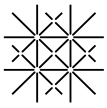


Universität
Basel

Komprimierte Pfaddatenbanken durch Lauflängenkodierung von optimal permutierten first-move Matrizen

Bachelorarbeit

Philosophisch-Naturwissenschaftliche Fakultät der Universität Basel
Departement Mathematik und Informatik
Künstliche Intelligenz
http://ai.cs.unibas.ch

Beurteiler: Prof. Dr. Malte Helmert
Zweitbeurteiler: Florian Pommerening

Patrick Zumsteg
patrick.zumsteg@stud.unibas.ch
2013-060-793

27. Juni 2016

Danksagung

Ich möchte mich herzlich bei Florian Pommerening bedanken, der mir im Verlaufe dieser Arbeit stets mit wertvoller Kritik und Betreuung zur Seite stand. Ausserdem gilt mein Dank Prof. Dr. Malte Helmert, für die Gelegenheit, auf dem Gebiet der Künstlichen Intelligenz meine Bachelorarbeit zu schreiben.
Mein Dank geht ferner an die Herren Ben Strasser vom Karlsruhe Institute of Technology, Adi Botea von der IBM Research in Dublin, und Daniel Harabor vom National ICT Australia, die freundlicherweise gewillt waren, mir ihre Implementation der Pfaddatenbankkompression mit Lauflängenkodierung zu Test- und Vergleichszwecken zur Verfügung zu stellen.

Zusammenfassung

Das Finden eines kürzesten Pfades zwischen zwei Punkten ist ein fundamentales Problem in der Graphentheorie. In der Praxis ist es oft wichtig, den Ressourcenverbrauch für das Ermitteln eines solchen Pfades minimal zu halten, was mithilfe einer komprimierten Pfaddatenbank erreicht werden kann. Im Rahmen dieser Arbeit bestimmen wir drei Verfahren, mit denen eine Pfaddatenbank möglichst platzsparend aufgestellt werden kann, und evaluieren die Effektivität dieser Verfahren anhand von Probleminstanzen verschiedener Grösse und Komplexität.

Inhaltsverzeichnis

1 Einleitung

Unter einer komprimierten Pfaddatenbank (engl. *Compressed Path Database*, im Folgenden **CPD**) versteht man eine Datenstruktur, welche alle optimalen Pfade zwischen allen möglichen Start- und Zielknoten in einem Graphen enthält. Sie erlaubt es also, den optimalen Pfad zwischen zwei Knoten in kürzester Zeit nachzuschlagen[1]. Eine CPD hat gegenüber herkömmlicher Pfadplanung zwei grosse Vorteile: Erstens muss ein Agent nicht warten, bis der gesamte Pfad berechnet ist, sondern kann seine Route sofort ermitteln und begehen. Vor allem bei einer Vielzahl an Agenten, die gleichzeitig eine Route planen wollen, kann hier ein Engpass in der verfügbaren Rechenleistung vermieden werden. Zweitens benötigt das Nachschlagen des gesamten Pfades einen Bruchteil der Zeit, die man für eine Berechnung des Pfades aufwenden müsste.
Dies bedeutet natürlich, dass für einen Graphen im Voraus für jeden Knoten die optimalen Pfade zu jedem anderen Knoten berechnet und gespeichert werden müssen. Die Anzahl dieser Pfade wächst mit dem Quadrat der Anzahl der Knoten im Graphen, der Speicherverbrauch einer "naiven" CPD, welche alle optimalen Pfade speichert, wird also für viele Anwendungsfälle schnell zu gross.
Es ist daher sinnvoll, als Kandidaten für die CPD eine sogenannte *first-move Matrix* zu betrachten. Diese enthält für jedes mögliche Knotenpaar nicht den optimalen Pfad selbst, sondern nur den *ersten Zug* besagten Pfades. Dies hat den Vorteil, dass eine grosse Platzersparnis erzielt wird, indem nur noch einzelne first moves und nicht ganze Pfade gespeichert werden müssen. Möchte man aus einer first-move Matrix den optimalen Pfad zwischen zwei Knoten auslesen, erfragt man zuerst, welcher Knoten vom Startknoten aus als nächstes besucht wird, dann den ersten Zug von diesem neuen Knoten aus, bis man schliesslich zum Zielknoten gelangt. Die in dieser Reihenfolge ausgelesenen *first-moves* repräsentieren einen optimalen Pfad zwischen zwei gegebenen Knoten.
In der Praxis stimmen benachbarte Einträge in einer first-move Matrix oft überein. Um den Platzverbrauch der CPD weiter zu reduzieren, wird die Matrix daher Zeile für Zeile mit *Lauflängenkodierung* (engl. *run length encoding*, im Folgenden RLE) komprimiert[2]. Der Kompressionsgrad hängt bei dieser Methode davon ab, wie unterschiedlich die einzelnen Zeileneinträge sind: lange Ketten identischer Zeichen lassen sich besser komprimieren als Sequenzen mit vielen verschiedenen, ungeordneten Zeichen.

Aus diesem Grund lässt sich eine first-move Matrix möglicherweise stärker komprimieren, wenn die Spalten zuerst permutiert werden. Das Ziel ist dabei, die Spalten der Matrix so zu sortieren, dass in benachbarten Spalten die Einträge in der selben Zeile möglichst oft übereinstimmen. Als Mass für die Ähnlichkeit zweier Spalten können verschiedene Metriken verwendet werden, wie beispielsweise die Hamming-Distanz. Im Rahmen dieser Arbeit werden drei unterschiedliche Distanzmasse und ihre Auswirkung auf die Qualität der Kompression analysiert.
Die Suche nach einer optimalen Spaltenordnung lässt sich als ein *Travelling Salesman Problem* (**TSP**) formulieren. Dabei entspricht jede Spalte einem Knoten, während die Distanzmasse zwischen zwei Spalten die Gewichte der Kanten zwischen den Knoten repräsentieren. In dieser Arbeit wird ein TSP-Solver verwendet, um die Spaltenordnung zu berechnen.

2
Hintergrund

2.1 Graphentheorie

Wir bezeichnen als Graphen ein Tupel $\langle V, E\rangle$. Hierbei ist V eine endliche Menge der Knoten. Die Kantenmenge $E = \{e_1, e_2, ..., e_n\} \subseteq V \times V$ ist eine Menge von Tupeln $e = \langle start(e), ziel(e)\rangle$, wobei $start(e), ziel(e) \in V$ und $start(e) \neq ziel(e)$. Einen solchen Graphen bezeichnen wir als *gerichteten Graphen*. Dies bedeutet, dass eine Kante e vom Knoten $start(e)$ zum Knoten $ziel(e)$ führt. Bei einem *ungerichteten Graphen* führt zudem jede Kante $e \in E$ vom Knoten $ziel(e)$ zum Knoten $start(e)$. Mit jeder Kante werden bestimmte Kosten durch eine Kostenfunktion $cost : E \to \mathbb{R}_{\geq 0}$, assoziiert.

Wir betrachten als Beispiel den gerichteten Graphen $G_{bsp} = \langle V, E\rangle$ mit $V = \{a, b, c, d, e\}$ und $E = \{\langle a, d\rangle, \langle d, a\rangle, \langle b, c\rangle, \langle c, b\rangle, \langle b, e\rangle, \langle d, c\rangle, \langle c, e\rangle, \langle e, d\rangle\}$. Die Zahl neben einer Kante bezeichnet deren Kosten.

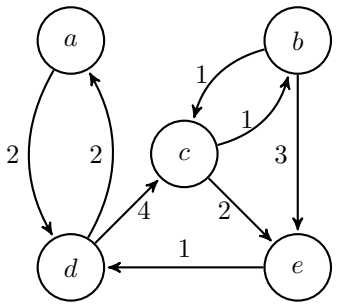


Abbildung 2.1: Der Graph G_{bsp}

Wir konzentrieren uns in dieser Arbeit auf gerichtete Graphen, die im Folgenden beschriebenen Prozeduren sind aber alle auch auf ungerichtete Graphen anwendbar: Für jeden ungerichteten Graphen $G = \langle V, E\rangle$ existiert ein gerichteter Graph $G' = \langle V, E'\rangle$ mit $\langle start(e), ziel(e)\rangle, \langle ziel(e), start(e)\rangle \in E'$ für jede Kante $e \in E$.

Ein st-Pfad π vom Knoten s zum Knoten t mit $s, t \in V$ ist ein Tupel von Kanten $\pi = \langle e_1, e_2, ..., e_N\rangle$ mit $e_i \in E$ für $i \in [1, N]$, für das gilt, dass $start(e_{j+1}) = ziel(e_j)$ für $j \in [1, N-1]$, $start(e_1) = s$, und $ziel(e_N) = t$.

Die *Kosten* eines Pfades π sind die Summe der Kosten aller Kanten in π. Ein Pfad π von s nach t ist genau dann optimal bzw. ein kürzester Pfad, wenn kein anderer Pfad π' von s nach t existiert, dessen Kosten geringer sind als die Kosten von π.

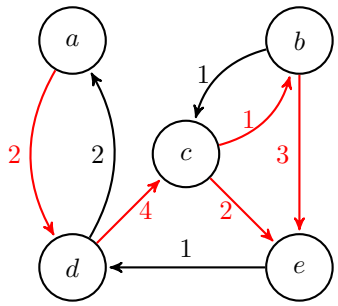


Abbildung 2.2: Kürzeste Pfade in G_{bsp}

Am Beispiel: Vom Startknoten a zum Zielknoten e gibt es mehrere st-Pfade. Pfade mit Zyklen, d.h. Pfade, in denen ein Knoten mehrmals besucht wird, sind hier vernachlässigt. Übrig bleiben die Pfade $\pi_1 = \langle\langle a,d\rangle,\langle d,c\rangle,\langle c,e\rangle\rangle$ mit Kosten $2+4+2 = 8$ und $\pi_2 = \langle\langle a,d\rangle,\langle d,c\rangle,\langle c,b\rangle,\langle b,e\rangle\rangle$ mit Kosten $2+4+1+3 = 10$. Für dieses Beispiel existiert also genau ein optimaler st-Pfad: π_1.

2.2 First moves

Gegeben sei ein Graph $G = \langle V, E\rangle$. Ein *first move* für ein bestimmtes Knotenpaar $s,t \in V$ ist jeweils der erste Element eines optimalen st-Pfades. Für ein gegebenes Knotenpaar können also mehrere verschiedene first moves existieren, wenn es auch mehrere verschiedene st-Pfade gibt.

Für diesen Graphen G existiert eine quadratische *first-move Matrix* F mit den Dimensionen $|V| \times |V|$. Jedes Element $F_{i,j}$ mit $i,j \in [1,|V|]$ und $i \neq j$ einer first-move Matrix enthält die Menge aller *first moves* vom Knoten s zum Knoten t.
Auf allen diagonalen Feldern $F_{i,i}$ der first-move Matrix befinden sich die first-moves der optimalen st-Pfade mit $s = t$. Diese Einträge der first-move Matrix sind definiert als die Menge aller Kanten, für die gilt, dass $start(e) = s$. Dies ist erlaubt, da die Einträge $F_{i,i}$ später nie ausgelesen werden (Es macht keinen Sinn, für einen tt-Pfad den first move nachzuschlagen). Ausserdem ist diese Definition für die später beschriebene Kompression günstiger.

Wir verwenden weiterhin als Beispiel den Graphen G_{bsp}. In diesem Beispiel steht der Zeilenindex für den Startknoten, der Spaltenindex für den Zielknoten der st-Pfade, aus denen die first moves stammen.

	a	b	c	d	e
a	$\langle a,d\rangle$	$\langle a,d\rangle$	$\langle a,d\rangle$	$\langle a,d\rangle$	$\langle a,d\rangle$
b	$\langle b,c\rangle$	$\langle b,c\rangle,\langle b,e\rangle$	$\langle b,c\rangle$	$\langle b,c\rangle,\langle b,e\rangle$	$\langle b,c\rangle,\langle b,e\rangle$
c	$\langle c,e\rangle$	$\langle c,b\rangle$	$\langle c,b\rangle,\langle c,e\rangle$	$\langle c,e\rangle$	$\langle c,e\rangle$
d	$\langle d,a\rangle$	$\langle d,c\rangle$	$\langle d,c\rangle$	$\langle d,a\rangle,\langle d,c\rangle$	$\langle d,c\rangle$
e	$\langle e,d\rangle$	$\langle e,d\rangle$	$\langle e,d\rangle$	$\langle e,d\rangle$	$\langle e,d\rangle$

Abbildung 2.3: Die aus G_{bsp} resultierende first-move Matrix

2.2.1 Platzverbrauch

Die first-move Matrix für einen Graphen $G = \langle V, E\rangle$ hat einen quadratischen Platzverbrauch, mit $|V|^2$ Einträgen. Diese Eigenschaft ist jedoch in der Praxis oft unerwünscht. Der Speicherverbrauch einer first-move Matrix, obwohl dieser im schlimmsten Fall immer quadratisch wächst, kann jedoch reduziert werden, indem die first-move Matrix komprimiert

wird[1]; das Ziel dabei ist es, den Speicherverbrauch zu verringern, ohne dass durch die Dekompression beim Auslesen der Matrix ein zu grosser Overhead entsteht. In dieser Arbeit wird als Kompressionsalgorithmus die Lauflängenkodierung verwendet.

2.3 Lauflängenkodierung

Das Ziel der Lauflängenkodierung ist es, eine Folge von Zeichen verlustfrei zu komprimieren, indem Ketten (oder "Läufe") von identischen Zeichen durch die Länge der Kette und das Zeichen, aus dem sie bestand, zu ersetzen. Wir betrachten zuerst einen Algorithmus, der auf Sequenzen von Zeichen arbeitet, und wenden diese Idee danach auf Sequenzen von Mengen an.

Ein auf Zeichenfolgen arbeitender RLE-Algorithmus akzeptiert im Folgenden eine beliebige, endliche Sequenz $S = \langle s_1, s_2, ..., s_n \rangle$ von Zeichen aus einer nicht-leeren, endlichen Zeichenmenge Σ und übergibt als Ausgabe eine Sequenz O von Tupeln $l_j = \langle x_j, c_j \rangle$ mit $j \in [1, |O|]$. Dabei ist $x_j \in \Sigma$ und $c_j \in \mathbb{N}_{>0}$. Für jede Folge in S von einem oder mehreren konsekutiven Zeichen, soll die Ausgabe O ein Tupel mit diesem Zeichen und der Länge der Folge enthalten. Würde man also jedes Tupel $\langle x, c \rangle$ der Ausgabe durch eine c-malige Aneinanderreihung von x ersetzen, erhielte man wieder die Eingabesequenz.

Zur Berechnung der Kompression wird die Eingabe S von Anfang bis Ende elementweise traversiert, wobei zu Beginn und jedes Mal, wenn ein Zeichen $s_i \in S$, $i \in [1, n]$ nicht gleich dem vorhergehenden Zeichen s_{i-1} ist, ein neuer *Lauf*, also ein neues Tupel $l_j = \langle s_i, 1 \rangle$ begonnen wird. Jedes Mal, wenn auf s_i ein Zeichen $s_{i+1} = s_i$ folgt, wird die Länge c_j dieses Laufes um eins inkrementiert. Stösst der Algorithmus auf das Ende der Zeichenkette oder ein Zeichen $s_{i+1} \neq s_i$, wird an die Ausgabe O des Algorithmus das Tupel l_j angehängt. Zum Beispiel enthält die Zeichenkette "*aaaabbbc*" drei Läufe: "*aaaa*" mit Länge 4, "*bbb*" mit Länge 3 und "*c*" mit Länge 1. Nach der Kompression lautet die resultierende Zeichenkette demnach "$\langle \langle a, 4 \rangle, \langle b, 3 \rangle, \langle c, 1 \rangle \rangle$".

2.4 Anwendung auf Zeilen der first-move Matrix

Wendet man Lauflängenkodierung auf eine Zeile in einer first-move Matrix an, zeigt sich schnell, dass mit dem vorhin beschriebenen Algorithmus für Sequenzen von Mengen nur eine unbefriedigende Kompression zu erreichen ist, weil viele Elemente von S zwar gewisse Kanten teilen, aber nicht identisch sind. Es ist jedoch für first-moves gar nicht nötig, aus der komprimierten Sequenz die ursprüngliche wieder zu rekonstruieren; vielmehr reicht es, wenn aus jeder Menge $s_i \in S$ ein einziges Element wieder rekonstruiert werden kann. Da alle first moves auf einem kürzesten Pfad liegen, spielt es keine Rolle, welcher am Ende effektiv ausgelesen wird.

Wir verwenden als Beispiel die aus G_{bsp} resultierende first-move Matrix F: Sei das Alphabet $\Sigma = E$. Die Zeichenkette S, die dem RLE-Algorithmus übergeben wird, ist die Sequenz der

Elemente, die in jedem Feld einer Zeile enthalten sind. Für die vierte Zeile lautet die Eingabe also $S = \langle\{e_2\}, \{e_6\}, \{e_6\}, \{e_2, e_6\}, \{e_6\}\rangle$, wobei $e_2, e_6 \in E$. Komprimiert man diese Zeile mit RLE für Mengen, erhält man die Ausgabe $O = \langle\langle\{e_2\}, 1\rangle, \langle\{e_6\}, 2\rangle, \langle\{e_2, e_6\}, 1\rangle, \langle\{e_6\}, 1\rangle\rangle$. Viel günstiger wäre es hier, wenn man die Elemente s_2 bis s_5 aus S zu einem einzigen Lauf zusammenfassen könnte.
Um dieses Problem zu lösen, wandeln wir die Lauflängenkodierung für Mengen folgendermassen ab: Wir versuchen nicht mehr, ausschliesslich identische Mengen in S zu einem Lauf zusammenzufassen. Vielmehr wählen wir aus jeder Menge $s_i \in S$ genau ein Element, und komprimieren die resultierende Sequenz von Kanten wie gewohnt. Naheliegenderweise wählt man das Element, auf welches jede Menge reduziert wird, nach dem greedy-Prinzip[2]: Die erste Kante wird beliebig gewählt. Für s_i, wähle wenn möglich die Kante, welche schon für s_{i-1} ausgewählt wurde. Ansonsten, wähle eine beliebige Kante. Diese Reduktion der Mengen auf eine Kante führt in jedem Fall zu einer mindestens so guten Kompression, da für zwei völlig disjunkte Mengen sowieso ein neuer Lauf begonnen werden muss, während es die neue Methode für zwei aufeinanderfolgende Mengen $s_i \neq s_{i+1}$, $s_i \cap s_{i+1} \neq \emptyset$ ermöglicht, einen bestehenden Lauf fortzuführen.

Am Beispiel: Wir betrachten wieder die vierte Zeile von F, also ist $S = \langle\{e_2\}, \{e_6\}, \{e_6\}, \{e_2, e_6\}, \{e_6\}\rangle$. Das vierte Element von S enthält zwei Einträge $e_2, e_6 \in E$. Da e_6 schon im dritten Element vorkommt, wird aus dem vierten Element die Kante e_6 ausgewählt, aus der Eingabe wird $\langle e_2, e_6, e_6, e_6, e_6\rangle$. Die Ausgabe des neuen Algorithmus lautet dann $O = \langle\langle e_2, 1\rangle, \langle e_6, 4\rangle\rangle$.

2.5 Relevanz der Spaltenordnung

Bei einer first-move Matrix ist es erlaubt, ihre einzelnen Spalten beliebig zu vertauschen, bevor die Matrix komprimiert wird[2]. Das bedeutet natürlich, dass die Permutation p der Spalten gespeichert werden muss, und dass bei jedem Zugriff auf die Matrix der Index, an welchem diese ausgelesen wird, der Index mit p permutiert werden muss. Dies ist unbedenklich, da der Platzverbrauch von p linear mit der Anzahl der Knoten wächst. Der zusätzliche Zeitaufwand ist konstant und kann vernachlässigt werden.
Da jede Zeile bzw. Spalte mit einem bestimmten Start- bzw. Zielknoten assoziiert ist, ist eine Vertauschung der Spalten nichts anderes als eine Umsortierung der Knoten im entsprechenden Graphen. Diese Eigenschaft lässt sich ausnutzen, um die Kompression der gesamten Matrix zu verbessern: Die Anzahl der Läufe, die in einer Sequenz von Mengen enthalten ist, bestimmt die Qualität der Kompression. Je weniger Läufe, desto weniger Elemente enthält die Ausgabe der Kompression. Wir sagen, dass die *Kosten* einer Kompression gleich der Kardinalität der Ausgabe O sind. Sortiert man nun die Knoten im Graphen in einer bestimmten Reihenfolge neu, müssen alle Zeilen der first-move Matrix in derselben Reihenfolge umsortiert werden. Dabei ist es möglich, dass die neue Knotenreihenfolge zu einer anderen Summe der Kompressionskosten der Zeilen führt.

Am Beispiel: ohne Umsortieren der Knoten in G_{bsp} resultiert die Kompression der dritten Zeile in der Ausgabe $O_3 = \langle\langle e_5, 1\rangle, \langle e_4, 2\rangle, \langle e_5, 2\rangle\rangle$ mit Kosten 3, während die vierte Zeile

zu $O_4 = \langle\langle e_2, 1\rangle, \langle e_6, 4\rangle\rangle$ mit Kosten 2 komprimiert wird. Vertauscht man nun den Knoten a mit dem Knoten c, erhält man zwar für die dritte Zeile die bessere Kompression $O_3 = \langle\langle e_4, 2\rangle, \langle e_5, 3\rangle\rangle$ mit Kosten 2, verschlechtert aber im Gegenzug die Kompression der vierten Zeile zu $O_4 = \langle\langle e_6, 2\rangle, \langle e_2, 2\rangle, \langle e_6, 1\rangle\rangle$ mit Kosten 3.

Das Ziel ist also, eine Knotenreihenfolge zu finden, welche die Summe der Kompressionskosten der Zeilen einer first-move Matrix minimiert. Damit man bestimmen kann, welche Reihenfolge über *alle* Zeilen die beste ist, benötigt man eine Metrik $dist(s, t)$ dafür, wie lohnend es ist, zwei Knoten s, t nebeneinander zu platzieren - oder anders gesagt, wie lohnend es ist, zwei Spalten der Matrix nebeneinander zu platzieren. Da das Ziel ist, die Summe der Kompressionskosten (also die Anzahl an Läufen) zu minimieren, sollte das Distanzmass widerspiegeln, wie viele Läufe das Nebeneinandersetzen zweier Spalten unterbricht. Da jede Spalte an eine beliebige Position gesetzt werden darf, existiert für jeden Startknoten s zu jedem Zielknoten $t \neq s$ eine solche Distanz.

Dieses Problem lässt sich als ungerichteter Graph $G_{dist} = \langle V_{dist}, E_{dist}\rangle$ formulieren: Die Knotenmenge V_{dist} des neuen Graphen entspricht genau der Knotenmenge V des ursprünglichen Graphen. Da für jedes Knotenpaar $s, t \in V_{dist}$, $s \neq t$ ein Distanzmass für das Nebeneinandersetzen der entsprechenden Spalten existiert, gibt es für jedes Knotenpaar s, t eine Kante $\langle s, t\rangle \in E_{dist}$ mit $cost(\langle s, t\rangle) = dist(s, t)$. Das bedeutet, dass jeder tt-Pfad π auf dem Graphen G_{dist}, welcher jeden Knoten genau einmal besucht (d.h. π enthält keine zwei Kanten e_1, e_2 mit $start(e_1) = start(e_2)$) und wieder beim Startknoten ankommt, einer Knotenreihenfolge bzw. Spaltenordnung für die first-move Matrix entspricht. Die Suche nach einer optimalen Spaltenordnung entspricht also genau der Suche nach einem *kürzesten* solchen Pfad π auf G_{dist}. Dieses Problem ist bekannt als das *Travelling Salesman Problem*. Es gilt nun also, ein gutes Distanzmass $dist(s, t)$ zu finden, um den Graphen G_{dist} aufzustellen.

3
Distanzmasse

Ein optimales Distanzmass $dist_{opt}(s,t)$ ist eine Metrik, unter deren Verwendung die Lösung des TSPs, also die aus $G_{dist_{opt}}$ erhaltene Spaltenordnung, zu einer Kompression C von F führt, sodass kein anderes Distanzmass $dist'(s,t)$ existiert, unter dessen Verwendung eine Kompression C' mit $|C'| < |C|$ möglich ist.
Um ein solches Distanzmass zu finden, wäre es nötig, im Voraus zu wissen, welche first-Moves bei der Lauflängenkodierung ausgewählt werden, damit diese in der resultierenden Spaltenordnung möglichst nebeneinander zu liegen kommen. Es ist daher nicht möglich, ein solches optimales Distanzmass zu bestimmen, ohne die komprimierte first-move Matrix zu kennen.

Wir betrachten deshalb im Folgenden drei suboptimale Distanzmasse und diskutieren deren Vorzüge und Mängel.

3.1 Hamming-Distanz

Als einfaches Distanzmass zwischen den beiden Knoten s und t liegt es nahe, die *Hamming-Distanz* der entsprechenden Spalten in F zu berechnen. Die Hamming-Distanz $dist_h(\sigma_1, \sigma_2)$ ist ein Mass für die Unterschiedlichkeit zweier Zeichenketten $\sigma_1 = \langle x_1, x_2, ..., x_n\rangle, \sigma_2 = \langle y_1, y_2, ..., y_n\rangle$ mit $x_k, y_k \in \Sigma$ für $k \in [1, n]$. Sie ist definiert als die Kardinalität der Menge aller Elemente $x_k \neq y_k$. Wir betrachten als Beispiel die Zeichenketten $\sigma_1 = \langle a, b, c, d, e, f, g\rangle$ und $\sigma_2 = \langle a, c, c, d, e, e, e\rangle$; hier ist $dist_h(\sigma_1, \sigma_2) = 3$.

Wir wenden nun dieses Distanzmass auf die Spalten einer first-move Matrix F an. Die beiden Zeichenketten σ_i und σ_j für jeweils die i-te und j-te Spalte von F sind Sequenzen von Kantenmengen, d.h. $\sigma_i = \langle x_1, x_2, ..., x_n\rangle$, $\sigma_j = \langle y_1, y_2, ..., y_n\rangle$ mit $x_k, y_k \subseteq E$, $k \in [1, n]$.

Am Beispiel der first-move Matrix F des Graphen G_{bsp}: Wir berechnen die Hamming-Distanz zwischen der zweiten und dritten Spalte. Hier ist

$\sigma_2 = \langle\{\langle a, d\rangle\}, \{\langle b, c\rangle, \langle b, e\rangle\}, \{\langle c, b\rangle\}, \{\langle d, c\rangle\}, \{\langle e, d\rangle\}\rangle$ und
$\sigma_3 = \langle\{\langle a, d\rangle\}, \{\langle b, c\rangle\}, \{\langle c, b\rangle, \langle c, e\rangle\}, \{\langle d, c\rangle\}, \{\langle e, d\rangle\}\rangle$.

Die Hamming-Distanz $dist_h(\sigma_2, \sigma_3)$ beträgt 2.

Es ist möglich, dass die Hamming-Distanz als Metrik die effektiven Kosten des Aneinanderreihens zweier Spalten überschätzt. Dies rührt daher, dass mit diesem Distanzmass zwei Spaltenelemente als ungleich angesehen werden, d.h. dass an dieser Stelle das Ende eines Laufes angenommen wird, auch wenn eine Kante in beiden Spalteneinträgen x_k, y_k vorkommt, ein Lauf also nicht zwangsläufig unterbrochen werden muss.
Die Hamming-Distanz kann die echten Kosten nicht unterschätzen: Dafür müsste sie für ein Paar x_k, y_k fälschlicherweise annehmen, dass an dieser Stelle ein Lauf unterbrochen wird. Falls $x_k \neq y_k$, gibt es mindestens eine Kante e in x_k bzw. y_k, welche in der anderen Menge nicht enthalten ist. Existiert an dieser Stelle bereits ein Lauf der Kante e, wird er also unterbrochen. Es ist deshalb immer dann möglich, dass ein bestehender Lauf abgebrochen wird, wenn die Kantenmengen nicht identisch sind. Da die Hamming-Distanz genau diese Identität prüft, wird sie niemals fälschlicherweise annehmen, dass ein Lauf unterbrochen werden muss, kann sich also auch nicht unterschätzen.

Für den Fall, dass σ_1 und σ_2 Tupel von einelementigen Mengen sind, entspricht die Hammingdistanz genau den effektiven Kosten: Wenn $x_k \neq y_k$, muss an dieser Stelle immer ein neuer Lauf begonnen werden, und wenn $x_k = y_k$ kann der bestehende Lauf immer fortgesetzt werden.

Unter Verwendung der Hamming-Distanz als Distanzmass entspricht die Lösung des TSPs also nicht immer der Spaltenordnung, unter welcher die Summe der Kompressionskosten minimal wird.

3.2 Elementweise Disjunktheit

Der grosse Nachteil der Hamming-Distanz als Metrik war die Tatsache, dass für zwei Spaltenelemente x_k, y_k fälschlicherweise der Beginn eines neuen Laufes angenommen wird. Da es möglich ist, dass die beiden Kantenmengen eines oder mehrere gleiche Elemente enthalten, betrachten wir als Metrik die Disjunktheits-Distanz $dist_d(\sigma_1, \sigma_2)$, welche dieser Besonderheit Rechnung trägt, indem sie nur dann einen Abbruch des Laufes annimmt, wenn kein Element aus x_k in y_k enthalten ist.
Dieses Distanzmass arbeitet mit derselben Eingabe wie die Hamming-Distanz, allerdings ist für die Disjunktheits-Distanz der Abstand zwischen zwei Sequenzen σ_1, σ_2 gleich der Kardinalität der Menge aller Elemente x_k, y_k, für die gilt, dass $x_k \cap y_k = \emptyset$.

Wir betrachten wieder dasselbe Beispiel und berechnen die Disjunktheits-Distanz zwischen der zweiten und dritten Spalte von F, mit

$\sigma_2 = \langle\{\langle a, d\rangle\}, \{\langle b, c\rangle, \langle b, e\rangle\}, \{\langle c, b\rangle\}, \{\langle d, c\rangle\}, \{\langle e, d\rangle\}\rangle$ und
$\sigma_3 = \langle\{\langle a, d\rangle\}, \{\langle b, c\rangle\}, \{\langle c, b\rangle, \langle c, e\rangle\}, \{\langle d, c\rangle\}, \{\langle e, d\rangle\}\rangle$.

Hier gibt es keine Elemente, deren Schnittmenge gleich der leeren Menge sind, die Disjunktheits-

Distanz $dist_d(\sigma_2, \sigma_3)$ beträgt also 0.

Im Gegensatz zur Hamming-Distanz wird die Disjunktheits-Distanz die wahren Kosten des Nebeneinandersetzens zweier Spalten stets unterschätzen. Der Grund dafür ist, dass allein die Tatsache, dass zwei Elemente x_k, y_k beide eine bestimmte Kante enthalten, nicht garantiert, dass diese Kante auch für die Kompression ausgewählt wird: Es ist möglich, dass in der k-ten Zeile der first-move Matrix ein Lauf von einer Kante $e \in x_k$ existiert, aber $e \notin y_k$. Der Lauf müsste dann also unterbrochen werden, auch wenn die Mengen x_k, y_k nicht disjunkt sind.
Genau wie die Hamming-Distanz entspricht die Disjunktheits-Distanz den effektiven Kosten, wenn σ_1 und σ_2 Tupel von einelementigen Mengen sind, denn: Zwei einelementige Mengen x_k, y_k sind genau dann disjunkt, wenn $x_k \neq y_k$, es gilt derselbe Beweis wie für die Hamming-Distanz.

3.3 Gewichtete Disjunktheit

Als letztes Distanzmass betrachten wir eine Variante der Disjunktheits-Distanz, welche deren Mängel zu einem gewissen Grad behebt. Anstatt anzunehmen, dass nur dann ein neuer Lauf begonnen wird, wenn die Elemente x_k, y_k völlig disjunkt sind, verwendet die Metrik $dist_{gd}(\sigma_1, \sigma_2)$ die elementweise Ähnlichkeit der Spalteneinträge. Die gewichtete Disjunktheits-Distanz beträgt $dist_{gd}(\sigma_1, \sigma_2) = \sum_{k=1}^{n} |(x_k \setminus y_k) \cup (y_k \setminus x_k)|$, also die Summe der Kardinalitäten der symmetrischen Differenzen zwischen x_k, y_k. Je mehr Kanten also in den Elementen der einen Spalte, aber nicht im korrespondierenden Element der anderen Spalte zu finden sind, desto grösser wird die gewichtete Disjunktheits-Distanz.

Am laufenden Beispiel: Die Eingabe lautet wieder
$\sigma_2 = \langle\{\langle a, d\rangle\}, \{\langle b, c\rangle, \langle b, e\rangle\}, \{\langle c, b\rangle\}, \{\langle d, c\rangle\}, \{\langle e, d\rangle\}\rangle$ und
$\sigma_3 = \langle\{\langle a, d\rangle\}, \{\langle b, c\rangle\}, \{\langle c, b\rangle, \langle c, e\rangle\}, \{\langle d, c\rangle\}, \{\langle e, d\rangle\}\rangle$,
die gewichtete Disjunktheits-Distanz $dist_{gd}(\sigma_2, \sigma_3)$ beträgt 2.

Die gewichtete Disjunktheits-Distanz wird die wahren Kosten zwar möglicherweise über-, aber nicht unterschätzen. Der Abstand kann beispielsweise stark überschätzt werden, wenn $x_k \subset y_k$ mit $|y_k| >> |x_k|$, obwohl an dieser Stelle ein Lauf einer Kante $e \in x_k$ existieren könnte.
Um die Distanz zu unterschätzen, müsste für zwei Mengen $x_k \in \sigma_1, y_k \in \sigma_2$ die gewichtete Disjunktheits-Distanz zwischen den Sequenzen σ_1, σ_2 nicht erhöht werden, obwohl dort zwingend ein Lauf abgebrochen werden muss. Dies ist nicht möglich, da ein Lauf nur dann sicher abgebrochen wird, wenn x_k und y_k disjunkt sind, und in diesem Falle wäre auch die symmetrische Differenz zwischen x_k und y_k eine nichtleere Menge, $dist_{gd}$ würde also erhöht.

Dies bedeutet, dass bei der gewichteten Disjunktheits-Distanz nicht mehr darüber entschieden wird, ob für zwei Mengen x_k, y_k mit Sicherheit ein neuer Lauf begonnen wird

(Disjunktheits-Distanz), oder ob für x_k, y_k garantiert *kein* neuer Lauf angefangen werden muss (Hamming-Distanz). Vielmehr repräsentiert die gewichtete Disjunktheits-Distanz die Wahrscheinlichkeit, dass an einer Stelle ein Lauf unterbrochen wird: Je mehr Kanten nur in einer Menge enthalten sind, desto eher erwartet man intuitiv, einen begonnenen Lauf abbrechen zu müssen.

Mit demselben Argument wie bei der Disjunktheits-Distanz kann man auch für die gewichtete Disjunktheits-Distanz sagen, dass sie den effektiven Kosten entspricht, wenn σ_1 und σ_2 Tupel von einelementigen Mengen sind.

4
Resultate

Wir analysieren die im letzten Kapitel beschriebenen Distanzmasse auf grid map-basierten Graphen. Jedes begehbare Feld der grid map ist repräsentiert durch einen Knoten, während für alle Felder, die auf der Karte benachbart sind, deren korrespondierende Knoten durch Kanten in beide Richtungen verbunden sind. Alle Kanten haben hier dieselben Kosten.

Alle Berechnungen wurden auf einer Maschine mit einem i5-4670K Prozessor mit 3.4GHZ und 6MB kombinierten Cache, 8GB Arbeitsspeicher und Ubuntu 16.04 durchgeführt. Die Programme sind mit g++ 5.3.1 kompiliert, alle Berechnungen liefen unter Verwendung eines einzelnen Prozessorkerns. Zum Lösen des TSPs wurde der TSP-Solver Concorde[1] benutzt. Die verwendeten Karten stammen aus dem Videospiel *Dragon Age: Origins*. Alle Karten sind erhältlich in Dr. Nathan Sturtevants Sammlung von Benchmark-Problemen[2].

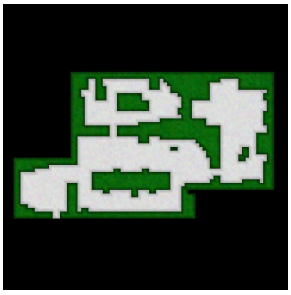
Der Algorithmus zur Berechnung der kürzesten Pfade und Lauflängenkodierung der einzelnen first-move Zeilen baut auf dem von Ben Strasser, Adi Botea und Daniel Harabor entwickelten Programm auf, das in deren Paper *Compressing Optimal Paths with Run Length Encoding*[2] verwendet wird.

[1] http://www.math.uwaterloo.ca/tsp/concorde/index.html
[2] http://www.movingai.com/benchmarks

4.1 Ergebnisse

Wir konzentrieren uns bei der Auswertung der Distanzmasse primär auf die Qualität der Kompression, d.h. die Grösse der RLE-komprimierten first-move Matrix. Als Massstab verwenden wir ausserdem die CPD, welche mit dem Algorithmus von Strasser et. al. berechnet wird. Dieser Algorithmus erreicht zwar keine optimale Kompression, ist aber äusserst performant[2]. Ferner vergleichen wir, wie effizient die Distanzmasse zu berechnen sind und wie gross der Speicherverbrauch und die Rechenzeit wird.

4.1.1 Ergebnis für den101d

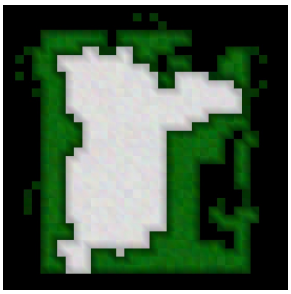


Wir betrachten zuerst eine Karte mit moderatem Detailreichtum. Der aus dieser Karte resultierende Graph enthält 1360 Knoten, die first move Matrix hat also $1360^2 = 1'849'600$ Elemente. Um dies in Kontext mit der Grösse der CPD zu bringen: Selbst unter der Annahme, dass die first move Matrix nur aus einelementigen Einträgen besteht, und dass jeder Eintrag mit 4 bits kodiert werden kann, wäre die resultierende, unkomprimierte first move Matrix immer noch 3.5278 Megabytes gross. Zum Vergleich: Strasser et. al.s Algorithmus erreicht für diese Karte immerhin eine Kompression auf 137.4 kB.

		Hamming-Distanz	Disjunktheit	gew. Disjunktheit
Rechenzeit	avg	350s	41s	152s
DistMasse	min	6.58806s	7.54519s	22.4139s
	avg	6.95229s	7.64552s	23.4894s
	max	7.17239s	7.74901s	24.0899s
Speicher	avg	25'740 kB	19'645 kB	22'600 kB
CPD	avg	116.6 kB	111.9 kB	115.2 kB

Tabelle 4.1: Daten für die Karte den101d. "Rechenzeit" bezeichnet die gesamte Laufzeit des Programs. Der grösste Teil der Rechenzeit wird hier jeweils vom TSP-Solver beansprucht. "DistMasse" enthält die minimal, durchschnittlich und maximal verbrauchte Zeit zur Berechnung des jeweiligen Distanzmasses. "Speicher" ist die durchschnittlich beanspruchte Menge an Arbeitsspeicher, "CPD" bezeichnet die Grösse der komprimierten Pfaddatenbank.

4.1.2 Ergebnis für lak101d

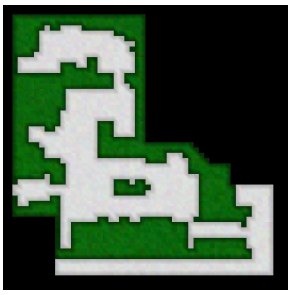


Als zweites Experiment verwenden wir eine deutlich detailärmere Karte. Der Graph enthält hier nur 318 Knoten. Analog zum letzten Beispiel wäre die unkomprimierte first-move Matrix 197.5 Kilobytes gross. Strasser et.al. erreichen für diese Karte eine CPD mit Grösse 30.6 Kilobytes.

		Hamming-Distanz	Disjunktheit	gew. Disjunktheit
Rechenzeit	avg	3s	1s	2s
DistMasse	min	0.070455s	0.066759s	0.249161s
	avg	0.071324s	0.066970s	0.250014s
	max	0.072058s	0.067297s	0.251333s
Speicher	avg	7'969 kB	4'910 kB	7'336 kB
CPD	avg	26.7 kB	27.6 kB	26.6 kB

Tabelle 4.2: Daten für die Karte lak101d.

4.1.3 Ergebnis für lak103d



Zuletzt betrachten wir eine ähnlich detailreiche Karte wie den101d, welche aber nur 861 begehbare Felder enthält. Die theoretische Grösse der unkomprimierten first-move Matrix beträgt in dieser Karte 1.413 Megabytes, Strasser et.al. komprimieren die CPD für diese Karte auf eine Grösse von 65.2 Kilobytes.

		Hamming-Distanz	Disjunktheit	gew. Disjunktheit
Rechenzeit	avg	7s	35s	30s
DistMasse	min	1.47375s	1.48498s	5.49935s
	avg	1.49710s	1.51262s	5.56296s
	max	1.52164s	1.52682s	5.60192s
Speicher	avg	13'457 kB	16'576 kB	19'809 kB
CPD	avg	58.4 kB	57.8 kB	58.3 kB

Tabelle 4.3: Daten für die Karte lak103d.

4.2 Evaluation der Distanzmasse

Als erstes fällt auf, dass für die beiden detailreicheren Karten den101d und lak101d ungeachtet des Distanzmasses eine massive Reduktion im Speicherverbrauch erreicht wurde: Für den101d ist eine Kompression auf rund 114 Kilobytes möglich, was gerade mal 3.2% der theoretischen $3'527$ Kilobytes entspricht. Bemerkenswert ist, dass für die detailärmere Karte lak101d die Kompression mit 13.7% deutlich weniger stark ausgefallen ist. Dies lässt sich dadurch erklären, dass Strukturen wie der lange "Korridor" in lak103d oder kleine, nur durch eine schmale "Tür" mit der restlichen Karte verbundene Kammern zu Zeilen in der first-move Matrix führen, die nur sehr wenige verschiedene Kanten enthalten: Sei ein beliebiger Punkt in einer solchem Struktur der Startknoten s. Die überwältigende Mehrheit der Zielknoten t liegt ausserhalb der Struktur, die meisten kürzesten st-Pfade führen also von s aus durch die "Tür" und haben mit sehr grosser Wahrscheinlichkeit denselben first move. Diese Eigenschaft, wenn auch bemerkenswert, ist nicht von der Wahl des Distanzmasses abhängig, denn alle drei Distanzmasse nähern sich dem optimalen Distanzmass $dist_{opt}(\sigma_1, \sigma_2)$, wenn die Kardinalitäten der Elemente von σ_1, σ_2 gegen 1 gehen.

Ferner scheint die Wahl des besten Distanzmasses vom Detailreichtum der Karte abhängig zu sein. Für die erste und letzte Karte ist die Disjunktheits-Distanz klar führend, während sie für die zweite, detailarme Karte merklich schlechter ist als die restlichen Distanzmasse. Dies macht Sinn, wenn man bedenkt, dass die Disjunktheits-Distanz als einzige die wahren Kosten stets unterschätzt. Auf detailreichen Karten sind die Elemente der first-move Zeilen tendenziell ähnlicher und enthalten eine geringere Kantenmenge als auf detailarmen Karten. Die bessere Performance für die Disjunktheits-Distanz kann also zumindest teilweise dadurch erklärt werden, dass auf detailreichen Karten weniger Raum für das Unterschätzen der Kosten existiert.

Zwischen der Kompression mithilfe der Hamming-Distanz und der gewichteten Disjunktheits-Distanz ist kaum ein Unterschied festzustellen, wobei die Hamming-Distanz marginal schlechter abschneidet. Die Hamming-Distanz ist zwar ca. 3- bis 4mal so schnell zu berechnen, ob diese Zeitersparnis jedoch die Einbusse in der Kompressionsqualität rechtfertigt, ist vom Anwendungsfall abhängig. Es lässt sich also nach den Experimenten nicht mit Sicherheit sagen, ob eines der beiden Distanzmasse dem anderen in irgend einer Hinsicht ausser der Rechenzeit klar überlegen ist.

4.2.1 Kritik

Die Rechenzeit des gesamten Programmes weist in allen Beispielen starke Schwankungen auf. Anhand der Experimente ist es nicht möglich, diese Schwankungen zu begründen. Es scheint so, als ob der Solver bestimmte TSP-Instanzen leichter löst als andere; dies zu ergründen, war im Rahmen dieser Arbeit nicht möglich.

Um die Distanzen zwischen den Spalten der first-move Matrix zu berechnen, muss logischerweise zuerst die gesamte Matrix aufgestellt werden. Das bedeutet, dass die $|V| \times |V|$ grosse

Matrix so lange im Arbeitsspeicher gehalten werden muss, bis alle Distanzen berechnet sind; dies ist für grosse Karten problematisch.

Der grösste Nachteil der in diesem Paper beschriebenen Kompressionsmethode ist so schwerwiegend wie offensichtlich. Während die Berechnung der Distanzmasse zwar eine quadratisch zur Knotenmenge wachsende Zeit braucht, geht der bei weitem grösste Teil der Rechenzeit am Lösen des TSPs verloren. Aus diesem Grund ist es völlig unrealistisch, Karten mit mehr als wenigen tausend begehbaren Feldern zu bearbeiten, was den Algorithmus für die Praxis eindeutig unbrauchbar macht. Es ist zwar immer eine merklich bessere Kompression als mit Strasser et. al.s Algorithmus möglich; die Karten, auf welchen der in diesem Paper beschriebene Algorithmus in absehbarer Zeit terminiert, sind jedoch so klein, dass die Platzersparnis nicht mehr ins Gewicht fällt.

5 Schlussfolgerung

In dieser Arbeit wurde erklärt, wie das Problem der Pfadplanung mithilfe einer komprimierten Pfaddatenbank gelöst werden kann. Wir haben gesehen, wie eine first-move Matrix aufgebaut und komprimiert wird, und haben drei Distanzmasse evaluiert, mithilfe derer wir das Travelling Salesman Problem gelöst haben, um eine optimale Permutation der first-move Matrix zu ermitteln.
Wir haben gesehen, dass die Disjunktheits-Distanz für detailarme Karten die ungünstigste Metrik ist, für detailreiche Karten jedoch die anderen Distanzmasse klar übertrifft.

5.1 Ausblick

Es bleibt zu ergründen, ob die Laufzeit des Algorithmus auf ein akzeptables Mass reduziert werden kann, indem heuristische TSP-Solver verwendet werden. Dies würde voraussichtlich eine Einbusse in der Kompressionsqualität zur Folge haben; die Frage ist dabei, ob es möglich ist, eine zufriedenstellende Balance zwischen Rechenzeit und Kompressionsqualität zu finden.
Da wir gezeigt haben, dass keines der drei Distanzmasse für jede Karte zur besten Kompression führt, bleibt die Frage, ob es ein Distanzmass gibt, welches die wahren Distanzen zwischen den Spalten besser repräsentiert.

Literaturverzeichnis

[1] Botea, A. et al. Ultra-Fast Optimal Pathfinding without Runtime Search. In *AIIDE* (2011).

[2] Strasser, B., Botea, A., and Harabor, D. Compressing optimal paths with run length encoding. *Journal of Artificial Intelligence Research*, 54:593–629 (2015).

Declaration on Scientific Integrity
Erklärung zur wissenschaftlichen Redlichkeit

includes Declaration on Plagiarism and Fraud
beinhaltet Erklärung zu Plagiat und Betrug

Author — Autor
Patrick Zumsteg

Matriculation number — Matrikelnummer
2013-060-793

Title of work — Titel der Arbeit
Komprimierte Pfaddatenbanken durch Lauflängenkodierung von optimal permutierten first-move Matrizen

Type of work — Typ der Arbeit
Bachelorarbeit

Declaration — Erklärung
I hereby declare that this submission is my own work and that I have fully acknowledged the assistance received in completing this work and that it contains no material that has not been formally acknowledged. I have mentioned all source materials used and have cited these in accordance with recognised scientific rules.

Hiermit erkläre ich, dass mir bei der Abfassung dieser Arbeit nur die darin angegebene Hilfe zuteil wurde und dass ich sie nur mit den in der Arbeit angegebenen Hilfsmitteln verfasst habe. Ich habe sämtliche verwendeten Quellen erwähnt und gemäss anerkannten wissenschaftlichen Regeln zitiert.

Basel, 27. Juni 2016

Signature — Unterschrift