die Zuordnung der Knoten zu den Ebenen auf, wodurch die Zeichnung wieder breiter wird. Dann sind die Reihenfolge der Prozessschritte sowie die Flussrichtung des Graphen eventuell wieder schwerer erkennbar.

Weitergehend könnte zusätzlich zu den statischen Zeichnungen ein Programm entwickelt werden, welches den Nutzern Zoom-Features sowie Möglichkeiten zur Interaktion mit dem Graphen, wie das dynamische Ein- und Ausblenden von Varianten, ermöglicht. Dies würde die Analyse des Prozesses erleichtern, da so kritische Varianten im Einzelnen betrachtet werden könnten.

Insgesamt bildet diese Arbeit eine solide Grundlage für die weitere Forschung und Entwicklung im Bereich der Prozessgraphen-Darstellung und bietet verschiedene Ansatzpunkte für zukünftige Verbesserungen und Erweiterungen.

Literaturverzeichnis

[AEHK10] Benjamin Albrecht, Philip Effinger, Markus Held und Michael Kaufmann: An automatic layout algorithm for BPEL processes. In: *Proceedings of the 5th International Symposium on Software Visualization*, SOFTVIS '10, Seite 173–182, New York, NY, USA, 2010. Association for Computing Machinery, 10.1145/1879211.1879237.

[BK02] Ulrik Brandes und Boris Köpf: Fast and Simple Horizontal Coordinate Assignment. In: *Graph Drawing*, Band 2265 der Reihe *Lecture Notes in Computer Science*, Seiten 31–44. Springer Berlin Heidelberg, 2002, 10.1007/3-540-45848-4_3.

[BP98] Sergey Brin und Lawrence Page: The Anatomy of a Large-Scale Hypertextual Web Search Engine. *Computer Networks*, 30(1-7):107–117, 1998, 10.1016/S0169-7552(98)00110-X.

[BWZ20] Ulrik Brandes, Julian Walter und Johannes Zink: Erratum: Fast and Simple Horizontal Coordinate Assignment. 2020. `https://arxiv.org/abs/2008.01252`.

[Die17] Reinhard Diestel: *Graph Theory*. Springer, Berlin, Germany, 5. Auflage, 2017, 10.1007/978-3-662-53622-3.

[ELS93] Peter Eades, Xuemin Lin und William F. Smyth: A fast and effective heuristic for the feedback arc set problem. *Information Processing Letters*, 47(6):319–323, 1993, 10.1016/0020-0190(93)90079-O.

[EW94] Peter Eades und Sue Whitesides: Drawing graphs in two layers. *Theoretical Computer Science*, 131(2):361–374, 1994, 10.1016/0304-3975(94)90179-1.

[Far02] Gerald Farin: *Curves and Surfaces for Computer-Aided Geometric Design: A Practical Guide*. Academic Press, San Diego, CA, USA, 5. Auflage, 2002, 10.1016/C2009-0-22351-8.

[Fou24] Python Software Foundation: *Python Documentation*. Python Software Foundation, 2024. `https://docs.python.org/3/`.

[GKNV93] Emden R. Gansner, Eleftherios Koutsofios, Stephen C. North und Kiem

Phong Vo: A technique for drawing directed graphs. *IEEE Transactions on Software Engineering*, 19(3):214–230, 1993, 10.1109/32.221135.

[GLT22] Vasileios Geladaris, Panagiotis Lionakis und Ioannis G. Tollis: Computing a Feedback Arc Set Using PageRank. In: *Graph Drawing and Network Visualization*, Band 13764 der Reihe *Lecture Notes in Computer Science*, Seiten 188–200. Springer International Publishing, 2022, 10.1007/978-3-031-22203-0_14.

[GPZ+14] Thomas Gschwind, Jakob Pinggera, Stefan Zugal, Hajo A. Reijers und Barbara Weber: A linear time layout algorithm for business process models. *Journal of Visual Languages & Computing*, 25(2):117–132, 2014, 10.1016/j.jvlc.2013.11.002.

[Gra69] Ronald L. Graham: Bounds on Multiprocessing Timing Anomalies. *SIAM Journal on Applied Mathematics*, 17(2):416–429, 1969, 10.1137/0117039.

[HW23] Tim Hegemann und Alexander Wolff: A Simple Pipeline for Orthogonal Graph Drawing. In: *Graph Drawing and Network Visualization*, Band 14466 der Reihe *Lecture Notes in Computer Science*, Seiten 170–186, Cham, 2023. Springer Nature Switzerland, 10.1007/978-3-031-49275-4_12.

[JSGB00] Bill Joy, Guy Steele, James Gosling und Gilad Bracha: *The Java language specification*, 2000. `https://docs.oracle.com/javase/specs/jls/se17/html/`.

[Kar72] Richard M. Karp: Reducibility among combinatorial problems. In: *Proceedings of a Symposium on the Complexity of Computer Computations*, Band 40, Seiten 85–103., 1972, 10.2307/2271828.

[MSW19] Robin J.P. Mennens, Roeland Scheepens und Michel A. Westenberg: A stable graph layout algorithm for processes. *Computer Graphics Forum*, 38(3):725–737, 2019, 10.1111/cgf.13723.

[Pur00] Helen C. Purchase: Effective information visualisation: a study of graph drawing aesthetics and algorithms. *Interacting with Computers*, 13(2):147–162, Dezember 2000, 10.1016/S0953-5438(00)00032-1.

[Sed12] Thomas W. Sederberg: *Computer Aided Geometric Design*, Kapitel 2.5. Brigham Young University, Provo, UT, USA, 2012. `http://hdl.lib.byu.edu/1877/2822`.

[SST16] Michael Simpson, Venkatesh Srinivasan und Alex Thomo: Efficient computation of feedback arc set at web-scale. *Proc. VLDB Endow.*, 10(3):133–144, 2016, 10.14778/3021924.3021930.

[STT81] Kozo Sugiyama, Shojiro Tagawa und Mitsuhiko Toda: Methods for Visual Understanding of Hierarchical System Structures. *IEEE Transactions on Systems, Man, and Cybernetics*, 11(2):109–125, 1981, 10.1109/TSMC.1981.4308636.

[vdA12] Wil van der Aalst: Process Mining: Overview and Opportunities. *ACM Trans. Manage. Inf. Syst.*, 3(2), 2012, 10.1145/2229156.2229157.

[vdAAdM+11] Wil van der Aalst, Arya Adriansyah, Ana Karla Alves de Medeiros, Franco Arcieri, Thomas Baier, Tobias Blickle, Jagadeesh Chandra Bose, Peter van den Brand, Ronald Brandtjen, Joos Buijs, Andrea Burattin, Josep Carmona, Malu Castellanos, Jan Claes, Jonathan Cook, Nicola Costantini, Francisco Curbera, Ernesto Damiani, Massimiliano de Leoni, Pavlos Delias, Boudewijn F. van Dongen, Marlon Dumas, Schahram Dustdar, Dirk Fahland, Diogo R. Ferreira, Walid Gaaloul, Frank van Geffen, Sukriti Goel, Christian Günther, Antonella Guzzo, Paul Harmon, Arthur ter Hofstede, John Hoogland, Jon Espen Ingvaldsen, Koki Kato, Rudolf Kuhn, Akhil Kumar, Marcello La Rosa, Fabrizio Maggi, Donato Malerba, Ronny S. Mans, Alberto Manuel, Martin McCreesh, Paola Mello, Jan Mendling, Marco Montali, Hamid R. Motahari-Nezhad, Michael zur Muehlen, Jorge Munoz-Gama, Luigi Pontieri, Joel Ribeiro, Anne Rozinat, Hugo Seguel Pérez, Ricardo Seguel Pérez, Marcos Sepúlveda, Jim Sinur, Pnina Soffer, Minseok Song, Alessandro Sperduti, Giovanni Stilo, Casper Stoel, Keith Swenson, Maurizio Talamo, Wei Tan, Chris Turner, Jan Vanthienen, George Varvaressos, Eric Verbeek, Marc Verdonk, Roberto Vigo, Jianmin Wang, Barbara Weber, Matthias Weidlich, Ton Weijters, Lijie Wen, Michael Westergaard und Moe Wynn: Process Mining Manifesto. In: *Business Process Management Workshops*, Band 99 der Reihe *Lecture Notes in Business Information Processing*, Seiten 169–194. Springer Berlin Heidelberg, 2011, 10.1007/978-3-642-28108-2_19.

[You63] D. Younger: Minimum Feedback Arc Sets for a Directed Graph. *IEEE Transactions on Circuit Theory*, 10(2):238–245, 1963, 10.1109/TCT.1963.1082116.

[ZWBW22] Johannes Zink, Julian Walter, Joachim Baumeister und Alexander Wolff: Layered Drawing of Undirected Graphs with Generalized Port Constraints. *Computational Geometry: Theory and Applications*, 105–106(101886):1–29, 2022, 10.1016/j.comgeo.2022.101886.

A. Anhang

A.1. Algorithmen

Algorithmus 1: GreedyFAS(Graph G)

Eingabe: Gerichteter Graph G
Ausgabe: Lineare Sortierung der Knoten s

```
1  s_1 ← ∅
2  s_2 ← ∅
3  while G ≠ ∅ do
4      while G enthält eine Senke do
5          wähle eine Senke u ∈ V(G)
6          s_2 ← u s_2
7          G ← G\u
8      while G enthält eine Quelle do
9          wähle eine Quelle u ∈ V(G)
10         s_1 ← s_1 u
11         G ← G\u
12     wähle einen Knoten u ∈ V(G), für welchen δ(u) maximal ist
13     s_1 ← s_1 u
14     G ← G\u
15 return s = s_1 s_2
```

Algorithmus 2: SortFAS(Vertex[] A)

Eingabe: Sortierung A von Knoten $V(G)$ eines gerichteten Graphen G

```
1  foreach v ∈ A do
2      val ← 0
3      min ← 0
4      loc ← Position von v
5      foreach Position j from loc − 1 down to 0 do
6          w ← Knoten auf Position j
7          if (v, w) ∈ E(G) then
8              val ← val − 1
9          else if (w, v) ∈ E(G) then
10             val ← val + 1
11         if val ≤ min then
12             min ← val
13             loc ← j
14     Füge v in A auf Position loc ein
```

Algorithmus 3: LineGraph(Graph G)

Eingabe: Gerichteter Graph G

Ausgabe: Gerichteter Kantengraph $L(G)$

```
1
2  v ← zufälliger Knoten aus V(G)
3  prev ← nil
4  L(G) ← ∅
5
6  Procedure GetLineGraph(G, L(G), v, prev)
7  Markiere v als besucht
8  foreach Kante (v, u) ausgehend aus v do
9      z ← Knoten aus L(G) zu Kante (v, u)
10     if prev ≠ nil then
11         E(L(G)) ← E(L(G)) ∪ {(prev, z)}
12     if u wurde noch nicht besucht then
13         GetLineGraph(G, L(G), u, z)
14     else
15         foreach Knoten k ∈ V(L(G)) mit Kante ausgehend aus u do
16             E(L(G)) ← E(L(G)) ∪ {(z, k)}
17 return L(G)
```

Algorithmus 4: PageRank(Graph G, int k)

Eingabe: Gerichteter Graph G, Anzahl an Iterationen k
Ausgabe: PageRank-Scores R der Knoten $V(G)$

```
1 foreach v ∈ V(G) do
2     R(v) ← 1/|V(G)|
3 for k Iterationen do
4     foreach v ∈ V(G) do
5         R_old(v) ← R(v)
6     foreach v ∈ V(G) do
7         R(v) ← Σ_{u: (u,v)∈E} R_old(u) / |{u: (v,u)∈E}|
8 return R
```

Algorithmus 5: PageRankFAS(Graph G)

Eingabe: Gerichteter Graph G
Ausgabe: Feedback Arc Set von G

```
1  fas ← ∅
2  while G enthält Kreise do
3      L(G) ← ∅
4      v ← zufälliger Knoten aus V(G)
5      GetLineGraph(G, L(G), v, nil)
6      PageRank(L(G))
7      u ← Knoten von L(G) mit dem höchsten PageRank-Score
8      e ← zu u gehörige Kante aus G
9      fas ← fas ∪ {e}
10     G ← G\e
11 return fas
```

A.2. Beispiellayouts von verschiedenen Prozessgraphen

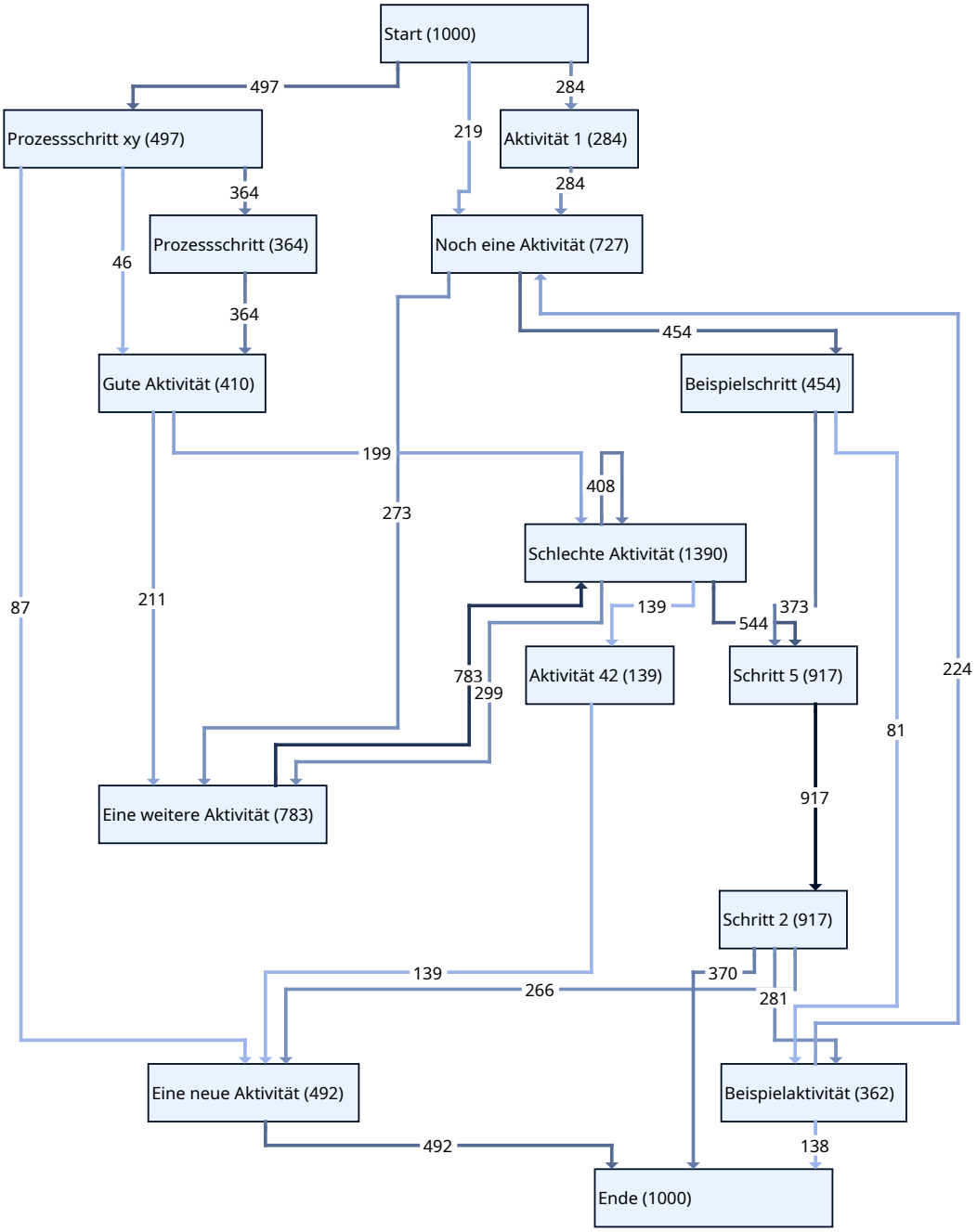


Abb. A.1.: Prozessgraph eines Beispiel-Prozesses mit orthogonalen Kanten, welcher mittels des Greedy-Algorithmus erstellt wurde

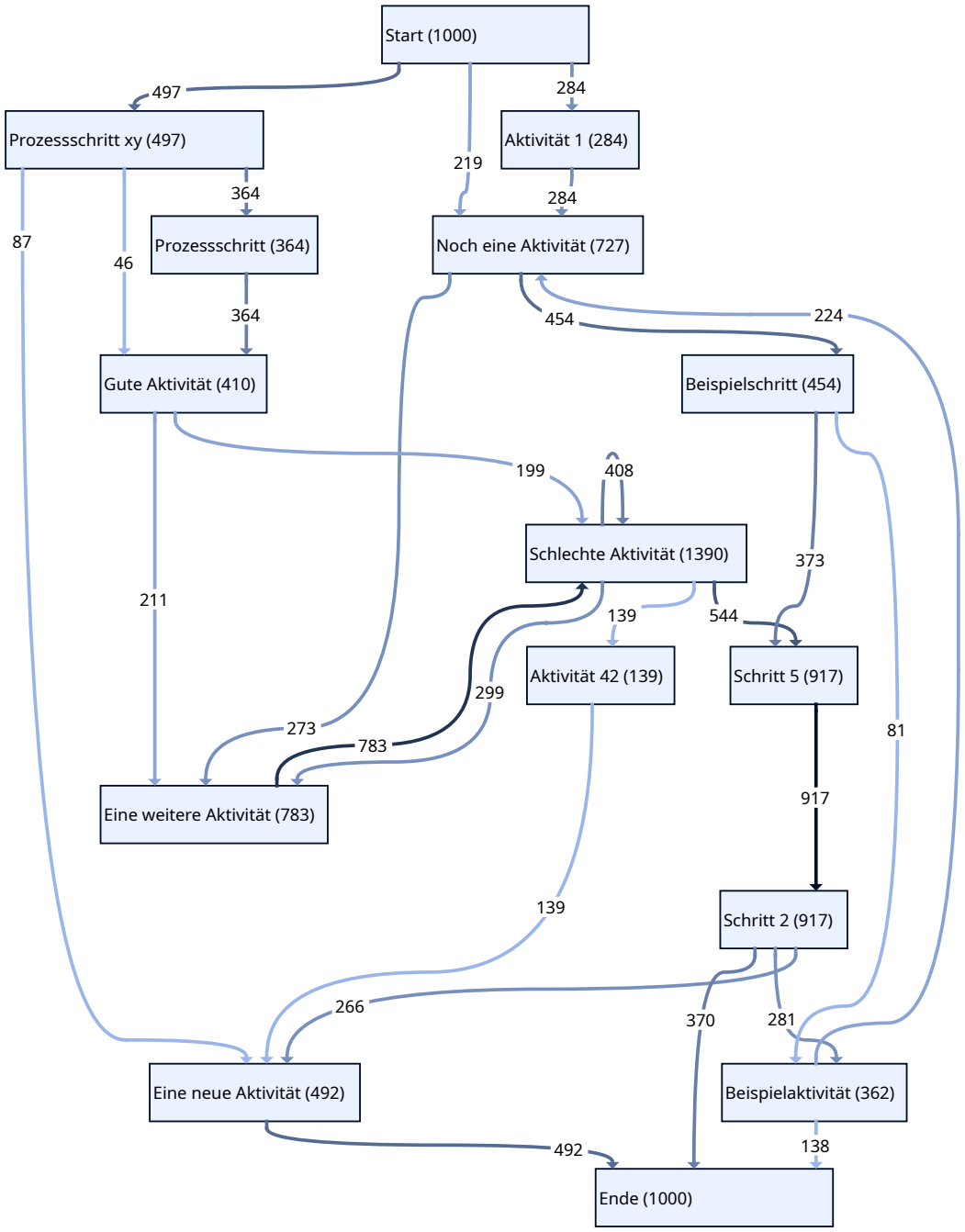


Abb. A.2.: Prozessgraph eines Beispiel-Prozesses mit Bézierkurven zweiten Grades, welcher mittels des Greedy-Algorithmus erstellt wurde

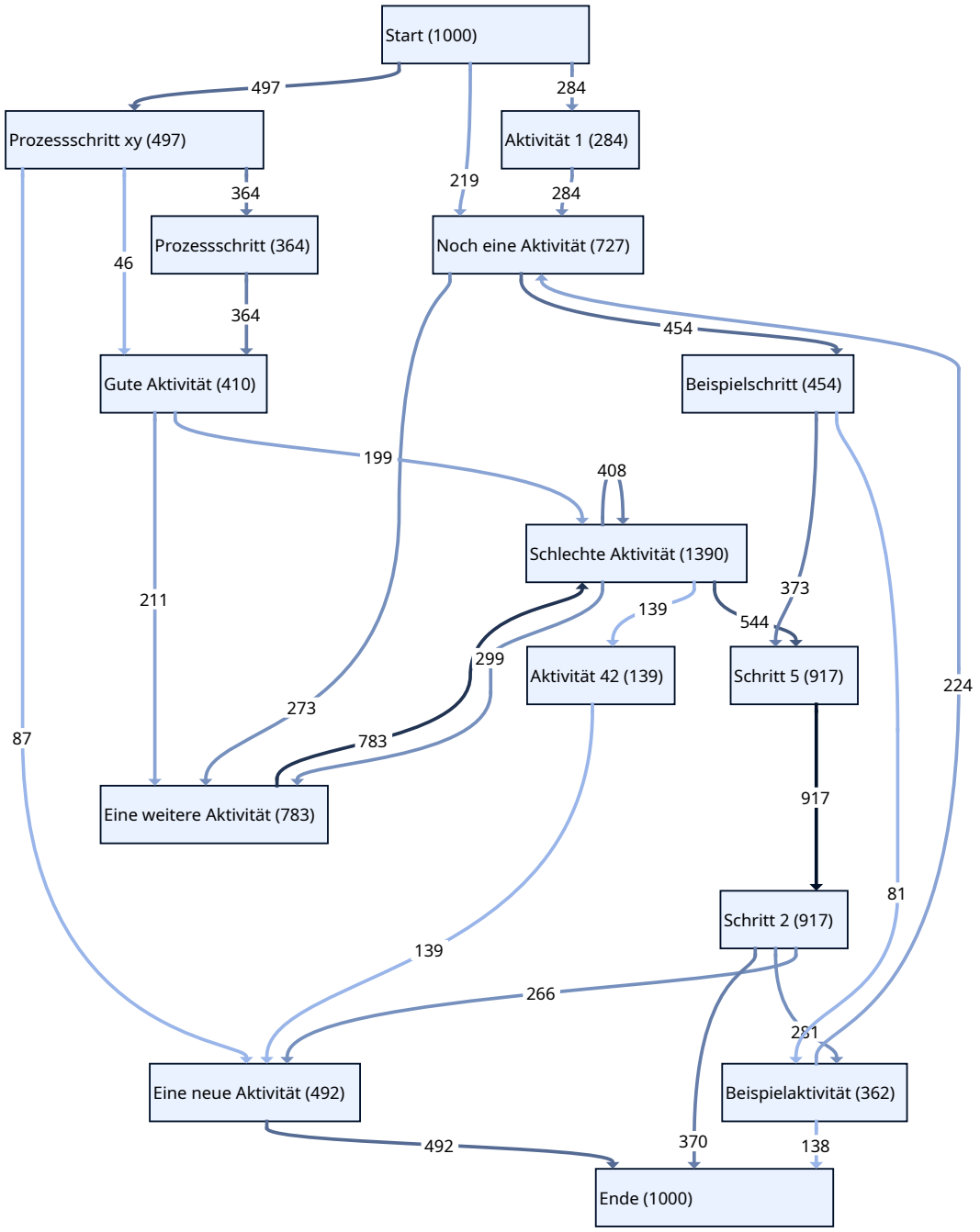


Abb. A.3.: Prozessgraph eines Beispiel-Prozesses mit Bézierkurven dritten Grades, welcher mittels des Greedy-Algorithmus erstellt wurde

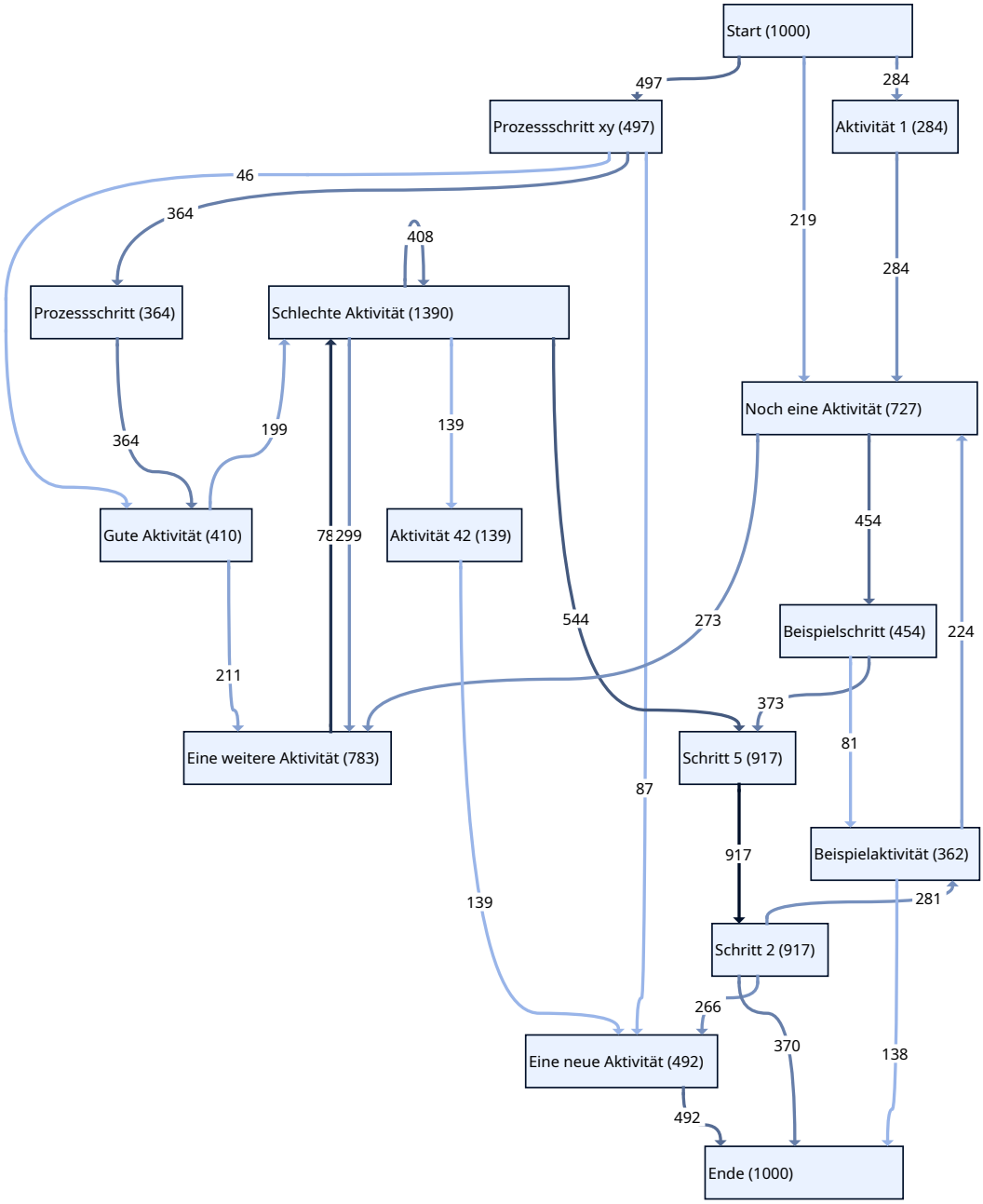


Abb. A.4.: Prozessgraph eines Beispiel-Prozesses mit Bézierkurven zweiten Grades, welcher mittels des Sort-Algorithmus erstellt wurde

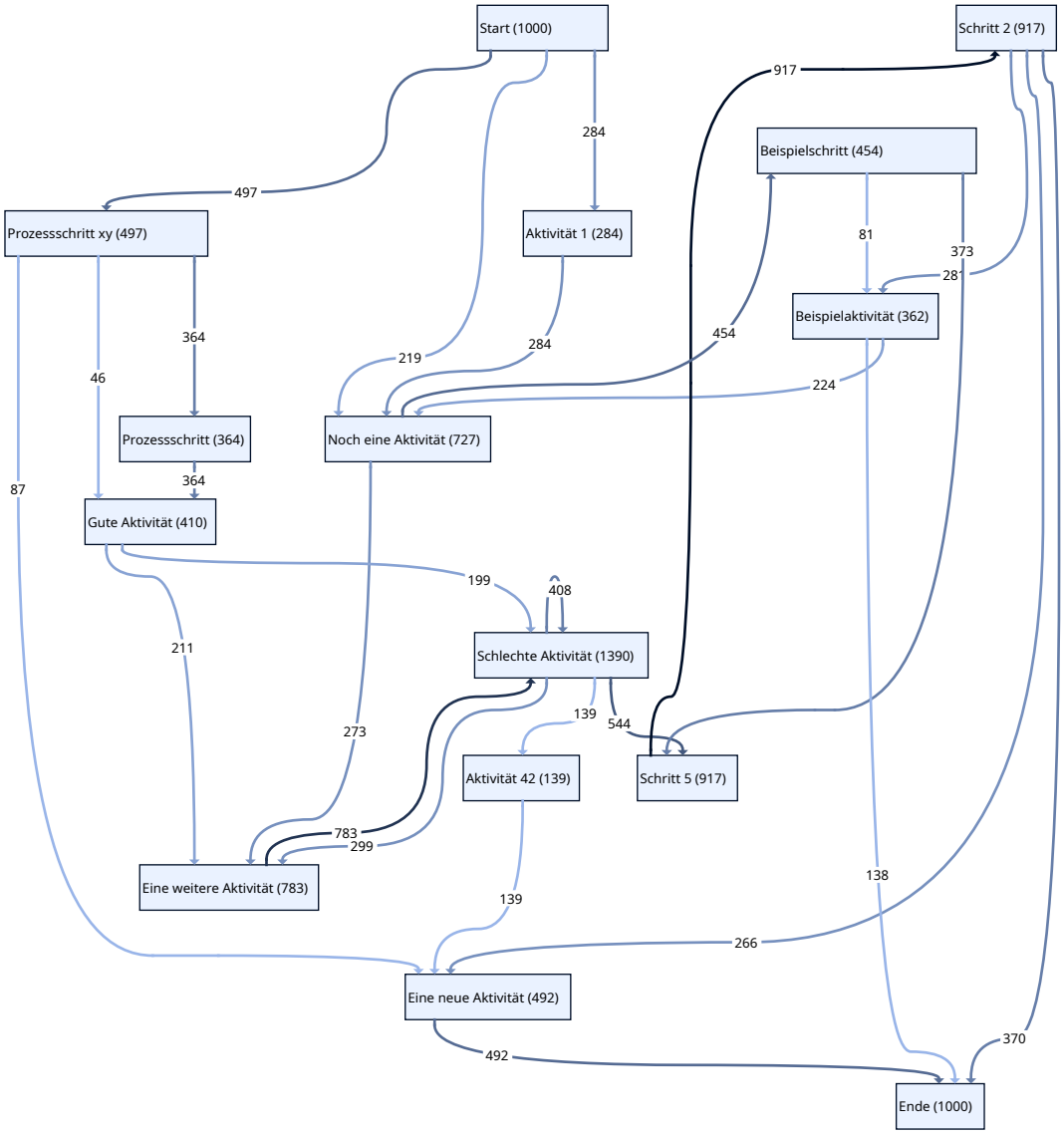


Abb. A.5.: Prozessgraph eines Beispiel-Prozesses mit Bézierkurven zweiten Grades, welcher mittels des Pagerank-Algorithmus erstellt wurde