1
Background

1.1 Graphen

Ein Graph repräsentiert eine Gruppe von Objekten und deren Verbindungen miteinander. Die einzelnen Objekte nennt man Knoten, während die Verbindungen als Kanten bezeichnet werden. Wie in Abbildung 1.1 werden Knoten oft als Kreise dargestellt und Kanten als Linien, die zwei der Kreise miteinander verbinden. Graphen können verwendet werden um beispielsweise Netzwerke von Computern zu modellieren. In der Abbildung stünde dann jeder Kreis für einen Rechner im Netzwerk und jede Linie für eine Netzwerkverbindung. Es ist allerdings nicht nur möglich physische Objekte und Verbindungen darzustellen, sondern auch abstraktere Konzepte. Satzstrukturen in einem Buch könnten beispielsweise mit einem Graphen analysiert werden, indem jeder Knoten ein Wort aus dem Text darstellt und jede Kante anzeigt, dass zwei Wörter häufig zusammen in einem Satz auftauchen. Ein bekanntes Beispiel für den Praxiseinsatz von Graphen ist der $PageRank - Algorithmus$ [1], welcher als Basis für die Suchmaschine Google diente.

1.1.1 Bäume

Die in dieser Arbeit beschriebene Visualisierung befasst sich mit Bäumen, welche eine spezielle Form von Graphen sind, für die zusätzliche Regeln gelten. Verbundene Knoten in einem Baum stehen in einer Vorgänger-Nachfolger Beziehung. Nachfolger eines Knotens werden auch als dessen Kinder bezeichnet. Zur Veranschaulichung ist auf Abbildung 1.2 ein Baum dargestellt. Auf dem Bild ist A der Vorgänger von Knoten B und dessen Nachfolger sind C und D. In einem Baum hat jeder Knoten genau einen Vorgänger und beliebig viele Nachfolger. Die einzige Ausnahme zu dieser Regel bildet die sogenannte Wurzel, die keinen Vorgänger hat und somit quasi der Ursprung des Baumes ist. In diesem Fall ist die Wurzel der rot dargestellte Knoten A. Von Blättern – hier in weiß dargestellt – spricht man, wenn Knoten keine Nachfolger haben. B hat sowohl einen Vorgänger als auch Kinder, was ihn zu einem der inneren Knoten macht, die in schwarz abgebildet sind. Genauer gesagt sind alle Knoten, die keine Blätter sind, innere Knoten, was auch die Wurzel mit einschließt. Die Höhe eines Baumes bezeichnet die Strecke von der Wurzel bis zu dem Blatt, welches am weitesten entfernt ist. Die Entfernung misst man

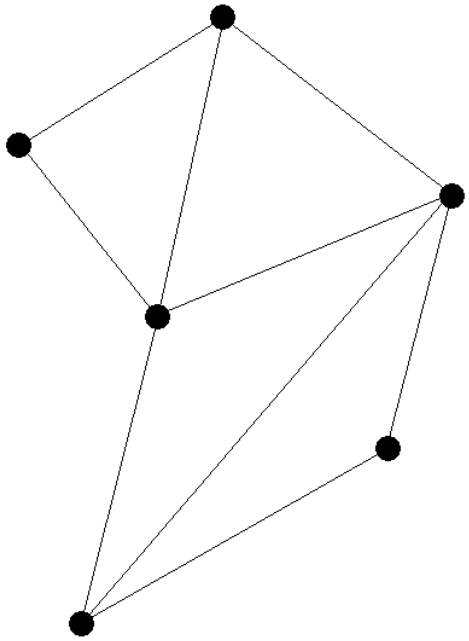

Abbildung 1.1: Ein Beispiel für einen Graphen

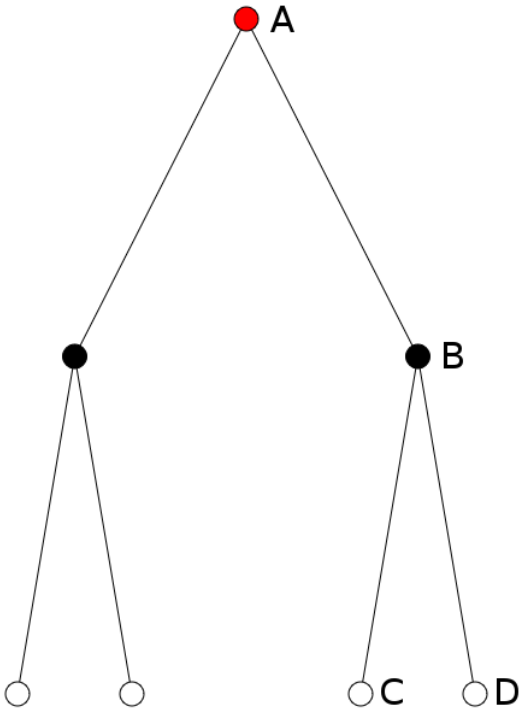

Abbildung 1.2: Ein Beispiel für einen Baum

in der Anzahl an Knoten, die zwischen den Beiden liegen, wobei man die Wurzel und das Blatt mitzählt. In Abbildung 1.2 wäre die Distanz von Knoten *A* zu *D* also 3. Das ist auch die Höhe des Baumes, da es kein Blatt gibt, das weiter von *A* entfernt ist.

Bäume finden Anwendung in den verschiedensten Gebieten. Im Bereich der Künstlichen Intelligenz werden sie zur Ermittlung optimaler Züge in Brettspielen wie Schach verwendet, in der Ahnenforschung kennt man sie als Stammbäume und sie können zur Modellierung von Entscheidungsabfolgen genutzt werden. Im zuletzt genannten Fall spricht man auch von Entscheidungsbäumen.

1.1.2 Entscheidungsbäume

In Abbildung 1.3 ist beispielhaft ein Entscheidungsbaum zu sehen, der in stark vereinfachter Form die Entscheidungsfindung zur Frage zeigt, ob beim Verlassen des Hauses ein Regenschirm mitgenommen werden sollte. Innere Knoten sind als Ellipsen dargestellt und Blätter, welche endgültige Ergebnisse repräsentieren, sind rechteckig. Die Beschriftung der Knoten zeigt an, welche Aussage oder Frage sie symbolisieren. Bei Verzweigung *A* muss zwischen den Möglichkeiten „Es regnet“ und „Es regnet nicht“ entschieden werden, wobei im Ersten Fall sofort das Ergebnis „Regenschirm mitnehmen“ erreicht wird. Regnet es nicht, so muss bei Verzweigung *B* der Wetterbericht oder das Bauchgefühl zurate gezogen werden um zwischen den Möglichkeiten „Es wird regnen“ und „Es wird nicht regnen“ zu entscheiden, welche dann zu den Ergebnissen „Regenschirm mitnehmen“ beziehungsweise „Regenschirm weglassen“ führen.

Natürlich handelt es sich hierbei zum erleichterten Verständnis um ein sehr simples Beispiel. Man kann sich vorstellen, das bei der Repräsentation von komplexeren Abläufen, wie

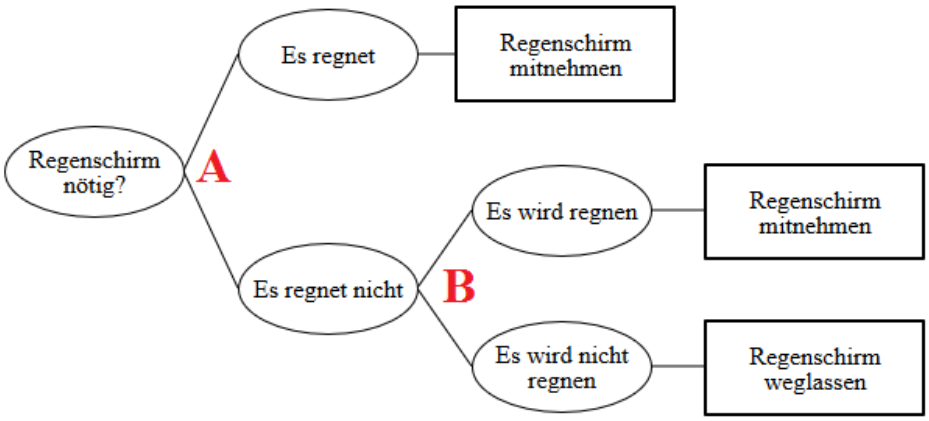

Abbildung 1.3: Ein simpler Entscheidungsbaum

dem Behandlungsverfahren von Krankheiten bedeutend größere Bäume entstehen.

1.2 HTML und CSS

1.3 SVG

SVG steht für „Scalable Vector Graphics“ (kurz: Vektorgrafik) und bezeichnet eine vom World Wide Web Consortium aufgestellte Spezifikation zur Definition von Vektorgrafiken. [todo: quelle] Durch Verwendung des HTML-Elements *svg* bietet sich damit eine einfache, von allen Browsern unterstützte Möglichkeit, Grafiken mit beliebigen Formen und Farben in eine Internetseite einzubinden. Dazu können im Inneren eines *svg* verschiedene Elemente benutzt werden. Im Folgenden bezieht sich die Schreibweise „SVG“ immer auf Vektorgrafiken im Allgemeinen, während mit *svg* das HTML-Element gemeint ist.

1.3.1 Geometrische Grundformen

Für einige Grundformen gibt es eigene Elemente. Zum Beispiel kann mit *circle* ein Kreis erzeugt werden, mit *rect* ein Rechteck. Dabei können genaue Angaben über Position, Größe, Farbe, Umrandung und einige Andere Eigenschaften gemacht werden.

1.3.2 Text

Ein *text*-Element dient zur Anzeige von Schrift. Es kann entweder reinen Text enthalten, oder Unter-Elemente wie *tspan*. Sowohl beim *text*- als auch beim *tspan*-Element können beispielsweise Position, Farbe und Schriftgröße bestimmt werden. Die *tspan*-Elemente dienen dazu, diese Angaben nur für den in ihnen enthaltenen Teil des Textes zu machen. So kann ein Text zum Beispiel in mehrere Zeilen aufgeteilt werden, indem jede Zeile innerhalb eines *tspan* steht und deren Positionierung entsprechend angepasst wird, was bei der Berechnung von Zeilenumbrüchen in Kapitel **??** zum Einsatz kommt.

1.3.3 Path

Zur Erzeugung komplexerer Formen dienen *path*-Elemente, die Pfade innerhalb des *svg* beschreiben. Ein Pfad ist zunächst eine Linie mit beliebig vielen Ecken und Wendungen. Der Pfad, den die Linie beschreibt wird von der Eigenschaft *d* festgelegt, welche eine Abfolge von Zeichenbefehlen enthält. Diese Liste von Anweisungen kann mit steigender Komplexität des Pfades sehr lang werden, weshalb das in Kapitel 1.4 beschriebene D3 einige Funktionen zu deren Generierung anbietet. Auch bei Pfaden gilt wieder, dass ihr Aussehen was zum Beispiel Farbe angeht frei wählbar ist. Ist ein Pfad geschlossen, das heißt endet er dort wo er begonnen hat, dann kann auch eine Füllfarbe angegeben werden, mit der die vom Pfad eingeschlossene Fläche gefüllt wird.

1.3.4 Bilder

Mit *image*-Elementen können Bilder in ein *svg* eingefügt werden. Gibt man Höhe und Breite über die Attribute *width* und *height* an, werden diese auf die entsprechende Größe skaliert.

1.3.5 Gruppieren von Elementen

Manchmal ist es wünschenswert zusammengehörende Elemente in Gruppen zusammenzufassen. Zu diesem Zweck gibt es *g*-Elemente, welche das Attribut $transform$ besitzen und beliebige andere Objekte, wie die zuvor beschriebenen *path*-, *circle*- oder *text*-Elemente enthalten können. Das $transform$-Attribut beschreibt Veränderungen, die an allen im *g* enthaltenen Elementen vorgenommen werden. Erlaubte Transformationen sind Verschiebung ($translate$), Skalierung (*scale*), Rotation (*rotation*) und Verzerrung (*skewX* bzw. *skewY*). Verschiebt man auf diese Weise alle gruppierten Elemente, bleibt deren relative Positionierung zueinander erhalten, das heißt ein Text, der vor der Verschiebung in der Gruppe zentriert ist, bleibt das auch nach der Verschiebung.

SVG können eingesetzt werden um Statistiken und andere Daten zu visualisieren. Zur Vereinfachung dieser Aufgabe liefert die Javascript Bibliothek D3 zahlreiche Hilfsmittel.

1.4 D3

D3 – kurz für Data-Driven Documents – ist eine unter BSD-Lizenz veröffentlichte Open-Source Javascript Bibliothek zur Visualisierung von Daten auf Internetseiten. Sie ist ein maßgebliches Hilfsmittel im Zuge dieser Arbeit. Das Konzept von D3 ist es, Datensätze mit Elementen eines HTML-Dokuments, wie *svg* und dessen Unterelemente, zu verknüpfen und dadurch großen Einfluss auf die Darstellung dieser Daten zu haben. Dabei werden neben Hilfsfunktionen zum Strukturieren der Datenanzeige auch diverse Möglichkeiten zur Erstellung und Manipulierung von Datenstrukturen angeboten.

1.4.1 Anwendungsbeispiele

Ein Vorteil von D3 besteht in seiner Verbreitung, die sich in der Menge verschiedenster Anwendungsbeispiele zeigt, welche im Internet zu finden sind. Auf Abbildung 1.4 sieht man eine Alterspyramide der US-Amerikanischen Bevölkerung im Jahr 2000. [todo: quelle] Daten zu

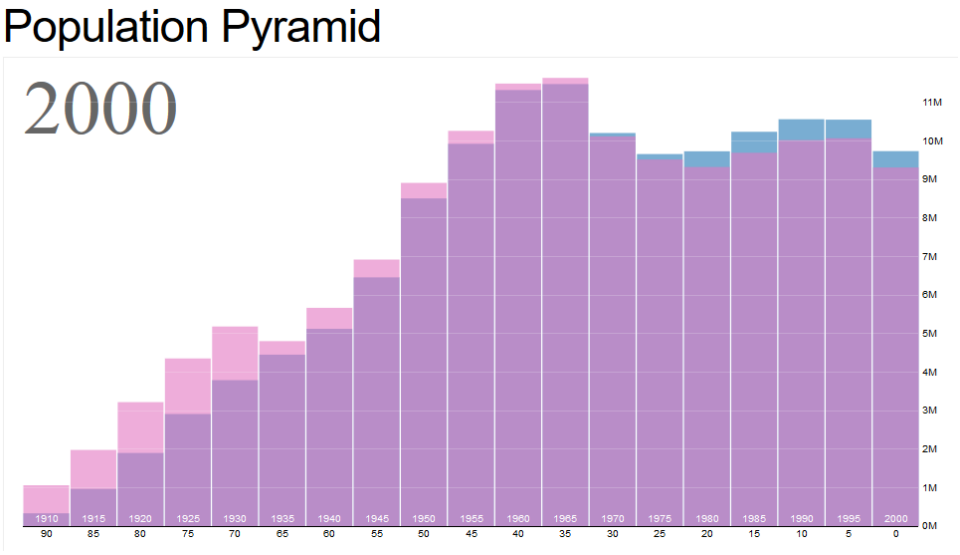

Abbildung 1.4: Darstellung einer Alterspyramide mit D3

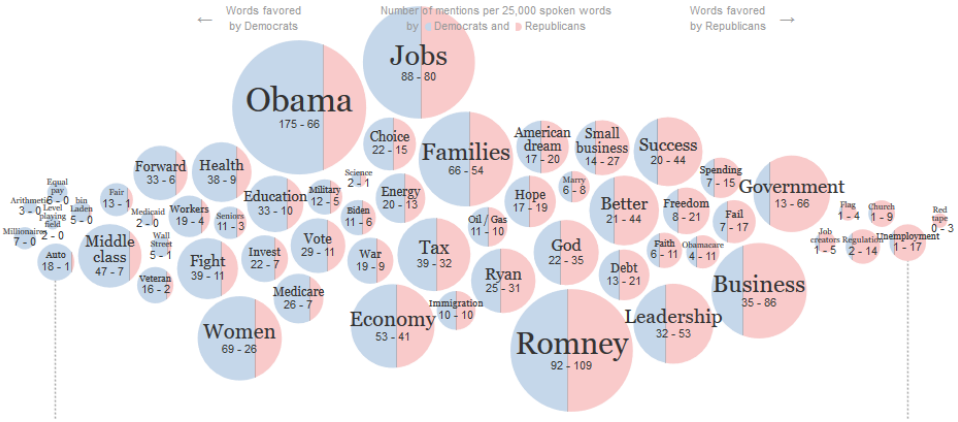

Abbildung 1.5: Benutzte Worte bei „National Conventions“ 2012 in den USA

Männern und Frauen sind dabei nicht – wie normalerweise üblich – getrennt voneinander dargestellt, sondern übereinander gelegt. Rosafarbene Balken stehen für Frauen, blaue für Männer und lila zeigt eine Überschneidung beider Farben an. Die x-Achse beschreibt das Alter der Personen und die y-Achse die Bevölkerungszahl. Mit Hilfe der Pfeiltasten kann die Jahreszahl schrittweise bis 1850 zurückgestellt werden, wobei die angezeigten Daten sich immer auf das eingestellte Jahr aktualisieren.

Ein anderes Beispiel, bei dem D3 zum Einsatz kommt, ist in Abbildung 1.5 zu sehen, die einem Onlineartikel der New York Times entnommen ist. [todo: quelle] Auf den „National Conventions“, welche in den USA traditionell von beiden Parteien vor den Präsidentschaftswahlen abgehalten werden, wurden die gesprochenen Wörter der Redner gezählt und in einem sogenannten Blasendiagramm dargestellt. Je häufiger ein Wort verwendet wurde, desto größer ist dessen Blase. Die Farbaufteilung in rot und blau beschreibt, wie häufig Republikaner bzw. Demokraten das Wort benutzten. Vorwiegend von Republikanern ausgesprochene Wörter finden sich eher auf der rechten Seite wieder, mehrheitlich von Demokraten verwendete Begriffe auf der linken.

Die Verschiedenheit dieser Beispiele macht die Vielseitigkeit von D3 deutlich, welche ein wichtiger Grund für dessen Verbreitung ist.

1.4.2 Module

Da D3 sehr umfangreich ist und die Meisten Anwendungen nicht alle Funktionen benötigen, ist es in Module unterteilt, die einzeln eingebunden werden können. Die für diese Arbeit besonders relevanten Module sind *Selections*, *Hierarchies*, *Shapes* und *Transitions*.

Selections

Selections sind einer der wichtigsten Bestandteile von D3. Sie dienen dazu, HTML-Elemente zu gruppieren, manipulieren und mit Daten zu verknüpfen. Man erstellt sie mit den Befehlen *d3.select* oder *d3.selectAll*, wobei *select* das erste auf die Übergabe passende und *selectAll* alle passenden Elemente in eine *selection* zusammenführt.

Über die Funktion *selection.data* können die Elemente einer *selection* mit beliebigen Daten verknüpft werden. Gibt man dabei jedem einzelnen Datensatz eine eindeutige *id* so teilt D3 automatisch Datensätze und HTML-Elemente in drei Gruppen auf:

1. Daten-Element-Paare, die bereits zuvor verknüpft wurden (*update selection*)
2. Daten, für die noch kein HTML-Objekt existiert (*enter selection*)
3. HTML-Elemente, für die kein Datensatz vorhanden ist (*exit selection*).

In vielen Fällen, wie zum Beispiel durch Nutzereingaben bei der in Kapitel 1.4.1 beschriebenen Alterspyramide, kann es zu Änderungen der Daten kommen, die bereits Auf dem Bildschirm zu sehen sind. In solchen Fällen kommt die *update selection* zum Einsatz. Sie ist die *selection*, die beim Aufruf von *selection.data* zurückgegeben wird und dient – wie der Name sagt – zur Aktualisierung der angezeigten Objekte. Beispielsweise können mit der Funktion *selection.attr* HTML-Attribute angepasst oder mit *selection.class* CSS-Klassen gesetzt werden. Jede auf einer *selection* aufgerufene Funktion wird dabei auf alle in ihr enthaltenen Objekte einzeln angewendet. Über die *update selection* hat man außerdem Zugriff auf die *enter* und *exit selection*.

Die *enter selection* erhält man durch den Funktionsaufruf *selection.enter*. Um HTML-Objekte für diese bisher nicht visualisierten Informationen hinzuzufügen wendet man die Methode *selection.append* auf die *enter selection* an und benennt dabei das anzuhängende Element. *Append* erstellt dann für jeden der Datensätze in der *selection* ein HTML-Element dieser Art und verknüpft die beiden miteinander.

Meist gilt für Objekte in der *exit selection*, dass sie nicht mehr angezeigt werden sollen, weil es keine korrespondierenden Daten mehr gibt. In diesem Fall kann mit einem Aufruf von *selection.exit* auf sie zugegriffen und alle Elemente durch *selection.remove* entfernt werden.

Hierarchies

Mit Hilfe von D3 *hierarchies* können Daten verarbeitet werden, die hierarchisch angeordnet sind, wie zum Beispiel Familienstammbäume oder Dateisysteme. Einzelne Datenpunkte werden dabei auch als Knoten (*node*) bezeichnet. Die Daten unterliegen den gleichen Regeln, wie

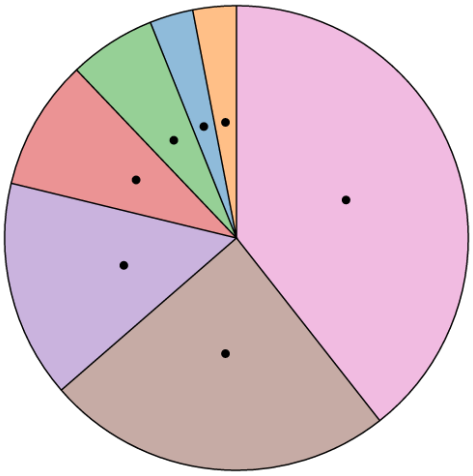

Abbildung 1.6: Ein Kreisdiagramm

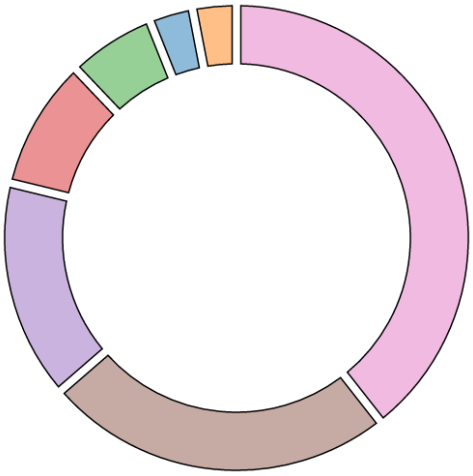

Abbildung 1.7: Ein Ringdiagramm

die Knoten eines Baumes (Kapitel 1.1.1), das heißt es gibt genau eine Wurzel und alle anderen Knoten haben einen Vorgänger und beliebig viele Nachfolger. Aufgrund dieser Eigenschaften kann eine *hierarchy* als Baum visualisiert werden. Die Funktion *d3.hierarchy* nimmt ein Javascript-Objekt entgegen, dass die Wurzel repräsentiert und erzeugt daraus eine *hierarchy*, in der jeder Knoten seine ursprünglichen Daten enthält und zusätzlich diverse Hilfsfunktionen, wie *node.descendants*, die eine Liste aller Nachfolgenden Knoten zurückgibt, oder *node.path* zur Berechnung des kürzesten Pfades zum übergebenen Zielknoten.

Die Funktion *d3.tree* berechnet unter Verwendung des Reingold-Tilford-Algorithmus (Reingold and Tilford [2]) eine Knotenanordnung für die Darstellung einer *hierarchy* als Baum. Dadurch erhält jeder Knoten eine x- und y-Koordinate im Bereich von 0 bis 1, die bei der Visualisierung beliebig skaliert werden können.

Shapes

Das *shapes* Modul dient zur Darstellung von beliebigen Formen. Es liefert dabei für verschiedene Anwendungsfälle unterstützende Funktionen. Es wird bei der hier beschriebenen Visualisierung von Bäumen verwendet, um die Kanten in Form von Linien zu zeichnen und für die Berechnung des Eingabemenüs (Kapitel 2.3.6 und 2.5). Dabei kommen insbesondere die Funktionen *d3.line*, *d3.pie* und *d3.arc* zum Einsatz.

D3.line generiert aus einem Start- und Endpunkt-Paar eine Reihe von Anweisungen zum Zeichnen einer Linie, die an ein *path*-Element in einem *svg* übergeben werden kann. Die Abbildungen 1.6 und 1.7 [todo: quelle] zeigen die Repräsentation einer Datenmenge als Kreis- und Ringdiagramm. Beide bestehen aus sieben Segmenten in unterschiedlichen Farben, wobei im Kreisdiagramm der Mittelpunkt jedes Segments mit einem schwarzen Punkt markiert ist. Im Ringdiagramm sind kleine Abstände zwischen den einzelnen Segmenten eingefügt.

D3.arc und *d3.pie* können zusammen benutzt werden, um solche Diagramme zu erzeugen. *D3.arc* wird verwendet um einen Bogengenerator zu erzeugen, der zu Übergaben von Start- und Endwinkeln passende Kreisbögen generiert. *D3.pie* erstellt aus einer Liste von Daten, wie zum Beispiel Stimmenanteilen bei einer Wahl, die nötigen Übergaben, damit der Bogengenerator automatisch ein passendes Ringdiagramm erzeugt. Man kann dabei unter Einsatz der Funktionen

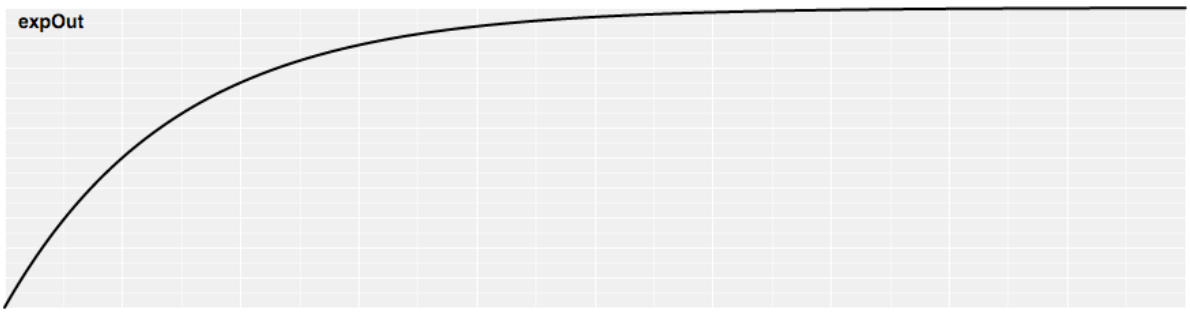

Abbildung 1.8: Die Übergangsfunktion *easeExpOut*

arc.innerRadius und *arc.outerRadius* den Innen- und Außenradius des erzeugten Rings festlegen, wobei ein Innenradius von 0 zur Erzeugung eines Kreisdiagramms führt (Abbildung 1.6). Eine nützliche Hilfsfunktion zur Positionierung von Beschriftungen oder Icons ist *arc.centroid*, welche den Mittelpunkt für jedes Segment liefert. Die schwarzen Punkte auf der Abbildung 1.6 wurden auf diese Weise generiert. Unter Verwendung der Funktion *pie.padAngle* können feste Abstände zwischen allen Segmenten eingefügt werden, was beim Beispiel des Ringdiagramms geschehen ist.

Transitions

Transitions bauen auf dem Konzept von *selections* auf und dienen dazu, Veränderungen an Angezeigten Objekten nicht abrupt vorzunehmen, sondern in gleichmäßigen Abstufungen vom Startzustand zum Endzustand überzugehen. Auf diese Weise können vielfältige Animationen realisiert werden. Man startet eine *transition* durch den Aufruf von *selection.transition* und kann anschließend alle Anzeigeänderungen, die man an der *selection* vornehmen kann auch über die zurückgegebene *transition* erreichen. Dabei kann mit der Funktion *transition.duration* eine Dauer und mit *transition.delay* eine Verzögerung des Übergangs in Millisekunden angegeben werden. Darüber hinaus bietet *transition.ease* die Möglichkeit eine Übergangsfunktion zu bestimmen, die vorgibt wie schnell sich der jeweilige Wert zu gegebenen Zeitpunkten ändert. Zum Beispiel kann eine Bewegung schnell beginnen und zum Ende hin langsamer werden, wofür die Übergangsfunktion *easeExpOut* geeignet wäre, deren Funktionsgraph in Abbildung 1.8 [todo: quelle] zu sehen ist. D3 bringt bereits eine Reihe solcher Funktionen mit, es können aber auch eigene definiert werden.

1.5 Struktur der verwendeten Daten

[todo: aufzählen, welche eigenschaften die übergebenen knoten haben. diagnosis, etc.]

2
Contribution

Most important chapter of the thesis. Describes what the author contributes as research. Discusses intuition, motivation, describes and reasons about necessity of proposed elements. Defines theses based on reasonable assumptions. Discusses relevant aspects of contribution. Approximately 30 to 40 pages. Can be split into multiple chapters.

2.1 Baumlayout

2.1.1 Linear vs. Radial

Aufgrund des begrenzten Platzes, der auf mobilen Geräten typischerweise zur Verfügung steht, ist zu entscheiden, wie Baumstrukturen möglichst platzsparend anzuordnen sind, ohne Übersichtlichkeit einzubüßen. Betrachtet werden dazu speziell die allgemein übliche Darstellung (linear) gegenüber einer kreisförmigen (radialen) Anordnung.

Die Abbildungen 2.1 und 2.2 zeigen einen Vergleich zwischen einem Baum in linearer (Abb. 2.1) und in radialer Anordnung (Abb. 2.2). In beiden Fällen ist derselbe Baum der Höhe 4 auf gleicher Fläche abgebildet. Er besteht aus 105 Knoten, wobei alle inneren Knoten jeweils vier Nachfolger haben. Der Wurzelknoten befindet sich beim linearen Layout am oberen Rand. Die übrigen Knoten sind in drei horizontalen Linien darunter angeordnet, wobei die Kinder der Wurzel auf der obersten Linie liegen, deren Kinder auf der mittleren und deren Kinder auf der untersten Linie. Beim radialen Layout ist die Wurzel in der Mitte abgebildet und alle anderen Knoten in drei konzentrischen Kreisen darum herum. Knoten der Tiefe 1 liegen auf dem innersten Kreis, der Tiefe 2 auf dem mittleren Kreis und die Blätter auf dem äußeren Kreis. Den Mittelpunkt der Kreise bildet die Wurzel. Man sieht deutlich, dass im Fall der vertikalen Anordnung (Abb. 2.1) nicht ausreichend Platz für alle Blätter zur Verfügung steht, wodurch es zu Überschneidungen kommt. In der radialen Anordnung (Abb. 2.2) können hingegen alle Knoten überschneidungsfrei dargestellt werden.

Um den Grund dafür zu veranschaulichen vergleicht man den Platz, der bei den beiden Herangehensweisen auf einer einzelnen Stufe des Baumes zur Verfügung steht. Unter der An-

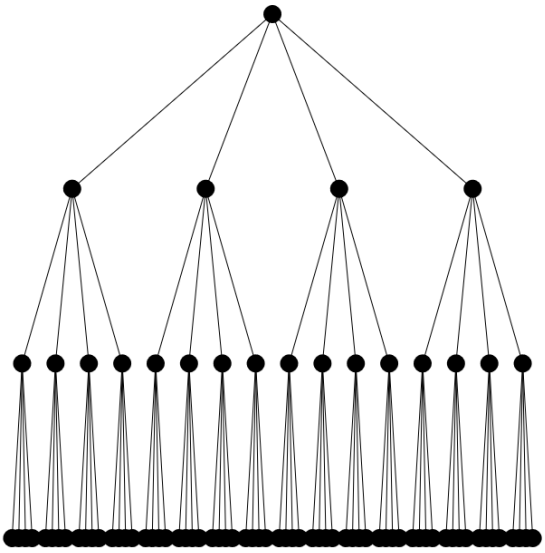

Abbildung 2.1: Lineare Baumdarstellung

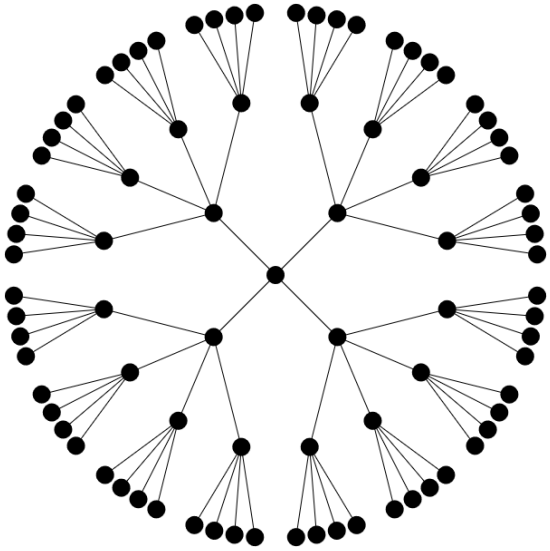

Abbildung 2.2: Radiale Baumdarstellung

nahme, dass für die gesamte Darstellung ein Quadrat mit Seitenlänge a zur Verfügung steht und alle Knoten einen Durchmesser von 1 haben ergibt sich für den Platz l_{linear} bzw. l_{radial}, der auf einer Stufe des Baumes verfügbar ist

$$\begin{aligned} l_{linear} &= a \\ l_{radial} &= 2 \cdot \pi \cdot r \qquad mit \quad 0 \leq r \leq \frac{a}{2}. \end{aligned}$$

Dabei bezeichnet r den Abstand einer Stufe zum Wurzelknoten. Setzt man l_{radial} und l_{linear} gleich und stellt nach r um, so ergibt sich

$$\begin{aligned} & l_{linear} = l_{radial} \\ \Rightarrow \quad & a = 2 \cdot \pi \cdot r \\ \Rightarrow \quad & \frac{a}{2 \cdot \pi} = r \\ \Rightarrow \quad & r \approx \frac{a}{6}. \end{aligned}$$

Da l_{linear} konstant ist kann man daraus ableiten

$$l_{radial} > l_{linear} \qquad \Leftrightarrow \qquad r > \frac{a}{6}.$$

Das bedeutet, dass ab einem Abstand zur Wurzel von mehr als $\frac{a}{6}$ im radialen Layout mehr Platz für jede Stufe verfügbar ist. Außerdem gilt: Je größer der Abstand r, desto mehr Platzersparnis. Das ist vor allem deshalb von Interesse, weil Bäume in der Regel die Eigenschaft haben, dass mit wachsender Baumtiefe auch die Anzahl der Knoten in der jeweiligen Tiefe wächst und diese tiefer liegenden Knoten in der kreisförmigen Darstellung weiter von der Wurzel entfernt sind. Das radiale Layout bietet somit genau dann mehr Platz, wenn üblicherweise mehr Platz benötigt wird. Aufgrund dieser Erkenntnis und der Tatsache, dass beide Darstellungsformen gute Übersichtlichkeit aufweisen, fiel die Wahl letztendlich auf das radiale Layout.

2.1.2 Polarkoordinaten

D3 bietet zur Berechnung einer sinnvollen Knotenanordnung von Bäumen die Funktion *d3.tree* an, welche unter Verwendung des Reingold-Tilford Algorithmus [todo: quelle] allen Knoten einer *hierarchy* x- und y-Koordinaten jeweils im Bereich von 0 bis 1 zuordnet. Es liegt dann in der Hand des Programmierers, diese sinnvoll zu interpretieren. Um die Knoten, wie in Abbildung 2.2 gezeigt in konzentrischen Kreisen anzuordnen, eignen sich Polarkoordinaten ausgezeichnet. Einem Vorschlag aus der Dokumentation von D3 folgend [todo: quelle] wird die y-Koordinate als Radius ρ, die x-Koordinate als Polarwinkel φ in Radiant interpretiert. [todo: bild]

$$\rho = y$$
$$\varphi = 2 \cdot \pi \cdot x$$

Zur Darstellung auf dem Bildschirm müssen ρ und φ anschließend in kartesische Koordinaten x_{screen} und y_{screen} umgerechnet werden. Die allgemeinen Umrechnungsformeln ergeben sich als

$$x_{screen} = \rho \cdot \cos(\varphi)$$
$$y_{screen} = \rho \cdot \sin(\varphi).$$

Bei dieser Umrechnung wird allerdings noch nicht berücksichtigt, dass die gegebenen Polarkoordinaten auf generischen Koordinaten im Bereich $[0,1]$ basieren. Zur korrekten Positionierung müssen noch eine Skalierung auf die verfügbare Breite (width) w und Höhe (height) h, sowie eine Verschiebung in die Mitte der Anzeige vorgenommen werden. Es entstehen die endgültigen Formeln:

$$x_{screen} = \rho \cdot \cos(\varphi) \cdot w + \frac{w}{2} \tag{2.1}$$
$$y_{screen} = \rho \cdot \sin(\varphi) \cdot h + \frac{h}{2}. \tag{2.2}$$

2.2 Reduzieren der angezeigten Knoten

Nachdem mit dem radialen Layout (Kapitel 2.1.1) eine erste Maßnahme zum Einsparen von Platz ergriffen wurde, stellt sich als nächstes die Frage, wie die Übersichtlichkeit weiter verbessert werden kann. Da Knoten auf dem Bildschirm Informationen in Form von Text beinhalten sollen, ist es abzusehen, dass jeder einzelne Knoten mehr Platz einnehmen wird, als z.B. in Abbildung 2.2 gezeigt ist. Es bietet sich an, immer nur aktuell relevante Knoten ein- und irrelevante auszublenden. Das erfordert je nach Eingabe dynamische Änderungen an der Anzeige der Baumstruktur (Kapitel 2.4.1). Zunächst muss jedoch entschieden werden, welche Knoten aktuell relevant oder irrelevant sind.

Dazu betrachten wir noch einmal den simplen Entscheidungsbaum in Abbildung 1.3. Die beiden Stellen, an denen Entscheidungen getroffen werden müssen, sind dort mit *A* und *B* markiert. In Situation *A* gilt es zu entscheiden, ob es regnet oder nicht. Um diese Entscheidung treffen zu können, ist nicht relevant, ob es zu einem späteren Zeitpunkt regnen wird oder nicht und welche Ergebnisse sich daraus ableiten lassen. Bei *B* wiederum sind vorherige Entscheidungsmöglichkeiten nicht von Interesse, ebenso wie die Folgen davon, ob es regnen wird oder nicht. Man kann sagen, dass bei jeder Verzweigung nur die zur Verfügung stehenden Alternativen relevant sind und angezeigt werden müssen. Da es jedoch nicht nur darum geht, so viel

Platz wie möglich zu sparen, sondern auch darum, eine übersichtliche und intuitiv verständliche Darstellung zu finden ist es hilfreich, außerdem noch den Knoten, von dem die Verzweigung ausgeht zu zeigen. Bei *A* also den Knoten „Regenschirm nötig“, bei *B* „Es regnet nicht“. Auf diese Art ist es leichter die Orientierung zu behalten, selbst wenn ein Großteil des Baumes nicht sichtbar ist.

Das Grundlegende Konzept, bestehend aus dem radialen Layout und dem Weglassen irrelevanter Knoten, kann nun unter Verwendung von D3 umgesetzt werden.

2.3 Baumdarstellung mit D3

Um mit D3 auf den Bildschirm zu zeichnen gibt es verschiedene Möglichkeiten. In der Regel werden die HTML-Elemente *canvas* oder *svg* als Zeichenfläche verwendet. Beide haben Vor- und Nachteile, die vor einer Entscheidung abzuwägen sind.

2.3.1 Vergleich zwischen Canvas und SVG

Der größte Unterschied zwischen *canvas* und *svg* Elementen besteht darin, dass ein *svg* jedes Objekt, dass es anzeigen soll einzeln als HTML-Element hinterlegt während ein *canvas* nur das insgesamt entstehende Bild speichert. Für den Browser bedeutet das, dass bei einem *canvas* nur die Darstellung eines einzelnen Bildes zu berechnen ist, während bei einem *svg* alle dargestellten Objekte getrennt behandelt werden. Das führt dazu, dass mit wachsender Zahl anzuzeigender Elemente die Berechnungsdauer des *svg* stärker ansteigt, als die des *canvas*. Das einzelne Speichern der Anzeigebausteine bringt aber auch Vorteile mit sich. Es ist einfach ein einzelnes Objekt auf dem Bildschirm zu verändern, indem man dessen *svg*-Attribute anpasst, weil der Browser sich dann selbst um die Aktualisierung der Anzeige kümmert. D3 bietet mit seinen *transitions* eine simple Möglichkeit beliebige Animationen auf einem *svg* Element durchzuführen, was auf einem *canvas* umständlicher mit D3 *interpolators* und einer Funktion zu lösen ist, die für jedes Einzelbild der Animation das Bild neu zeichnet. Allgemein scheint die Nutzung eines *svg*-Elements zur Darstellung von Daten mit D3 die am häufigsten gewählte Herangehensweise, was zu einer großen Menge Beispiele führt, die zur Inspiration genutzt werden können. Auch in diesem Fall fällt die Wahl darauf D3 in Verbindung mit *svg* einzusetzen. Diese Entscheidung wird vor allem durch die in Kapitel 2.2 beschriebene Reduzierung der angezeigten Knoten untermauert, denn der große Vorteil schnellerer Berechnung auf *canvas*-Elementen fällt damit kaum noch ins Gewicht.

2.3.2 Datenstruktur

Zur Speicherung der Baumdaten werden die Knoten in eine D3 *hierarchy* Umgewandelt. Es ist zu beachten, dass eine *hierarchy* nur durch ein einzelnes Objekt repräsentiert wird, welches die Wurzel (*root*) des Baumes darstellt. Um eine Liste aller Knoten zu erhalten kann *root.descendants* aufgerufen werden. Jeder einzelne Knoten ist dabei ein Javascript-Objekt, das die in Kapitel 1.5 beschriebenen Daten enthält. Wendet man die Funktion *d3.tree* (Kapitel 1.4.2) auf die *hierarchy* an, so kommen zusätzlich noch x- und y-Koordinaten dazu. Unter Einsatz dieser Daten kann der Baum in einem *svg*-Element dargestellt werden.

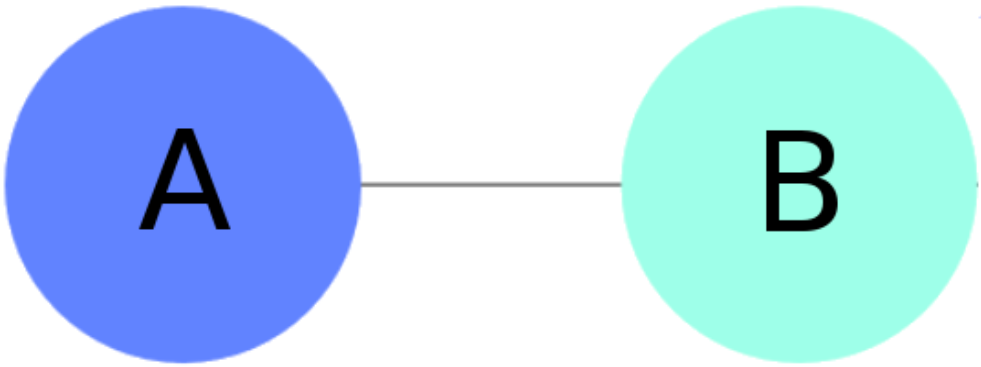

Abbildung 2.3: Konzept für Knoten- und Kantendarstellung

2.3.3 Erzeugung eines SVG

Zu Beginn wird dem HTML-Dokument mit *d3.select* und *selection.append* ein *svg*-Element angehängt. Die Rückgabe von *append* – eine *selection*, die nur das *svg* enthält – wird in einer Variable zwischengespeichert.

Abbildung 2.3 zeigt das gewünschte Aussehen für Knoten und Kanten. Zu sehen sind zwei Knoten als Kreise mit Beschriftung A und B. Die Kante zwischen den Beiden Knoten ist dargestellt als Linie, die beide Kreise miteinander verbindet. Die Unterschiedlichen Farben der Knoten zeigen an, dass sie verschiedene Knotentypen haben. Im medizinischen Kontext könnte es sich zum Beispiel um ein Symptom A und eine Diagnose B handeln. Es ergibt Sinn, für Knoten und Kanten getrennte Gruppen zu erstellen. Dazu werden unter der Verwendung der zuvor gespeicherten *selection* zwei *g*-Elemente hinzugefügt, die im Folgenden als Knoten- und Kantengruppe bezeichnet werden.

2.3.4 Präparieren der Knotendaten für die Darstellung

Bevor die Knoten auf den Bildschirm gebracht werden können, müssen zunächst deren Koordinaten bestimmt werden. Außerdem soll darauf geachtet werden, dass wie in Kapitel 2.2 erörtert nur der aktuelle Verzweigungsknoten und dessen Kinder angezeigt werden. Die zur Koordinatenberechnung verwendete Funktion *d3.tree* erhält als Übergabe einen Startknoten und läuft bei der Berechnung von dort den gesamten nachfolgenden Baum ab, wobei jedem Knoten x- und y-Koordinaten gegeben werden. Dazu verwendet sie immer die Liste *children*, die in jedem Knoten enthalten ist und dessen Kinder aufzählt. Erreicht die Funktion einen Knoten, der kein *children*-Objekt enthält, dann wird dieser als Blatt betrachtet. Da nur ein Knoten und dessen Nachfolger in die Berechnung einbezogen werden sollen, muss *d3.tree* die Nachfolger dieses Knotens als Blätter betrachten. Das wird erreicht, indem die *children*-Objekte der Kinder entfernt und in einem *childrenBackup* benannten Objekt zwischengespeichert werden. Hat *d3.tree* anschließend allen anzuzeigenden Knoten ihre Koordinaten zugewiesen, werden diese noch mit dem in Kapitel 2.1.2 beschriebenen Verfahren umgerechnet, um eine Kreisförmige Anordnung zu erzeugen.

Ein weiterer Voreilt dieses Verfahrens liegt darin, dass zum Erhalten einer Liste der anzuzeigenden Knoten *currentRoot.descendants* aufgerufen werden kann, weil auch diese Funktion die *children* Objekte zum ablaufen des Baumes verwendet. Das Objekt *currentRoot* bezeichnet dabei den aktuellen Verzweigungsknoten.

2.3.5 Erzeugen der Knotendarstellung

Wie auf Abbildung 2.3 zu sehen ist, soll die Knotenvisualisierung aus einem Kreis mit einer Aufschrift bestehen. Für jeden Knoten muss das *svg* also ein *circle* und ein *text*-Element enthalten. Um leichter beide zusammen ein- und ausblenden sowie verschieben zu können, sollen diese jeweils Gruppiert werden. Mit *selectAll* wählt man zuerst alle *g*-Elemente in der Knotengruppe aus und Verknüpft diese unter Einsatz von *selection.data* mit den gewünschten Knotendaten, welche nach dem gerade beschriebenen Präparieren der Daten mit *currentRoot.descendants* zu erhalten sind. Die daraus entstehende *enter selection* verwendet man dann um für alle neu hinzugekommenen Knoten *g*-Elemente einzufügen, die einen *circle* mit zentriertem *text* enthalten. Der Radius des Kreises ist abhängig von der Bildschirmgröße und der Text ist der Name des Knotens, welcher den Knotendaten entnommen wird. Mit dem *transform*-Attribut der *g*-Elemente werden die Gruppen an die zuvor berechneten Koordinaten geschoben.

Wegen der begrenzten Bildschirmgröße von mobilen Geräten soll die Knotenaufschrift zur besseren Lesbarkeit so groß wie möglich sein, ohne den Kreis zu überschreiten, in dem sie zentriert ist.

Berechnen der Schriftgröße

[todo: aufteilung auf zwei zeilen implementieren] [todo: bild von Knoten mit langem text] Angaben von Schriftgrößen beziehen sich auf die Höhe der Buchstaben [todo: quelle]. Das bedeutet nicht notwendigerweise, dass eine Schriftgröße von 10px zu Buchstaben führt, die exakt Zehn Pixel hoch sind, sondern dass die Höhe der Schriftzeichen proportional zur angegebenen Schriftgröße ist. Das bedeutet auch, dass die Breite eines Buchstabens dazu proportional ist, da er beim Vergrößern nicht verzerrt werden soll, wodurch das Verhältnis von Höhe zu Breite immer gleich bleibt. Diese Tatsache lässt sich auf ganze Texte übertragen...

2.3.6 Erzeugen der Kantendarstellung

Die Darstellung der Kanten verläuft analog zur Knotendarstellung in Kapitel 2.3.5, allerdings werden anstelle von *g*-Elementen mit *circle* und *text* jetzt *path*-Elemente verwendet. Zur Erzeugung der *enter selection* wird als Daten die Rückgabe der *hierarchy*-Funktion *node.links* verwendet, welche eine Liste aller Kanten, die von einem Knoten ausgehen, zurückgibt. Aus den Einträgen dieser Liste lassen sich mit *d3.line* die Zeichenanweisungen für das *d*-Attribut der *path*-Elemente berechnen.

2.4 Navigation im Baum

Durch das Ausblenden vieler der Knoten muss nun eine Möglichkeit geschaffen werden, von einer Entscheidung zur nächsten zu navigieren, wobei immer wieder irrelevante Knoten aus-

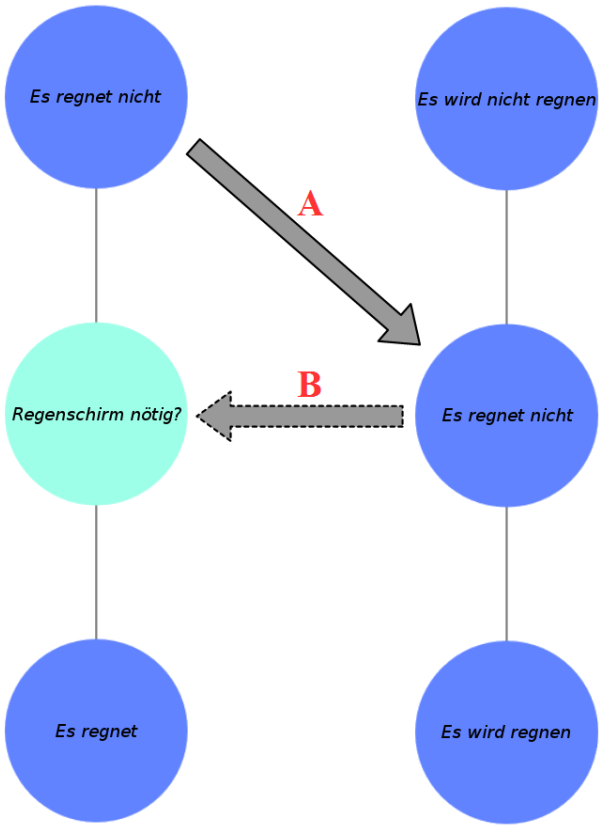

Abbildung 2.4: Beispiel zur Baumnavigation

und relevant gewordene eingeblendet werden. Das wird möglich gemacht, indem die Baumanzeige auf Berührung hin dynamisch aktualisiert wird. Es gilt dabei, dass der Knoten, von dem die aktuelle Verzweigung ausgeht, in der Mitte des Bildschirms zu sehen ist und die nachfolgenden Entscheidungsmöglichkeiten im Kreis darum herum angeordnet werden. Abbildung 2.4 zeigt die Reaktion des Baumes auf verschiedene Nutzereingaben. Auf der linken Seite ist der Ausgangszustand zu sehen und rechts der Nachfolgezustand. Links ist der mittlere Knoten mit „Regenschirm nötig?“ beschriftet und dessen Nachfolger mit „Es regnet nicht“ sowie „es regnet“. Rechts sieht man „Es regnet nicht“ in der Mitte, „Es wird nicht regnen“ und „Es wird regnen“ als Folgeknoten. Die mit *A* und *B* beschrifteten Pfeile symbolisieren Reaktionen auf Eingaben, die im Folgenden genauer beschrieben werden.

Durch Antippen eines der äußeren Knoten wird dieser in die Mitte verschoben und es zeigen sich dessen Nachfolger. Tippt man zum Beispiel in Abbildung 2.4 links „Es regnet nicht“ an, dann wird dieser, wie Pfeil *A* zeigt, zum mittleren Knoten, die anderen verschwinden und es erscheinen die neuen Nachfolgeknoten. Möchte man einen Schritt zurück machen, kann der Mittelknoten berührt werden und man wird wieder zur links gezeigten Situation geleitet (illustriert durch Pfeil *B*). Es soll dadurch das Gefühl entstehen, dass man sich durch die Baumstruktur bewegt, wobei besonders die gewählten Animationen (Kapitel 2.6) von großer Wichtigkeit sind. Zuerst ist aber zu klären, wie die interne Datenstruktur verwendet werden kann, um dynamische Aktualisierungen zu ermöglichen.

2.4.1 Dynamische Aktualisierung

Mit dem Präparieren der Daten in Kapitel 2.3.4 ist bereits der Grundstein für das Aktualisieren der Anzeige auf Nutzereingaben hin gelegt worden. Wird einer der Entscheidungsknoten berührt, so soll dieser zum neuen mittleren Knoten werden und dessen Nachfolger darum herum erscheinen (Abbildung 2.4). Dazu wird als erstes das *children*-Objekt des ausgewählten Knotens, welches beim Präparieren der vorherigen Daten in das Objekt *childrenBackup* verschoben wurde, wiederhergestellt. Anschließend werden mit dem angetippten Knoten als Verzweigungsknoten die Schritte vom Präparieren der Daten bis zur Anzeige erneut durchgeführt (Kapitel 2.3.4 bis 2.3.6). Der Unterschied besteht diesmal aber darin, dass nicht nur die *enter selections* beim Verknüpfen der Daten betrachtet werden, sondern auch die *update*− und *exit selections*. Die *update selection* enthält dann den neuen Mittelknoten, weil er bereits vorher auf dem Bildschirm war. Dieser muss nur an seine neue Position in der Mitte der Anzeige verschoben werden, indem das *transform*-Attribut seines *g*-Elements angepasst wird. In der *exit selection* befinden sich der vorherige Verzweigungsknoten und dessen Kinder, die nicht ausgewählt wurden. Sie werden mit der Funktion *selection.remove* vom Bildschirm entfernt. [todo: verweise auf abbildung zur veranschaulichung]

2.5 Eingabemenü

Für die medizinische Nutzung der Visualisierung soll die Anwendung mit einem Server kommunizieren, der Ergebnisse von Untersuchungen und getroffene Behandlungsentscheidungen speichert. Es soll aber nicht immer, wenn ein Knoten berührt wird, sofort eine Änderung in der Datenbank des Servers vorgenommen werden, damit die Möglichkeit erhalten bleibt, ohne Konsequenzen den Baum zu erkunden. Das ist zum Beispiel bei der Einarbeitung neuer Mitarbeiter nützlich oder wenn anhand der Visualisierung einem Patienten ein Behandlungsablauf erklärt werden soll. Deshalb wird das in Abbildung 2.5 zu sehende Eingabemenü eingeführt. [todo: besser lesbares Bild einfügen sobald zweizeilige anzeige implementiert ist] Zu sehen ist ein Knoten mit der Aufschrift „Has calprotectin in stool“ und dessen Nachfolger „Inflammatory diarrhea“, „negative stool bacteriology“ und „positive stool bacteriology“, um welchen das Eingabemenü angezeigt wird. Es besteht aus drei Ringsegmenten in den Farben Grün, Rot und Gelb. Das grüne enthält ein Häkchen, das rote ein Kreuz und das gelbe ein Fragezeichen. Sie dienen dazu, Knoten als positiv, negativ oder neutral zu markieren, was im Hintergrund an den Datenbankserver weitergeleitet wird. Der Knoten, um den auf der Abbildung das Eingabemenü geöffnet wurde, betrifft eine bakteriologische Stuhlprobenuntersuchung. Nach Erhalt des Ergebnisses, kann der Knoten durch Antippen des grünen oder roten Segments als positiv beziehungsweise negativ markiert werden. Die Markierung wird anschließend durch eine Farbige Umrandung deutlich gemacht, welche die gleiche Farbe hat wie die berührte Schaltfläche. In Abbildung 2.5 zeigt die gründe Umrandung des mittleren Knotens beispielsweise an, dass er als positiv markiert wurde. Die gelbe Schaltfläche mit dem Fragezeichensymbol kann im Falle einer Fehlerhaften Eingabe genutzt werden, um die Markierung eines Knotens zusammen mit dem Eintrag in der Datenbank wieder rückgängig zu machen.

Das Eingabemenü wird geöffnet, indem man einen der Knoten mit dem Finger gedrückt hält.

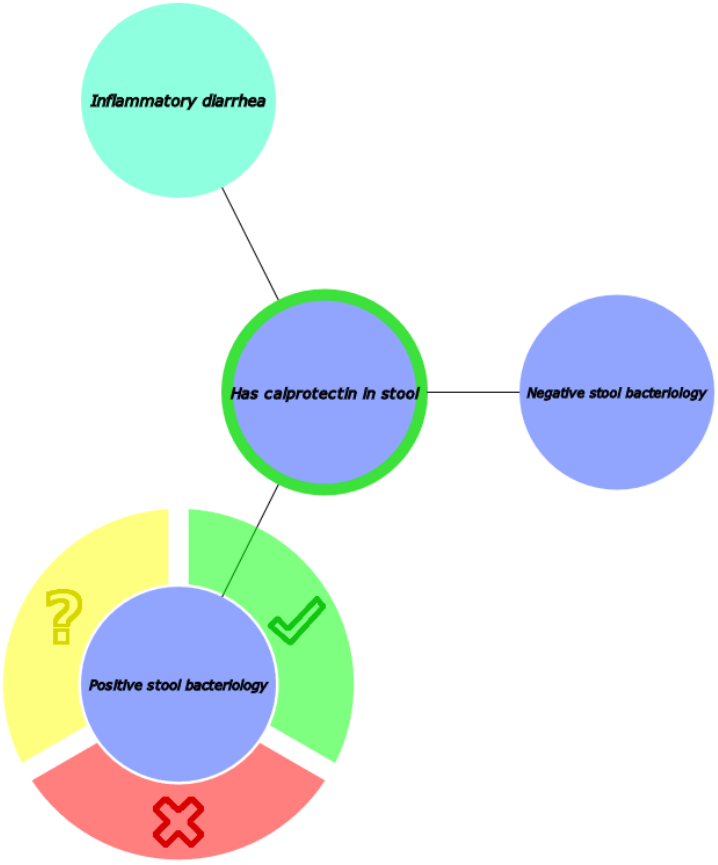

Abbildung 2.5: Eingabemenü zum Eintragen von Behandlungsergebnissen

2.5.1 Erzeugung des Eingabemenüs

Das Eingabemenü gleicht in seiner Form einem Ringdiagramm mit drei gleichgroßen Segmenten. Die in Kapitel 1.4.2 beschriebenen Funktionen *d3.arc* und *d3.pie* dienen zur Erzeugung eines solchen Diagramms. Als Daten erhält *d3.pie* eine Liste mit drei Objekten, die jeweils das Attribut *value* mit Wert 1 und *cssClass* mit einem der Werte „positive“, „negative“ und „unknown“ enthalten. *D3.pie* verwendet dabei *value* als Datenwert, welcher für alle drei Objekte gleich ist, wodurch drei gleichgroße Ringsegmente entstehen. Mit der Funktion *pie.padAngle* wird noch angegeben, wie viel Abstand diese voneinander haben sollen. Die von *d3.pie* generierte Rückgabe kann anschließend an *d3.arc* übergeben werden, welches daraus Zeichenangaben für *path*-Elemente macht.

Zuerst wird dem *svg* eine neue Gruppe in Form eines *g*-Elements hinzugefügt. Dann wird ähnlich wie beim erzeugen der Knoten- und Kantendarstellung (Kapitel 2.3.5 und 2.3.6) vorgegangen. Mit *selectAll* wird eine *selection* aller *path*-Elemente in der neu erzeugten Gruppe erstellt und diese dann mit den von *d3.pie* generierten Daten verknüpft. Über die *enter selection* fügt man dann neue Pfade ein und gibt diesen als *d*-Attribut die mit *d3.arc* aus den Daten entstehenden Zeichenbefehle. Zusätzlich erhält jedes der drei *path*-Elemente als CSS-Klasse den Wert der Variable *cssClass* , welche im verknüpften Datensatz hinterlegt ist. [todo: muss html und css im background erklärt werden?] Über diese wird die Füllfarbe des jeweiligen Pfades festgelegt. Des Weiteren fügt man der Gruppe die in Abbildung 2.5 zu sehenden Icons als

image-Elemente ein, deren Positionierung mit *arc.centroid* berechnet werden.

2.6 Einsatz von Animationen

Normalerweise wird zum Verändern der Anzeige eine *selection* mit allen zu verändernden Objekten erstellt und diese dann mit der Funktion *selection.attr* angepasst. Dabei verändern sich die Ausgewählten Elemente sofort. Fließende Übergänge werden unter Einsatz von *transitions* (Kapitel 1.4.2) realisiert. Anstatt ein Objekt per *selection.attr* zu bearbeiten erzeugt man mit *selection.transition* eine Animation und wendet *transition.attr* an, wobei verschiedene Parameter, wie Dauer und Übergangsfunktion zur Konfigurierung verfügbar sind.

In Googles „Material design guidelines“ heißt es „Motion provides meaning“ – zu deutsch „Bewegung verleiht Bedeutung“. [todo: quelle matrial.io introduction] Diesem Prinzip folgend kommen bei der hier beschriebenen interaktiven Visualisierung Animationen zum Einsatz und spielen dabei für Interaktivität und verbesserte Orientierung eine zentrale Rolle.

2.6.1 Interaktivität

Für eine interaktive Anwendung ist es wichtig, dass Nutzer immer das Gefühl haben, dass sie das Geschehen auf dem Bildschirm kontrollieren. Wenn sich Elemente auf Eingaben hin unvorhersehbar oder nicht nachvollziehbar verhalten, ist das nicht gewährleistet. Durch kontinuierliche Bewegungen, wie in Abbildung 2.6, anstelle von plötzlichen Sprüngen kann immer nachvollzogen werden, wie die Visualisierung sich verändert, womit eine interaktivere Umgebung geschaffen wird.

2.6.2 Verbesserung von Orientierung und Übersichtlichkeit

Kapitel 2.2 beschreibt, wie zum Einsparen von Platz Knoten weggelassen werden können. Ein daraus folgender Nachteil besteht im Orientierungsverlust. Obwohl die Menge an Informationen auf dem Bildschirm sinkt, was die Übersichtlichkeit verbessert, wird es schwerer zu wissen, wo man sich im Baum gerade befindet. Durch passende Animationen beim in Kapitel 2.4.1 beschriebenen dynamischen Aktualisieren des Baumes kann dem Abhilfe geschaffen werden. Es soll dabei durch die Bewegungen auf dem Bildschirm deutlich werden, welche Veränderung eine Eingabe bewirkt. Die gewählte Animation beim Navigieren von einem Knoten zum nächsten ist in Abbildung 2.6 zu sehen. Von links nach rechts mit 1 bis 4 nummeriert werden die vier wichtigen Zeitpunkte im Bewegungsablauf gezeigt. Situation 1 zeigt den Ausgangszustand: Ein Knoten A mit dessen Nachfolgerknoten B und C. Die anderen drei Stadien zeigen was geschieht, wenn man B antippt. Zunächst werden bei 2 die Knoten A und C durchsichtiger, bis sie nicht mehr zu sehen sind. Anschließend bewegt sich B in die Mitte der Anzeige (zu sehen bei 3) und zuletzt bewegen sich zu Zeitpunkt 4 von B aus die Knoten D und E nach außen.

Das Ausblenden von A und C verdeutlicht, dass beide nicht mehr relevant sind, geschieht aber zur Vermeidung von Verwirrung nicht von einem Moment auf den anderen. Die Bewegung von B in die Mitte lässt leicht nachverfolgen, dass B zum neuen Verzweigungspunkt wird. Wichtig ist dabei, dass B zu keinem Zeitpunkt vom Bildschirm verschwindet oder umherspringt, wodurch klar ist, dass es sich noch immer um denselben Knoten handelt. Zuletzt verdeutlicht die von B ausgehende Bewegung von D und E nach außen, dass diese die Nachfolger von B

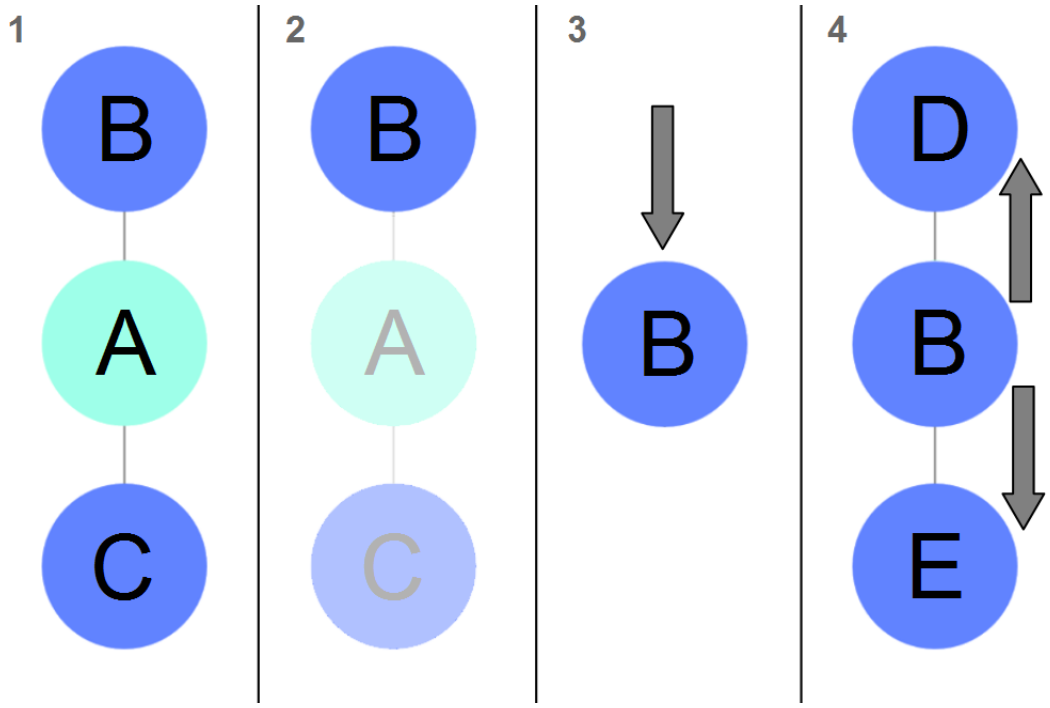

Abbildung 2.6: Animation beim Berühren eines Knotens

sind. Der allgemeine Schwerpunkt liegt darauf, es durch kontinuierliche Bewegungen leichter zu machen, dem Geschehen zu folgen, was zu einem intuitiven Verständnis der Vorgänge auf dem Bildschirm führt.

2.7 Detailansicht

In der bisherigen Darstellung, welche im weiteren Verlauf als Navigation oder Navigationsansicht bezeichnet wird, werden Knoten nur mit ihrem Namen beschriftet. Dabei wird vernachlässigt, dass die Übergebenen Daten, welche in Kapitel 1.5 beschrieben werden, auch eine Ausführliche Beschreibung des Knotens beinhalten. Um diese anzuzeigen wird dem Dokument eine Detailansicht hinzugefügt. Abbildung 2.7 zeigt das Konzept dazu. Es handelt sich um Knoten aus einem Entscheidungsbaum zur Behandlung von Verdauungsstörungen – im Englischen „dyspepsia “. Die Knoten werden als Rechtecke dargestellt, die als Überschrift den Namen und darunter den ausführlichen Beschreibungstext des Knotens enthalten. Im diesem Beispiel wurde von der Wurzel „Dyspepsia“ aus der Knoten „Age ¿ 55 years“ und von dort „New alarm signal“ ausgewählt. Die Hintergrundfarben der Rechtecke gleichen den Farben der korrespondierenden Kreise in der Navigationsansicht. Wie hier zu sehen ist bietet die Detailansicht neben der Knotenbeschreibung auch eine Anzeige des zurückgelegten Pfades durch den Entscheidungsbaum. Damit dient sie gleichzeitig dem Verständnis und der Orientierung.

2.7.1 HTML-Struktur

Das HTML-Dokument ist so aufgebaut, dass es vertikal in zwei Hälften aufgeteilt ist. Auf der linken Seite soll die Detailansicht und auf der rechten die Navigationsansicht zu sehen sein.

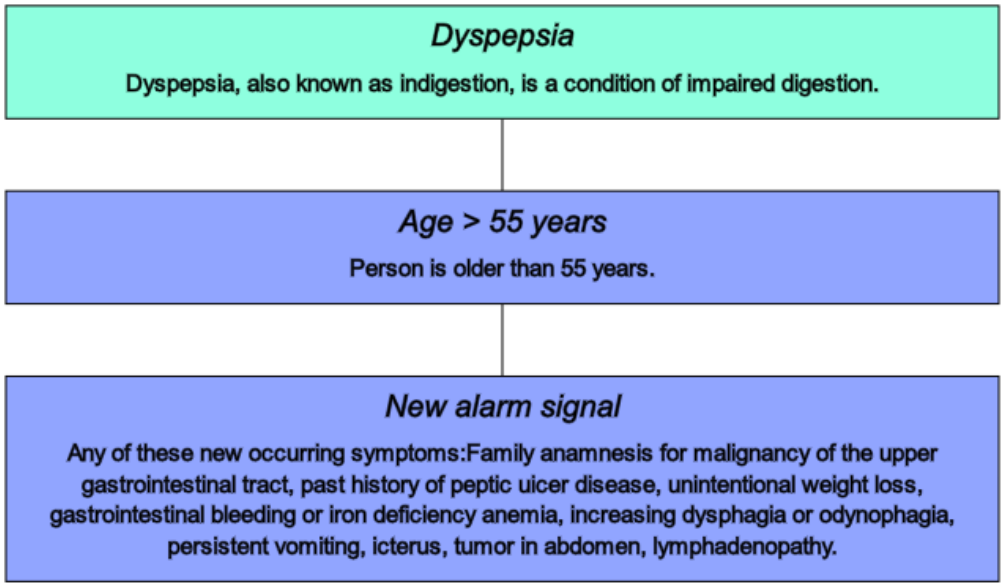

Abbildung 2.7: Detailansicht dreier Knoten

Reicht die Bildschirmgröße nicht aus, um beide Ansichten nebeneinander zu positionieren, wird die Detailansicht ausgeblendet und die Navigation nimmt den gesamten Bildschirm ein. Es kann dann per Knopfdruck zwischen beiden Ansichten gewechselt werden.

2.8 Erzeugen der Detailansicht

Für die Detailansicht wird auf der linken Seite ein eigenes *svg*-Element hinzugefügt. Das verläuft analog zur in Kapitel 2.3.3 beschriebenen Erzeugung des *svg* für die Navigation. Auch hier werden für Knoten und Kanten eigene Gruppen erstellt.

Für die Knotendarstellung wird die Funktion *hierarchy.path* benutzt, die in Form einer Liste aller auf dem Weg passierten Knoten den Pfad von einem zum anderen zurückgibt. Da es in einem Baum nur genau einen möglichen Pfad zwischen zwei Knoten gibt, ist ein so berechneter Pfad von der Wurzel zum aktuellen Verzweigungsknoten immer auch der Pfad, welcher in der Navigationsansicht bisher gewählt wurde. Eine *selection* aller *g*-Elemente in der Knotengruppe wird per *selection.data* mit der von *hierarchy.path* erzeugten Liste verknüpft. Anhand der entstehenden *selection* werden nach dem in D3 üblichen Verfahren bereits existierende Elemente mit der *update selection* aktualisiert, neue mit der *enter selection* erzeugt und überflüssig gewordene Elemente unter Einsatz der *exit selection* entfernt.

Die Erzeugung neuer Knoten beginnt mit dem Hinzufügen eines *text*-Elements, in welches sowohl die Überschrift als auch der Beschreibungstext eingefügt werden. Dabei wird mit dem gleichen Vorgehen, wie in Kapitel 2.3.5 die Schriftgröße der Überschrift so berechnet, dass sie den ihr zur Verfügung stehenden Platz nicht überschreitet. Da der Beschreibungstext eine beliebige Länge haben kann, macht es keinen Sinn, ihn so weit zu verkleinern, dass er auf eine Zeile passt. Stattdessen bekommt er eine feste Schriftgröße und es werden Zeilenumbrüche eingefügt.

2.8.1 Einfügen von Zeilenumbrüchen

Einem Beispiel von Mike Bostock folgend [todo: quelle textwrapping] werden Zeilenumbrüche mit Hilfe von *tspan*-Elementen innerhalb des *text*-Elements eingefügt. Dabei wird zunächst ein erstes *tspan* unterhalb der Überschrift platziert. Der Beschreibungstext wird umgewandelt in eine Liste aller einzelnen Wörter, und ein Wort nach dem anderen wird dem *tspan* hinzugefügt. Nach jedem Wort wird überprüft, ob das Element die erlaubte Breite überschreitet. Ist das der Fall, so wird das zuletzt zugefügte Wort wieder entfernt und in ein neues *tspan* eingefügt, welches unterhalb des Vorherigen platziert wird. Dieser Vorgang wird solange wiederholt, bis der vollständige Text untergebracht ist.

asd

Literaturverzeichnis

[1] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999.

[2] Edward M. Reingold and John S. Tilford. Tidier drawings of trees. *IEEE Transactions on software Engineering*, (2):223–228, 1981.