

Humboldt-Universität zu Berlin



Institut für Informatik
Mathematisch-Naturwissenschaftliche Fakultät II

Studienarbeit

Kürzeste Pfade mit GRIPP

Verfasser:

Leonid Snurnikov

13. April 2008

Betreuer:

Prof. Dr. Ulf Leser

Snurnikov, Leonid:

Kürzeste Pfade mit GRIPP

Studienarbeit, Humboldt-Universität zu Berlin, 2008

Inhaltsverzeichnis

Inhaltsverzeichnis	i
Abbildungsverzeichnis	iii
Tabellenverzeichnis	v
Verzeichnis der Abkürzungen	vii
1 Einleitung	3
1.1 Hintergrund	3
1.2 GRIPP-Indexstruktur	4
1.2.1 Grundoperationen auf GRIPP	6
1.3 Distanzsuche und GRIPP	7
1.4 Zielstellung	7
2 Technischer Hintergrund	9
2.1 Arten von relevanten Graphen	9
2.2 Distanzsuche auf Graphen	9
2.3 Distanzsuche auf GRIPP	10
3 Analysen und Optimierungen	15
3.1 Testumgebung	15
3.2 Optimierung gegebener Routinen	15
3.2.1 Knotenbasierte Breitensuche auf GRIPP	15
3.2.2 Mengenbasierte Breitensuche auf GRIPP	17
3.3 Erste Tests	19
3.4 Die beste obere Schranke	20

3.5	Distanzenverteilung und Hopcount	21
4	Obere Schranken	25
4.1	Shortcut-Reachability	26
4.2	Obere Schranke und der k Index	27
4.3	Fiktive obere Schranke	30
4.4	Vorhersage der BFS-Laufzeit	32
5	Related Work	35
6	Zusammenfassung und Ausblick	39
	Literaturverzeichnis	43

Abbildungsverzeichnis

3.1	Laufzeiten verschiedener Breitensuchalgorithmen	21
3.2	Verteilung kürzester Distanzen	23
3.3	Potenzielle und tatsächliche Hops bei BFS-GRIPP	24
3.4	Laufzeiten in Abhängigkeit von gesuchter Distanz	24
4.1	Optimierungsspanne durch obere Schranken	25
4.2	Verteilung durchschnittlich kürzester Distanzen	27
4.3	Laufzeiten der BFS und obere SR-Schranken	34

Tabellenverzeichnis

3.1	Alte und neue Implementierung knotenbasierter Breitensuche auf GRIPP	17
3.2	Alte und neue Implementierung mengenbasierter Breitensuche auf GRIPP	19
3.3	Laufzeiten verschiedener Distanzalgorithmen	20
3.4	GRIPP-basierte Distanzalgorithmen mit optimaler oberer Schranke . . .	22
4.1	Vergleich oberer Schranken der normalen und Shortcut Reachability . . .	28
4.2	GRIPP-basierte Algorithmen mit der Shortcut Reachability Schranke . .	29
4.3	Obere Schranken mittels k Index	29
4.4	Qualität der statistischen oberen Schranke \hat{s}	32
4.5	GRIPP-basierte Algorithmen mit fiktiver Schranke \hat{s}	33
4.6	Wirkung fiktive Schranke \hat{s} bei isolierter Zeitmessung	33

Verzeichnis der Abkürzungen

BFS	node-based breadth-first search
DFS	depth-first search
GRIPP	graph indexing based on pre- and postorder numbering
RIS	reachable instance set
SB_BFS	set-based breadth first search
SR	shortcut reachability algorithms
SSSP	single-source shortest path
UB_NR	upper bound of normal reachability algorithm
UB_SR	upper bound of shortcut reachability algorithm

Zusammenfassung

In der vorliegenden Studienarbeit werden unterschiedliche Ansätze für eine effiziente Distanzsuche auf einem gerichteten Graphen untersucht. Der Schwerpunkt liegt auf der Ausnutzung einer von U. Leser und S. Trißl entwickelten Indexstruktur GRIPP [TL07, TL06]. Dieser Index ist speziell für eine sehr effiziente Beantwortung von Erreichbarkeitsanfragen konzipiert worden. Hier soll seine Tauglichkeit für die Beantwortung von Distanzfragen untersucht werden. Es werden diesbezüglich verschiedene Suchstrategien vorgestellt sowie deren Optimierungen sowohl durch die Einführung einer zusätzlichen Indexstruktur als auch die Ausnutzung statistischer Eigenschaften der vorliegenden Graphen untersucht.

Kapitel 1

Einleitung

In zahlreichen Anwendungen werden Netzwerke als Organisations- und Strukturierungsformen verwendet. Sie lassen sich mathematisch als Graphen modellieren. Mit zunehmender Digitalisierung und den sich eröffnenden technischen Möglichkeiten werden immer mehr und vor allem immer größere Graphen aufgebaut und verwaltet. Auch wenn die Speicherung sehr großer Datenstrukturen heutzutage wenig Schwierigkeiten darstellt, werden Algorithmen zur effizienten Beantwortung diverser Anfragen an diese Strukturen um so wichtiger. So werden zum Beispiel Systematiken, Ontologien, Taxomonien im Allgemeinen sowie metabolische Netzwerke und Genomdaten im Rahmen der Bioinformatik gepflegt und analysiert. Wohl am weitesten verbreitet ist die Verwendung relationaler Datenbank Management Systeme, die durch die Reifezeit der letzten Jahrzehnte viel Effizienz, Robustheit und Flexibilität entwickelt haben. Nichtsdestotrotz bereiten allgemeine Graphen aufgrund ihrer Struktur und der damit verbundenen inhärenten Rekursion immer noch Schwierigkeiten bezüglich effizienter Anfragenausführung. Im Fokus der vorliegenden Studienarbeit stehen allgemeine gerichtete Graphen, auf denen Erreichbarkeits- und Distanzanfragen möglichst effizient ausgeführt werden sollen. Gerichtete Graphen sind aufgrund der fehlenden Symmetrie (wenn es einen Pfad von A nach B gibt, dann nicht notwendigerweise einen in umgekehrter Richtung) im Hinblick auf die beiden oberen Problemstellungen schwieriger zu behandeln.

1.1 Hintergrund

Im weiteren Verlauf dieser Arbeit wird ausschließlich von gerichteten Graphen gesprochen, die in einer relationalen Datenbank verwaltet werden. Konkret handelt es sich um Oracle 10g, wobei auch viele andere relationale Datenbanksysteme, die Stored Procedures unterstützen, geeignet wäre. Bei den untersuchten Graphen handelt es sich um

sogenannte skalenfreie Graphen. Aufgrund ihres häufigen Vorkommens in der Natur und Praxis gewinnen sie ein besonderes Interesse. Ein skalenfreier Graph zeichnet sich dadurch aus, dass die Verteilung der Knotengrade dem Potenzgesetz folgt [CH03], wobei der Grad eines Knotens gleich der Summe der ein- und ausgehenden Kanten ist. Die Wahrscheinlichkeit für das Vorkommen eines bestimmten Knotengrades k wird durch die folgende Wahrscheinlichkeitsfunktionen beschrieben:

$$p(k) \sim k^{-\gamma} \quad (1.1)$$

Dabei ist γ eine einheitslose Zahl, die für viele natürliche skalenfreie Netzwerke einen Wert zwischen 2 und 3 annimmt.

Die Knoten der Graphen sind numerisch benannt und in einer Relation mit der Struktur *node(nodename)* gespeichert. Die gerichteten Kanten sind durch ihre Start- und Endknoten in der Relation *node(startnode, endnode)* dargestellt. Um Erreichbarkeits- oder Distanzanfragen auf einer solchen Struktur beantworten zu können, sind entweder rekursives SQL aus dem SQL99-Standard oder iterative Algorithmen in Form von Stored Procedures notwendig, die im Wesentlichen eine Breiten- und/oder Tiefensuche implementieren [THCR01]. Beide Ansätze haben ihre Nachteile. So unterstützen viele Datenbanksysteme rekursives SQL (noch) nicht. Weiterhin sind beide Ansätze nicht besonders effizient.

Die GRIPP-Indexstruktur, die die Basis für die Distanzberechnungen im Rahmen dieser Arbeit darstellt, ist eine Indexstruktur, die in linearer Zeit (bezüglich der Größe des Graphen) aufgebaut werden kann. Mit ihrer Unterstützung können in fast konstanter Zeit Erreichbarkeitsanfragen auch für sehr große Graphen beantwortet werden [TL07].

1.2 GRIPP-Indexstruktur

Die wohl wichtigste Idee hinter GRIPP ist die Überführung des gerichteten Graphen $G(V, E)$ in eine Baumstruktur ähnlich einem Spannbaum auf ungerichteten Graphen. Ausgehend von einem Anfangsknoten r_1 wird der Graph durch Tiefensuche traversiert. Die besuchten Knoten werden sukzessive einer Baumstruktur hinzugefügt. Der resultierende Baum enthält zwei Arten von Knoten - *Bauminstanzen* und *Nichtbauminstanzen*. Wenn ein Knoten des Graphen während der Tiefensuche zum ersten Mal besucht wird, wird für ihn im Baum eine Bauminstanz erzeugt. Wenn ein Knoten wiederholt besucht wird (das ist dann der Fall, wenn ein Knoten im Graphen über eine weitere eingehende Kante erreicht wird), dann wird für ihn im Baum eine Nichtbauminstanz erstellt, und die Tiefensuche macht ein Backtracking.

Wenn es im Graphen Knoten gibt, die nicht vom Ausgangsknoten r_1 erreicht werden können, muss für die verbleibende und noch nicht besuchte Knotenmenge wieder ein Ausgangsknoten r_2 ausgewählt und eine Tiefensuche vollzogen werden. Das Verfahren wird so lange wiederholt, bis alle Knoten des Graphen besucht und entsprechende Baum- bzw. Nichtbauminstanzen erzeugt wurden. Anschließend werden die einzelnen Teilbäume mit den Wurzelknoten r_1 bis r_m durch einen virtuellen Wurzelknoten r zu einem Baum zusammengefügt.

Durch diese Konstruktion entspricht die Anzahl der Knoten im GRIPP-Baum

$$M = |E| + m \tag{1.2}$$

Für das Ziel einer jeden Kante wird entweder eine Baum- oder eine Nichtbauminstanz erzeugt. Hinzu kommen die Bauminstanzen der Knoten r_1 bis r_m . Bei der Auswahl der Ausgangsknoten als auch bei der Reihenfolge der zu besuchenden Nachfolgerknoten werden die Knotengrade betrachtet und Knoten in absteigender Reihenfolge ihrer Grade verwendet, wobei auch andere Strategien denkbar wären.

Die Knoten des GRIPP-Baums werden während der Konstruktion mit *Pre-* und *Postnummern* sowie einer Tiefenangabe (*depth*) versehen. Der GRIPP-Baum weist konstruktionsbedingt folgende Eigenschaften auf:

- Nichtbauminstanzen treten nur als Blattknoten auf.
- Der Baum ist linksschief, da die Knoten in absteigender Reihenfolge ihrer Grade verarbeitet werden.
- Die Bauminstanzen zu entsprechenden Nichtbauminstanzen befinden sich immer weiter links im Baum bzw. besitzen kleinere Preorder-Nummern. Sie werden bei der Tiefensuche früher besucht und somit als Bauminstanzen zum GRIPP-Baum hinzugefügt.
- Zu jedem Knoten des Graphen existieren im GRIPP-Baum genau eine Bauminstanz und Null oder mehr Nichtbauminstanzen.

Physikalisch wird die GRIPP-Indexstruktur in einer Relation mit folgender Struktur abgelegt: *gripp(nodename, preorder, postorder, depth, nodeinst)*. Dabei zeigt *nodeinst* an, ob es sich um eine Baum- oder Nichtbauminstanz handelt.

Der GRIPP-Index hat eine lineare Zeit- und Platzkomplexität in Bezug auf die Größe des Graphen, die der Summe aus Knoten- und Kantenanzahl entspricht.

1.2.1 Grundoperationen auf GRIPP

Durch die Pre- und Postordernummerierung kann man zu einem vorgegebenen Knoten v mit $[v_{pre}, v_{post}]$ im Allgemeinen mit einer einzigen SQL-Anfrage eine große Menge der von ihm erreichbaren Knoten erhalten. Diese Menge wird als *RIS-Menge* (*Reachable Instance Set*) des Knotens v bezeichnet und umfasst formal alle Knoten der Menge

$$\{w \mid v_{pre} < w_{pre} < v_{post}\}. \quad (1.3)$$

Wenn man feststellen möchte, ob es einen Pfad von Knoten v zu einem anderen Knoten t gibt, reicht es allerdings nicht aus, t nur in $RIS(v)$ zu suchen. Es muss auch den durch Nichtbauminstanzen dargestellten Verweisen iterativ gefolgt werden.

Einen weiteren im Zusammenhang im GRIPP wichtiger Begriff ist der *Sprungknoten*. Von einem Sprungknoten spricht man bei der Analyse einer Menge $RIS(v)$. Dabei ist die Bauminstanz w ein Sprungknoten von v , wenn eine Nichtbauminstanz w^N von w in der untersuchten Menge $RIS(v)$ existiert.

Wie auch auf einem normalen Graphen können auf der GRIPP-Indexstruktur zwei grundlegende Suchalgorithmen implementiert werden - *Tiefensuche* und *Breitensuche*. In der Arbeit von Leser und Trißl wird Tiefensuche auf dem GRIPP-Baum angewendet [TL07], um auch für große Graphen sehr effizient Erreichbarkeitsanfragen beantworten zu können.

Einen interessanten Optimierungsaspekt stellen dabei sogenannte *Stopknoten* da. Ein Stopknoten s ist dadurch charakterisiert, dass für alle Nichtbauminstanzen $w \in RIS(s)$ die entsprechenden Bauminstanzen w' ebenfalls in derselben Menge liegen:

$$s \text{ ist ein Stopknoten} \iff \forall w \in RIS(s) : w' \in RIS(s) \quad (1.4)$$

Wenn man also bei einer Suche im Rahmen einer Erreichbarkeitsanfrage $RIS(s)$ untersucht hat, kann die Verfolgung der innerhalb von $RIS(s)$ liegenden Nichtbauminstanzen verworfen werden. Um diese Tatsache auszunutzen, werden bei der Indexerzeugung eine Menge solcher Stopknoten berechnet. Um den Aufwand für die Berechnung im Rahmen zu halten und diesen zum späteren Nutzen in ein gesundes Verhältnis zu setzen, wird eine entsprechende Überprüfung nur für solche potenzielle Stopknoten s vorgenommen, deren *RIS-Mengen* eine bestimmte Größe überschreiten. Diese kann leicht durch

$$|RIS(s)| = \lfloor (s_{post} - s_{pre})/2 \rfloor \quad (1.5)$$

berechnet werden.

1.3 Distanzsuche und GRIPP

Bei einer Distanzanfrage auf einem gerichteten Graphen wird nach der Länge des kürzesten Pfades von einem Startknoten $node_s$ zu einem Zielknoten $node_e$ gefragt. Dabei ergibt sich die Distanz aus der Anzahl der auf dem Pfad verwendeten Kanten. Distanzsuchen auf einem Graphen werden als Breitensuchen realisiert. Somit werden zuerst alle Knoten mit einer Entfernung von eins, dann alle mit einer Entfernung von zwei usw. untersucht. Eine entsprechende Suche kann auch unter Verwendung der GRIPP-Indexstruktur durchgeführt werden. Im Rahmen des Technical Reports [TL06] wurde allerdings festgestellt, dass die Distanzsuchen auf der GRIPP-Struktur nur für sehr kleine Graphen eine höhere Performance aufweisen als die Breitensuchen auf den Originalgraphen.

1.4 Zielstellung

Im Rahmen dieser Studienarbeit soll untersucht werden, inwieweit die Distanzsuchen auf der GRIPP-Indexstruktur verbessert oder sogar performanter als die Breitensuche auf dem Originalgraphen gemacht werden können. Eine der grundlegenden Überlegungen war es, möglichst schnell eine *gute* obere Schranke für die tatsächlich gesuchte Distanz zu bestimmen. Der Nutzen einer solchen oberen Schranke resultiert daraus, dass die Breitensuche auf der GRIPP-Indexstruktur anders als eine Breitensuche auf einem normalen Graphen nicht genau ebenenweise vorgeht. Sie ist eher als eine Mischung aus Breiten- und Tiefensuche anzusehen. Unter einer Ebene ist die Menge aller Knoten mit einem bestimmten Abstand zum Startknoten der Anfrage zu verstehen.

Kapitel 2

Technischer Hintergrund

2.1 Arten von relevanten Graphen

In dieser Arbeit werden gerichtete zusammenhängende skalenfreie Graphen $G(V, E)$ mit der Knotenmenge V und der Kantenmenge E untersucht. Diese Graphen variieren in ihrer Größe

$$|G| = |V| + |E| \quad (2.1)$$

als auch ihrer Dichte

$$D = |E|/|V|. \quad (2.2)$$

Es ist intuitiv ersichtlich, dass sich bei gleichbleibender Knotenanzahl und zunehmender Dichte die durchschnittlichen Knotengrade erhöhen und die durchschnittlichen kürzesten Distanzen verringern.

2.2 Distanzsuche auf Graphen

Der Standardalgorithmus für die Distanzberechnung auf einem (ungewichteten) Graphen ist die *Breitensuche*. Dabei werden zunächst alle Knoten mit der Distanz 1 zum Ausgangsknoten nach dem Vorkommen des Zielknotens untersucht, dann alle mit der Distanz 2 usw. Mit dem Hintergedanken, dass eine solche Breitensuche auf einer relationalen Datenbank implementiert werden soll, existieren zwei Variationen der Breitensuche - die knoten- und mengenbasierte Breitensuche. Bei der mengenbasierten Breitensuche werden zu der Menge M der Knoten mit einer bestimmten Distanz n vom Ausgangsknoten der Anfrage mit einer einzigen SQL-Anfrage die Menge M' der Knoten mit der Distanz $n + 1$ ermittelt. Dabei wird darauf geachtet, dass kein Knoten $k \in M'$ schon

früher mit einer Distanz $n' < n + 1$ betrachtet wurde. Bei der knotenbasierten Breitensuche wird hingegen jeder Knoten $k \in M$ einzeln untersucht und mit einer SQL-Anfrage seine noch nicht betrachteten Nachbarknoten gesucht. Bei dem mengenbasierten Ansatz werden zwar weniger SQL-Anweisungen ausgeführt, dafür sind die Zwischenergebnisse viel größer. Mit zunehmender Entfernung vom Startknoten der Anfrage steigt die Größe dieser Zwischenergebnisse exponentiell an. Knoten- und mengenbasierte Breitensuche auf dem Originalgraphen sind in Algorithmus 1 und 2 dargestellt.

Algorithmus 1 : Knotenbasierte Breitensuche

```

FUNCTION breadth_first(nodeS, nodeE)
  /* E is the set of graph edges */
  dist ← 0
  seen ← (nodeS, dist)
  while not all nodes considered do
    for (v, dist) ∈ seen do
      new ← {(w, dist + 1) | w ∉ seen ∧ (v, w) ∈ E}
      if nodeE ∈ new then
        | return dist + 1
      end
      seen ← seen ∪ new;
    end
    dist ← dist + 1
  end
  return null
end

```

2.3 Distanzsuche auf GRIPP

Auf der GRIPP-Indexstruktur kann genauso wie auf dem Originalgraphen zur Beantwortung einer Distanzfrage eine knoten- oder mengenbasierte Breitensuche durchgeführt werden. Anders als bei einem normalen Graphen wird für einen Kontextknoten q nicht nur seine unmittelbare Nachbarschaft, sondern alle Knoten $w \in RIS(q)$ untersucht. Diese können unter Ausnutzung der Eigenschaft der Pre-/Postnummerierung mit einer einzigen SQL-Anfrage berechnet werden. Dabei entsteht allerdings das Problem, dass auch Knoten betrachtet werden, die vom Startknoten der Anfrage weiter entfernt sind als die tatsächlich gesuchte Distanz (vgl. Aussage aus 1.4). Wenn innerhalb der $RIS(q)$ -Menge

Algorithmus 2 : Mengenbasierte Breitensuche

```

FUNCTION breadth_first_set(nodeS, nodeE)
  /* E is the set of graph edges */
  dist ← 0
  seen ← (nodeS, dist)
  while not all nodes considered do
    new ← { (w, dist + 1) | w ∉ seen ∧ (v, w) ∈ E ∧ (v, dist) ∈ seen }
    if nodeE ∈ new then
      | return dist + 1
    end
    seen ← seen ∪ new
    dist ← dist + 1
  end
  return null
end

```

eine Instanz des Endknotens der Anfrage gefunden wurde, heisst es nicht, dass die Suche nach der kürzesten Distanz beendet ist. Es müssen noch die durch die Nichtbauminstanzen angezeigten Verweise mit einer kürzeren Entfernung zum aktuellen Kontextknoten q verfolgt werden, denn diese könnten zu einem insgesamt noch kürzeren Pfad führen. Um der Charakteristik ‘Breitensuche’ gerecht zu werden, müssen die Verweise der Nichtbauminstanzen in aufsteigender Reihenfolge ihrer Entfernung von dem Startknoten der Anfrage weiterverfolgt werden.

Die knoten- und mengenbasierte Breitensuche auf der GRIPP-Indexstruktur unterscheiden sich dadurch, dass bei dem mengenbasierten Ansatz alle Verweise der Nichtbauminstanzen mit derselben Entfernung vom Startknoten der Anfrage gleichzeitig behandelt werden. Dies impliziert:

1. Berechnen der *RIS*-Mengen für die entsprechenden Bauminstanzen,
2. Suchen nach den naheliegendsten Instanzen des Zielknotens in den *RIS*-Mengen,
3. Hinzufügen der noch zu behandelnder Nichtbauminstanzen innerhalb der *RIS*-Mengen zu einer globalen Liste.

Die Information über bereits verfolgte Verweise darf dabei nicht verloren gehen. Wenn der Verweis einer Nichtbauminstanz des Knotens k bereits verfolgt wurde und im späteren Verlauf eine andere Nichtbauminstanz von k in einer der untersuchten *RIS*-Mengen gefunden wird, braucht der Verweis nicht erneut verfolgt werden, da er zu höchstens

längeren Distanzen führen kann.

Ähnlich kann eine Nichtbauminstanz ignoriert werden, deren entsprechende Bauminstanz sich in einer bereits betrachteten *RIS*-Menge befindet und zu höchstens längeren Pfaden führen kann. Die Algorithmen 3 und 4 zeigen die knoten- und mengenbasierte Breitensuche auf der GRIPP-Indexstruktur.

Algorithmus 3 : Knotenbasierte Breitensuche auf GRIPP

```

FUNCTION gripp_breadth_first(nodeS, nodeE)
  /* level(w) is the distance of node w to nodeS */
  /* treeInst(w) is the tree instance of the non-tree instance w */
  /* inst(w) = 1 iff w is a non-tree instance else 0 */
  tempHopNodes ← {(treeInst(w), level(w)) | w ∈ RIS(nodeS) ∧ inst(w) = 1}
  if nodeE ∈ RIS(nodeS) then
    | dist ← min{level(nodeE ∈ RIS(nodeS))}
  end
  else dist ← ∞
  level ← 0
  while ∃v ∈ tempHopNodes: level(v) > level ∧ level(v) < dist do
    | level ← getNextMinLevel(tempHopNodes, level)
    | for {v | v ∈ tempHopNodes ∧ level(v) = level} do
      | if nodeE ∈ RIS(v) ∧ min{level(nodeE ∈ RIS(v))} < dist then
        | | dist ← min{level(nodeE ∈ RIS(v))}
      | end
      | tempHopNodes ← tempHopNodes ∪ {(treeInst(k), level(k)) |
        | | k ∈ RIS(v) ∧ level(k) < (dist - 1) ∧ inst(k) = 1}
    | end
  end
  return dist
end

```

Algorithmus 4 : Mengenbasierte Breitensuche auf GRIPP

```

FUNCTION gripp_breadth_first_set(nodeS, nodeE)
  /* level(w) is the distance of node w to nodeS */
  /* treeInst(w) is the tree instance of the non-tree instance w */
  /* inst(w) = 1 iff w is a non-tree instance else 0 */
  tempHopNodes  $\leftarrow$   $\{(treeInst(w), level(w)) \mid w \in RIS(nodeS) \wedge inst(w) = 1\}$ 
  if nodeE  $\in$  RIS(nodeS) then
    | dist  $\leftarrow$  min{level(nodeE  $\in$  RIS(nodeS))}
  end
  else dist  $\leftarrow$   $\infty$ 
  unifiedRIS  $\leftarrow$   $\emptyset$ 
  nextHopNodes  $\leftarrow$   $\emptyset$ 
  level  $\leftarrow$  0
  while  $\exists w \in tempHopNodes : level(w) > level \wedge level(w) < dist$  do
    | level  $\leftarrow$  getNextMinLevel(tempHopNodes, level)
    | nextHopNodes  $\leftarrow$  getHopNodesWithLevel(tempHopNodes, level)
    | unifiedRIS  $\leftarrow$  getRISets(nextHopNodes)
    | if nodeE  $\in$  unifiedRIS  $\wedge$  min{level(nodeE)}  $<$  dist then
      | | dist  $\leftarrow$  min{level(nodeE)}
    | end
    | tempHopNodes  $\leftarrow$  tempHopNodes  $\cup$   $\{(treeInst(k), level(k)) \mid$ 
      | |  $k \in unifiedRIS \wedge level(k) < dist \wedge inst(k) = 1\}$ 
    | end
  return dist
end

```

Kapitel 3

Analysen und Optimierungen

3.1 Testumgebung

Für alle im folgenden Verlauf durchgeführten Performance- und Qualitätstests wurden im Voraus für verschiedene Graphengrößen und -dichten zufällig jeweils 100 Knotenpaare generiert und die kürzesten Distanzen mittels der knotenbasierten Breitensuche auf dem Originalgraphen berechnet. Bei den verwendeten Graphen handelt es sich um synthetisch generierte zusammenhängende gerichtete skalenfreie Graphen, die durch den Graphgenerator von S. Trißl erzeugt wurden. Die vorausberechneten Distanzen wurden in relationalen Tabellen mit der Struktur *samples(nodes, nodee, distance)* abgelegt. Die Einschränkung auf nur Knotenpaare, für die es tatsächlich einen Pfad gibt, resultiert aus der Überlegung, dass für jede Distanzanfrage zunächst ein Erreichbarkeitstest durchgeführt werden kann. Dieser ist einerseits extrem performant [TL06], andererseits erhält man für nicht durch einen Pfad verbundene Knoten sehr schnell eine (negative) Antwort. Alle verwendeten Algorithmen liegen als PL/SQL-Routinen vor.

3.2 Optimierung gegebener Routinen

Im Folgenden werden Optimierungen an den überlassenen Implementierungen beschrieben, die insgesamt zu merklichen Performanceverbesserungen geführt haben.

3.2.1 Knotenbasierte Breitensuche auf GRIPP

Im Verlauf der knotenbasierten Breitensuche auf GRIPP wird eine Liste (*temp_hop_nodes_dist*) der noch zu untersuchenden Sprungknoten geführt. Allerdings

werden bei der vorliegenden Implementierung nicht die referenzierten Bauminstanzen, sondern direkt die gefundenen Nichtbauminstanzen zu der Liste hinzugefügt. Damit eine solche Nichtbauminstanz in die Liste aufgenommen werden kann, darf ihre Entfernung vom Startknoten der Anfrage nicht länger sein als die kürzeste bislang gefundene Distanz zwischen dem Startknoten der Anfrage und einer Instanz des Zielknotens. Beim späteren Entnehmen der Nichtbauminstanzen aus der Liste zwecks Untersuchung der *RIS*-Menge der entsprechenden Bauminstanz wurde in der ursprünglichen Implementierung nicht überprüft, ob diese Bauminstanz nicht schon früher und damit über einen kürzeren Pfad erreicht wurde und somit nicht mehr untersucht werden braucht:

```

1  CURSOR breadth_next_node(min_dist INTEGER) IS
2    SELECT A.node_name, A.preorder, A.postorder, A.distance
3    FROM (
4      SELECT node_name, preorder, postorder, distance
5      FROM temp_hop_nodes_dist
6      WHERE used_as_hop = 'f'
7      AND distance < min_dist-1
8      ORDER BY distance, preorder asc) A
9    WHERE rownum = 1;

```

Die entsprechende SQL-Anweisung wurde diesbezüglich angepasst:

```

1  CURSOR breadth_next_node(min_dist INTEGER) IS
2    SELECT A.node_name, A.preorder, A.postorder, A.distance
3    FROM (
4      SELECT t1.node_name, t1.preorder, t1.postorder, t1.distance
5      FROM temp_hop_nodes_dist t1
6      WHERE t1.used_as_hop = 'f'
7      AND t1.distance < (min_dist - 1)
8      AND NOT EXISTS (SELECT 1 FROM temp_hop_nodes_dist t2
9                       WHERE t2.node_name=t1.node_name AND t2.used_as_hop='t')
10     ORDER BY t1.distance, t1.preorder asc) A
11   WHERE rownum = 1;

```

Die ursprüngliche Implementierung enthielt folgende SQL-Anweisung zum Hinzufügen von Nichtbauminstanzen zu der Liste *temp_hop_nodes_dist*. Es werden aus der *RIS*-Menge eines Sprungknotens mit den Pre-/Postwerten [*query_{pre}*, *query_{post}*] diejenigen Nichtbauminstanzen selektiert, die nicht weiter vom Startknoten der Anfrage entfernt sind als die bislang gefundene kürzeste Distanz. Intuitiv ist klar, dass beim Vorliegen mehrerer Nichtbauminstanzen zu einem Knoten nur diejenige interessant ist, welche die kürzeste Entfernung besitzt. In der ursprünglichen Implementierung wird eine GROUP-BY-Klausel samt Min-Aggregationsfunktion verwendet, die nicht das gewünschte Ergebnis erzielt. Innerhalb von *RIS*-Mengen treten keine zwei Instanzen mit derselben Preorder-/Postordernummer auf. Somit hat die Anwendung der Gruppierung keine Auswirkungen:

```

1 INSERT INTO temp_hop_nodes_dist
2   SELECT node_name, preorder, postorder, depth, MIN(depth - query_depth), 'f'
3   FROM gripp_depth
4   WHERE preorder > query_pre
5         AND preorder < query_post
6         AND node_inst = 1
7         AND depth-query_depth < min_dist-1
8   GROUP BY node_name, preorder, postorder, depth;

```

Da SQL diesbezüglich keine Möglichkeiten bietet, das Problem einfach und elegant zu lösen, wurde die GROUP-BY-Klausel einfach entfernt, um dem datenbankinternen Optimierer mehr Freiheitsgrade zu geben. Die uninteressanten Nichtbauminstanzen werden stattdessen beim späteren Auslesen der Liste verworfen (siehe dazu den oben erwähnten Cursor *breadth_next_node*):

```

1 INSERT INTO temp_hop_nodes_dist
2   SELECT node_name, preorder, postorder, depth, (depth - query_depth), 'f'
3   FROM gripp_depth
4   WHERE preorder > query_pre AND preorder < query_post
5         AND node_inst = 1 AND depth-query_depth < (min_dist - 1);

```

Tabelle 3.1 zeigt die Laufzeiten der ursprünglichen und der optimierten Version der knotenbasierten Breitensuche im Vergleich.

GRAPH	ALTE VERSION	NEUE VERSION
50T_100T	62s 70s 45s	49s 51s 37s
100T_200T	106s 57s 118s	85s 44s 95s

Tabelle 3.1: Laufzeiten der ursprünglichen und der optimierten Implementierung knotenbasierter Breitensuche auf GRIPP (*avg|med|stddev*)

3.2.2 Mengenbasierte Breitensuche auf GRIPP

Bei der Implementierung der mengenbasierten Breitensuche auf GRIPP wird in einer temporären Relation *temp_hop_nodes_dist* die Liste aller noch zu untersuchenden Sprungknoten (*hop nodes*) verwaltet. Die Relation *temp_hop_nodes_dist* hat folgende Struktur: *temp_hop_nodes_dist*(*node_name*, *preorder*, *postorder*, *depth*, *distance*, *used_as_hop*). Dabei beschreibt *depth* die Tiefe des Knotens innerhalb des GRIPP-Baums und *distance* seine Entfernung vom Startknoten der Anfrage. Im Verlauf des Algorithmus müssen aus allen *RIS*-Mengen der Sprungknoten mit derselben Entfernung (*hop_level*) die noch zu untersuchenden Nichtbauminstanzen, genauer gesagt ihre entsprechenden

Bauminstanzen (Sprungknoten), zu *temp_hop_nodes_dist* hinzugefügt werden. In der ursprünglichen Version wurde dies durch das folgende PL/SQL-Statement realisiert:

```

INSERT INTO temp_hop_nodes_dist
2  SELECT p2.node_name, p2.preorder, p2.postorder, p2.depth,
      (p1.depth - t.depth + t.distance) AS distance, 'f'
4  FROM temp_hop_nodes_dist t, gripp_depth p1, gripp_depth p2
   WHERE t.distance = hop_level
6  AND t.preorder < p1.preorder AND t.postorder > p1.preorder
   AND p1.node_inst = 1 AND p1.node_name = p2.node_name
8  AND (p1.depth - t.depth + t.distance) < (cur_dist - 1)
   AND p2.node_inst = 0 AND (p2.postorder - p2.preorder) > 1;

```

Es hat sich herausgestellt, dass das Aufbrechen dieser Operation in zwei SQL-Anfragen und das Umsetzen in Form einer FOR-Schleife unter Verwendung eines Cursors zu sichtlicher Performanceverbesserung führt. Außerdem brauchen nur solche Sprungknoten zur Liste hinzugefügt werden, bei denen die Möglichkeit, einen kürzeren Pfad als bereits durch *cur_dist* beschrieben, zumindest im Zeitpunkt des Hinzufügens zu der Liste noch besteht:

```

FOR w IN (SELECT MIN(p1.depth - t.depth + t.distance) AS distance, p1.node_name
2  FROM temp_hop_nodes_dist t, gripp_depth p1
   WHERE t.distance = hop_level
4  AND t.preorder < p1.preorder AND t.postorder > p1.preorder
   AND p1.node_inst = 1
6  AND (p1.depth - t.depth + t.distance) < (cur_dist - 1)
   AND NOT EXISTS
8  (SELECT 1 FROM temp_hop_nodes_dist t1
   WHERE t1.node_name = p1.node_name
10  AND t1.distance < (p1.depth - t.depth + t.distance) )
12  GROUP BY p1.node_name
   ) LOOP

14  INSERT INTO temp_hop_nodes_dist
   SELECT p2.node_name, p2.preorder, p2.postorder, p2.depth, w.distance, 'f'
16  FROM gripp_depth p2
   WHERE w.node_name = p2.node_name
18  AND p2.node_inst = 0 AND (p2.postorder - p2.preorder) > 1;
20 END LOOP;

```

Weiterhin wird im Verlauf der Suche jeweils das nächste *hop level* ermittelt. Dieses entspricht der Entfernung der als nächstes zu untersuchenden Nichtbauminstanzen aus *temp_hop_nodes_dist*. Es wird in der ursprünglichen Version durch die folgende Anweisung berechnet:

```

CURSOR next_distance(min_dist integer, hop_level integer) IS
2  SELECT min(t.distance)
   FROM temp_hop_nodes_dist t
4  WHERE t.distance > hop_level
   AND t.distance < min_dist;

```

Dabei beschreibt *min_dist* die aktuell kürzeste gefundene Distanz zu einer Instanz des Zielknotens der Anfrage und *hop_level* das letzte untersuchte *hop level*. Es kann im Verlauf des Algorithmus passieren, dass zu einem Sprungknoten *s* zuerst eine Nichtbauminstanz *s'* mit einer Entfernung s'_d gefunden und zu der Liste *temp_hop_nodes_dist* hinzugefügt wird und später eine andere Nichtbauminstanz *s''* mit einer Entfernung $s''_d < s'_d$. Da die Nichtbauminstanzen in aufsteigender Reihenfolge ihrer Entfernung vom Startknoten der Anfrage bearbeitet werden, wird zuerst *s''* und dann später *s'* bearbeitet. Dabei kann *s'* aber aufgrund seiner größeren Entfernung vom Startknoten der Anfrage nicht zu einer kürzeren Gesamtdistanz führen. Somit sollte die obige Anweisung wie folgt verändert werden:

```

CURSOR next_distance(min_dist integer, hop_level integer) IS
2  SELECT min(d1) FROM (
      SELECT t.distance d1, t2.distance d2
4     FROM temp_hop_nodes_dist t, temp_hop_nodes_dist t2
      WHERE t.distance > hop_level AND t.distance < min_dist
6         AND t.node_name = t2.node_name(+)
          AND t.distance != t2.distance(+)
8         AND t2.distance(+) <= hop_level)
      WHERE d2 IS NULL;

```

Dadurch werden der Liste *temp_hop_nodes_dist* nur solche Nichtbauminstanzen *s'* zur Bearbeitung entnommen, für die keine andere Nichtbauminstanz *s''* existiert, wobei *s'* und *s''* sich auf denselben Sprungknoten *s* beziehen und *s''* eine geringere Entfernung vom Startknoten der Anfrage als *s'* besitzt.

Tabelle 3.2 zeigt die Laufzeiten der ursprünglichen und der optimierten Version der mengenbasierten Breitensuche und damit deutlich die Verbesserung der neuen Version.

GRAPH	ALTE VERSION	NEUE VERSION
50T_100T	134s 117s 124s	22s 25s 13s
100T_200T	311s 316s 298s	62s 75s 51s

Tabelle 3.2: Laufzeiten der ursprünglichen und der optimierten Implementierung der mengenbasierten Breitensuche auf GRIPP (*avg|med|stddev*)

3.3 Erste Tests

Zu allererst musste ein Überblick über das Laufzeitverhalten der verschiedenen Algorithmen auf dem Originalgraphen und der GRIPP-Indexstruktur gewonnen werden. In Tabelle 3.3 sind die durchschnittlichen Ausführungszeiten der verschiedenen Breitensuchen

dargestellt. Es sind jeweils die durchschnittliche Zeit, der Median und die Standardabweichung angegeben. Man sieht, dass schon für nicht sehr große Graphen die knotenbasierte Breitensuche auf dem Originalgraphen schneller als die mengenbasierte Breitensuche auf der GRIPP-Indexstruktur ist. Die mengenbasierte Breitensuche auf dem Originalgraphen sowie die knotenbasierte Breitensuche auf der GRIPP-Indexstruktur benötigten bereits für mittelgroße Graphen wesentlich länger, so dass sie auf größeren Graphen nicht mehr getestet wurden.

Im Abschnitt 2.3 ist bereits das Problem mit der Ineffizienz der Breitensuche auf der GRIPP-Indexstruktur angesprochen worden. Es werden unter anderem Knoten untersucht, die von dem Startknoten der Anfrage weiter entfernt sind als die gesuchte kürzeste Distanz. Den Grad dieser Effizienz kann man dadurch steuern, dass den GRIPP-basierten Routinen eine obere Schranke für die gesuchte Distanz übergeben wird. Welchen Einfluss eine solche Schranke hat und wie man sie berechnen kann, wird in nächsten Abschnitten behandelt.

In Abbildung 3.1 sind auf einer logarithmischen Skala die Laufzeiten der verschiedenen

GRAPH	BFS	SB_BFS	BFS_GRIPP	SB_BFS_GRIPP
1T_2T	366 312 293	26 16 32	58 31 71	41 31 35
5T_10T	1,5s 1,3s 1s	0,5s 0,3s 0,5s	1s 0,4s 1s	0,4s 0,3s 0,2s
10T_20T	3s 2s 2s	2s 1s 3s	3s 1s 4s	1s 1s 1s
10T_30T	3s 3s 2s	7s 3s 8s	10s 4s 19s	4s 3s 3s
10T_40T	3s 3s 2s	11s 3s 14s	14s 5s 32s	7s 6s 6s
50T_100T	15s 16s 10s	107s 83s 101s	89s 46s 136s	24s 23s 16s
100T_200T	35s 30s 26s	—	—	87s 76s 63s
100T_400T	30s 20s 25s	—	—	826s 688s 673s
500T_1M	164s 155s 100s	—	—	2120s 1800s 1652s

Tabelle 3.3: Durchschnittliche Dauer der Distanzanfragen für verschiedene Algorithmen.

Algorithmen für Graphen mit 100% mehr Kanten als Knoten dargestellt.

3.4 Die beste obere Schranke

Bevor es darum geht, wie man eine möglichst gute, d.h. kleine und schnell ermittelte, obere Schranke erhalten kann, muss untersucht werden, welchen positiven Effekt eine solche Schranke auf die Laufzeit der GRIPP-basierten Algorithmen zur Distanzberechnung maximal haben kann. Dazu wurde die knotenbasierte und mengenbasierte Distanzsuche

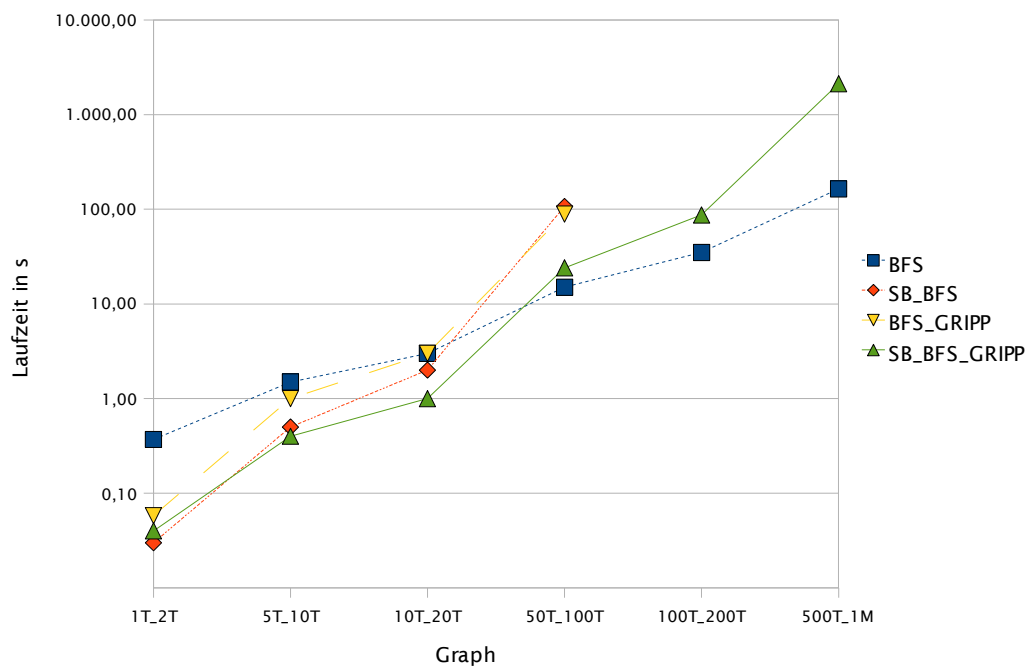


Abbildung 3.1: Durchschnittliche Laufzeiten verschiedener Breitensuchealgorithmen.

auf GRIPP mit der bestmöglichen oberen Schranke - der tatsächlichen kürzesten Distanz - parametrisiert. Tabelle 3.4 zeigt die dabei gemessenen Laufzeiten und zum Vergleich die Zeiten der Suchen ohne obere Schranke.

Man sieht, dass eine gute obere Schranke einen großen Einfluss auf das Laufzeitverhalten hat. Jedoch sind die neuen Zeiten immer noch schlechter als diejenigen der knotenbasierten Breitensuche auf dem Originalgraphen aus der Tabelle 3.3. Dies zeigt, dass die vorliegenden GRIPP-basierten Algorithmen auch abgesehen von den in 2.3 beschriebenen Problemen weniger effizient sind als die knotenbasierte Breitensuche auf dem Originalgraphen.

3.5 Distanzenverteilung und Hopcount

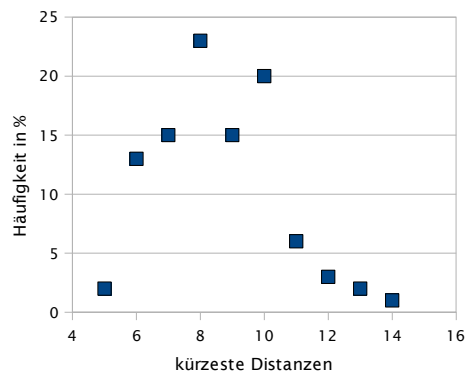
Im nachfolgenden Kapitel werden verschiedene Ansätze zur Bestimmung einer oberen Schranke vorgestellt. Vorab liefert Abbildung 3.2 einen Überblick über die Verteilung der kürzesten Distanzen für ausgewählte Testgraphen. Man kann erkennen, dass mit zunehmender Graphgröße die durchschnittliche kürzeste Distanz nur geringfügig ansteigt. Abbildung 3.3 zeigt für die mengenbasierte Breitensuche auf GRIPP (1) die Anzahl der zur Liste der noch zu untersuchenden Sprungknoten hinzugefügten Knoten und (2) die Anzahl der tatsächlich untersuchten Sprungknoten bzw. *RIS*-Mengen, wobei die

GRAPH	BFS_GRIPP	SB_BFS_GRIPP	BFS_GRIPP	SB_BFS_GRIPP
1T_2T	34 16 51	18 16 19	58 31 71	41 31 35
5T_10T	489 172 864	158 94 177	1s 0, 4s 1s	0, 4s 0, 3s 0, 2s
10T_20T	1, 5s 0, 4s 3s	525 281 603	3s 1s 4s	1s 1s 1s
10T_30T	5s 1s 10s	2s 0, 8s 2, 5s	10s 4s 19s	4s 3s 3s
10T_40T	5s 1s 13s	3s 1s 4s	14s 5s 32s	7s 6s 6s
50T_100T	56s 22s 99s	15s 11s 14s	89s 46s 136s	24s 23s 16s
100T_200T	189s 49s 311s	57s 36s 57	–	87s 76s 63s
100T_400T	799s 173s 2057s	345s 168s 469s	–	826s 688s 673s
500T_1M	–	1564s 898s 1570s	–	2120s 1800s 1652s

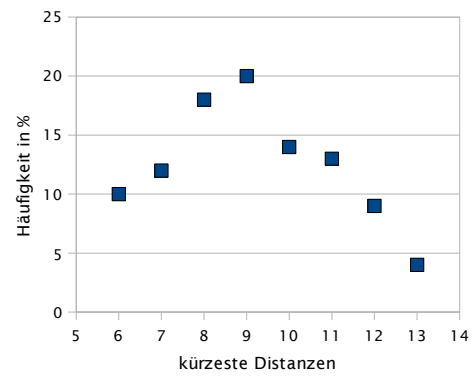
Tabelle 3.4: Durchschnittliche Dauer der Distanzanfragen für GRIPP-basierte Algorithmen mit der *bestmöglichen* oberen Schranke. In Grau zum Vergleich die Zeiten der Breitensuchen ohne obere Schranke.

RIS-Menge des Startknotens mitgezählt wird. Man sieht die deutliche Korrelation zwischen der Anzahl der tatsächlich durchgeführten Sprünge und der tatsächlich kürzesten Distanz. Dies kann wie folgt erklärt werden. Wenn man sich den kürzesten Weg zwischen dem Start- und Zielknoten der Anfrage im Graph als eine Abfolge der Knoten s, n_1, n_2, \dots, z vorstellt und diese Schritte auf den GRIPP-Baum projiziert, dann ist ausgehend von s und entsprechender *RIS*-Menge viel wahrscheinlicher n_1 zunächst als eine Nichtbauminstanz zu finden. Dies wird dadurch erklärt, dass zu einem beliebigen Knoten des Graphen im GRIPP-Baum zwar genau eine Bauminstanz, aber dafür im Allgemeinen viele Nichtbauminstanzen existieren. Somit wird für jeden Knoten des kürzesten Pfades ein Sprung im GRIPP-Baum sehr wahrscheinlich. Ausserdem ist es unwahrscheinlich, dass die vom aktuell untersuchten Sprungknoten (innerhalb seiner *RIS*-Menge) weiter entfernte Nichtbauminstanzen zum kürzesten Pfad beitragen. An dieser Stelle könnte eine obere Schranke die Verarbeitung unnötiger Sprungknoten verhindern.

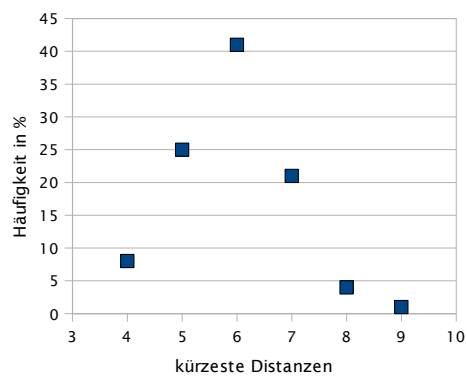
In Abbildung 3.4 sind für die knotenbasierte Breitensuche und mengenbasierte Breitensuche auf GRIPP beispielhaft die Suchzeiten in Abhängigkeit von der gesuchten kürzesten Distanz dargestellt. Man kann deutlich erkennen, dass es nur einen Bereich der Stärke gibt - der GRIPP-basierte Ansatz ist hier durchweg weniger effizient.



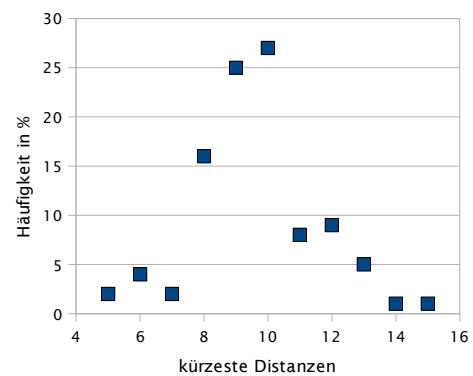
(a) Graph 50 000_100 000



(b) Graph 100 000_200 000

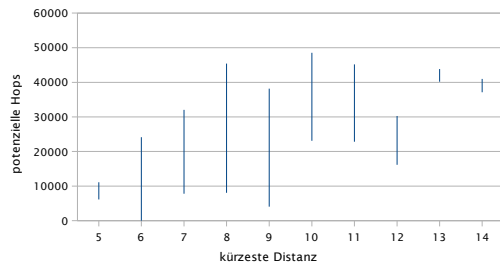


(c) Graph 100 000_400 000

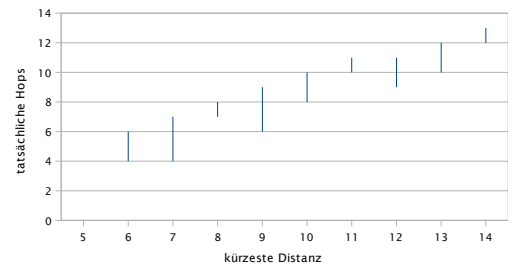


(d) Graph 500 000_1 000 000

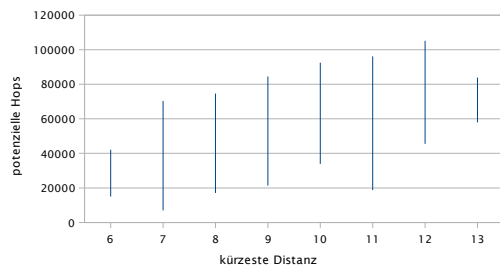
Abbildung 3.2: Verteilung kürzester Distanzen für ausgewählte Testgraphen und 100 zufällige Knotenpaare.



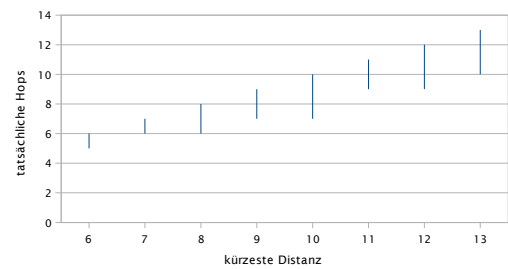
(a) Graph 50 000_100 000 - potenzielle Hops



(b) Graph 50 000_100 000 - tatsächliche Hops

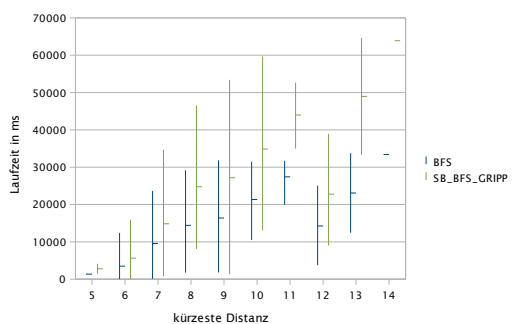


(c) Graph 100 000_200 000 - potenzielle Hops

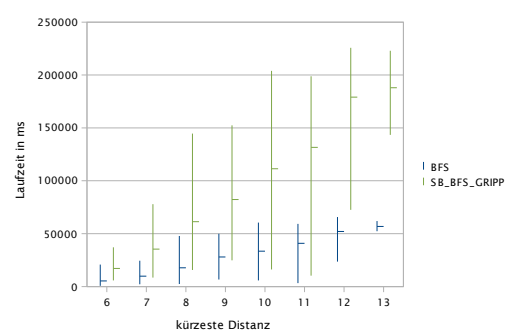


(d) Graph 100 000_200 000 - tatsächliche Hops

Abbildung 3.3: Anzahl potenzieller und tatsächlicher Hops für zwei unterschiedliche Graphen und je 100 Distanzsuchen.



(a) Graph 50 000_100 000



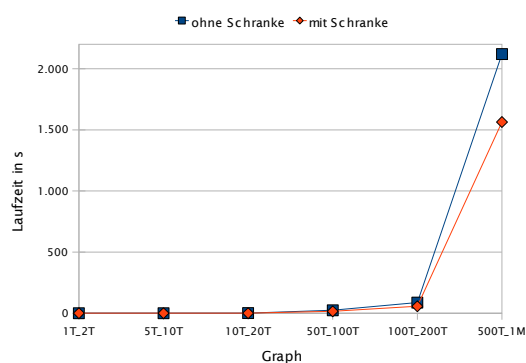
(b) Graph 100 000_200 000

Abbildung 3.4: Laufzeiten knotenbasierter Breitensuche und mengenbasierter Breitensuche auf GRIPP in Abhängigkeit von der gesuchten kürzesten Distanz (*min, max, avg*).

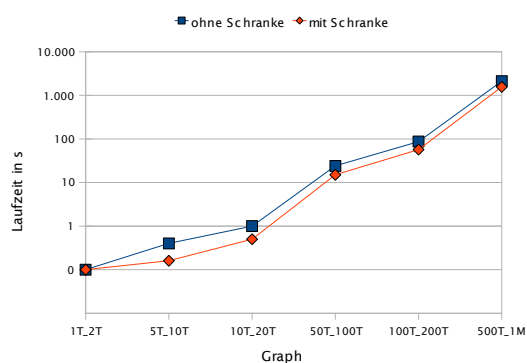
Kapitel 4

Obere Schranken

Auch wenn in 3.3 gezeigt wurde, dass die Distanzsuche unter Verwendung der GRIPP-Indexstruktur im Punkt Performance auch mit der bestmöglichen oberen Schranke nicht diejenige auf dem Originalgraphen schlagen konnte, ist aufgrund ihres Einflusses doch interessant zu untersuchen, wie man eine gute obere Schranke für die GRIPP-basierten Distanzalgorithmen erhalten kann. In diesem Kapitel werden verschiedene Ansätze beschrieben und untersucht. Abbildung 4.1 stellt für die mengenbasierte Breitensuche auf GRIPP für verschiedene Graphgrößen das Laufzeitverhalten ohne und mit bestmöglicher oberer Schranke dar und zeigt damit die durch eine obere Schranke mögliche Optimierungsspanne an.



(a) lineare Skala



(b) logarithmische Skala

Abbildung 4.1: Durchschnittliche Laufzeit der mengenbasierten Breitensuche auf GRIPP ohne und mit bestmöglicher oberer Schranke.

4.1 Shortcut-Reachability

Bei den diversen Variationen der GRIPP-basierten Distanzsuche von S. Trißl verbarg sich unter anderem eine Variation des Algorithmus für die Beantwortung von Erreichbarkeitsanfragen, die im weiteren Verlauf als *Shortcut Reachability* bezeichnet wird. Der Standardalgorithmus zur Beantwortung von Erreichbarkeitsanfragen - *GRIPP-Reachability* - läuft die GRIPP-Baumstruktur in Tiefensuche ab und sucht nach der ersten Instanz des Zielknotens der Anfrage. Dabei kann die Länge des Weges protokolliert werden. Im Allgemeinen ist die Länge eines solchen Weges sehr groß und übersteigt die gesuchte kürzeste Distanz um das Vielfache, weswegen sie sich kaum als eine obere Schranke für die nachfolgende Breitensuche eignet. *Shortcut Reachability* erweitert den Algorithmus wie folgt:

- Alle Knoten p auf dem Pfad zwischen dem Startknoten der Anfrage und der durch die Tiefensuche gefundenen Instanz des Zielknotens werden mit ihrer Entfernung zum Zielknoten der Anfrage ($dist_{node_e}(p)$) zu der Menge P hinzugefügt.
- In jeder auf dem oben erwähnten Pfad untersuchten *RIS*-Menge werden Nichtbauminstanzen s gesucht, die theoretisch zu einer kürzeren Gesamtdistanz führen könnten. Solche Nichtbauminstanzen werden mit ihrer Entfernung zum Startknoten der Anfrage ($dist_{node_s}(s)$) zu einer Menge S hinzugefügt.
- Anschließend wird geprüft, ob unter Verwendung der Nichtbauminstanzen aus S eine Abkürzung (*Shortcut*) für den langen Pfad der Tiefensuche gefunden werden kann: $d' = \min\{dist_{node_s}(s) + dist_{node_e}(p) | s \in S \wedge p \in P\}$.

Der Algorithmus der *Shortcut Reachability* liefert neben der Information über die Existenz eines Weges auch eine um Größenordnungen bessere obere Schranke als der normale *Reachability* Algorithmus. Dabei hat der Algorithmus offensichtlich eine auf die Größe des Graphen bezogene lineare Zeitkomplexität. Leider verschlechtert sich die Qualität der oberen Schranken mit zunehmender Graphgröße, wodurch abzusehen ist, dass auch dieses Verfahren für große Graphen nicht effizient genug sein wird.

In der Tabelle 4.1 sind für verschiedene Graphen die durchschnittlichen oberen Schranken der normalen *Reachability* (UB_NR), der *Shortcut Reachability* (UB_SR) sowie die tatsächlichen kürzesten Distanzen (REAL_DIST) und benötigten Berechnungszeiten dargestellt. Abbildung 4.2 zeigt die schwache logarithmische Zunahme der durchschnittlichen kürzesten Distanzen für verschiedene Graphen mit 100% mehr Kanten als Knoten.

In der Tabelle 4.2 sind die Suchzeiten der GRIPP-basierten Breitensuchen unter Ver-

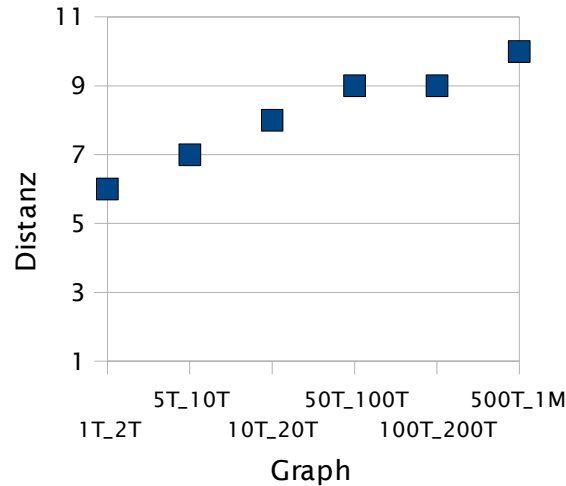


Abbildung 4.2: Verteilung durchschnittlich kürzester Distanzen für verschiedene Graphen mit 100% mehr Kanten als Knoten.

wendung der durch die Shortcut-Reachability berechneten oberen Schranken abgebildet. Der Vergleich der Zeiten mit denen ohne eine obere Schranke (Tabelle 3.3) zeigt, dass die durch Shortcut Reachability berechnete obere Schranke trotz ihrer sichtlichen Verbesserung gegenüber der oberen Schranke des normalen Reachability-Algorithmus (siehe Tabelle 4.1) kaum Auswirkungen auf die Laufzeit der GRIPP-basierten Distanzalgorithmen hat.

4.2 Obere Schranke und der k Index

Ein anderer untersuchter Ansatz zur Bestimmung einer guten oberen Schranke verwendet einen zusätzlichen Index, hier als $kIndex$ bezeichnet. Dabei werden k Knotenpaare des Graphen ausgewählt, für die im Voraus die Distanzen berechnet werden. Zu Anfragezeit wird dieser $kIndex$ verwendet, um die Länge des kürzesten Pfades zwischen den Knoten der Distanzanfrage möglichst gut abzuschätzen. Der $kIndex$ wird mittels einer Stored Procedure erzeugt und in einer relationalen Tabelle mit der Struktur $kIndex(sNode, eNode, kDist)$ gespeichert. Der Index wird unter Verwendung der GRIPP-Struktur wie folgt verwendet:

- Ausgehend von dem Startknoten der Distanzanfrage ($node_s$) werden in $RIS(node_s)$ Instanzen der $sNode$ -Knoten des $kIndex$ gesucht und in der temporären Relation $ks(sNode, level)$ gespeichert. Formal ergibt sich diese Menge durch

$$\{(w, level(w) \mid w \in kIndex_{sNode} \wedge w \in RIS(node_s)\}. \quad (4.1)$$

GRAPH	UB_NR	UB_SR	REAL_DIST	TIME_NR	TIME_SR
1T_2T	44 45 29	12 9 7	6 6 2	5 0 11	10 0 33
5T_10T	187 173 128	19 16 12	7 7 2	5 0 7	18 16 14
10T_20T	384 357 244	28 20 18	8 8 2	5 0 7	28 31 23
10T_30T	619 477 507	32 24 26	6 6 1	6 0 8	42 39 21
10T_40T	737 526 667	27 23 20	5 5 1	2 0 5	61 47 47
50T_100T	1789 1505 1359	56 51 37	9 8 2	14 16 9	104 94 47
100T_200T	3317 3059 2500	71 53 50	9 9 2	4 0 7	190 187 68
100T_400T	7457 5259 7129	69 47 64	6 6 1	3 0 6	525 515 220
500T_1M	16947 16783 11615	186 167 142	10 10 2	5 0 10	1204 1109 501

Tabelle 4.1: Durchschnittliche obere Schranken der normalen und Shortcut Reachability im Vergleich. Weiterhin sind die gesuchten kürzesten Distanzen und die Berechnungszeiten der oberen Schranken (in ms) dargestellt ($avg|med|stddev$).

Unter $level(w)$ wird die Entfernung des Knotens w vom Startknoten der Anfrage verstanden. Da es sein kann, dass die Menge $RIS(node_s)$ aufgrund der ungünstigen Lage von $node_s$ sehr klein ist, wird zusätzlich die größte RIS -Menge untersucht, die von einer in $RIS(node_s)$ befindlichen Nichtbauminstanz referenziert wird. Dazu werden zunächst zu allen Nichtbauminstanzen in $RIS(node_s)$ die entsprechenden Bauminstanzen p angeschaut und diejenige mit $max\{p_{postorder} - p_{preorder}\}$ ausgewählt.

- Ähnlich wird auch in der Umgebung des Zielknotens der Distanzanfrage ($node_e$) nach Instanzen der $eNode$ -Knoten des $kIndex$ gesucht. Dazu werden zu *allen* Instanzen des Zielknotens $node_e$ alle entsprechenden Vorfahrknoten untersucht. Wenn e eine Instanz des $node_e$ ist, dann ergeben sich alle ihre Vorfahren als

$$\{v \mid v_{pre} < e_{pre} \wedge v_{post} > e_{pre}\}. \quad (4.2)$$

Alle so gefundenen $eNode$ -Knoten werden in der temporären Relation $ke(eNode, level)$ gespeichert, wobei auch hier $level$ die Entfernung der $eNode$ -Instanz von der $node_e$ -Instanz beschreibt.

- Zuletzt wird folgendes Minimum bestimmt:

$$\min\{level(sNode) + kDist + level(eNode) \mid sNode \in ks \wedge eNode \in ke \wedge (kNode, eNode, kDist) \in kIndex\}. \quad (4.3)$$

GRAPH	BFS_GRIPP	SB_BFS_GRIPP	BFS_GRIPP	SB_BFS_GRIPP
1T_2T	52 16 76	31 31 27	58 31 71	41 31 35
5T_10T	664 328 977	276 250 220	1s 0,4s 1s	0,4s 0,3s 0,2s
10T_20T	2.5s 1s 4s	972 774 768	3s 1s 4s	1s 1s 1s
10T_30T	10s 3s 21s	3,5s 3s 3s	10s 4s 19s	4s 3s 3s
10T_40T	11s 3s 26s	6s 5s 5,5s	14s 5s 32s	7s 6s 6s
50T_100T	80s 40s 119s	22s 23s 15s	89s 46s 136s	24s 23s 16s
100T_200T	323s 83s 644s	85s 74s 64s	–	87s 76s 63s
100T_400T	–	675s 477s 604s	–	826s 688s 673s
500T_1M	–	2094s 1800s 1645s	–	2120s 1800s 1652s

Tabelle 4.2: Durchschnittliche Ausführungszeiten der GRIPP-basierten Distanzalgorithmien unter Verwendung der oberen Schranken der *Shortcut Reachability*. In Grau zum Vergleich die Zeiten der Breitensuchen ohne obere Schranke.

Aus der oben geschilderten Anwendung ergeben sich Kriterien für die Auswahl der Knoten für den k Index. Ein solcher Index ist nur dann wirklich sinnvoll, wenn er für viele Anfragen verwendet werden kann. Somit müssen die k Index-Knoten so ausgewählt werden, dass sie möglichst häufig gefunden werden. Für einen $sNode$ -Knoten ist dies der Fall, wenn dieser viele Instanzen in der GRIPP-Baumstruktur besitzt, d.h. der Knoten im Originalgraphen viele Eingangskanten bzw. einen hohen Eingangsgrad hat. Ein $eNode$ -Knoten sollte wiederum einen hohen Ausgangsgrad besitzen.

Nach diesen Kriterien werden \sqrt{k} Knoten mit dem höchsten Eingangsgrad als $sNodes$ und \sqrt{k} Knoten mit dem höchsten Ausgangsgrad als $eNodes$ ermittelt. Das Testergebnis für einen skalenfreien Graphen mit 10 000 Knoten und 20 000 Kanten und $k = 10\,000$ präsentiert die Tabelle 4.3.

GRAPH	UB_KINDEX	QUOTE	TIME_KINDEX	BFS_GRIPP	SB_BFS_GRIPP
10T_20T	15 9 14	84%	71 71 60	1901 625 3903	803 602 754

Tabelle 4.3: Durchschnittliche obere Schranke durch den k Index und ihre Auswirkung auf die GRIPP-basierten Distanzsuchen¹².

Die durchschnittliche obere Schranke (UB_KINDEX) konnte durch den Einsatz des k Index auf 54% der oberen Schranke der *Shortcut Reachability* gesenkt werden. Aller-

¹(*avg|med|stddev*)

²TIME_KINDEX ist die Dauer für die Berechnung der oberen Schranke unter Verwendung des k Index. BFS_GRIPP und SB_BFS_GRIPP sind die Zeiten der knoten- und mengenbasierten Breitensuche auf GRIPP unter Verwendung dieser oberen Schranke.

dings lieferte der k Index-Ansatz in nur 84% der Fälle (QUOTE) überhaupt eine obere Schranke, obwohl für den Indexaufbau $k = 10\,000$ Distanzberechnungen und 8 Stunden Berechnungszeit notwendig waren. An diesem Beispiel ist bereits abzusehen, dass der zeitliche Aufwand für den Aufbau eines k Index mit einer vergleichbaren Trefferquote für größere Graphen nicht vertretbar ist. Man stelle sich einen entsprechenden k Index für einen Graphen mit 100 000 Knoten, 200 000 Kanten und $k = 10\,000$ vor. Dies würde unter Beachtung der durchschnittlichen Distanzberechnungsdauer mittels knotenbasierter Breitensuche auf dem Originalgraphen (siehe Tabelle 3.3) einen Berechnungsaufwand von Wochen beanspruchen. Für $k = 100\,000$ entsprechend vier Tage.

Die wesentlichen Konfigurationsparameter des k Index sind:

- Auswahlkriterium der $sNode$ - und $eNode$ -Knoten,
- Größe k des k Index,
- Verwendung des k Index für die Suche nach der oberen Schranke, dabei speziell die Auswahl und Anzahl der zu untersuchenden RIS -Mengen auf der Suche nach $sNode$ -Instanzen.

4.3 Fiktive obere Schranke

Eine weitere Methode, die GRIPP-basierten Distanzsuchen zu beschleunigen, besteht darin, die statistische Verteilung der kürzesten Distanzen in skalenfreien Graphen auszunutzen. Anhand vorhergegangener Beispiele kann man beobachten, dass mit zunehmender Größe der Graphen die durchschnittlichen kürzesten Distanzen nur geringfügig zunehmen. Cohen und Havlin zeigen in [CH03] in Anlehnung an die Arbeit von Bollobas and Riordan [BR04] aus dem Jahr 2000, dass skalenfreie Graphen einen sehr kleinen Diameter besitzen, der logarithmisch proportional zur Anzahl der Knoten ist. Unter Diameter verstehen sie, anders als in mathematischer Literatur, die durchschnittliche kürzeste Entfernung zwischen zwei beliebigen Knoten in einem Graphen. Die Verteilung der Knotengrade in einem skalenfreien Graph folgt dem Potenzgesetz:

$$p(k) \sim k^{-\gamma} \tag{4.4}$$

Zu den Aussagen der Arbeit von Cohen und Havlin gehören folgende Zusammenhänge zwischen dem Diameter und der Anzahl der Knoten in einem skalenfreien unkorrelierten Graphen:

- $2 < \gamma < 3$: $d \sim \ln \ln |V|$

- $\gamma = 3$: $d \sim \ln|V| / \ln \ln|V|$
- $\gamma > 3$: $d \sim \ln|V|$

Die Abschätzung des Diameters $d \sim \ln|V|$ kann als (konservative) Grundlage für eine Abschätzung einer fiktiven oberen Schranke ausgenutzt werden. Eine solche fiktive obere Schranke wird für einen Graphen und nicht pro Distanzanfrage berechnet. Für sie gilt im Gegensatz zu den oberen Schranken früherer Abschnitte, dass die gesuchte kürzeste Distanz nicht immer kleiner oder gleich der (fiktiven) oberen Schranke ist. Dies soll (nur) für den Großteil der Anfragen gelten. Für diese Anfragen werden die GRIPP-basierten Breitensuchen durch die relativ kleine obere Schranke mehr beschleunigt. Da im Vorhinein die gesuchte kürzeste Distanz unbekannt ist, muss bei einem negativen Ergebnis (d.h. Instanz des Zielknotens innerhalb der Distanz der fiktiven oberen Schranke nicht gefunden) die Suche ohne Schranke wiederholt werden.

Um zu berücksichtigen, dass mit zunehmender Graphdichte die durchschnittliche kürzeste Distanz abnimmt (vgl. Tabelle 4.1), geht das gewichtete Reziproke der Dichte in die Abschätzung der fiktiven oberen Schranke ein:

$$\hat{s} = \lfloor \ln|V| \left(1 + \frac{|V|}{|E|} \frac{1}{\ln \ln|E|} \right) \rfloor \quad (4.5)$$

Die Abschätzung von \hat{s} enthält einen Kompromiss - einerseits wird durch ein kleiner werdendes \hat{s} die Breitensuche auf GRIPP umso mehr beschleunigt, andererseits steigt dabei die Wahrscheinlichkeit für eine gegebene Anfrage, dass die gesuchte Distanz größer als \hat{s} ist und somit die Suche wiederholt werden muss.

Tabelle 4.4 stellt für verschiedene Graphen die Schranke \hat{s} im Vergleich zu den durchschnittlichen kürzesten Distanzen und den entsprechenden Standardabweichungen (vgl. Tabelle 4.1) dar. Weiterhin wird prozentuell die Anzahl der Pfade mit einer größeren Distanz als \hat{s} angegeben (AUSFALL).

In der Tabelle 4.5 sind die durchschnittlichen Laufzeiten der GRIPP-basierten Breitensuchen unter Verwendung der fiktiven oberen Schranke \hat{s} dargestellt. Der Vergleich der Zeiten mit den vorangegangenen Testergebnissen zeigt, dass dieser Ansatz im Durchschnitt nicht effizienter ist als die Suche gänzlich ohne eine obere Schranke durchzuführen. Dieses Verhalten wird dadurch erklärt, dass die Anfragen nach langen Distanzen vor allem durch mehrfache Ausführung übermäßig längere Laufzeiten haben. Die größere Diskrepanz zwischen dem Median und dem Durchschnittswert stützt diese Vermutung.

GRAPH	SCHRANKE \hat{s}	REAL_DIST	STDDEV(REAL_DIST)	AUSFALL
1T_2T	8	6	2	23%
5T_10T	10	7	2	12%
10T_20T	11	8	2	7%
10T_30T	10	6	1	2%
10T_40T	10	5	1	0%
50T_100T	13	9	2	3%
100T_200T	13	9	2	4%
100T_400T	12	6	1	0%
500T_1M	15	10	2	1%

Tabelle 4.4: Statistisch begründete obere Schranke \hat{s} im Vergleich

4.4 Vorhersage der BFS-Laufzeit

In diesem Abschnitt wird untersucht, ob die durch die Shortcut Reachability (vgl. dazu Seite 26) berechnete obere Schranke mit der Laufzeit der normalen Breitensuche auf dem Originalgraphen korreliert. Eine solcher Zusammenhang könnte bei der Bewertung verschiedener Ausführungspläne im Rahmen eines kostenbasierten Optimierers bei der Analyse komplexer Graphanfragen verwendet werden. Hierzu werden für zwei repräsentative Graphen anhand von jeweils 100 Knotenpaaren die durch den Shortcut Reachability Algorithmus ermittelte obere Schranke sowie die Laufzeit der Breitensuche auf dem Originalgraphen untersucht.

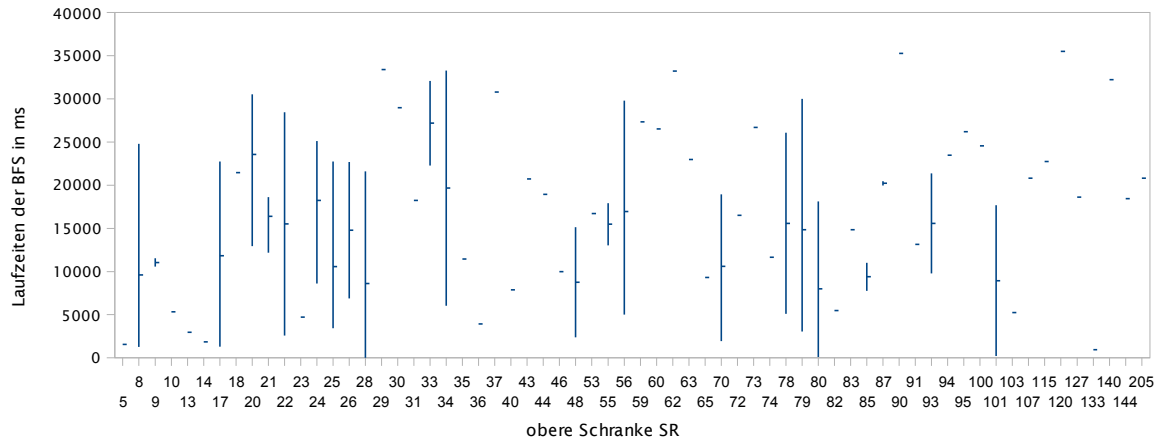
In Abbildung 4.3 sind die Laufzeiten der knotenbasierten Breitensuche auf dem Originalgraph in Abhängigkeit von den durch Shortcut Reachability berechneten oberen Schranken dargestellt. Für jede aufgetretene obere Schranke wird jeweils die kleinste und größte Suchdauer sowie die Durchschnittszeit (waagerechte Markierung) abgebildet. Man sieht deutlich, dass zwischen den beiden Größen kaum Korrelation besteht.

GRAPH	BFS_GRIPP	SB_BFS_GRIPP	BFS_GRIPP	SB_BFS_GRIPP
1T_2T	98 16 203	67 31 141	58 31 71	41 31 35
5T_10T	1047 297 2212	379 250 466	1s 0, 4s 1s	0, 4s 0, 3s 0, 2s
10T_20T	2761 688 6837	1214 836 1331	3s 1s 4s	1s 1s 1s
10T_30T	8s 3s 24s	4s 3s 4s	10s 4s 19s	4s 3s 3s
10T_40T	7s 3s 15s	7s 5s 6s	14s 5s 32s	7s 6s 6s
50T_100T	89s 32s 227s	24s 23s 21s	89s 46s 136s	24s 23s 16s
100T_200T	312s 74s 727s	93s 74s 82s	—	87s 76s 63s
100T_400T	—	799s 624s 698s	—	826s 688s 673s
500T_1M	—	2208s 1872s 1742s	—	2120s 1800s 1652s

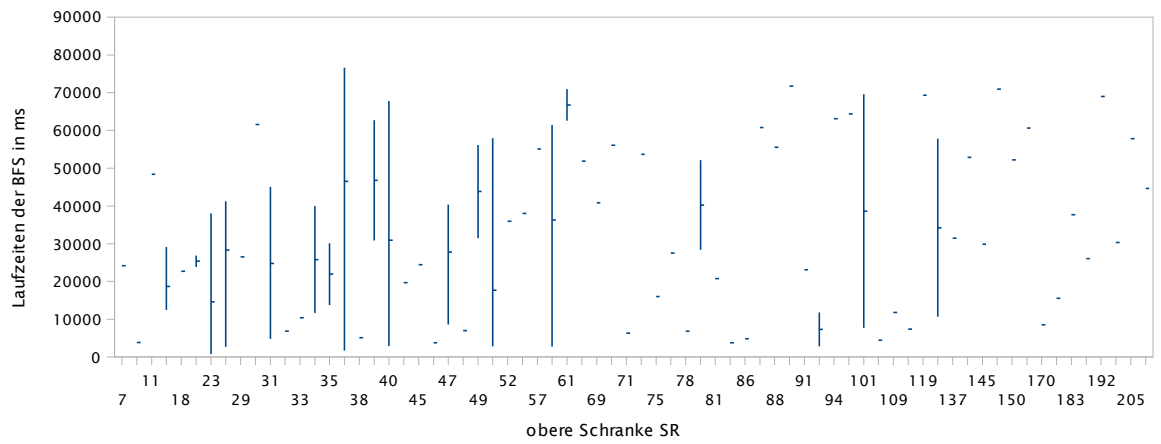
Tabelle 4.5: Durchschnittliche Ausführungszeiten der GRIPP-basierten Distanzalgorithmien unter Verwendung der fiktiven oberen Schranke \hat{s} . In Grau zum Vergleich die Zeiten der Breitensuchen ohne obere Schranke.

GRAPH	BFS_GRIPP	SB_BFS_GRIPP	BFS_GRIPP	SB_BFS_GRIPP
1T_2T	25 16 31	21 16 15	58 31 71	41 31 35
5T_10T	362 235 414	202 164 138	1s 0, 4s 1s	0, 4s 0, 3s 0, 2s
10T_20T	1335 609 1953	799 656 611	3s 1s 4s	1s 1s 1s
10T_30T	5, 7s 2, 6s 9, 3s	3, 3s 2, 4s 2, 8s	10s 4s 19s	4s 3s 3s
10T_40T	8s 3s 20s	6, 2s 5s 5, 4s	14s 5s 32s	7s 6s 6s
50T_100T	60s 29s 86s	21s 21s 14s	89s 46s 136s	24s 23s 16s
100T_200T	214s 62s 329s	77s 67s 59s	—	87s 76s 63s
100T_400T	—	752s 588s 656s	—	826s 688s 673s
500T_1M	—	1862s 1572s 1651s	—	2120s 1800s 1652s

Tabelle 4.6: Durchschnittliche Ausführungszeiten der GRIPP-basierten Distanzalgorithmien unter Verwendung der fiktiven oberen Schranke \hat{s} . Durchschnittliche Zeiten basieren auf Distanzanfragen mit einem Ergebnis kleiner als \hat{s} . In Grau zum Vergleich die Zeiten der Breitensuchen ohne obere Schranke



(a) Graph 50 000_100 000



(b) Graph 100 000_200 000

Abbildung 4.3: Laufzeiten der knotenbasierten Breitensuche auf dem Originalgraphen in Abhängigkeit von der oberen Schranke der Shortcut Reachability

Kapitel 5

Related Work

Der wohl effizienteste Weg Distanzanfragen zur Laufzeit zu beantworten, ist das Vorberechnen aller Distanzen, d.h. der transitiven Hülle des Graphen. Obwohl es zur Berechnung der transitiven Hülle E^* effiziente Algorithmen gibt, deren Effizienz unabhängig von den Pfadlängen des zugrundeliegenden Graphen ist [AJ87], benötigt ein solcher Index $O(|V|^2)$ Platz.

Ein Ansatz mit dem Namen 2-Hop-Cover von Cohen et al. [ECZ03] verwendet eine kompaktere Darstellung der transitiven Hülle. Dabei wird für zwei Knoten $(u, v) \in E^*$ diese Information nicht direkt gespeichert, sondern in zwei Teilpfade aufgespalten $(u, w) \in E^*$ und $(w, v) \in E^*$. Wenn es nun n -viele Knoten u' mit $(u', w) \in E^*$ und m -viele v' Knoten mit $(w, v') \in E^*$ gibt, dann kann die Information der transitiven Hülle durch $n + m$ -viele Einträge im entsprechenden Index repräsentiert werden. Dabei hätte die traditionelle Darstellung $n * m + n + m$ -viele Einträge benötigt. Der Knoten w wird als *Hop* bezeichnet, und es sind zur Feststellung der Existenz eines Pfades zwei Anfragen notwendig. Beide Aspekte verleihen dem Verfahren seinen Namen. 2-Hop-Cover ermöglicht eine kompaktere Speicherung der transitiven Hülle durch die Minimierung der notwendigen Hops und benötigt im Worst-Case $O(|V| * |E|^{1/2})$. Leider ist die Berechnung eines minimalen Hop-Cover NP-vollständig [ECZ03] und benötigt als Eingabe die transitive Hülle. Das normale 2-Hop-Cover kann nur zur Beantwortung von Erreichbarkeitsanfragen verwendet werden. Um es auch für Distanzanfragen verwenden zu können, müssen neben der Speicherung der Distanzinformationen die Hops so gewählt werden, dass sie sich auf kürzesten Pfaden befinden. Erst dadurch wird gewährleistet, dass $dist(u, v) = dist(u, w) + dist(w, v)$. Diese Anforderung führt zu Verringerung der Freiheitsgrade bei der Auswahl der Hops und damit zu einer wesentlich schlechteren Komprimierung der Darstellung der transitiven Hülle bzw. zu einer höheren Anzahl der Hops. Dadurch verliert 2-Hop-Cover für Distanzanfragen an Attraktivität.

In einer von Schenkel et al. vorgestellten Variation wird das 2-Hop-Cover ohne vorhergehende Berechnung der vollständigen transitiven Hülle erzeugt [RSW05]. Dazu wird der Graph in Partitionen zerlegt, für die jeweils ein 2-Hop-Cover berechnet wird. Anschliessend werden die Teillösungen unter Beachtung der die Partitionen verbindenden Kanten zur einer Gesamtlösung zusammengeführt. Die Anzahl der Kanten zwischen den Partitionen und somit die Partitionierung selber hat dabei einen großen Einfluß auf die Kompaktheit des resultierenden 2-Hop-Covers.

Der nach seinem Erfinder benannte Dijkstra-Algorithmus berechnet kürzeste Pfade in kantengewichteten Graphen [THCR01] und entspricht für ungewichtete Graphen einer Breitensuche, weswegen er in reiner Form keine wirkliche Alternative darstellt.

Sanders und Schultes stellen in ihrer Arbeit *Highway Hierarchies Hasten* [SS05, SS07] eine interessante und sehr performante Methode zur Berechnung exakter kürzester Pfade in großen Straßennetzwerken vor. Es handelt sich dabei im Gegensatz zu den in dieser Studienarbeit betrachteten Graphen im Wesentlichen um planare und gewichtete Graphen. Bei *Highway Hierarchies Hasten* wird der Originalgraph im Rahmen der Vorverarbeitung um zusätzliche Knoten- und Kanteninformationen sowie um weitere Kanten, die Abkürzungen darstellen, erweitert. Ausgehend vom Originalgraphen werden iterativ parametrisierbare Operationen angewendet, die unter Berücksichtigung von Distanzinformationen schrittweise reduzierte Subgraphen berechnen. Bei jeder Reduktion werden Kanten, die nur bei lokalen Suchen Bestandteil kürzester Pfade sind, entfernt. Das Ergebnis einer Reduktion wird als *highway network* bezeichnet. Dieses wird anschließend unter Zusammenfassung von Linien (Pfad aus mehreren Knoten mit Grad zwei) und Einführung von substituierenden Abkürzungskanten zu einem *Kern* verdichtet, der nur Knoten mit einem Grad von mindestens drei besitzt. Jeder so erzeugte Kern wird als Hierarchie bzw. Level bezeichnet. Die Anzahl der Hierarchien ist konfigurierbar. Die Konstruktionsvorschrift garantiert, dass beim Fortsetzen einer Distanzsuche in einem höheren Level die Eigenschaft, sich auf einem kürzesten Weg zum Zielknoten der Anfrage zu befinden, nicht verloren geht. Diese Eigenschaft wird im Wesentlichen dadurch sichergestellt, dass eine Kante nur dann zu einem *highway network* einer höheren Stufe gezählt wird, wenn sie Bestandteil eines (hinreichend langen) kürzesten Pfades zwischen zwei Knoten der vorhergehenden Stufe ist. Die einzelnen Hierarchien werden nicht separat gespeichert, sondern der Originalgraph, der auch die 0-te Hierarchie darstellt mit entsprechenden Informationen angereichert. Dabei muss unter anderem die Zugehörigkeit einer Kante zu einer Hierarchiestufe bzw. Angabe der höchsten Stufe, zu der die Kante noch gehört, gespeichert werden. Als Grundlage für den Suchalgorithmus dient der bidirektionale Dijkstra-Algorithmus. Dabei wird die Suche gleichzeitig von beiden

Enden der Anfrage durchgeführt und beendet, wenn die Suchfronten zusammentreffen. Der Algorithmus für die Auswahl der zu relaxierenden Kanten wird um Restriktionen erweitert. Dadurch wird zunächst die lokale Umgebung des Ausgangsknotens (Nachbarschaftssuche) betrachtet und dann die Suche in einer höheren Hierarchiestufe fortgesetzt. Dort wird ebenfalls nur die lokale Umgebung des Eintrittsknotens durchsucht usw. In [SS07] beschreiben die Autoren detailliert das Grundkonzept und die Aspekte der Implementierung für gerichtete gewichtete Straßennetze. Die Vorverarbeitung nimmt auch für sehr große Graphen mit vielen Millionen Knoten nur wenige Stunden in Anspruch und die Suchzeiten sind im einstelligen Millisekundenbereich.

Möhring et al. haben in [MSS⁺05] verschiedene Methoden der Partitionierung des Graphen zur Beschleunigung des Dijkstra-Algorithmus auf gerichteten gewichteten Graphen mit relativ wenigen Kanten untersucht. Fast alle zur Partitionierung eingesetzte Verfahren setzen eine zweidimensionale Anordnung des Graphen voraus. Der Graph wird in Regionen partitioniert. Zu jeder Kante speichert ein Bitvektor, in welchen Regionen sich die Endknoten der mit dieser Kante beginnenden kürzesten Pfade befinden. Durch diese Information wird der Dijkstra-Algorithmus beim Aufbau der SSSPs gelenkt, indem er nur solche Kanten weiterverfolgt, deren Bitvektoren u.a. die Region des Zielknotens der Anfrage durch eine 1 anzeigen. Dies führt zur wesentlichen Einschränkung des Suchraums und damit zur Beschleunigung der Suche. Das korrekte Ergebnis durch dieses Vorgehen wird dadurch gewährleistet, dass der Teilpfad eines kürzesten Pfades auch ein kürzester Pfad ist. Die besten Ergebnisse erreichten Möhring et al. bei der Kombination des beschriebenen Kantenlabelings mit der *bidirektionalen* Dijkstra-Suche. Einen großen Einfluss auf die Dauer der Vorverarbeitung sowie die Effizienz der Suche hat die Wahl der Art der Partitionierung, die Anzahl der Partitionen und die mit der Partitionierung sich ergebende Anzahl der Kanten zwischen den Partitionen. Köhler et al. beschreiben in [KMS04] ein Verfahren zur Berechnung der Vektoren für die Kantenlabels. Um für alle Kanten des Graphen jeweils das Flag einer bestimmten Region zu berechnen, geht man von der Menge der Kanten aus, die die Grenze zu dieser Region kreuzen. Zu jeder dieser Kanten erzeugt man auf dem inversen Graphen mittels Dijkstra den Baum der kürzesten Pfade. Bei allen durch diese Pfade erreichten Knoten setzt man das Flag für die besagte Region. Es ist klar, dass bei günstiger Wahl der Regionen die Anzahl der die Regionengrenzen schneidenden Kanten und damit die Dauer der Vorverarbeitung minimiert werden kann. In [KMS04] hat sich METIS [Kar95, KK95] als ein gutes Verfahren zur Berechnung der Partitionen ergeben. Dieses setzt zudem eine zweidimensionale Anordnung des Graphen nicht voraus. Bei diesem als *multilevel graph partitioning* bezeichneten Verfahren wird der gewichtete Graph in k Partitionen so unterteilt, dass die

Summe der grenzübergreifenden Kanten möglichst klein ist (*edge-cut*). Dabei wird der Graph zuerst schrittweise verdichtet, dann partitioniert und anschliessend die Verdichtung wieder stufenweise rückgängig gemacht und dabei über Heuristiken der *edge-cut* weiter verbessert.

Kapitel 6

Zusammenfassung und Ausblick

In dieser Studienarbeit wurde die Suche nach kürzesten Distanzen auf gerichteten zusammenhängenden skalenfreien Graphen unter Verwendung von GRIPP untersucht. Diese von Leser und Trißl entwickelte Indexstruktur kann für eine sehr effiziente Beantwortung von Erreichbarkeitsanfragen auch für sehr große Graphen verwendet werden. Es wurde untersucht, inwieweit sich diese Indexstruktur auch für die Beantwortung von Distanzanfragen eignet. Zunächst sind vorhandene Implementierungen der knoten- und mengenbasierten Breitensuche auf GRIPP durch eine Einschränkung des Suchraums und die Umformulierung einiger PL/SQL-Programmstrukturen optimiert und für verschiedene synthetische Graphen mit den Breitensuchen auf den Originalgraphen verglichen worden. Dabei hat sich für Originalgraphen die knotenbasierte und für GRIPP die mengenbasierte Breitensuche als die jeweils performantere Implementierung herauskristallisiert. Die Breitensuche auf dem Originalgraphen vor allem für größere Graphen ist wesentlich schneller. Die Breitensuche auf GRIPP ist keine reine Breitensuche, da im Verlauf auch Knoten mit einer größeren Entfernung vom Startknoten der Anfrage als die tatsächlich gesuchte Distanz betrachtet werden. Der naheliegende Ansatz war, zu einer Distanzanfrage möglichst schnell eine gute obere Schranke für die Länge der gesuchten Distanz zu ermitteln und damit die Ineffizienz der Breitensuche auf GRIPP einzuschränken. Eingangs wurde ein Test durchgeführt, der zeigen sollte, welche maximale Verbesserung durch eine solche obere Schranke möglich ist. Das Ergebnis zeigte, dass auch mit der bestmöglichen oberen Schranke - der tatsächlich gesuchten kürzesten Distanz - die Breitensuche auf GRIPP langsamer als auf dem Originalgraphen ist. Eine gute obere Schranke kann zwar das Zeitverhalten der Breitensuche auf GRIPP beachtlich verbessern, es muss allerdings nach anderen Suchstrategien bzw. Optimierungen gesucht werden, um die GRIPP-Indexstruktur auch für Distanzanfragen konkurrenzfähig nutzen zu können. Für die Bestimmung einer oberen Schranke sind mehrere Verfahren unter-

sucht worden. Beim k Index-Ansatz werden zunächst für eine bestimmte Auswahl an Knoten kürzeste Distanzen mittels der knotenbasierten Breitensuche auf dem Originalgraphen vorberechnet. Zur Anfragezeit wird versucht, möglichst schnell einen Pfad vom Start- zum Zielknoten der Anfrage zu finden, auf dem sich auch Knoten des k Index befinden. Die durch diesen Ansatz gelieferten Schranken sind zwar um Größenordnungen kleiner als die Länge eines durch die Tiefensuche gefundenen Pfades, allerdings sind damit im wesentlichen drei Probleme verbunden. Die Schranke ist einerseits nicht ausreichend gut genug, um die sich anschließende Breitensuche signifikant zu beschleunigen. Andererseits ist nicht garantiert, dass überhaupt ein Pfad vom Start- zum Zielknoten der Anfrage gefunden wird, auf dem sich k Index-Knoten befinden. Bei den Tests ergab sich trotz eines (proportional) großen k Index nur eine Nutzquote von 84%. Schlussendlich dauert die Vorberechnung einer für eine gute Nutzquote notwendigen Anzahl von Distanzen und damit eines hinreichend großen k Index für große Graphen untragbar lange. Ein anderer adaptierter Ansatz für die Berechnung einer oberen Schranke stellte *Shortcut Reachability* dar, der die Tiefensuche auf GRIPP um die Protokollierung von Pfad- von Sprungknoten erweitert und damit eine Abkürzung für den durch die Tiefensuche gefundenen Pfad sucht. *Shortcut Reachability* benötigt zwar keine Vorberechnungen und liefert immer eine obere Schranke, allerdings ist die Schranke wesentlich schlechter als diejenige des k Index, wodurch die Optimierung der Breitensuche nur marginal ist. Weiterhin wurde ein Ansatz untersucht, bei dem in Anlehnung an die Diameterabschätzung für skalenfreie Graphen von Cohen und Havlin eine fiktive obere Schranke für kürzeste Distanzen in einem Graph entwickelt wurde. Für viele Anfragen ist dies eine gute Schranke und beschleunigt merklich die Distanzsuche. In den sehr wenigen anderen Fällen musste die Suche ohne eine Schranke erneut durchgeführt werden. Da es sich dabei um überdurchschnittlich lange Distanzen handelt, entsteht ein schwerwiegender Overhead. Dieser führte bei den Tests dazu, dass die durchschnittlichen Suchzeiten dieses Ansatzes sogar schlechter als diejenigen der Breitensuche ohne eine obere Schranke waren.

Eine Untersuchung der Korrelation der von der *Shortcut Reachability* berechneten oberen Schranken mit der Suchdauer der knotenbasierten Breitensuche auf dem Originalgraphen brachte ein negatives Ergebnis. Diese obere Schranke kann nicht als zuverlässige Abschätzung der Laufzeit der besagten Breitensuche verwendet werden.

Es blieb ungeklärt, ob GRIPP für Distanzanfragen auf eine Weise benutzt werden kann, die Operationen auf dem Originalgraphen überflüssig macht. Vorstellbar ist die Anreicherung des GRIPP-Baums mit weiteren Informationen, wodurch die Anzahl der zu betrachtenden Sprungknoten während der Breitensuche und damit die Größe des Suchraums merklich abnimmt. Zu überlegen wäre auch, ob ein mehrstufiges Vorgehen ähnlich

dem von Sanders und Schultes [SS05] auch für skalenfreie Graphen sinnvoll wäre. Dabei könnte zunächst um die Start- und Zielknoten der Anfrage herum jeweils eine Breiten-
suche bis zu einer bestimmten Schwellenwerttiefe durchgeführt werden. Wenn dadurch
kein Pfad und somit keine kürzeste Distanz gefunden wird, könnte in der zweiten Stu-
fe eine hierarchisch gröbere Struktur oder ein Komponentengraph durchsucht werden.
Eine Komponente des Komponentengraphen könnte ein Teilgraph sein, der nur über
einen Knoten an den restlichen Graphen angebunden ist und dieser Knoten die Stop-
knoteneigenschaften erfüllt. Sehr interessant wäre auch zu untersuchen, welchen Perfor-
manceeffekt eine bidirektionale Suche auf GRIPP hat und ob sich skalenfreie Graphen
so geschickt partitionieren lassen, dass man daraus einen Vorteil für die Distanzsuche
ziehen kann.

Literaturverzeichnis

- [AJ87] Agrawal, R.; Jagadish, H. V.: Direct algorithms for computing the transitive closure of database relations. In *Proceedings of the 13th International Conference on Very Large Data Bases (VLDB)*, S. 255–266. Morgan Kaufmann, 1987.
- [BR04] Bollobás, B.; Riordan, O.: The diameter of a scale-free random graph. *Combinatorica*, Band 24, Nr. 1, S. 5–34, 2004.
- [CH03] Cohen, R.; Havlin, S.: Scale-free networks are ultrasmall. *PHYS.REV.LETT*, Band 90, S. 058701, 2003.
- [ECZ03] E. Cohen, H. K., E. H.; Zwick, U.: Reachability and distance queries via 2-hop labels. *SIAM Journal on Computing*, Nr. 32(5), S. 1338–1355, 2003.
- [Kar95] Karypis, G.: Metis: Family of multilevel partitioning algorithms, 1995.
- [KK95] Karypis, G.; Kumar, V.: Multilevel graph partitioning schemes. In *Proceedings of the 1995 Intl. Symp. on Physical Design*, S. 113–112, 1995.
- [KMS04] Köhler, E.; Möhring, R. H.; Schilling, H.: Acceleration of shortest path computation. Article Nr. 42, Technische Universität Berlin, Fakultät II Mathematik und Naturwissenschaften, 2004.
- [MSS⁺05] Möhring, R. H.; Schilling, H.; Schütz, B.; Wagner, D.; Willhalm, T.: Partitioning graphs to speed up Dijkstra’s algorithm. In *WEA*, S. 189–202, 2005.
- [RSW05] R. Schenkel, A. T.; Weikum, G.: Efficient creation and incremental maintenance of the hopi index for complex xml document collections. In *Proceedings of the 21st International Conference on Data Engineering (ICDE)*, S. 360–371. IEEE Computer Society, 2005.

- [SS05] Sanders, P.; Schultes, D.: Highway hierarchies hasten exact shortest path queries. In *Proceedings 17th European Symposium on Algorithms (ESA)*, Springer LNCS, Band 3669, S. 568–579. Springer, 2005.
- [SS07] Sanders, P.; Schultes, D.: Engineering highway hierarchies. preliminary version, September 2007.
- [THCR01] T. H. Cormen, C. E. L.; Rivest, R. L.: *Introduction to Algorithms*. MIT Press, 2001.
- [TL06] Trißl, S.; Leser, U.: Gripp - indexing and querying graphs based on pre- and postorder numbering. Technischer Bericht, Humboldt Universität zu Berlin, 2006.
- [TL07] Trißl, S.; Leser, U.: Fast and practical indexing and querying of very large graphs. *SIGMOD*, 2007.

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und nur mit erlaubten Hilfsmitteln angefertigt habe.

Berlin, den 13. April 2008

Leonid Snurnikov

