Adaptive Suchverfahren

Dissertation

zur Erlangung des Grades
Doktor der Ingenieurwissenschaften (Dr.-Ing.)
der Technischen Fakultät
der Universität des Saarlandes

Frank Schulz

Saarbrücken
April 1999

Tag des Kolloquiums:	29. Oktober 1999
Dekan:	Prof. Dr. Wolfgang J. Paul
Vorsitzender:	Prof. Dr. Kurt Mehlhorn
Gutachter:	Prof. Dr. Dr. h. c. mult. Günter Hotz
	Prof. Dr. Susanne Albers

Inhaltsverzeichnis

Kurze Zusammenfassung **v**

Einleitung **1**

1. Adaptive Listenalgorithmen **7**

1.1. Einführung 7
1.1.1. Definitionen 8
1.1.2. Verschiedene Listenalgorithmen 9
1.2. Simple(k)- und Batched(k)-Verfahren 15
1.2.1. Unabhängige Zugriffe 15
1.2.2. Abhängige Zugriffe 16
1.3. Vorwärts-Regeln 22
1.4. Drei neue Familien von Listenalgorithmen 25
1.4.1. Definition der Algorithmen 26
1.4.2. Analyse der SORT-BY-RANK-Verfahren 28
1.4.3. Analyse der SORT-BY-DELAY-Verfahren 35
1.4.4. Analyse der SORT-BY-TIME-Verfahren 52
1.4.5. Empirische Ergebnisse 55
1.5. Konvergenzgeschwindigkeit 57
1.5.1. Batched(k)-MOVE-TO-FRONT 57
1.5.2. Sort-by-Delay(k) 64
1.6. Gewinnung statischer Listen 65
1.7. Randomisierte Listenalgorithmen 73
1.7.1. RMTF(p) 74
1.7.2. BIT 76
1.8. Das gewichtete Listenproblem 83
1.9. Bidirektionale Suche in Listen 89
1.9.1. Mittleres Verhalten der DMTF-Regel 89
1.9.2. Kompetitives Verhalten der DMTF-Regel 94
1.10. Suche in mehreren Listen 95

2. Anwendungen **101**

2.1. Datenkompression 101
2.1.1. Theoretische Ergebnisse 103
2.1.2. Empirische Ergebnisse 105

2.2. Konvexe Hüllen . . . 113
2.3. Lokalität . . . 118
2.3.1. Lokalität bei Markov-Ketten . . . 119
2.4. Adaptive Suchbäume . . . 122

3. Kodes und Suchbäume mit geometrischen Kosten **123**
3.1. Kodebäume für nicht geordnete Alphabete . . . 123
3.1.1. Schranken für die mittleren Kodewortkosten . . . 124
3.1.2. Verallgemeinerter Huffman-Algorithmus . . . 126
3.2. Suchbäume . . . 130
3.2.1. Alphabetische Kodes . . . 131
3.2.2. Eine untere Schranke für die mittlere Suchzeit . . . 133
3.2.3. Optimale Suchbäume . . . 134
3.3. Zufällig gewachsene Bäume . . . 135
3.4. Tries . . . 140

4. Zusammenfassung und Ausblick **145**
4.1. Zusammenfassung . . . 145
4.2. Ausblick . . . 146
4.2.1. Tradeoff zwischen den Kostenmodellen? . . . 146
4.2.2. Beschränkte Gegner . . . 147

A. Grundlagen über Markov-Ketten **149**
A.1. Wahrscheinlichkeitsverteilungen . . . 149
A.2. Markov-Ketten . . . 150
A.2.1. Definitionen und Grundlagen . . . 150
A.2.2. Beispiele . . . 154

Literaturverzeichnis **157**

Index **169**

Kurze Zusammenfassung

In der vorliegenden Arbeit untersuchen wir adaptive Suchalgorithmen. Diese Verfahren arbeiten *online* und versuchen, ohne Kenntnis zukünftiger Anfragen gute Antwortzeiten zu erzielen, indem sie ihre Suchstrategie dynamisch ändern. Wir analysieren bekannte und entwickeln neue Verfahren für sequentielle Suche unter kompetitiven und stochastischen Kostenmodellen. Ein Schwerpunkt der Arbeit liegt auf Markov-Ketten als Zugriffsmodell, die eine Verallgemeinerung unabhängiger Zugriffe darstellen und praktisch beobachtete Phänomene gut beschreiben können. Verschiedene Varianten des Problems wie randomisierte Algorithmen, gewichtete Zugriffskosten oder bidirektionale Suche werden ebenfalls betrachtet, außerdem vergleichen wir die Konvergenzgeschwindigkeit der einzelnen Verfahren.

Adaptive Verfahren ermöglichen es implizit, statistische Informationen über die Anfragefolgen zu sammeln, die wir zur Datenkompression einsetzen. Außerdem schlagen wir eine Verallgemeinerung des Kostenmaßes für Kodes und Suchbäume vor, bei dem die Vergleichskosten nicht konstant sind, sondern exponentiell mit der Tiefe wachsen, und analysieren die Auswirkungen dieses Modells.

Abstract

This thesis deals with adaptive search algorithms. These algorithms work *online* and strive to achieve fast response time without knowledge of future requests by modifying their search strategy dynamically. We analyse known and devise new algorithms for sequential search problems in the competitive and average case settings. Special emphasis is placed on Markovian request sequences, as they provide a generalization of the independent request model and are well suited to describe real data. We also consider several variations of the problem, like randomised algorithms, weighted access costs or bidirectional search, and we investigate the convergence speeds of the algorithms.

Adaptive algorithms implicitly collect statistical information on the request sequence that can be used for data compression. Furthermore we propose a generalization of the cost functions on codes and trees. While the comparison costs are constant in the standard model, they now increase exponentially with the depth. We analyse and discuss the impacts of this model.

Einleitung

Wer sucht, der findet.
Matthäus 7,8

Suchverfahren bilden einen zentralen Bestandteil vieler algorithmischer Problemlösungen. Eines der Hauptanwendungsgebiete liegt im Bereich des Speicherns von Informationen und des Zugriffs auf gespeicherte Daten. Wenn die Daten durch einen Schlüssel identifiziert werden, so sollen bei einer Nachfrage nach einem bestimmten Schlüssel die Daten möglichst effizient gefunden werden. Für diese Aufgabe der Implementierung eines Wörterbuchs sind zahlreiche klassische Datenstrukturen bekannt, die entweder linear suchen oder durch Ausnutzen einer Ordnung auf den Schlüsseln in einer Baumstruktur arbeiten. Neben der direkten Anwendung von Suchverfahren in Wörterbüchern treten Suchprozesse in vielen anderen Bereichen auf, zum Beispiel bei geometrischen Problemstellungen (Punktlokalisierung) oder Textverarbeitung (Mustererkennung).

Der Schwerpunkt dieser Arbeit liegt auf *adaptiven* Datenstrukturen und Algorithmen. Damit sind Verfahren gemeint, die sich an die Suchanfragen anpassen können und dadurch ihre Suchstrategie optimieren. Wir betrachten adaptive Verfahren aus dem Blickfeld der Online-Algorithmen und unter dem Gesichtspunkt der statistischen Informationstheorie.

Kompetitive Analyse

Ein Algorithmus, der eine Folge von Anfragen bearbeitet, arbeitet *online*, wenn er jede Anfrage bedient, ohne die zukünftigen Anfragen zu kennen. Im Gegensatz dazu stehen *Offline-Algorithmen*, die eine gesamte Eingabefolge im voraus sehen dürfen und dieses Wissen bei der Bearbeitung der einzelnen Anfragen ausnutzen können. In der kompetitiven Analyse vergleicht man die Kosten eines Online-Algorithmus mit den Kosten eines optimalen Offline-Algorithmus. Wenn für alle Anfragefolgen das Verhältnis dieser Kosten durch eine Konstante c beschränkt ist, so erhält man damit die Garantie, daß der Online-Algorithmus höchstens das c-fache der optimalen Kosten erzeugt, auch wenn er bei der Bearbeitung der Anfragen die Zukunft nicht kennt. Die Schranke c muß für alle Anfragefolgen gelten, und für die Intuition ist oft die Vorstellung hilfreich, daß die Anfragen von einem Gegner erzeugt werden, der beliebig schwierige Eingaben produzieren kann, sie dann aber auch selbst offline bearbeiten muß.

Das Gebiet der kompetitiven Analyse von Online-Algorithmen wurde 1985 von *D. Sleator* und *R. Tarjan* mit der Untersuchung des MOVE-TO-FRONT-Algorithmus begründet [ST85] und hat seitdem eine rasante Entwicklung genommen, vgl. [BE98, FW98] für einen aktuellen Überblick.

Mittlere Analyse

Wenn man ein stochastisches Modell für die Suchanfragen annimmt, werden dadurch die Suchkosten oder andere charakteristische Werte des untersuchten Algorithmus zu Zufallsvariablen. Dann ist man an Aussagen über diese Zufallsvariablen interessiert, in erster Linie über den Erwartungswert der Suchkosten.

Oft wird bei probabilistischen Untersuchungen die Annahme gemacht, daß die Elemente der Eingabefolgen unabhängig voneinander sind (stochastische Quelle ohne Gedächtnis, Bernoulli-Modell) und sogar gleichverteilt sind (symmetrisches Bernoulli-Modell). Wenn man aber Abhängigkeiten zuläßt, erhält man nicht nur eine Verallgemeinerung der Theorie, sondern auch Resultate, die die in der Praxis beobachteten Verhalten der Algorithmen wesentlich besser beschreiben können. *Markov-Ketten* bieten ein solches Modell von stochastischen Quellen mit Gedächtnis. Die Abhängigkeiten in der Eingabefolge bestehen hierbei nur lokal, indem die Verteilung eines Eingabeelementes von dem direkt vorausgehenden Element abhängt, aber nicht von den weiter zurückliegenden Vorkommen. Da solche Phänomene aber in der Praxis erscheinen, bzw. eine hinreichend genaue Approximation praktisch auftretender Eingabefolgen ermöglichen, ist die Anwendbarkeit von Markov-Ketten gerechtfertigt.

Die Betrachtung von Markov-Quellen im Zusammenhang mit Suchbäumen wurde erstmals von *G. Hotz* durchgeführt [H93]. Es hat sich gezeigt, daß die Kodierungssätze zur Quellenkodierung der von *C. Shannon* begründeten statistischen Informationstheorie [S48] übertragen werden können. Mit dem Konzept der Suchgraphen kann eine mittlere Suchzeit realisiert werden, die durch die Entropie der Markov-Kette der Zugriffe bestimmt ist.

Vergleich der Analysemodelle

Bis Mitte der 80er Jahre wurden adaptive Suchalgorithmen nur unter dem probabilistischen Analysemodell betrachtet. Die dann eingeführte kompetitive Analyse erscheint zunächst überlegen, da sie nicht nur Aussagen über Mittelwerte macht, sondern Leistungsgarantien für die einzelnen Algorithmen auch im schlechtesten Fall gibt. Es zeigt sich aber, daß es viele Verfahren gibt, die sich zwar unterscheiden, aber im kompetitiven Modell gleich bewertet werden. Die mittlere Analyse kann wesentlich genauere Aussagen über das Verhalten der Algorithmen geben.

Aus diesem Grund betrachten wir in dieser Arbeit beide Analysemodelle parallel. Einerseits werden die Untersuchungen bekannter Verfahren auf Markov-Modelle erweitert, andererseits entwickeln wir eine Reihe neuer Algorithmen, die wir – soweit möglich – in beiden Kostenmaßen analysieren.

Unter anderem geben wir zwei neue überabzählbare Familien von kompetitiv-optimalen Listenalgorithmen an, die die umfassendsten Klassen solcher Verfahren darstellen. Eine weitere Familie von Verfahren, die wir einführen, ermöglicht einen Trade-off zwischen den Kosten im kompetitiven Modell und im probabilistischen Modell.

Randomisierte Algorithmen dürfen Zufallsbits benutzen. Dadurch haben randomisierte Online-Algorithmen die Möglichkeit, ihre Strategie gegenüber dem Offline-

Gegner zu verschleiern, und bessere Ergebnisse im kompetitiven Modell zu erzielen. Wir untersuchen, wieviel Zufall dazu nötig ist und geben eine derandomisierte Variante eines bekannten Verfahrens an, die nur noch wenige zufällige Bits verwendet.

Anwendungen

Indem adaptive Suchverfahren sich der Anfragefolge anpassen, optimieren sie nicht nur ihre Suchstrategie, sondern lernen gleichzeitig statistische Informationen über die Quelle der Anfragen. Dieser implizite Lernvorgang kann ausgenutzt werden. Bei stochastischen, unabhängigen Anfragen versucht ein Listenalgorithmus, seine Elemente in die Reihenfolge absteigender Zugriffshäufigkeiten zu bringen. Diese Eigenschaft kann für eine effiziente Datenkompression genutzt werden.

Unter dem Gesichtspunkt der Informationstheorie stellt sich die Frage, wie gut man eine anstehende Suchaufgabe lösen kann, wenn man statistische Informationen über die Anfragequelle kennt oder sie mit einem adaptiven Verfahren erlernt. Von *G. Hotz* stammt das Konzept, die Ausführungspfade eines Programms als präfixfreie Kodes zu betrachten [H97]. Bei bekannten Wahrscheinlichkeiten für die einzelnen Ausführungspfade kann mit Hilfe der Kodierungssätze der Informationstheorie das Programm so transformiert werden, daß die mittlere Laufzeit minimiert wird und durch die Entropie der Verteilung abgeschätzt werden kann. Wir untersuchen diese Ideen an der von *G. Hotz* vorgeschlagenen effizienten Berechnung der konvexen Hülle von Punkten in der Ebene.

Bei Eingabefolgen mit abhängigen Symbolen kann man die Kosten von adaptiven Verfahren als Maß für die Abhängigkeiten auffassen. Dadurch erhalten wir eine Möglichkeit, das Phänomen der Lokalität in Datenströmen festzustellen und zu beschreiben.

Verallgemeinerte Kostenmaße

Eine wichtige Verallgemeinerung von Suchverfahren ist es, variable Kosten für die einzelnen Suchschritte zu betrachten, anstatt Einheitskosten anzunehmen. Dabei kann man entweder annehmen, daß die Kosten eines Vergleichs von dem Element selbst abhängen, oder daß sie durch die aktuelle Position des Elements in der Datenstruktur gegeben sind. Wir untersuchen adaptive Listenalgorithmen für den ersten Fall, d.h. Vergleichskosten $w_x > 0$ für ein Element x. Den zweiten Fall behandeln wir am Beispiel von Kodes und Suchbäumen, bei denen die Kosten für einen Vergleich exponentiell mit der Tiefe im Baum wachsen. Dadurch kann die Suche in hierarchischen Strukturen modelliert werden, bei denen die Vergleichskosten exponentiell ansteigen, je mehr man in die Struktur hinabsteigt. Zur Analyse solcher Kostenmodelle erweist sich ein verallgemeinerter Entropie-Begriff, die *Rényi-Entropie*, als sinnvoll [R61]. Obwohl dieser Begriff und entsprechende Kodierungstheoreme schon seit den 60er Jahren bekannt sind, gab es bisher noch keine direkte Anwendung auf Suchbäume und Suchalgorithmen.

Gliederung und Resultate

In **Kapitel 1**, dem Hauptkapitel der Arbeit, werden adaptive Listenalgorithmen ausgiebig analysiert. Nach einer Einführung in die Problemstellung und die verwendeten Analysemodelle untersuchen wir in *Abschnitt 1.2* und *1.3* bekannte Verfahren bei Markov-Zugriffen. Im Falle eines trägen Nachfrageverhaltens beobachten wir die interessante Eigenschaft, daß sich bekannte Hierarchien der Simple(k)- bzw. Batched(k)-MOVE-TO-FRONT-Verfahren oder der Vorwärtsregeln gerade umkehren, verglichen mit stochastisch unabhängigen Zugriffen.

Abschnitt 1.4 präsentiert drei neue Familien von Listenalgorithmen. Sie beruhen auf der von uns eingeführten Idee, *Rangfunktionen* zu verwenden, die die Ordnung der Liste definieren. Dadurch werden die bekannten Algorithmen MOVE-TO-FRONT, TIMESTAMP [A98] und ABSTEIGENDE-HÄUFIGKEIT auf verschiedene Weise verallgemeinert: Die SORT-BY-RANK(α)-Strategien interpolieren das Verhalten von MTF und TS, während SORT-BY-DELAY(k) über TS hinaus extrapoliert. Die SORT-BY-TIME(q)-Algorithmen erlauben einen stetigen Übergang zwischen MTF und ABSTEIGENDE-HÄUFIGKEIT.
Die SORT-BY-RANK(α)-Verfahren weisen für alle $0 \leq \alpha \leq 1$ ein optimales kompetitives Verhältnis von 2 auf, ebenso wie die SORT-BY-TIME(q)-Regeln für $0 \leq q \leq \sqrt{1/2}$. Diese Algorithmen stellen damit die umfassendsten zur Zeit bekannten Klassen von kompetitiv-optimalen Verfahren dar. Die SORT-BY-DELAY(k)-Regeln sind k-kompetitiv für $k \geq 2$. Mit $k \to \infty$ arbeiten sie auf gedächtnislosen stochastischen Zugriffen beliebig gut. Dies ist die einzige bekannte Familie von Verfahren mit dieser Eigenschaft, wobei jeder Algorithmus einen konstanten kompetitiven Faktor aufweist. Satz 1.24 schätzt ab, wie schnell die asymptotischen Kosten von SBD(k) mit k gegen die optimalen asymptotischen Kosten konvergieren. Wir verallgemeinern zunächst einen bekannten Ansatz, der auf der Ungleichung von Hilbert basiert [CHS88], können aber mit einem anderen Vorgehen noch bessere Schranken zeigen. Eine allgemeine untere Schranke, die die minimalen erzielbaren Kosten eines Listenalgorithmus zu der Größe seines Gedächtnisses in Beziehung setzt, wird in Satz 1.27 entwickelt. In Satz 1.28 untersuchen wir SBD(k) ebenfalls bei Markov-Zugriffen, so daß eine umfassende Analyse dieser neuen Verfahren hiermit vorgestellt werden kann.

Der *Abschnitt 1.5* stellt eine Analyse der Konvergenzgeschwindigkeit der Batched(k)-MTF-Verfahren und der SBD(k)-Regeln vor. Wir zeigen verteilungsabhängige und verteilungsunabhängige Schranken für die erwarteten Kosten nach t Zugriffen (Sätze 1.33 und 1.36) und verallgemeinern die Formel von *J. Fill* für die Übergangswahrscheinlichkeiten in t Schritten auf die Batched(k)-MTF-Regeln (Satz 1.39).

Sektion 1.6 behandelt absorbierende Verfahren, die nach einer anfänglichen Lernphase in einer bestimmten Listenkonfiguration „hängenbleiben" und diese dann ausschließlich benutzen. Wir analysieren die Qualität der erzielten statischen Konfigurationen und diskutieren die Wartezeit, bis Absorption eintritt.

In *Abschnitt 1.7* untersuchen wir randomisierte Algorithmen im kompetitiven und stochastischen Modell. Wir zeigen unter anderem, wie man den BIT-Algorithmus [RWS94] auf einer n-elementigen Liste mit nur $\lceil \log(n+1) \rceil$ anstatt n zufälligen Bits initialisieren kann, so daß das kompetitive Verhältnis weiterhin 1.75 beträgt. Diese derandomisierte Variante von BIT stellt damit den Listenalgorithmus mit dem kleinsten Verbrauch der Ressource Zufall dar, dessen kompetitiver Faktor die untere Schranke 2 für den kompetitiven Faktor deterministischer Verfahren unterbietet.

Die restlichen drei Abschnitte von Kapitel 1 behandeln Varianten von sequentiellen Suchverfahren. In *Sektion 1.8* diskutieren wir das gewichtete Listenproblem, bei dem das Lesen von Element x Kosten von w_x verursacht. Satz 1.55 enthält eine untere Schranke für deterministische Algorithmen im kompetitiven Modell. Außerdem untersuchen wir den randomisierten BIT-Algorithmus und präsentieren eine randomisierte Variante von Batched(k)-MTF, die im gewichteten Listenproblem auf gedächtnislosen stochastischen Quellen beliebig gut arbeitet.

Abschnitt 1.9 behandelt bidirektionale lineare Suche, bei der die Suche von jedem Ende der Liste aus starten kann. Für dieses Problem geben wir Ergebnisse im kompetitiven Modell und diskutieren das Verhalten von Algorithmen bei unabhängigen und bei Markov-Zugriffen.

In *Abschnitt 1.10* untersuchen wir die Aufgabenstellung, die Daten auf eine feste Anzahl von Listen zu verteilen und dynamisch umzusortieren, um die erwartete Zugriffszeit zu minimieren. Wir geben deterministische und randomisierte Verfahren für unabhängige Zugriffe und eine untere Schranke im kompetitiven Modell an.

Kapitel 2 behandelt Anwendungen von adaptiven Suchverfahren. In *Abschnitt 2.1* nutzen wir aus, daß ein Listenalgorithmus durch das Umsortieren der Liste implizit die Ordnung der relativen Häufigkeiten approximiert. Somit liefert er ein statistisches Modell der Zugriffe, das für Datenkompression genutzt werden kann. Wir analysieren die in Abschnitt 1.4 eingeführten SBD(k)-Regeln hinsichtlich ihrer Fähigkeit zur Datenkompression und fassen die Ergebnisse von Experimenten auf praktisch vorkommenden Daten, die dem *Calgary/Canterbury Compression Corpus* entnommen sind, zusammen. Da der Parameter k steuert, wie schnell der Listenalgorithmus reagiert, hängt seine geeignete Wahl von der Zugriffsfolge ab. Um einen guten Wert für die SBD(k)-Verfahren zu finden, untersuchen wir verschiedene Methoden, den Parameter online zu schätzen. Außerdem diskutieren wir in diesem Abschnitt die SBR(α)-Regeln in Zusammenhang mit der Burrows-Wheeler-Transformation für die Textkompression.

Sektion 2.2 wendet sich geometrischen Anwendungen zu. Wir analysieren einen Algorithmus von *G. Hotz* zur effizienten Berechnung der konvexen Hülle bei bestimmten Verteilungsannahmen über die Punkte. Anschließend erweitern wir das Verfahren, so daß es dynamisch arbeitet (d.h. ohne die Anzahl n der Punkte im voraus zu kennen).

In *Abschnitt 2.3* diskutieren wir das Phänomen der Lokalität. Insbesondere interessieren wir uns für Kriterien, den Grad der Lokalität in Markov-Ketten zu bestimmen. Dazu verfolgen wir einen pragmatischen Ansatz und betrachten die Kosten von adaptiven Suchverfahren als Maß für die Lokalität. *Abschnitt 2.4* geht kurz auf adaptive Suchbäume ein und grenzt selbst-organisierende Listen und Bäume voneinander ab.

In **Kapitel 3** definieren und untersuchen wir ein neues Kostenmaß für Kodes und Suchbäume. Die Kosten für das Durchlaufen einer Kante bzw. für einen Vergleich in einem Knoten sind nicht wie bisher konstant, sondern wachsen exponentiell mit der Tiefe. Damit erhalten wir eine Verallgemeinerung des bekannten Modells mit Einheitskosten, das als Grenzfall in unserer Konstruktion enthalten ist. Mit Hilfe der verallgemeinerten Kostenfunktion können beispielsweise Suchprozesse in hierarchischen Strukturen dargestellt werden, bei denen mit zunehmender Suchtiefe die Komplexität und damit die Kosten für einen weiteren Suchschritt ansteigen.

Wir verallgemeinern zunächst den Huffman-Algorithmus, um optimale Kodes zu berechnen. Dann betrachten wir ordnungserhaltende Kodes und Suchbäume, und geben Schranken für die erwarteten Kosten an. In *Abschnitt 3.3* untersuchen wir zufällig gewachsene Bäume, die durch das Einfügen von Elementen in zufälliger Reihenfolge entstehen, und berechnen die mittlere externe Pfadlänge bzw. die erwartete Tiefe eines Knotens in dem verallgemeinerten Kostenmaß. *Abschnitt 3.4* schließlich betrachtet zufällige Tries (digitale Suchbäume) unter dem neuen Kostenmaß und leitet die erwartete externe Pfadlänge im Bernoulli- und Markov-Modell für die Schlüssel her.

Kapitel 4 enthält eine Zusammenfassung der vorliegenden Arbeit und gibt einen Ausblick auf Erweiterungsmöglichkeiten und offene Fragen.

Teile der Arbeit wurden in [SS96, S98] veröffentlicht.

Danksagung

An dieser Stelle möchte ich mich herzlich bei Herrn Prof. Dr. Günter Hotz für die Vergabe des interessanten und fruchtbaren Themas bedanken, für den Freiraum bei der Bearbeitung und für viele entscheidende Ideen und Anregungen. Herr Prof. Hotz hat meine wissenschaftliche Ausbildung vom ersten Studientag an geprägt [H90] und durch zahlreiche hilfreiche Gespräche das Gelingen dieser Arbeit ermöglicht.

Wertvolle Ratschläge und Diskussionen zu einzelnen Themen verdanke ich insbesondere den Herren Dr. Thomas Chadzelek, Martin Nest und Dr. Elmar Schömer. Für das sorgfältige Korrekturlesen der Arbeit möchte ich mich bei den Herren Dr. Elmar Schömer, Dr. Thomas Chadzelek und Jung-Bae Son herzlich bedanken; für eventuell noch verbleibende Fehler bin selbstverständlich ich alleine verantwortlich.

Frau Katrin Klose und allen Kollegen an diesem Lehrstuhl danke ich für das hervorragende Arbeitsklima. Insbesondere aber danke ich meiner Familie, die mir während meines Studiums ein großer Rückhalt war und mich stets tatkräftig unterstützte.

1. Adaptive Listenalgorithmen

In diesem Kapitel werden adaptive Methoden für sequentielle Suchprobleme behandelt. Diese Verfahren zielen darauf ab, die Testreihenfolge möglichst gut den Nachfragen anzupassen, um die Suchzeiten zu reduzieren. Gleichzeitig erlernen sie dabei statistische Informationen über die Quelle der Nachfragen.

Nach einer Einführung in die Problemstellung und Literaturübersicht werden bekannte Verfahren bei Markov-Zugriffsfolgen untersucht. Anschließend stellen wir drei neue Familien von Listenalgorithmen vor und analysieren ihr kompetitives und mittleres Verhalten. Wir betrachten die Konvergenzgeschwindigkeit von ergodischen und absorbierenden Listenalgorithmen, und analysieren verschiedene randomisierte Verfahren. Abschließend werden Varianten des Problems behandelt, wie etwa verallgemeinerte Zugriffskosten, bidirektionale Suche und Suche in Feldern.

1.1. Einführung

Das Listenproblem oder *list update problem* wurde erstmals 1963 von *M. L. Tsetlin* [T63] behandelt, weshalb es manchmal auch als *Tsetlin library problem* bezeichnet wird. Seit 1965 wird es im Kontext der informationstechnischen Anwendungen betrachtet [M65].

Von *Tsetlin* wurde die Nach-Vorne-Schieben-Regel oder MOVE-TO-FRONT (MTF) untersucht, die er an folgendem Beispiel darstellt: in einer Bibliothek gibt es ein einziges langes Bücherregal. Jedesmal wenn ein Buch nachgefragt wird, läuft der Bibliothekar an dem Regal entlang, bis er das gewünschte Buch gefunden hat. Er entnimmt es, und es entsteht eine Lücke. Bis zur Rückgabe dieses Buches kann kein weiteres Buch ausgeliehen werden. Wenn das Buch zurückgegeben wird, setzt der Bibliothekar es an den Anfang des Regals und schiebt die Bücher nach hinten, bis die Lücke wieder verschwindet.

Die Nach-Vorne-Schieben-Regel stellt den wichtigsten Listenalgorithmus dar, da sie sehr einfach arbeitet und damit auch relativ leicht zu analysieren ist. Je nach Fragestellung oder verwendetem Kostenmaß können aber andere Listenalgorithmen angegeben werden, die sich besser verhalten.

1.1.1. Definitionen

Die Menge der Listenelemente sei ohne Beschränkung der Allgemeinheit stets

$$\mathcal{A} = \{1, 2, \dots, n\},$$

da bei sequentieller Suche keine Ordnung auf den Elementen benötigt wird. Eine bestimmte Sortierung der Liste stellen wir dar durch einen Vektor

$$\langle x_1, x_2, \dots, x_n \rangle,$$

der eine Permutation der Elemente aus $\mathcal{A}$ beschreibt. Dabei soll sich der Listenkopf stets am linken Rand befinden, und die Suche nach einem Listenelement von links nach rechts, also in der Reihenfolge $x_1, x_2, \dots$ ablaufen.

Der Aufwand für das anschließende Umsortieren durch den Listenalgorithmus wird vernachlässigt, wenn das nachgefragte Element ein Stück zum Listenkopf bewegt wird (*freier Austausch, free exchange*). Die Kosten sind nämlich proportional zu den vorher entstandenen Suchkosten. Jede andere Transposition zweier benachbarter Listenelemente hat Einheitskosten und wird als *bezahlter Austausch, paid exchange* bezeichnet.

Die Kosten für einen Zugriff auf das Element x_k seien gleich der Anzahl der dafür benötigten Vergleiche, also k, im sogenannten *Standardmodell* oder *full cost model*. Manchmal ist es für die Analyse einfacher, das sogenannte $(i-1)$-*Kostenmodell* zu betrachten, bei dem man für einen Zugriff auf x_k nur die Kosten $k-1$ bezahlen muß. Gelegentlich wird auch das P^d-*Kostenmodell* analysiert, bei dem die Vergleiche beim Zugriff Einheitskosten erzeugen, aber das anschließende Umsortieren mit bezahlten Austauschen die Kosten d verursacht [RWS94].

Weitere Varianten sind denkbar: die Kosten für den Zugriff auf ein Element auf Position ℓ können durch eine Funktion $f(\ell)$ gegeben sein. Im Standardmodell ist $f(\ell) = \ell$. Es ist bekannt, daß verschiedene Aussagen des Standardmodells auch gelten, wenn man voraussetzt, daß f konvex (rechtsgekrümmt) ist [ST85].

Es ist auch möglich, für das Lesen der einzelnen Listenelemente verschiedene Kosten anzusetzen. Seien w_x die Kosten für das Lesen von x. Bei einer Listenanordnung $\langle 1, 2, \dots, n \rangle$ entstehen dann Kosten w_1 beim Zugriff auf 1, $w_1 + w_2$ beim Zugriff auf 2, usw. Damit kann man beispielsweise das sequentielle Lesen verschieden langer Datensätze von einem Band modellieren. Wir werden dieses Modell in Abschnitt 1.8 untersuchen.

Wenn die Menge $\mathcal{A}$ der Listenelemente stets gleich bleibt, spricht man vom statischen Listenproblem, das wir im folgenden zugrunde legen. Wenn man das dynamische Listenproblem betrachtet, sind neben den Zugriffen auch Einfügen und Löschen von Elementen möglich. Eine Einfügeoperation besteht aus einem Durchlaufen der Liste, um sicherzustellen, daß das Element noch nicht enthalten ist, und dem Anhängen des neuen Elements an das Listenende. Sie verursacht also Kosten in Höhe der aktuellen Listenlänge (im Standardmodell). Das Löschen eines Elements besteht aus der Suche des Elements und dem anschließenden Entfernen. Die Kosten sind gleich den Kosten eines Zugriffs auf das Element.

Analysemodelle

Im Verlauf der gesamten Arbeit wenden wir zwei Analysemodelle für die betrachteten Verfahren an: mittlere Analyse und kompetitive Analyse.

Bei der mittleren Analyse der Kosten legt man ein Wahrscheinlichkeitsmodell für die Anfragefolgen zugrunde. Dies kann eine gedächtnislose Quelle sein, die eine Folge von unabhängigen, identisch verteilten Zufallsvariablen produziert, oder eine Quelle mit Gedächtnis, beispielsweise eine Markov-Kette erster Ordnung, bei der die Verteilung einer Zufallsvariablen von der Realisierung der direkt vorausgehenden Variablen abhängt. Von Hauptinteresse sind hier die asymptotischen erwarteten Kosten $E_{A}(\mathbf{p})$ eines Algorithmus A unter dem Modell $\mathbf{p}$ und die mittleren Kosten eines optimalen Verfahrens $E_{OPT}(\mathbf{p})$, das das Modell $\mathbf{p}$ kennt, und der Vergleich dieser Kosten.

Sei $\sigma = \sigma_1, \ldots, \sigma_m \in \mathcal{A}^m$ eine beliebige Anfragefolge. Ein *Online-Algorithmus* muß jede Anfrage σ_t bearbeiten, ohne Kenntnis von zukünftigen Zugriffen $\sigma_{t+1}, \ldots, \sigma_m$ zu haben. Ein *Offline-Algorithmus* darf dagegen die gesamte Anfragefolge σ sehen und seine Entscheidungen darauf aufbauen. Bei der kompetitiven Analyse vergleicht man die Kosten $C_A(\sigma)$ eines Online-Algorithmus A mit den Kosten von optimalen Offline-Algorithmen $C_{OPT}(\sigma)$ bei beliebigen Anfragefolgen σ. Falls gilt

$$\exists a\ \forall \sigma : \quad C_A(\sigma) \leq c \cdot C_{OPT}(\sigma) + a ,$$

so sagt man: A ist c-kompetitiv. Der kompetitive Faktor eines Online-Algorithmus A ist das Infimum über alle c, für die der Algorithmus c-kompetitiv ist.

Der kompetitive Faktor beschränkt das Verhältnis von Online-Kosten zu optimalen Offline-Kosten und gibt somit eine Leistungsgarantie des Online-Algorithmus. Man beachte, daß der Faktor für Listen und Zugriffsfolgen beliebiger Länge gelten muß.

Lemma 1.1. *(Karp/Raghavan 1990 [KR90])*
Wenn ein deterministischer Listenalgorithmus c-kompetitiv ist, so gilt $c \geq 2$.

In einem verallgemeinerten Kostenmodell verwenden wir die gleiche Beweismethode, siehe Satz 1.55, und erhalten dieses Lemma als Spezialfall.

Obere Schranken für den kompetitiven Faktor im $(i-1)$-Kostenmodell sind auch obere Schranken im Standardmodell. Denn seien $m = |\sigma|$ die Länge der Anfragefolge und $\overline{C}_A(\sigma)$ die Kosten im $(i-1)$-Modell. Mit $C_A(\sigma) = \overline{C}_A(\sigma) + m$ folgt

$$\frac{C_A(\sigma)}{C_{OPT}(\sigma)} = \frac{\overline{C}_A(\sigma) + m}{\overline{C}_{OPT}(\sigma) + m} \leq \frac{\overline{C}_A(\sigma)}{\overline{C}_{OPT}(\sigma)} ,$$

da stets $C_{OPT}(\sigma) \leq C_A(\sigma)$. Umgekehrt ist eine untere Schranke im Standardmodell auch eine untere Schranke im $(i-1)$-Kostenmodell.

1.1.2. Verschiedene Listenalgorithmen

Je nachdem, ob Listenalgorithmen aus vergangenen Zugriffen lernen dürfen oder nicht, unterscheidet man gedächtnislose Verfahren (*memoryless heuristics*) und solche mit Gedächtnis. Außerdem können Listenalgorithmen deterministisch oder randomisiert sein, wobei sie durch letzteres ihre Schwachpunkte gegenüber Gegnern verschleiern können.

Gedächtnislose deterministische Algorithmen

Die oben erwähnte Regel für das Bücherregal lautet:

> **Nach-Vorne-Schieben-Regel**, MOVE-TO-FRONT, MTF: Wenn auf ein Listenelement x zugegriffen wird, dann setze x an den Listenkopf. Wenn es sich schon dort befindet, so tue nichts.

Eine andere bekannte Regel, die eine langsamere Änderung der Liste erreicht, ist die

> **Vertauschen-Regel**, TRANSPOSTION RULE, TR: Wenn auf ein Listenelement x zugegriffen wird, das noch nicht am Listenkopf steht, dann vertausche es mit seinem Vorgänger in der Liste. Wenn x bereits am Listenkopf steht, so tue nichts.

Allgemein läßt sich ein gedächtnisloses Verfahren durch eine Folge von n Permutationen $(\tau_1, \dots, \tau_n)$ über der Menge $\{1, \dots, n\}$ beschreiben. Wenn sich die Liste in Zustand $\sigma = \langle x_1, \dots, x_n \rangle$ befindet und auf das Listenelement x_j zugegriffen wird, dann wird die Liste umsortiert zu $\sigma \circ \tau_j = \langle x_{\tau_j(1)}, x_{\tau_j(2)}, \dots, x_{\tau_j(n)} \rangle$. Ein gedächtnisloses Verfahren trifft seine Entscheidung über das Umsortieren der Liste nur aufgrund der Position, auf der das gesuchte Element gefunden wurde. Auf diese Weise wird die Nach-Vorne-Schieben-Regel definiert als

$$\tau_j(k) = \left\{ \begin{array}{ccl} j & : & \text{falls } k = 1 \\ k-1 & : & \text{falls } 1 < k \le j \\ k & : & \text{falls } j < k \le n \end{array} \right\} \text{ für } j = 1, \dots, n.$$

Die Vertauschen-Regel kann folgendermaßen beschrieben werden:

$$\tau_j(k) = \left\{ \begin{array}{ccl} j & : & \text{falls } k = j-1 \\ j-1 & : & \text{falls } 1 < k = j \\ 1 & : & \text{falls } 1 = k = j \\ k & : & \text{falls } 1 \le k < j-1 \text{ oder } j < k \le n \end{array} \right\} \text{ für } j = 1, \dots, n.$$

Allgemeiner wird die Klasse der **Vorwärts-Regeln** (MOVE-FORWARD RULES) definiert durch n Zahlen $1 = m_1, m_2, \dots, m_n$ mit $m_j < j$ und $m_{j-1} \le m_j$ für $j \ge 2$. Wenn auf das Element auf Position j zugegriffen wird, dann wird es auf Position m_j der Liste gesetzt:

$$\tau_j(k) = \left\{ \begin{array}{ccl} j & : & \text{falls } k = m_j \\ k-1 & : & \text{falls } m_j < k \le j \\ k & : & \text{sonst} \end{array} \right\} \text{ für } j = 1, \dots, n.$$

Ein Beispiel ist die Familie der k-Plätze-Vorwärts-Regeln für $k = 1, \dots, n-1$.

k**-Plätze-Vorwärts-Regel**, MOVE-UP(k), MU(k) [R76]: Wenn auf ein Listenelement x zugegriffen wird, das auf einer der k ersten Positionen steht, so setze x an den Listenkopf. Ansonsten schiebe x um k Positionen nach vorne.

Für $k = 1$ erhält man die Vertauschen-Regel und für $k = n-1$ die Nach-Vorne-Schieben-Regel. Eine andere Familie von Verfahren, die den Betrag des Nach-Vorne-Schiebens von der Position des Elementes abhängig macht, ist

Fraktion-d**-Nach-Vorne-Regel**, MOVE-FRACTION(d), MF(d) [ST85]: Wenn auf ein Listenelement x auf Position ℓ zugegriffen wurde, dann schiebe es um $\lceil \ell/d - 1 \rceil$ Plätze nach vorne.

Hier ergibt sich für $d = 1$ die Nach-Vorne-Schieben-Regel, während für größere d das Verfahren immer langsamer reagiert.

Eine partielle Ordnung auf den Vorwärts-Regeln und Vergleiche der erwarteten Kosten sind in [L84] für den Fall unabhängiger Zugriffe angegeben. Für die Situation von Markov-Zugriffen fassen wir unsere Ergebnisse in Abschnitt 1.3 zusammen, siehe auch [SS96].

Deterministische Verfahren mit Gedächtnis

Listenalgorithmen mit Gedächtnis verwerten die bisherige Zugriffsfolge, um bei der nächsten Anfrage eine gute Entscheidung zu treffen. Dabei werden meist nicht die bisherigen Zugriffe explizit gespeichert, sondern der Funktionswert einer fest vorgegebenen Funktion über den Zugriffsfolgen. Dies kann beispielsweise für jedes Element die Anzahl der bisherigen Nachfragen sein, oder die Zeitpunkte der k letzten Nachfragen. Das bekannteste Verfahren mit Gedächtnis ist

Absteigende Häufigkeit, FREQUENCY-COUNT, FC: Sortiere die Liste stets nach absteigenden Zugriffshäufigkeiten.

Im Gegensatz zu den folgenden Algorithmen benutzt FC ein unendliches Gedächtnis. Eine wichtige Regel mit Gedächtnis wurde von *S. Albers* eingeführt:

Timestamp, TS [A98]: Wenn auf x zugegriffen wird, dann füge x vor dem ersten Listenelement ein, das seit dem letzten Zugriff auf x höchstens einmal nachgefragt wurde. Bei der ersten Anfrage nach x bleibt die Position unverändert.

Dieses Verfahren benötigt für jedes Listenelement zwei Zeitstempel, um die Zeiten des letzten und vorletzten Zugriffs zu speichern. Eine Verallgemeinerung der Timestamp-Regel stammt von *R. El-Yaniv*:

Move-to-Recent-Item, MRI(k) [E96]: Wenn auf x zugegriffen wird, dann füge x hinter das letzte Listenelement ein, das vor x steht und seit dem letzten Zugriff auf x höchstens $k+1$ mal nachgefragt wurde. Wenn kein solches Element existiert oder bei der ersten Anfrage nach x, setze x an den Listenkopf.

In Abschnitt 1.4 werden wir drei weitere Familien von Listenalgorithmen einführen.

Randomisierte Listenalgorithmen

In der Literatur wurde eine Reihe von Verfahren vorgestellt, die für ihre Entscheidungen zufällige Bits benutzen dürfen. Wichtige Vertreter sind RANDOM-MTF, BIT und COMB. Randomisierte Listenalgorithmen werden in Abschnitt 1.6 untersucht und diskutiert.

Random-Move-to-Front(p), RANDOM-MTF(p): Mit Wahrscheinlichkeit p setze das nachgefragte Element an den Listenkopf, mit Wahrscheinlichkeit $1-p$ tue nichts.

Dieses Verfahren ist jedoch höchstens 2-kompetitiv (siehe Satz 1.45) und damit nicht besser als deterministische Verfahren. Dagegen ist der folgende Listenalgorithmus ein einfaches Verfahren, das einen kompetitiven Faktor von 1.75 erzielt.

Bit [RWS94]: Zu jedem Listenelement x gehört ein Bit $b(x) \in \{0,1\}$, das bei jedem Zugriff auf x gekippt wird. Wenn $b(x) = 0$ ist, wird x an den Listenanfang gesetzt, ansonsten passiert nichts. Die n Bits werden zufällig unabhängig und gleichverteilt initialisiert.

Der beste bisher bekannte Listenalgorithmus COMB von *S. Albers et.al.* erreicht einen kompetitiven Faktor von 1.6.

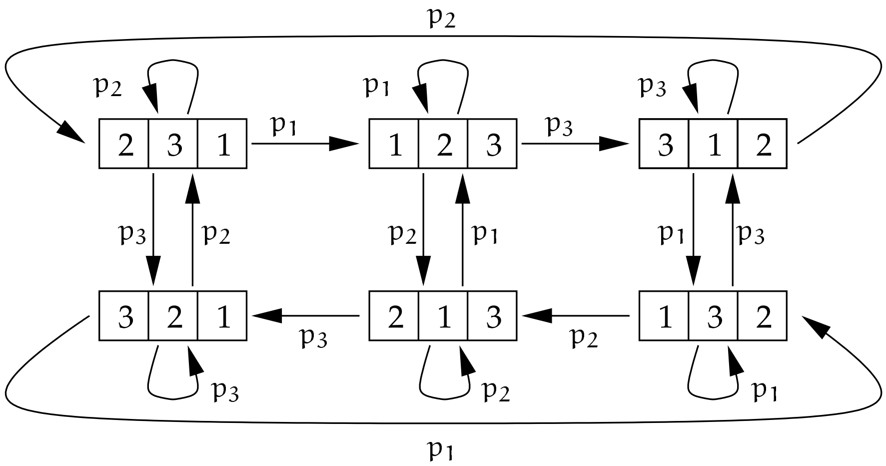
Combination, COMB [ASW95]: Mit Wahrscheinlichkeit $1/5$ wende den deterministischen Timestamp-Algorithmus an und mit Wahrscheinlichkeit $4/5$ benutze BIT.

Analyse von Move-to-Front

In diesem Abschnitt stellen wir die bekannten Analysen des MTF-Verfahrens bei unabhängigen und abhängigen stochastischen Zugriffen und im kompetitiven Modell vor. Die hierbei verwendeten Vorgehensweisen werden im weiteren Verlauf dieser Arbeit immer wieder auftreten.

Unabhängige Zugriffe. Diese Analysen wurden erstmals von *J. McCabe* [M65], *W. Hendricks* [H72] und *P. Burville* und *J. Kingman* [BK73] durchgeführt. Vergleiche auch [ANW82, A77, B79, CSO89, CHS88, F84, H73, H76, HH85, HH87, HM93, N82, OD97, P91, P94, R76] für weitere Untersuchungen im Modell der unabhängigen Zugriffe.

Sei $\mathbf{p} = (p_1, \dots, p_n)$ eine Zugriffsverteilung. Ohne Beschränkung der Allgemeinheit setzen wir voraus, daß $p_1 \geq p_2 \geq \cdots \geq p_n > 0$, da keine Sortierung der Elemente benötigt wird, und für die asymptotischen Kosten die Elemente mit Wahrscheinlichkeit 0 keine Rolle spielen. Diese Wahrscheinlichkeiten definieren eine Markov-Kette auf den möglichen Listenkonfigurationen, die hier am Beispiel $n = 3$ dargestellt ist.



Da nach Voraussetzung alle Übergangswahrscheinlichkeiten positiv sind, ist diese Markov-Kette positiv-rekurrent und damit insbesondere ergodisch (siehe Anhang A.2). Somit sind die stationären Wahrscheinlichkeiten $\pi(\tau)$ jeder Permutation $\tau \in \mathcal{S}_n$ wohldefiniert. Die Listenkonfiguration τ bedeutet, daß sich das Element $j \in \mathcal{A}$ an der Position $\tau(j)$ befindet. Für eine solche Permutation $\tau = (\tau(1), \dots, \tau(n))$ seien $c(\tau) = \sum_{j=1}^{n} j \cdot p_{\tau(j)}$ die erwarteten Kosten. Dann folgt für die asymptotischen erwarteten Kosten des Verfahrens

$$E_{\text{MTF}}(\mathbf{p}) = \sum_{\tau \in \mathcal{S}_n} \pi(\tau) \cdot c(\tau).$$

Zur Berechnung dieses Wertes genügt es, Paare i, j von Elementen zu betrachten [BK73]. Für die asymptotische Wahrscheinlichkeit $b(j, i)$, daß j vor i steht, folgt

$$\begin{aligned}
b(j,i) &= \sum_{\substack{\tau \in \mathcal{S}_n \\ \tau(j) < \tau(i)}} \pi(\tau) \\
&= p_j \cdot \sum_{t \geq 0} (1 - p_i - p_j)^t \\
&= \frac{p_j}{p_i + p_j}.
\end{aligned}$$

Dazu beobachtet man, daß j genau dann vor i steht, wenn seit dem letzten Zugriff auf j nicht mehr i nachgefragt wurde. Indem man danach konditioniert, daß genau t Anfragen nach dem letzten Zugriff auf j aufgetreten sind, folgt obiges Ergebnis. Nun ist die mittlere Position eines Elementes i gerade $1+$ die erwartete Anzahl von Elementen j, die vor i stehen, und man erhält

$$E_{\text{MTF}}(\mathbf{p}) \;=\; \sum_{i=1}^{n} p_i \cdot \Big(1 + \sum_{j\neq i} \frac{p_j}{p_i + p_j}\Big) \;=\; 1 + 2 \cdot \sum_{1\leq i<j\leq n} \frac{p_i p_j}{p_i + p_j}.$$

Die optimalen erwarteten Kosten werden mit der Listenordnung $\langle 1, 2, \dots, n\rangle$ erreicht und betragen $M(\mathbf{p}) = \sum_{i=1}^{n} i p_i$. Das Verhältnis ist durch 2 beschränkt, wie man folgendermaßen sieht:

$$\begin{aligned} E_{\text{MTF}}(\mathbf{p}) &= 1 + 2 \cdot \sum_{1\leq i<j\leq n} p_j \cdot \frac{p_i}{p_i + p_j} \\ &\leq 1 + 2 \cdot \sum_{1\leq i<j\leq n} p_j \cdot 1 \;=\; 1 + 2 \cdot \sum_{j=1}^{n} p_j \cdot (j-1) \\ &= 2 \cdot M(\mathbf{p}) - 1. \end{aligned}$$

Diese Abschätzung läßt sich verbessern, und in [CHS88] wurde gezeigt, daß die Schranke $\pi/2$ für $E_{\text{MTF}}(\mathbf{p})/M(\mathbf{p})$ scharf ist. Für eine Verallgemeinerung dieser Abschätzung auf neue Familien von Listenalgorithmen siehe Satz 1.24 auf Seite 40.

Abhängige Zugriffe. Die MTF-Regel bei Markov-Zugriffen wurde erstmals von *K. Lam et.al.* [LLS84] untersucht, siehe auch [B83, Ch93, DF95, D94, KV81, SS96] für zusätzliche Ergebnisse in diesem Modell.

Sei $\mathbf{P}$ eine positiv-rekurrente Markov-Kette (siehe Anhang A.2). Die wesentliche Idee zur Untersuchung ist die Rückwärtsanalyse: beim aktuellen Zugriff auf i steht ein Element j genau dann vor i, wenn wir die Zugriffsfolge zurückverfolgen und auf ein j treffen, bevor ein i aufgetreten ist. Mit Hilfe von Tabu-Mengen und der Notation aus Anhang A.2 können wir die bedingte Wahrscheinlichkeit, daß j vor i steht, wenn gerade auf j zugegriffen wird, ausdrücken als

$$b(j, i|i) = {}_i\hat{f}^*_{ij} = \frac{\hat{m}_{ii}}{\hat{m}_{ij} + \hat{m}_{ji}} = \frac{m_{ii}}{m_{ij} + m_{ji}}\,.$$

Mit $m_{ii} = 1/\pi_i$ folgt für die asymptotische erwartete Suchzeit

$$E_{\text{MTF}}(\mathbf{P}) = 1 + \sum_{j\neq i} \pi_i b(j, i|i) = 1 + 2 \cdot \sum_{j<i} \frac{1}{m_{ij} + m_{ji}}\,.$$

Kompetitives Modell. Mit der Analyse des MTF-Algorithmus gegen beliebige dynamische Offline-Gegner begründeten *D. Sleator* und *R. Tarjan* das Gebiet der kompetitiven Analyse von Online-Algorithmen [ST85]. Der Name *kompetitive Analyse* wurde erst später in [KMMO94] geprägt. Vergleiche auch [A98a, AW98, BM85] und andere.

Das entscheidende Resultat ist, daß MTF einen kompetitiven Faktor von 2 aufweist. Im Gegensatz zur mittleren Analyse erhält man hierbei die Garantie, daß für jede Zugriffsfolge die Kosten höchstens doppelt so hoch sind wie die eines optimalen Verfahrens, auch wenn dieses die gesamte Zugriffsfolge im voraus kennt. Zum Beweis mittels amortisierter Analyse verwendet man eine Potentialfunktion Φ, die zu jedem Zeitpunkt den Unterschied zwischen der Liste von MTF und der Liste des optimalen Gegners OPT mißt. Ein Paar $\langle y, x \rangle$ von Listenelementen heißt Inversion, wenn y vor x in MTF's Liste steht, aber hinter x in der Liste von OPT. Der Wert der Potentialfunktion ist dann gerade die Anzahl der Inversionen. Sei nun der Zugriff zum Zeitpunkt t auf x gerichtet. Sei A die Anzahl der Elemente, die sowohl bei MTF als auch bei OPT vor x stehen, und sei B die Anzahl der Inversionen $\langle y, x \rangle$. Dann sind die Kosten von MTF in diesem Schritt gerade $A + B + 1$, und die Kosten OPT_t des optimalen Verfahrens betragen mindestens $A + 1$. Indem MTF das Element x nach vorne bewegt, verschwinden B Inversionen. Je nachdem, wie weit OPT sein x nach vorne schiebt, können bis zu A neue Inversionen entstehen. Für die amortisierten Kosten a_t in diesem Schritt folgt also

$$a_t \;=\; A + B + 1 + \Delta\Phi \;\leq\; A + B + 1 + A - B \;\leq\; 2 \cdot (A + 1) \;=\; 2 \cdot \text{OPT}_t.$$

Da diese Abschätzung in jedem Schritt t gilt, folgt der kompetitive Faktor 2 für beliebige Zugriffsfolgen.

1.2. Simple(k)- und Batched(k)-Verfahren

Im folgenden beschreiben wir ein allgemeines Prinzip, wie man aus einem gedächtnislosen Listenalgorithmus eine Familie von Verfahren mit Gedächtnis erhält [GMS81, KR80]. Die beiden Regelklassen sind mit $k \in \mathbb{N}$ parametrisiert und benutzen einen Zähler, der von 0 bis k zählt, und einen Speicherplatz für das zuletzt gefragte Element. Der Platzbedarf liegt also bei $\Theta(\log n + \log k)$ Bit.

Definition 1.1. Sei $k > 0$ und R ein beliebiger gedächtnisloser Listenalgorithmus.

Die **Simple(k)-Heuristik** führt die Aktion der Regel R nur aus, wenn die letzten k Zugriffe auf das gleiche Element gerichtet waren.

Die **Batched(k)-Heuristik** gruppiert die Zugriffe in Blöcke der Länge k und ruft die Aktion der Regel R nur dann auf, wenn alle Zugriffe eines Blocks auf das gleiche Element gerichtet waren. □

Die Einteilung in Blöcke beim Batched(k)-Verfahren wird durch einen fortlaufenden Zähler modulo k realisiert. Da nicht nur k aufeinanderfolgende Zugriffe auf ein bestimmtes Element gerichtet sein müssen, sondern diese auch noch genau in den Blockgrenzen liegen müssen, wird die Aktion der Regel R noch seltener angewendet als bei der Simple(k)-Heuristik. Für $k = 1$ erhält man die gewöhnliche Regel R.

1.2.1. Unabhängige Zugriffe

In der bisherigen Literatur wurden ausschließlich die Regeln R=MTF und R=TS betrachtet. Bei unabhängigen Zugriffen sind folgende Resultate bekannt.

Sei $\mathbf{p} = (p_1, \dots, p_n)$ eine Zugriffsverteilung auf $\{1, \dots, n\}$ mit $p_1 \geq \cdots \geq p_n$. Für Simple(k)-MTF bezeichne $b_k(j,i)$ die asymptotische Wahrscheinlichkeit, daß j vor i steht, für Batched(k)-MTF sei $b'_k(j,i)$ diese Wahrscheinlichkeit. Dann gilt [GMS81]

$$b_k(j,i) = \frac{p_j^k \sum_{\ell=0}^{k-1} p_i^\ell}{p_j^k \sum_{\ell=0}^{k-1} p_i^\ell + p_i^k \sum_{\ell=0}^{k-1} p_j^\ell}, \qquad b'_k(j,i) = \frac{p_j^k}{p_i^k + p_j^k}. \tag{1.1}$$

Damit kann man zeigen, daß für $k \to \infty$ in beiden Fällen die erwartete Suchzeit gegen die optimale Suchzeit $M(\mathbf{p}) = \sum_{i=1}^n i p_i$ konvergiert.

$$\left.\begin{array}{lcl} E_{\text{Simple(k)-MTF}}(\mathbf{p}) & \longrightarrow & M(\mathbf{p}) \\ E_{\text{Batched(k)-MTF}}(\mathbf{p}) & \longrightarrow & M(\mathbf{p}) \end{array}\right\} \text{für } k \to \infty. \tag{1.2}$$

Für kleine Werte von k kann man das Verhältnis $E_{\text{Simple/Batched(k)-MTF}}(\mathbf{p})/M(\mathbf{p})$ durch numerische Abschätzungen beschränken; diese Schranken sind unabhängig von der speziellen Verteilung $\mathbf{p}$. Es gilt [GMS81]:

$$\begin{array}{rcl} E_{\text{Simple(1)-MTF}}(\mathbf{p})/M(\mathbf{p}) & \leq & \pi/2 \leq 1.571 \\ E_{\text{Simple(2)-MTF}}(\mathbf{p})/M(\mathbf{p}) & \leq & 1.367 \\ E_{\text{Simple(3)-MTF}}(\mathbf{p})/M(\mathbf{p}) & \leq & 1.274 \\ E_{\text{Simple(4)-MTF}}(\mathbf{p})/M(\mathbf{p}) & \leq & 1.228 \\ \\ E_{\text{Batched(1)-MTF}}(\mathbf{p})/M(\mathbf{p}) & \leq & \pi/2 \leq 1.571 \\ E_{\text{Batched(2)-MTF}}(\mathbf{p})/M(\mathbf{p}) & \leq & 1.208 \\ E_{\text{Batched(3)-MTF}}(\mathbf{p})/M(\mathbf{p}) & \leq & 1.119 \\ E_{\text{Batched(4)-MTF}}(\mathbf{p})/M(\mathbf{p}) & \leq & 1.084. \end{array}$$

1.2.2. Abhängige Zugriffe

In diesem Abschnitt analysieren wir die Simple(k)- und Batched(k)-MTF Regeln bei Markov-Zugriffsfolgen [SS96]. Dazu benutzen wir das Konzept der Rückwärtsanalyse: das gegenwärtige Aussehen der Datenstruktur wird durch ihre Vergangenheit bestimmt. Wir stellen fest, daß durch eine Verzögerung der Anpassung (großes k) die Verfahren schlechter werden, wenn das Phänomen der Lokalität vorliegt. In dieser Situation verhalten sich die Algorithmen also genau umgekehrt als im Fall unabhängiger Zugriffe, wo eine verzögerte Anpassung die Suchzeit minimiert wird.

Simple(k)-Nach-Vorne-Schieben

Lemma 1.2. *Das Listenelement j steht vor Element i, wenn die letzte Folge von k aufeinanderfolgenden Zugriffen auf j nach der letzten Folge von k aufeinanderfolgenden Zugriffen auf i aufgetreten ist.*

Oder: Beim Zugriff auf l steht j vor i, wenn wir bei l startend, die Zugriffsfolge zurückverfolgen und auf k aufeinanderfolgende Zugriffe auf j treffen, bevor k aufeinanderfolgende Zugriffe auf i aufgetreten sind.

Diese Beobachtung zeigt, daß es sinnvoll ist, k aufeinanderfolgende Anfragen zusammenzufassen. Dies leistet die folgende Konstruktion, die in [KS60, Seiten 140-144] am Beispiel von Zwei-Tupeln erläutert ist.

Sei eine ergodische Markov-Kette mit Zustandsmenge S und Übergangsmatrix $\mathbf{P}$ und ein $k > 1$ gegeben. Wir wollen k-Tupel von Elementen aus S als Zustände der neuen *expandierten* Markov-Kette $\overline{\mathbf{P}}$ auffassen. Dabei sollen $k-1$ Übergänge der ursprünglichen Kette durch einen Zustand der neuen Kette ausgedrückt werden. Es dürfen also nur die Zustände auftreten, deren korrespondierende Folgen von Übergängen in der alten Kette möglich sind. Das bedeutet, ein Zustand $[i_1 i_2 \dots i_k]$ der neuen Kette ist nur erlaubt, falls $p_{i_1 i_2} \cdot p_{i_2 i_3} \cdots p_{i_{k-1} i_k} > 0$ gilt. Diese Einschränkung ist nicht unmittelbar notwendig; wir brauchen sie jedoch, wenn wir ergodische Ketten betrachten. Denn bei einer ergodischen Kette muß jeder Zustand mit Wahrscheinlichkeit > 0 auftreten.

Die Zustandsmenge der expandierten Kette ist

$$\overline{S} = \{\, [i_1 i_2 \dots i_k] \mid i_l \in S \text{ für } l = 1, \dots, k \text{ und } p_{i_1 i_2} \cdots p_{i_{k-1} i_k} > 0 \,\}.$$

Einen Übergang kann man sich dadurch vorstellen, daß ein Fenster der Länge k um eine Position nach rechts auf der ursprünglichen Kette geschoben wird:

$$\overline{p}_{[i_1 i_2 \dots i_k][i_2 i_3 \dots i_{k+1}]} = p_{i_k i_{k+1}},$$

ansonsten ist die Übergangswahrscheinlichkeit 0. Allgemeiner kann man dies schreiben als

$$\overline{p}_{[i_1 \dots i_k][j_1 \dots j_k]} = p_{i_k j_k} \cdot \delta_{[i_2 \dots i_k][j_1 \dots j_{k-1}]},$$

wobei δ das Kronecker-Symbol bezeichnet,

$$\delta_{[a_1 \dots a_j][b_1 \dots b_j]} = \left\{ \begin{array}{lcl} 1 & : & a_i = b_i \text{ für } i = 1, \dots, j \\ 0 & : & \text{sonst} \end{array} \right\}.$$

Lemma 1.3. *Die expandierte Kette $\overline{\mathbf{P}}$ ist genau dann ergodisch, wenn die ursprüngliche Kette $\mathbf{P}$ ergodisch ist. In diesem Fall gilt: wenn $\{\pi_i\}_{i \in S}$ die stationäre Verteilung der ursprünglichen Kette ist, dann folgt für die stationäre Verteilung $\{\pi_{[i_1 \dots i_k]}\}_{[i_1 \dots i_k] \in \overline{S}}$ der expandierten Kette:*

$$\pi_{[i_1 \dots i_k]} = \pi_{i_1} \cdot p_{i_1 i_2} \cdot p_{i_2 i_3} \cdots p_{i_{k-1} i_k}.$$

Sei $b([j \dots j][i \dots i] \mid [l_1 \dots l_{k-1} i])$ die asymptotische Wahrscheinlichkeit, daß $[j \dots j]$ vor $[i \dots i]$ auftritt, wenn die zeitumgekehrte expandierte Kette im Zustand $[l_1 \dots l_{k-1} i]$ gestartet wird. Gemäs A.3 kann man sie folgendermaßen berechnen,

$$b([j \dots j], [i \dots i] | [l_1 \dots l_{k-1} i]) = \frac{\hat{m}_{[l_1 \dots l_{k-1} i][i \dots i]} + \hat{m}_{[i \dots i][j \dots j]} - \hat{m}_{[l_1 \dots l_{k-1} i][j \dots j]}}{\hat{m}_{[i \dots i][j \dots j]} + \hat{m}_{[j \dots j][i \dots i]}},$$

wobei $\hat{m}_{[i_1 \dots i_k][j_1 \dots j_k]}$ die erwarteten Ersteintrittszeiten in der rückwärtslaufenden expandierten Kette bezeichnet. Diese erhält man mit Hilfe der Fundamentalmatrix

$\mathbf{Z} = (z_{ij}) = (\mathbf{I} - \mathbf{P} + \mathbf{A})^{-1}$ als

$$\hat{m}_{[i_1 \dots i_k][j_1 \dots j_k]} = z_{j_k j_1}/\pi_{j_1} - z_{j_k i_1}/\pi_{i_1} + \frac{1 - \sum_{l=1}^{k-2} \pi_{j_1} p_{j_1 j_2} \cdots p_{j_l j_{l+1}} (\delta_{j_{l+1} i_1} \cdots \delta_{j_k i_{k-l}}/\pi_{i_1} - \delta_{j_{l+1} j_1} \cdots \delta_{j_k j_{k-l}}/\pi_{j_1})}{\pi_{j_1} p_{j_1 j_2} \cdots p_{j_{k-1} j_k}} .$$

Bildet man nun den Erwartungswert über alle $l_1, \dots, l_{k-1}$, so erhält man die asymptotische Wahrscheinlichkeit $b_k(j, i|i)$, daß j vor i steht, wenn gerade auf i zugegriffen wird und die Simple(k)-Nach-Vorne-Schieben-Regel benutzt wird. Die Wahrscheinlichkeit des Auftretens von $[l_1 \dots l_{k-1} i]$ unter der Bedingung, daß zuletzt Element i nachgefragt wurde, ist die Wahrscheinlichkeit, beim Zurückverfolgen der Zugriffe von i aus nacheinander die Elemente $l_{k-1}, \dots, l_1$ zu finden.

$$b_k(j, i|i) = \sum_{l_1 \dots l_{k-1}} \hat{p}_{i l_{k-1}} \cdot \hat{p}_{l_{k-1} l_{k-2}} \cdots \hat{p}_{l_2 l_1} \cdot b([j \dots j], [i \dots i] | [l_1 \dots l_{k-1} i]) .$$

Wenn man die oben angegebenen Ausdrücke in diese Formel einsetzt, so erhält man

Satz 1.4. *Die asymptotischen erwarteten Suchkosten der Simple(k)-Nach-Vorne-Schieben-Regel bei Markov-Zugriffsfolgen sind*

$$E_{\text{Simple(k)-MTF}}(\mathbf{P}) = 1 + \sum_{i=1}^{n} \sum_{j \neq i} \pi_i b_k(j, i|i),$$

mit

$$b_k(j, i|i) = \frac{\pi_j p_{jj}^{k-1} \left[\sum_{t=0}^{k-2} p_{ii}^t + p_{ii}^{k-1} \sum_{t=1}^{k-1} (p_{ji}^{(t)} - p_{ii}^{(t)}) + p_{ii}^{k-1} \left(\pi_i m_{ji} + \sum_{\ell} \pi_\ell p_{\ell i}^{(k-1)} (m_{i\ell} - m_{j\ell}) \right) \right]}{\pi_i p_{ii}^{k-1} \sum_{t=0}^{k-2} p_{jj}^t \; + \; \pi_j p_{jj}^{k-1} \sum_{t=0}^{k-2} p_{ii}^t \; + \; \pi_i \pi_j p_{ii}^{k-1} p_{jj}^{k-1} (m_{ij} + m_{ji})} .$$

Für unabhängige Zugriffe gilt $p_{ij} = \pi_j$ und damit auch $p_{ij}^{(t)} = \pi_j$ für alle $t \geq 1$, und $m_{ij} = 1/\pi_j$. Damit erhält man aus obigem Ergebnis wieder den Ausdruck (1.1) zurück. Zur Illustration betrachten wir das Beispiel der folgenden Übergangswahrscheinlichkeiten:

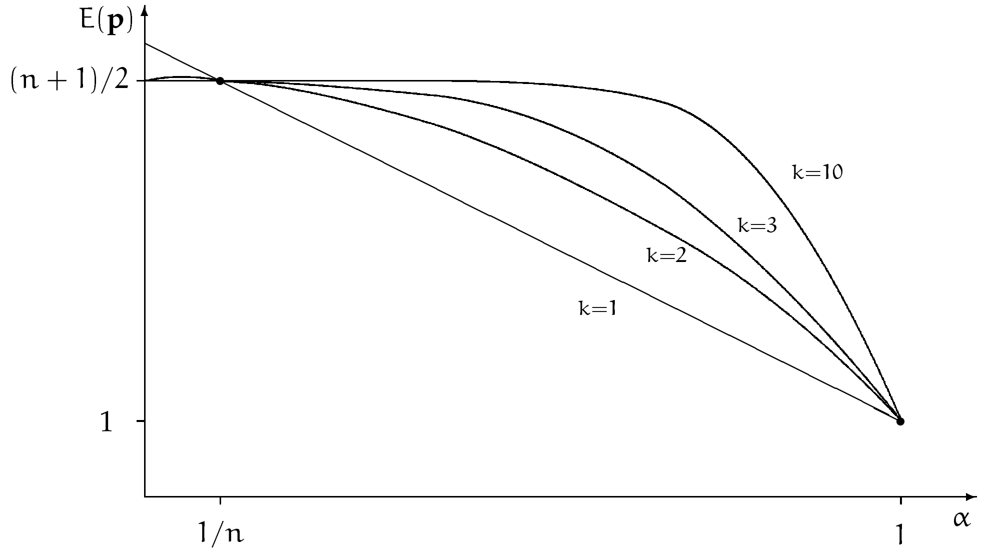
$$p_{ij} = \left\{ \begin{array}{ccc} \alpha & : & i = j \\ \beta & : & i \neq j \end{array} \right\} \tag{1.3}$$

mit $0 \leq \alpha < 1$ und $\beta = (1 - \alpha)/(n - 1)$; siehe auch Anhang A.2.2. Für $\alpha > 1/n$ zeigen die durch diese Markov-Kette beschriebenen Folgen das Phänomen der elementweisen Lokalität, und der Grad der Lokalitat hängt vom Parameter α ab.

Satz 1.5. *Für die oben definierte Markov-Quelle weist die Simple(*k*)-Nach-Vorne-Schieben-Regel die folgenden erwarteten Suchkosten auf:*

$$E_{\text{Simple(k)-MTF}}(\mathbf{P}) = 1 + \frac{n(n-1)(1-\alpha^k)}{2(n-\alpha^{k-1})} \; .$$

Man erkennt, daß für $\alpha > 1/n$ (Lokalität) die erwartete Suchzeit monoton in k wächst. Für alle $\alpha < 1$ liegt der Grenzwert bei $(n+1)/2$ für $k \to \infty$. Die Grafik stellt die erwarteten Suchkosten in Abhängigkeit von α für verschiedene Werte von k am Beispiel $n = 10$ dar.



Batched(k)-Nach-Vorne-Schieben

Die Analyse dieses Verfahrens verläuft ähnlich wie die des Simple(k)-Nach-Vorne-Schiebens.

Lemma 1.6. *Das Listenelement* j *steht vor Element* i*, wenn der letzte Block von Zugriffen auf* j *nach dem letzten Block von Zugriffen auf* i *aufgetreten ist.*

Oder: Beim Zugriff auf l *steht* j *vor* i*, wenn wir beim aktuellen Block (der* l *enthält) startend, die Anfragefolge zurückverfolgen und auf einen Block von Zugriffen auf* j *treffen, bevor ein Block von Zugriffen auf* i *aufgetreten ist.*

Anstatt die Folge elementweise rückwärts zu laufen, können wir hier blockweise rückwärts gehen, und es treten keine Probleme mit der Überlappung von Zustands-Tupeln auf. Wir betrachten wieder die expandierte Kette mit den Zuständen

$$\overline{S} = \{\, [i_1 i_2 \ldots i_k] \mid i_l \in S \text{ für } l = 1, \ldots, k \text{ und } p_{i_1 i_2} \cdots p_{i_{k-1} i_k} > 0 \,\}.$$

Die Übergangswahrscheinlichkeiten sind jetzt

$$\overline{p}_{[i_1 \ldots i_k][j_1 \ldots j_k]} = p_{i_k j_1} \cdot p_{j_1 j_2} \cdots p_{j_{k-1} j_k} \; .$$

Wir berechnen die asymptotische Wahrscheinlichkeit, daß $[j \dots j]$ vor $[i \dots i]$ auftritt, wenn die rückwärtslaufende Kette in einem Zustand $[l \dots]$ gestartet wird. Da sich keine Tupel überlappen, genügt es, das erste Element l des aktuellen Blocks zu kennen. Wir erhalten

$$b([j \dots j], [i \dots i] | l) = \frac{\hat{m}_{[l \dots][i \dots i]} + \hat{m}_{[i \dots i][j \dots j]} - \hat{m}_{[l \dots][j \dots j]}}{\hat{m}_{[i \dots i][j \dots j]} + \hat{m}_{[j \dots j][i \dots i]}}.$$

Im aktuellen Block kann der aktuelle Zugriff auf i an Position $1, \dots, k$ stehen, jeweils mit Wahrscheinlichkeit $1/k$. Wenn der Zugriff an Position m erfolgt, dann gehen wir $m-1$ Schritte rückwärts, um das erste Element des aktuellen Blocks zu erhalten. Dieses Element ist also l mit Wahrscheinlichkeit $\hat{p}_{il}^{(m)}$. Zusammen erhalten wir die asymptotische Wahrscheinlichkeit, daß j vor i steht, wenn gerade auf i zugegriffen wird und die Batched(k)-Nach-Vorne-Schieben-Regel angewendet wird.

$$b'_k(j, i | i) \quad = \quad \frac{1}{k} \sum_{m=1}^{k} \sum_{l} \hat{p}_{il}^{(m-1)} \cdot b([j \dots j], [i \dots i] | l)\ .$$

Man beachte, daß $\overline{\mathbf{P}} = \mathbf{P}^k$ die Markov-Kette beschreibt, die k Schritte der Kette $\mathbf{P}$ in einen Schritt kombiniert. Seien $(\bar{z}_{ij})$ die Einträge der dazugehörigen Fundamentalmatrix.

Satz 1.7. *Die asymptotischen erwarteten Suchkosten der Batched(k)-Nach-Vorne-Schieben-Regel bei Markov-Zugriffsfolgen sind*

$$E_{\text{Batched(k)-MTF}}(\mathbf{P}) = 1 + \sum_{i=1}^{n} \sum_{j \neq i} \pi_i b'_k(j, i | i),$$

mit

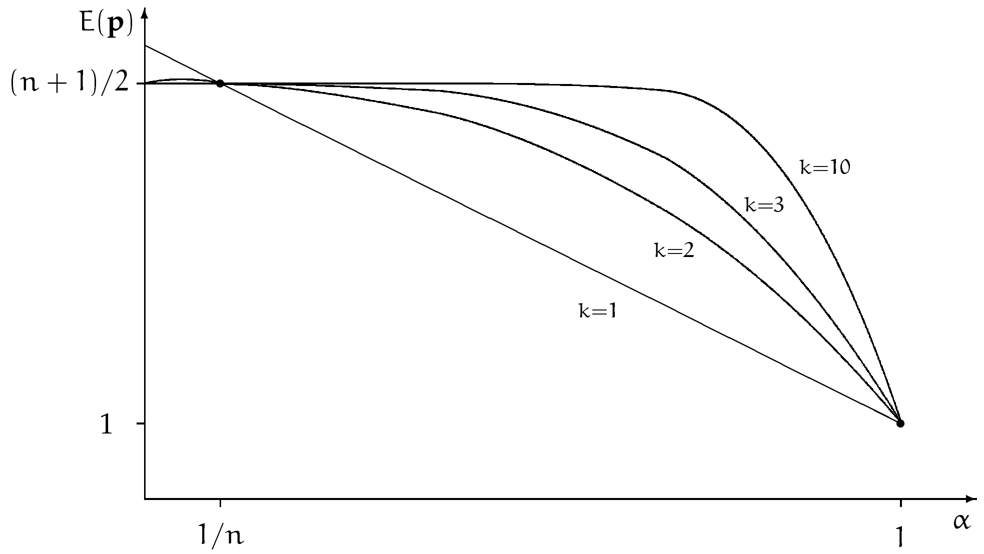
$$b'_k(j, i | i) \quad = \quad \frac{\pi_j p_{jj}^{k-1} - \pi_j p_{ii}^{k-1} p_{jj}^{k-1} \sum_{\ell} \left(\bar{z}_{j\ell} - \bar{z}_{i\ell}\right)\left(p_{\ell i} - \sum_{m=1}^{k} p_{\ell i}^{(m)}/k\right)}{\pi_i p_{ii}^{k-1} + \pi_j p_{jj}^{k-1} - p_{ii}^{k-1} p_{jj}^{k-1} \sum_{\ell} \left(\bar{z}_{j\ell} - \bar{z}_{i\ell}\right)\left(\pi_j p_{\ell i} - \pi_i p_{\ell j}\right)}\ .$$

Im oben definierten Spezialfall folgt

Satz 1.8. *Für Zugriffsfolgen gemäß der Markov-Quelle (1.3) weist die Batched(k)-Nach-Vorne-Schieben-Regel die folgenden erwarteten Suchkosten auf:*

$$E_{\text{Batched(k)-MTF}}(\mathbf{P}) =$$
$$1 + \frac{n-1}{2} \cdot \left(1 - \frac{\alpha^{k-1}(n\alpha - 1)\left((n-1)^k - (n\alpha - 1)^k\right)}{k(n - n\alpha)\left((n-1)^k - (n\alpha - 1)^k + \alpha^{k-1}(n\alpha - 1)(n-1)^{k-1}\right)}\right)\ .$$

Die Grafik stellt die erwarteten Suchkosten für verschiedene Werte von k am Beispiel $n = 10$ dar.



Der folgende Satz zeigt, daß die Batched(k)-Regel noch träger reagiert als die Simple(k)-Regel.

Satz 1.9. *Für Zugriffsfolgen gemäß der Markov-Quelle (1.3) und alle $n \geq 2$ und $k \geq 2$ hat im Falle von Lokalität die Simple(k)-Regel geringere erwartete Kosten als die Batched(k)-Regel:*

$$\alpha > \beta \implies E_{\text{Simple(k)-MTF}}(\mathbf{P}) < E_{\text{Batched(k)-MTF}}(\mathbf{P})$$

Da bekannt ist, daß bei diesem Beispiel und im Falle von Lokalität ($\alpha \geq \beta$) die MTF-Regel optimal ist, siehe [Ch93], können wir das Verhältnis der Kosten von Simple(k)-MTF zu den optimalen Kosten exakt angeben:

$$\begin{aligned} \frac{E_{\text{Simple(k)-MTF}}(\mathbf{P})}{E_{\text{MTF}}(\mathbf{P})} &= \frac{(1-\alpha^k)n^2 + (1+\alpha^k)n - 2\alpha^{k-1}}{(1-\alpha)n^2 + (2+\alpha^k - \alpha^{k-1})n - 2\alpha^{k-1}} \\ &= \frac{1-\alpha^k}{1-\alpha} + O(1/n) \qquad \text{für} \quad n \to \infty \end{aligned}$$

Die Funktion $f(\alpha) = (1-\alpha^k)/(1-\alpha)$ besitzt in $\alpha = 1$ eine hebbare Singularität und nimmt dort ihr Maximum k über dem Intervall $[0, 1]$ an. Also betragen bei starker elementweiser Lokalität und großen Listenlängen die erwarteten Kosten von Simple(k)-MTF bis zum k-fachen der optimalen Kosten.

Man beachte, daß die Simple(k)-MTF- bzw. Batched(k)-MTF-Regeln bereits für $k = 2$ nicht mehr c-kompetitiv für beliebige c sind. Denn durch alternierende Zugriffe auf das letzte und vorletzte Element der Liste werden diese nicht bewegt, erzeugen aber Kosten n bzw. $n-1$. Ein optimaler Algorithmus schiebt diese beiden Elemente an den

Listenkopf und bezahlt dann nur noch Kosten 1 bzw. 2. Da der kompetitive Faktor für beliebige Listenlängen gelten muß, folgt, daß Simple(k)-MTF und Batched(k)-MTF für $k \geq 2$ nicht konstant-kompetitiv sein können.

1.3. Vorwärts-Regeln

Um beliebige Vorwärts-Regeln analysieren zu können, muß man die Markov-Kette untersuchen, die die Zustände der Datenstruktur und ihre Übergänge beschreibt. Jede Listenpermutation ist ein Zustand, und ein Übergang von Zustand σ nach Zustand τ ist möglich, wenn ein Element i existiert, so daß τ erreicht wird, wenn beim Zugriff auf i die Aktion der Regel auf σ angewendet wird.

Im Falle von unabhängigen und identisch verteilten Zugriffen gemäß einer Verteilung $\mathbf{p} = (p_1, \ldots, p_n)$ hat dieser Übergang die Wahrscheinlichkeit $p_{\sigma\tau} = p_i$. Wenn die Zugriffe abhängig sind und durch die Markov-Übergangsmatrix $\mathbf{P} = (p_{ij})$ gegeben werden, muß man die erweiterte Markov-Kette mit Zuständen $(\sigma|i)$ betrachten, wobei σ die aktuelle Listenordnung ist und i das Element, auf das als nächstes zugegriffen wird. Alternativ könnte man auch das Element betrachten, auf das zuletzt zugegriffen wurde. Bei MTF ist dieses Element implizit gegeben durch das erste Symbol in der Liste, was z.B. in [D94, DF95] ausgenutzt wurde. Bei der allgemeineren Klasse der Vorwärts-Regeln müssen wir diese Information aber explizit in den Zustand der Markov-Kette einkodieren. Die Übergänge haben dann die Wahrscheinlichkeit

$$p_{(\sigma|i)(\tau|j)} = \left\{ \begin{array}{ll} p_{ij} & : \text{ nach dem Zugriff auf } i \text{ wird } \sigma \text{ zu } \tau \text{ permutiert} \\ 0 & : \text{ sonst} \end{array} \right\}.$$

Seien R und R' zwei beliebige Vorwärtsregeln, die durch $m_1, \ldots, m_n$ bzw. $m'_1, \ldots, m'_n$ spezifiziert werden ($m_j \leq j$ bezeichnet die Listenposition, auf die das Element bewegt wird, das auf Position j gefunden wird, vgl. Seite 10). Dann kann eine partielle Ordnung auf den Vorwärts-Regeln folgendermaßen definiert werden: $R \leq R'$ falls $m'_i \leq m_i$ für alle $i = 1, \ldots, n$. Die folgenden in der Literatur bekannten Regeln können verglichen werden:

$$\begin{array}{rcll} \text{TR} & \leq & R \;\leq\; \text{MTF} & \text{für alle Vorwärts-Regeln } R \\ \text{MOVE-UP}(k) & \leq & \text{MOVE-UP}(k+1) & \\ \text{POS}(k+1) & \leq & \text{POS}(k) & \text{siehe [A77]} \\ \text{SWITCH}(k) & \leq & \text{SWITCH}(k+1). & \end{array}$$

Für die spezielle Zugriffsverteilung $p_1 = \alpha, p_2 = \cdots = p_n = \beta$ wurde von *Lam* [L84] gezeigt, daß für die asymptotischen erwarteten Suchzeiten $E(R)$ und $E(R')$ gilt

$$R \leq R' \implies E(R) \leq E(R'). \tag{1.4}$$

Daraus folgen die entsprechenden Relationen der erwarteten Suchzeiten für die oben genannten Regel-Spektren, siehe *Kan/Ross* [KR80], *Lam* [L84], *Phelps/Thomas* [PT80], *Tenenbaum/Nemes* [TN82].

Der Vorteil der obigen speziellen Zugriffsverteilung ist, daß es ausreicht, die Position von Element 1 zu kennen. Damit kann die Größe des Zustandsraumes von $n!$ auf n reduziert werden, indem man die Markov-Kette auf den Listenzuständen partitioniert (*lumping*).

Bei abhängigen Zugriffen verfolgen wir den gleichen Ansatz. Indem wir die Zugriffsfolge auf Markov-Ketten mit Übergangswahrscheinlichkeiten der Form

$$p_{ij} = \left\{ \begin{array}{ccc} \alpha & : & i = j \\ \beta & : & i \neq j \end{array} \right\}, \tag{1.5}$$

beschränken, können wir die Größe des Zustandsraums der erweiterten Markov-Kette von $n \cdot n!$ auf n reduzieren. Es ist ausreichend, die Position des Elementes zu kennen, das als nächstes nachgefragt wird.

Satz 1.10. *Für zwei Vorwärts-Regeln R, R' gilt für ihre asymptotische erwartete Suchzeit bei der Zugriffsfolge (1.5) und $\alpha \geq \beta$, daß*

$$R \leq R' \implies E(R') \leq E(R).$$

Sei $\pi(i_1, \ldots, i_n | i_j)$ die asymptotische Wahrscheinlichkeit, daß sich die Liste in Konfiguration $\sigma = (i_1, \ldots, i_n)$ befindet und der nächste Zugriff auf i_j gerichtet ist. Wir werden zeigen, daß die Definition

$$\pi(j) = n! \cdot \pi(i_1, \ldots, i_n | i_j)$$

für beliebige Vorwärts-Regeln eine wohldefinierte Wahrscheinlichkeitsverteilung darstellt. Sei R eine Vorwärts-Regel, die durch Zahlen $1 = m_1 \leq \cdots \leq m_n$ mit $m_i < i$ für $i \geq 2$ gegeben ist, siehe die Definition auf Seite 10. Sei $W(k) = \{l \mid m_l = k\}$ die Menge aller Positionen, deren Elemente bei Zugriff auf Position k geschoben werden. Das Gleichungssystem für die stationären Wahrscheinlichkeiten (*steady-state equations*) der erweiterten Markov-Kette lautet

$$\pi(i_1, \ldots, i_n | i_j) \quad = \quad \sum_{k=1}^{n} \sum_{l \in W(k)} p_{i_k i_j} \cdot \pi(i_1, \ldots, i_{k-1}, i_{k+1}, \ldots, i_l, i_k, i_{l+1}, \ldots, i_n | i_k)$$

mit der Konvention $\pi(i_1, \ldots, i_{k-1}, i_{k+1}, \ldots, i_l, i_k, i_{l+1}, \ldots, i_n | i_k) = \pi(i_1, \ldots, i_n | i_1)$ im Fall $k = l = 1$. Aus obiger Reduktion und mit $\sum_{l=1}^{n} \pi(l) = 1$ erhalten wir

$$\pi(j) = \beta + (\alpha - \beta) \sum_{l \in W(j)} \pi(l) \qquad \text{für } j = 1, \ldots, n\,. \tag{1.6}$$

Dieses Gleichungssystem besitzt eine eindeutige Lösung. Die $n \times n$-Übergangsmatrix $\mathbf{P}$ der erweiterten, aber gemäß obiger Überlegung reduzierten Kette sieht folgendermaßen aus:

$$\mathbf{P} = \begin{pmatrix} \alpha & \beta & \beta & \beta & \cdots & \beta \\ \vdots & \beta & \beta & \beta & \cdots & \beta \\ \alpha & \beta & \beta & \beta & \cdots & \beta \\ \beta & \alpha & \beta & \beta & \cdots & \beta \\ \beta & \vdots & \beta & \beta & \cdots & \beta \\ \beta & \alpha & \beta & \beta & \cdots & \beta \\ & & \vdots & & & \\ \beta & \cdots & \beta & \alpha & \beta & \cdots \end{pmatrix}. \tag{1.7}$$

In Spalte k sind nur die Elemente p_{ik} mit $i \in W(k)$ gleich α, alle anderen sind β. Da $\pi(j)$ die Wahrscheinlichkeit ist, daß bei der Regel R das Element auf Position j als nächstes nachgefragt wird, ist die erwartete Suchzeit gerade $E(R) = \sum_{j=1}^{n} j\pi(j)$.

Für den Beweis des Satzes benötigen wir die Konzepte der Vektor-Dominanz und der monotonen Matrizen, siehe [KK77] für die Theorie, [L84] für die Anwendung auf unabhängige Zugriffe. Ein Wahrscheinlichkeitsvektor $\mathbf{p} = (p(1), \ldots, p(n))$ dominiert einen anderen Wahrscheinlichkeitsvektor $\mathbf{p}' = (p'(1), \ldots, p'(n))$, falls gilt:

$$\sum_{j=i}^{n} p'(j) \leq \sum_{j=i}^{n} p(j) \qquad \text{für alle } i = 1, \ldots, n.$$

Dafür schreiben wir auch $\mathbf{p}' \prec \mathbf{p}$. Eine Markov-Kette mit Übergangsmatrix $\mathbf{P} = (p_{ij})$ heißt stochastisch monoton, wenn für alle festen s die Partialsummen der i-ten Zeile, $P_{is} = \sum_{j=1}^{s} p_{ij}$, monoton fallend in i sind.

Lemma 1.11. *Seien* $\mathbf{P}$ *und* $\mathbf{P}'$ *zwei ergodische und stochastisch monotone Markov-Ketten mit stationären Wahrscheinlichkeiten* π *bzw.* π'. *Dann gilt*

$$\pi \cdot \mathbf{P}' \prec \pi \implies \pi' \prec \pi.$$

Zum Beweis des Lemmas siehe [KK77]. Nun zeigen wir den Satz.

Beweis. Gegeben seien zwei Vorwärtsregeln $R \leq R'$ unter der Zugriffsfolge (1.5). Wir zeigen, daß in der Situation $\alpha \geq \beta$ ihre Übergangsmatrizen, wie sie durch (1.7) gegeben sind, stochastisch monoton sind, und daß die stationären Wahrscheinlichkeiten die Beziehung $\pi\mathbf{P}' \prec \pi$ erfüllen. Mit Lemma 1.11 folgt dann $\pi' \prec \pi$, d.h. $\sum_{j=i}^{n} \pi'(j) \leq \sum_{j=i}^{n} \pi(j)$ für alle i. Summation über i ergibt $\sum_{i=1}^{n} i\pi'(i) \leq \sum_{i=1}^{n} i\pi(i)$, also $E(R') \leq E(R)$, was den Satz zeigt.

Sei R durch $m_1 \leq \cdots \leq m_n$ gegeben und $W(k)$ wie oben definiert. Jede Zeile i der Übergangsmatrix $\mathbf{P}$ enthält β außer Position m_i, wo α steht. Also ist die Partialsumme $P_{is} = \sum_{j=1}^{s} p_{ij}$ gerade $(s-1)\beta + \alpha$ für $i \leq \max W(s)$ und $s\beta$ für $i > \max W(s)$. Für $\alpha \geq \beta$ folgt, daß die Partialsummen P_{is} monoton nicht-wachsend in i sind, und $\mathbf{P}$ ist stochastisch monoton. Für $\mathbf{P}'$ verläuft die Begründung analog.

Da π die stationäre Verteilung für $\mathbf{P}$ ist, also $\pi = \pi\mathbf{P}$, ist $\pi\mathbf{P}' \prec \pi$ gleichbedeutend mit $\pi\mathbf{P}' \prec \pi\mathbf{P}$. Sei

$$V(i) \;=\; \bigcup_{j=i}^{n} W(j) \;=\; \{l \mid m_l \geq i\}.$$

Wir zeigen, daß im Fall $\alpha \geq \beta$ die Vektor-Dominanz $\pi\mathbf{P}' \prec \pi\mathbf{P}$ gilt. Für alle $i = 1, \ldots, n$ sind die folgenden Aussagen äquivalent:

$$\begin{aligned}
\sum_{j=i}^{n} (\pi\mathbf{P}')_j &\leq \sum_{j=i}^{n} (\pi\mathbf{P})_j \\
\sum_{j=i}^{n} \Big(\beta + (\alpha-\beta) \sum_{l \in W'(j)} \pi(l)\Big) &\leq \sum_{j=i}^{n} \Big(\beta + (\alpha-\beta) \sum_{l \in W(j)} \pi(l)\Big) \\
\sum_{j=i}^{n} \sum_{l \in W'(j)} \pi(l) &\leq \sum_{j=i}^{n} \sum_{l \in W(j)} \pi(l) \\
\sum_{l \in V'(i)} \pi(l) &\leq \sum_{l \in V(i)} \pi(l)\;.
\end{aligned}$$

Die letzte Aussage ist wahr, da aus $m_l' \leq m_l$ für alle $l = 1, \ldots, n$ folgt, daß auch $V'(i) \subseteq V(i)$ für alle $i = 1, \ldots, n$. Damit sind die Voraussetzungen für das Lemma erfüllt und der Beweis abgeschlossen. □

1.4. Drei neue Familien von Listenalgorithmen

In diesem Abschnitt stellen wir drei neue Klassen von deterministischen Verfahren zum dynamischen Umsortieren von Listen vor [S98]. Diese Algorithmen arbeiten nicht nur auf stochastischen Quellen, sondern auch auf beliebigen Anfragefolgen gut.

Die Familie SORT-BY-RANK(SBR) ist mit einem rellen $\alpha \geq 0$ parametrisiert, wobei SBR(0) identisch mit der NACH-VORNE-SCHIEBEN-REGEL und SBR(1) äquivalent mit der TIMESTAMP-Regel von *S. Albers* [A98] ist. Für $0 \leq \alpha \leq 1$ ergibt sich ein Verhalten, das zwischen der radikalen Strategie von MOVE-TO-FRONT und dem mehr konservativen Vorgehen von TIMESTAMP liegt. In diesem Parameterbereich sind die Verfahren auf beliebigen nicht-stochastischen Anfragefolgen optimal (d.h. 2-kompetitiv) und stellen damit die umfassendste bisher bekannte Klasse optimaler Listenalgorithmen dar. Sie werden in Abschnitt 1.4.2 analysiert.

Im darauffolgenden Abschnitt 1.4.3 werden die SORT-BY-DELAY-Regeln eingeführt. Sie sind mit den natürlichen Zahlen parametrisiert, und hier ist SBD(1) identisch mit MOVE-TO-FRONT und SBD(2) äquivalent zu TIMESTAMP. Allgemein gilt, daß für $k \geq 2$ die Regel SBD(k) k-kompetitiv ist. Außerdem läßt sich ihr Verhalten auf stochastischen Anfragequellen berechnen, und es zeigt sich, daß bei gedächtnislosen Quellen die erwarteten Suchkosten für $k \to \infty$ gegen die optimalen Suchkosten konvergieren.

Die SORT-BY-DELAY-Regeln bilden somit die einzige bekannte Klasse von Regeln, die asymptotisch optimal sind, und bei der jedes einzelne Verfahren einen konstanten kompetitiven Faktor (unabhängig von der Listenlänge) aufweist.

Die Familie der SORT-BY-TIME(q)-Algorithmen ist mit einem reellen $0 \leq q \leq 1$ parametrisiert. Sie definiert eine Verallgemeinerung des FREQUENCY-COUNT-Verfahrens, wobei die Häufigkeit der Listenelemente mit q diskontiert wird. Für $0 \leq q \leq 1/2$ erhält man die MTF-Regel und für $q = 1$ den FREQUENCY-COUNT-Algorithmus. Im Bereich $1/2 < q < 1$ ist SBT(q) noch $\lceil 1/\log(1/q) \rceil$-kompetitiv, wie in Abschnitt 1.4.4 gezeigt wird. Insbesondere folgt daraus, daß für $0 \leq q \leq \sqrt{1/2}$ auch die SBT(q)-Regeln kompetitiv-optimal.

In Abschnitt 1.4.5 werden empirische Ergebnisse für die drei Familien vorgestellt. Diese sind sinnvoll, um die Intuition zu bestätigen, daß SORT-BY-RANK auf stochastischen Quellen einen stetigen Übergang zwischen MOVE-TO-FRONT und TIMESTAMP bietet, da für diesen Fall analytische Resultate nur schwierig zu erhalten sind. Im Rahmen von Experimenten zur Datenkompression werden empirische Ergebnisse mit den SBR(α)- und SBD(k)-Verfahren in Abschnitt 2.1.2 dargestellt.

1.4.1. Definition der Algorithmen

Sei $\sigma = \sigma_1, \sigma_2, \ldots, \sigma_m \in \mathcal{A}^m$ eine beliebige Folge von Anfragen auf die Listenelemente $\mathcal{A}$. Wir betrachten einen Zeitpunkt t mit $1 \leq t \leq m$ und definieren für jedes Listenelement $x \in S$ die Zeit des letzten Zugriffs $w_1(x,t)$,

$$w_1(x,t) \;=\; \max(\, \{t' \leq t : \sigma_{t'} = x\} \cup \{0\} \,) \;.$$

Für $k > 1$ definieren wir den k-letzten Zugriff auf x zur Zeit t,

$$w_k(x,t) \;=\; \max(\, \{t' \leq w_{k-1}(x,t) : \sigma_{t'} = x\} \cup \{0\} \,) \;.$$

Damit definieren wir für $k \geq 1$ die *Rangfunktionen*

$$s_k(x,t) \;=\; t - w_k(x,t) \;.$$

Lemma 1.12. *(a) Für alle Zeitpunkte t, bei denen jedes Listenelement mindestens einmal nachgefragt wurde, gilt: Wenn die Liste mit* MOVE-TO-FRONT *verwaltet wird, dann ist die aktuelle Listenordnung genau dann $\langle x_1, x_2, \ldots, x_n \rangle$, wenn*

$$s_1(x_1,t) \;<\; s_1(x_2,t) \;<\; \cdots \;<\; s_1(x_n,t) \;.$$

(b) Für alle Zeitpunkte t, bei denen jedes Listenelement mindestens zweimal nachgefragt wurde, gilt: Wenn die Liste mit TIMESTAMP *verwaltet wird, dann ist die aktuelle Listenordnung $\langle x_1, x_2, \ldots, x_n \rangle$ genau dann, wenn*

$$s_2(x_1,t) \;<\; s_2(x_2,t) \;<\; \cdots \;<\; s_2(x_n,t) \;.$$

Beweis. (a) ist klar, da MTF die Listenelemente nach den Zeitpunkten ihres letzten Zugriffs (*recency rank*) sortiert.

(b) Die Bedingung *füge Element x vor das erste Element y ein, das vor x steht und das seit dem letzten Zugriff auf x höchstens einmal nachgefragt wurde* zur Zeit t ist äquivalent zu *füge x vor das erste Element y ein, das vor x steht und das $s_2(x,t) < s_2(y,t)$ erfüllt*. Diese Aussage ist äquivalent zu *füge x hinter dem letzten Element z vor x ein, das $s_2(z,t) < s_2(x,t)$ erfüllt*, was aus der Tatsache folgt, daß alle Elemente, die seit dem letzten Zugriff auf x höchstens einmal nachgefragt wurden, einen zusammenhängenden Block vor x bilden [A98].

□

Diese Charakterisierung liefert im übrigen eine einfache Definition des TIMESTAMP-Verfahrens. Für $\alpha \geq 0$ bertrachten wir jetzt eine neue Rangfunktion als die Konvexkombination von $s_1(x,t)$ und $s_2(x,t)$:

$$r_t(x) = r_t(x,\alpha) \;=\; (1-\alpha)\cdot s_1(x,t) \;+\; \alpha\cdot s_2(x,t)\;.$$

Definition 1.2. SORTIERE-NACH-RANG(α), SORT-BY-RANK(α), SBR(α)

> Wenn zum Zeitpunkt t das Listenelement x nachgefragt wird, dann füge x direkt hinter dem letzten Element y vor x ein, für das $r_t(y,\alpha) < r_t(x,\alpha)$ gilt. Setze x an den Listenanfang, wenn kein solches Element y existiert.

□

Dabei gilt SBR(0) = MTF, und SBR(1) ist äquivalent zu TS bis auf die Behandlung des ersten Zugriffs auf ein Element. Das bedeutet, daß SBR(1) und TS genau gleich arbeiten würden, wenn TS dahingehend geändert würde, daß ein Element beim ersten Zugriff an den Listenkopf gesetzt wird. Beim ersten Zugriff auf ein x ist $r_t(x) = 0$, daher wird x von SBR(1) an den Anfang der Liste gesetzt.

Für $0 \leq \alpha \leq 1$ interpolieren diese Regeln zwischen MTF und TS. Man beachte, daß es sich nicht um eine *randomisierte* Mischung zweier Verfahren handelt, wie z.B. die TIMESTAMP(p)-Verfahren [A98], sondern um eine deterministische Kombination der zwei Algorithmen. Für $\alpha > 1$ „extrapolieren" die Regeln, und im Limit $\alpha \to \infty$ wird die Liste nach aufsteigenden Rang*differenzen* $s_2(x,t) - s_1(x,t)$ zwischen dem letzten und vorletzten Zugriff auf x sortiert. Für größere α sind diese Verfahren jedoch nicht mehr 2-kompetitiv, siehe Satz 1.16.

Die zweite Familie von Listenalgorithmen, die wir betrachten, ist gegeben durch die folgende

Definition 1.3. SORTIERE-NACH-ZUGRIFF(k), SORT-BY-DELAY(k), SBD(k)

> Wenn zum Zeitpunkt t das Listenelement x nachgefragt wird, dann füge x direkt hinter dem letzten Element y vor x ein, für das $s_k(y,t) < s_k(x,t)$ gilt. Setze x an den Listenanfang, wenn kein solches Element y existiert, oder wenn es der erste Zugriff auf x ist.

□

Hier ist SBD(1) = MTF, und SBD(2) ist äquivalent zu TS im obigen Sinne. Somit erscheint SBD(k) als eine natürliche Verallgemeinerung dieser Regeln. Unabhängig von unseren Algorithmen wurden ähnliche Regeln für die effiziente Pufferung von Plattenzugriffen bei Datenbankoperationen von *G. Weikum et.al.* eingeführt. Die LRU(k)-Verfahren werden in [OOW93, OOW98] analysiert.

Die dritte Familie von Listenalgorithmen ist mit einem $0 \leq q \leq 1$ parametrisiert. Der Wert der Rangfunktion für ein Element x zum Zeitpunkt t ergibt sich als die diskontierte Summe der Zugriffszeitpunkte. Sei $\sigma = \sigma_1, \ldots, \sigma_m$ die Zugriffsfolge und $x \in \mathcal{A}$. Für $1 \leq t \leq m$ sei

$$u_t(x) = \sum_{\substack{\sigma_{t-s}=x \\ 0 \leq s \leq t-1}} q^s.$$

Dabei verschwindet eine leere Summe, und $0^0 := 1$ (im Gegensatz zu [GKP94, Seite 162], wo $0^0 := 0$).

Definition 1.4. SORTIERE-NACH-ZEIT(q), SORT-BY-TIME(q), SBT(q)

> Wenn zum Zeitpunkt t das Listenelement x nachgefragt wird, dann füge x direkt hinter dem letzten Element y vor x ein, für das $u_t(y) > u_t(x)$ gilt. Setze x an den Listenanfang, wenn kein solches Element y existiert.

□

Im Fall $q = 1$ werden lediglich die Zugriffe gezählt, das Verfahren ist dann identisch mit FREQUENCY-COUNT. Für $q < 1$ werden kurz zurückliegende Zugriffe höher bewertet als lange zurückliegende Nachfragen. Mit abnehmendem q werden die SBT(q)-Algorithmen zunehmend reaktiver. Im Fall $q = 0$ hat das Element des letzten Zugriffs den Rang $0^0 = 1$ und alle anderen Elemente den Rang 0, es liegt also MTF vor.

1.4.2. Analyse der SORT-BY-RANK-Verfahren

Das Hauptergebnis über die SORT-BY-RANK-Verfahren ist der folgende Satz.

Satz 1.13. *Für $0 \leq \alpha \leq 1$ ist SBR(α) 2-kompetitiv.*

Somit stellen diese Verfahren eine überabzählbare Klasse kompetitiv-optimaler Listenalgorithmen dar. Im Vergleich dazu bilden die MRI(k)-Regeln von *El-Yaniv* [E96] (siehe Seite 12) eine abzählbare Familie kompetitiv-optimaler Verfahren. Die SBR(α)-Algorithmen können mit $2n$ Zeitstempeln implementiert werden, wogegen die MRI(k)-Verfahren nk Zeitstempel benötigen, was viel ist, wenn man Regeln „nahe“ an MTF betrachtet.

Vor dem Beweis des Satzes geben wir ein Beispiel für die Arbeitsweise von SBR(α) anhand von $\alpha = 1/2$ und einer Liste $\langle a, b, c, d \rangle$.

$$\sigma = \underset{9}{a}, \underset{8}{b}, \underset{7}{c}, \underset{6}{d}, \underset{5}{a}, \underset{4}{b}, \underset{3}{c}, \underset{2}{d}, \underset{1}{b}, \underset{0}{a}$$

Zum Zeitpunkt $t = 10$ entsteht die Liste $\langle a, b, d, c\rangle$, nachdem die Zugriffe mit SBR(1/2) bearbeitet wurden. Man beachte, daß zum Zeitpunkt $t + 1$ der Rang von Elementen $x \neq \sigma_{t+1}$ sich um 1 erhöht, und der Rang von $y = \sigma_{t+1}$ durch $r_{t+1}(y) = \alpha s_2(y, t + 1) = \alpha(1 + s_1(y, t))$ gegeben ist.

Sei A/B eine rationale Zahl mit $A, B \in \mathbb{N}$ und $0 < A < B$. Für die Anfragefolge

$$\sigma \; = \; x, a^{B-A-1}, y, y, a^{A-1}, x \quad \in \{x, y, a\}^{B+2}$$

gilt dann zum Zeitpunkt $t = B + 2$ für die Differenz der Ränge von x und y

$$r_t(x) - r_t(y) = \alpha B - A.$$

An diesem Beispiel erkennt man zwei Punkte:

Jedes α definiert tatsächlich einen eigenen Listenalgorithmus, d.h. man kann für $\alpha \neq \alpha'$ eine Zugriffsfolge finden, die bei Bearbeitung mit SBR(α) und SBR(α') zu verschiedenen Ergebnissen führt. Denn seien α und α' gegeben mit $0 \leq \alpha < \alpha' \leq 1$. Wähle eine rationale Zahl A/B mit $\alpha < A/B < \alpha'$. Wenn obige Folge mit SBR(α) abgearbeitet wurde, dann steht x vor y in der Liste, bei SBR(α') steht aber y vor x.

Wenn $\alpha = A/B$ rational ist, dann existieren stets Anfragefolgen, die für zwei Elemente den gleichen Rang ergeben. Zum Beispiel gilt für obige Anfragefolge, daß zum Zeitpunkt $t = B+2$ die Ränge für x und y identisch sind: $r_t(x) = r_t(y) = A \cdot (B+1)/B$. Man beachte, daß für MTF ($\alpha = 0$) oder TS ($\alpha = 1$) die Ränge zweier Elemente niemals gleich sind. In der Definition von SBR(α) wird diese Mehrdeutigkeit umgangen, indem das nachgefragte Element so weit wie möglich nach vorne geschoben wird, ohne die Ordnung

$$r_t(x_1) \leq r_t(x_2) \leq \cdots \leq r_t(x_n)$$

zu verletzen. Wenn α irrational ist, dann sind stets alle Ränge verschieden.

Paarweise Unabhängigkeit

Eine wichtige Frage für die Analyse ist, ob ein Listenalgorithmus die *paarweise Unabhängigkeits-Bedingung* erfüllt (*pairwise independence property*). Diese Eigenschaft wurde von *Bentley/McGeoch* [BM85] als eine wichtige Voraussetzung für die einfache Analyse von MTF gefunden. Sie bedeutet, daß für jede Anfragefolge σ und zwei beliebige verschiedene Elemente x und y aus der Liste L gilt, daß ihre relative Reihenfolge nach dem Abarbeiten von σ die gleiche ist wie die relative Reihenfolge in L_{xy}, wenn σ_{xy} abgearbeitet wurde. Dabei ist L_{xy} die Liste, die nur die zwei Elemente x und y enthält, und σ_{xy} ist die auf $\{x, y\}$ projizierte Anfragefolge, die entsteht, wenn man alle Zugriffe auf Elemente verschieden von x oder y aus σ löscht.

Das folgende Beispiel zeigt, daß die SBR(α)-Regeln die paarweise Unabhängigkeits-Bedingung nicht erfüllen. Betrachte die Zugriffsfolgen σ_1 und σ_2 über den Elementen $\{x, y, a\}$,

$$\sigma_1 = x, y, y, a, x \qquad \sigma_2 = x, a, y, y, x\,.$$

Die Projektion auf $\{x, y\}$ ergibt in beiden Fällen $\sigma_{xy} = x, y, y, x$. Aber für $\alpha = 1/2$ und $t = 5$ ist im ersten Fall $r_t(x) = 2 < 2.5 = r_t(y)$, und im zweiten Fall $r_t(x) = 2 > 1.5 = r_t(y)$. Da die Bedingung der paarweisen Unabhängigkeit nicht erfüllt ist, muß der Beweis von Satz 1.13 auf anderen Konzepten aufbauen als die paarweise Analyse, nämlich auf einer amortisierten Analyse mit Hilfe einer Potentialfunktion und auf einer sogenannten Listenfaktorisierung.

Listenfaktorisierung

Sei $\sigma = \sigma_1, \sigma_2, \ldots, \sigma_m \in \mathcal{A}^m$ eine beliebige Zugriffsfolge. Mit OPT bezeichnen wir einen optimalen Offline-Algorithmus für diese Folge, und mit $\text{OPT}(\sigma)$ seine Gesamtkosten. Weiter sei FOPT (faktorisierter OPT) ein optimaler Offline-Algorithmus, der alle $\binom{n}{2}$ zwei-elementigen Listen L_{xy} für alle Mengen $\{x, y\} \subseteq \mathcal{A}$ verwaltet. Das bedeutet, jede solche Liste L_{xy} wird von einem optimalen Offline-Algorithmus kontrolliert, der die Zugriffsfolge σ_{xy} bearbeitet. Mit FOPT_t bezeichnen wir die Gesamtkosten, die beim Bearbeiten von Zugriff σ_t entstehen, $1 \le t \le m$. Dies ist gerade 1 plus die Anzahl der Listen L_{xy}, bei denen σ_t an zweiter Stelle steht. Dann gilt

$$\sum_{t=1}^{m} \text{FOPT}_t \;\le\; \text{OPT}(\sigma)\ . \tag{1.8}$$

Um diese wichtige Ungleichung herzuleiten, betrachten wir zunächst das $(i-1)$-Kostenmodell (*partial cost model*), und bezeichnen die Kosten eines Algorithmus A in diesem Modell mit $\overline{\text{A}}(.)$. Weiter setzen wir voraus, daß A niemals bezahlte Austausche vornimmt. Die Kosten beim Zugriff auf σ_t sind dann

$$\overline{\text{A}}_t(x, \sigma_t) = \left\{ \begin{array}{lll} 1 & : & \text{falls } x \text{ zum Zeitpunkt } t \text{ vor } \sigma_t \text{ steht} \\ 0 & : & \text{sonst} \end{array} \right\},$$

und die Gesamtkosten der Zugriffsfolge σ

$$\begin{aligned} \overline{\text{A}}(\sigma) &= \sum_{t=1}^{m} \sum_{x \in \mathcal{A}} \overline{\text{A}}(x, \sigma_t) = \sum_{x \in \mathcal{A}} \sum_{t=1}^{m} \overline{\text{A}}(x, \sigma_t) \\ &= \sum_{x \in \mathcal{A}} \sum_{y \in \mathcal{A}} \sum_{\substack{1 \le t \le m \\ \sigma_t = y}} \overline{\text{A}}(x, \sigma_t) = \sum_{\substack{x, y \in \mathcal{A} \\ x \ne y}} \sum_{\substack{1 \le t \le m \\ \sigma_t = y}} \overline{\text{A}}(x, \sigma_t) \\ &= \sum_{\substack{\{x, y\} \subset \mathcal{A} \\ x \ne y}} \underbrace{\sum_{\substack{1 \le t \le m \\ \sigma_t \in \{x, y\}}} \overline{\text{A}}(x, \sigma_t) + \overline{\text{A}}(y, \sigma_t)}_{=: \overline{\text{A}}_{xy}(\sigma)}\ . \end{aligned}$$

Für einen optimalen Offline-Algorithmus gilt diese Beziehung nicht unbedingt, da ein optimaler Algorithmus im allgemeinen nicht auf bezahlte Austausche verzichten kann [RW90]. Indem man die Kosten in Suchkosten und bezahlte Austausche aufspaltet, kann man aber zeigen [E96], daß

$$\sum_{\substack{\{x,y\}\subset\mathcal{A}\\ x\neq y}} \overline{\mathrm{OPT}}(\sigma_{xy}) \le \overline{\mathrm{OPT}}(\sigma).$$

Mit Hilfe einer modifizierten Kostenfunktion anstatt $\overline{A}(x, \sigma_t)$, die die Kosten des letzten Vergleiches im Standardkostenmodell über alle Vergleiche amortisiert, können obige Aussagen des $(i-1)$-Modells übertragen werden. Insbesondere gilt

$$\sum_{\substack{\{x,y\}\subset\mathcal{A}\\ x\neq y}} \mathrm{OPT}(\sigma_{xy}) \le \mathrm{OPT}(\sigma).$$

Da die linke Seite dieser Ungleichung gerade die summierten Kosten $\sum_{t=1}^{m} \mathrm{FOPT}_t$ eines optimalen faktorisierten Algorithmus bezeichnet, folgt auch Ungleichung (1.8).

Amortisierte Analyse

Sei L eine Liste, die von einem beliebigen Listenalgorithmus verwaltet wird. Wenn Element x vor y steht, dann sagen wir $\langle x, y\rangle$ in L. Wenn $\langle x, y\rangle$ in der Liste von $\mathrm{SBR}(\alpha)$ und $\langle y, x\rangle$ in der Liste L_{xy} von FOPT, dann nennen wir $\langle x, y\rangle$ eine Inversion.

Sei nun L die Liste, die von $\mathrm{SBR}(\alpha)$ verwaltet wird. Für jedes Paar verschiedener Elemente x, y definieren wir die Gewichtsfunktion

$$w(x,y) = \left\{\begin{array}{lll} 0 & : \text{ falls } \langle x,y\rangle & \text{keine Inversion ist} \\ 1 & : \text{ falls } \langle x,y\rangle & \text{Inversion ist und } y \text{ wird } x \text{ in } L \text{ überholen,} \\ & & \text{wenn als nächstes auf } y \text{ zugegriffen wird} \\ 2 & : \text{ falls } \langle x,y\rangle & \text{Inversion ist und } y \text{ wird } x \text{ in } L \text{ genau dann über-} \\ & & \text{holen, wenn jetzt zweimal auf } y \text{ zugegriffen wird} \end{array}\right\}.$$

Dann definieren wir die Potentialfunktion

$$\Phi = \sum_{x\neq y} w(x,y)\ .$$

Das Potential ist stets nicht-negativ. Fixiere eine Zugriffsfolge $\sigma = \sigma_1, \ldots, \sigma_m$. Die *amortisierten Kosten* a_t von $\mathrm{SBR}(\alpha)$ zum Bearbeiten von σ_t sind definiert als

$$a_t = \mathrm{SBR}(\alpha)_t + \Phi_t - \Phi_{t-1}\ ,$$

wobei $\mathrm{SBR}(\alpha)_t$ die tatsächlichen Kosten von $\mathrm{SBR}(\alpha)$ in diesem Schritt sind, und Φ_t das Potential zum Zeitpunkt t. Wir werden zeigen, daß eine Konstante c existiert, so daß $a_t \le c \cdot \mathrm{FOPT}_t$ für alle $t = 1, \ldots, m$. Daraus folgt mit Hilfe von (1.8), daß $\mathrm{SBR}(\alpha)$ dann c-kompetitiv ist.

Wir zeigen zuerst, daß die Gewichtsfunktion die folgende wichtige Eigenschaft besitzt, die besagt, daß das Gewicht einer Inversion $\langle z, y\rangle$ sich nicht von 1 auf 2 erhöhen kann, wenn ein drittes Element x nachgefragt wird.

Lemma 1.14. *Fixiere $0 \leq \alpha \leq 1$. Seien y und z zwei Elemente, so daß $\langle z, y\rangle$ eine Inversion in L mit dem Gewicht $w(z, y) = 1$ ist. Sei x ein von z und y verschiedenes Element. Dann bleibt $w(z, y) = 1$, nachdem eine Nachfrage nach x von SBR(α) bearbeitet wurde.*

Beweis. Wenn die Bedingung $w(z, y) = 1$ zum Zeitpunkt t gilt, dann würde y das Element z überholen, wenn es jetzt nachgefragt würde, d.h. $r_t(y) \leq r_t(z)$. In diesem Falle wäre also $\sigma_t = y$, und $r_t(y)$ reduziert sich zu $\alpha s_2(y, t)$, so daß gilt

$$\alpha s_2(y, t) \leq r_t(z). \tag{1.9}$$

Für ein Widerspruchsargument nehmen wir an, daß $w(z, y) = 2$ gilt, nachdem $\sigma_t = x$ bearbeitet wurde. Das bedeutet, daß z nicht von y überholt wird, wenn $\sigma_{t+1} = y$. Das kann nur dann passieren, wenn x zum Zeitpunkt t zwischen z und y gesetzt wird, d.h. $r_t(z) < r_t(x) < r_t(y)$. Nach dem Zugriff $\sigma_{t+1} = y$ wird y direkt hinter x gesetzt werden, also

$$r_{t+1}(x) < r_{t+1}(y) = \alpha s_2(y, t+1) = \alpha(s_1(y, t) + 1) \leq \alpha(s_2(y, t) + 1)\ .$$

Da $\sigma_{t+1} \notin \{x, z\}$ ist, gilt $r_{t+1}(z) = 1 + r_t(z)$ und $r_{t+1}(x) = 1 + r_t(x)$. Insgesamt finden wir $1 + r_t(z) < \alpha + \alpha s_2(y, t)$ und

$$r_t(z) < \alpha s_2(y, t) + \alpha - 1 \leq \alpha s_2(y, t)\ ,$$

was im Widerspruch zu (1.9) steht. Andererseits kann das Gewicht der Inversion auch nicht abnehmen, deshalb bleibt $w(z, y) = 1$. □

Wir setzen nun voraus, daß jedes Element schon mindestens einmal nachgefragt wurde. Dies kann immer sichergestellt werden, indem man vor die Zugriffsfolge σ ein Präfix der Länge n setzt, für jedes Listenelement einen Zugriff. Die gesamten Suchkosten ändern sich dadurch nur um einen Summanden, und der kompetitive Faktor bleibt gleich.

Sei der aktuelle Zugriff nun auf das Element $\sigma_t = x$ gerichtet. Wir definieren die Menge $\mathcal{A}$ als die Menge aller z, so daß $\langle z, x\rangle$ in L keine Inversion ist; die Menge $\mathcal{B}$ als die Zusammenfassung aller z, so daß $\langle z, x\rangle$ in L eine Inversion ist; und schließlich die Menge $\mathcal{C}$, die aus allen z besteht, für die $\langle x, z\rangle$ in L eine Inversion ist. Man beachte, daß alle drei Mengen paarweise disjunkt sind. Setze $A = |\mathcal{A}|$, $B = |\mathcal{B}|$ und $C = |\mathcal{C}|$.

Für die tatsächlichen Kosten von SBR(α) zum Zeitpunkt t gilt SBR$(\alpha)_t = A + B + 1$, da die Elemente vor x gerade die Elemente aus $\mathcal{A} \cup \mathcal{B}$ sind., Die Kosten, die in der optimalen faktorisierten Liste entstehen, sind FOPT$_t = A + C + 1$, da genau die Mengen $\{x, z\}$ mit $z \in \mathcal{A} \cup \mathcal{C}$ einen Beitrag dazu bringen.

Jetzt betrachten wir die Potentialänderung, die bei der Bearbeitung von $\sigma_t = x$ durch SBR(α) entsteht. Wenn ein Element $z \in \mathcal{B}$ von x überholt wird, dann war das Gewicht der Inversion $\langle z, x\rangle$ in L gleich 1, und ist jetzt eliminiert. Wenn ein Element $z \in \mathcal{B}$ nicht von x überholt wurde, dann war das Gewicht der Inversion $\langle z, x\rangle$ gleich 2, und ist jetzt auf 1 reduziert. Insgesamt nimmt aufgrund der Elemente in $\mathcal{B}$ das Potential um genau B ab. Auf der anderen Seite kann durch die Aktion von SBR(α) das Gewicht jeder Inversion $\langle x, z \rangle$ mit $z \in \mathcal{C}$ um 1 steigen, was einen Zuwachs von höchstens C ausmacht.

Als nächstes untersuchen wir die Potentialänderung, die durch Aktionen des optimalen Algorithmus FOPT für die faktorisierte Liste entstehen kann. Angenommen, eine neue Inversion $\langle x, z \rangle$ in L entsteht, weil x das Element z in L überholt hat, aber nicht in L_{xz}. Dann war $z \in \mathcal{A}$ vor dem Zugriff auf x, und wir benutzen das folgenden Lemma:

Lemma 1.15. *Wenn zum Zeitpunkt t in der Liste L das Element $\sigma_t = x$ ein Element $z \in \mathcal{A}$ überholt hat, dann ist*

$$s_1(z, t) \leq s_2(x, t).$$

Wenn dieser Zugriff eine Inversion $\langle x, z \rangle$ in L geschaffen hat, und der nächste Zugriff auf $\sigma_{t+1} = z$ gerichtet ist, dann wird x wieder von z überholt werden. Mit anderen Worten, diese Inversion hat das Gewicht $w(x, z) = 1$.

Beweis. Für ein Widerspruchsargument nehmen wir an, daß $s_1(z, t) > s_2(x, t)$. Das bedeutet, daß z seit dem vorherigen Zugriff auf x (dem letzten Zugriff vor σ_t) nicht mehr nachgefragt wurde. Aber dann würde FOPT das Element x schon vor dem aktuellen Zugriff σ_t vor z in der Liste L_{xz} gesetzt haben. Das bedeutet aber $z \notin \mathcal{A}$, ein Widerspruch zur Voraussetzung.

Sei $\sigma_{t+1} = z$. Wir wollen zeigen, daß $r_{t+1}(z) \leq r_{t+1}(x)$. Aber es ist $r_{t+1}(z) = \alpha s_2(z, t+1) = \alpha s_1(z, t)$ und $r_{t+1}(x) = 1 + r_t(x) = 1 + (1 - \alpha) s_1(x, t) + \alpha s_2(x, t)$. Aus dem oben gezeigten $s_1(z, t) \leq s_2(x, t)$ sowie aus $s_1(x, t) = 0$ folgern wir $r_{t+1}(z) \leq \alpha s_2(x, t) < r_{t+1}(x)$, also wird z wieder vor x gesetzt werden. □

Als nächstes betrachten wir den Fall, daß eine neue Inversion $\langle x, z \rangle$ in L entsteht, weil z von x in L_{xz} überholt wurde, aber nicht in L. Dann war $z \in \mathcal{A}$ vor dem Zugriff, und das Gewicht der neuen Inversion kann nur 1 sein, da ein darauffolgender Zugriff auf $\sigma_{t+1} = x$ wieder x an den Listenkopf setzt und die Inversion damit beseitigt. Insgesamt ergibt sich für die Aktionen von FOPT, daß das Gewicht der neuen Inversionen höchstens A sein kann.

Zusammengefaßt gilt für die Potentialänderung durch das Bearbeiten von σ_t, daß

$$\Phi_t - \Phi_{t-1} \leq C - B + A .$$

Die amortisierten Kosten für SBR(α) beim Zugriff auf σ_t sind somit

$$\begin{aligned} a_t &= \mathrm{SBR}(\alpha)_t \ + \ \Phi_t - \Phi_{t-1} \\ &\leq A + B + 1 \ + \ C - B + A \\ &= A + C + 1 + A \\ &= (2 - (C+1)/(A+C+1)) \cdot (A + C + 1) \\ &\leq (2 - 1/n) \cdot \mathrm{FOPT}_t. \end{aligned}$$

Damit ist der Beweis, daß SBR(α) 2-kompetitiv ist, abgeschlossen. □

Als nächstes betrachten wir den Fall $\alpha > 1$ und zeigen, daß bei hinreichend großen α die SBR(α)-Verfahren nicht mehr c-kompetitiv für beliebige c sind.

Satz 1.16. *Für* $c \geq 2$ *und*

$$\alpha > 2 \cdot \left(2c - 1 + \frac{2c}{c+2} \cdot \left(2c^2 + 5c + 2\right)\right)$$

ist $\mathrm{SBR}(\alpha)$ *nicht mehr* c*-kompetitiv.*

Beweis. Wir geben eine spezielle Zugriffsfolge σ an und untersuchen die dabei entstehenden Kosten. Dazu teilen wir die n Listenelemente in $a_1, \dots, a_m$ und $b_1, \dots, b_{n-m}$ auf. Die b_j dienen als Füller, um die Kosten der Zugriffe auf die a_i hoch zu machen.

$$\sigma = a_1, \dots, a_m, b_1, b_1, \dots, b_m, b_m, \underbrace{a_1, \dots, a_m}_{1}, \underbrace{a_1, \dots, a_m}_{2}, \dots, \underbrace{a_1, \dots, a_m}_{k+1}.$$

Nach den Zugriffen auf die b_j stehen diese vor den a_i und falls α hinreichend groß ist, werden sie auch für die $k+1$ folgenden Zugriffe auf a_i dort bleiben. Der Rang von a_m im k-ten Zugriffsblock ist $r(a_m) = m\alpha$. Zu diesem Zeitpunkt gilt $r(b_1) = km + 2(n - m - 1) + \alpha$. Also wird a_m nicht vor b_1 (und damit auch nicht vor die anderen b_j) gesetzt werden, falls

$$\alpha > \frac{km + 2(n - m - 1)}{m - 1}.$$

Die Kosten $C(\sigma)$ von $\mathrm{SBR}(\alpha)$ sind mindestens die Kosten der Zugriffe auf die a_i, also $C(\sigma) \geq m(k+1)(n-m+1)$. Ein optimaler Offline-Algorithmus produziert Kosten von höchstens $mn + (n - m)(n + 1) + mn + km(m + 1)/2$, so daß

$$\frac{C(\sigma)}{\mathrm{OPT}(\sigma)} \geq \frac{(k+1)(n-m+1)m}{n^2 + mn + m - n + km(m+1)/2} \overset{!}{>} c. \tag{1.10}$$

Für große k und n ist $2(n - m)/(m + 1)$ der bestimmende Anteil dieses Ausdrucks. Wir setzen also $n = m(1 + c)$ fest.

Betrachten wir speziell $m = 2$, so folgt, daß $k \geq (4c^3 + 10c^2 + 4c)/(c+2)$ hinreichend ist für (1.10). Durch Einsetzen in $\alpha > 2(k+n-3)$ erhält man den Satz. Speziell für $c = 2$ wird (1.10) mit $m = 2$, $n = 7$, $k = 18$ erfüllt, und man erhält $\alpha > 44$. Andere Werte m bringen hier keine Verbesserung. $\square$

Somit folgt für $\alpha > 44$, daß $\mathrm{SBR}(\alpha)$ nicht mehr 2-kompetitiv ist. Für Parameter $1 < \alpha < 44$ vermuten wir, daß der kompetitive Faktor ebenfalls > 2 ist.

Da die $\mathrm{SBR}(\alpha)$-Verfahren nicht die paarweise Unabhängigkeit erfüllen, ist es schwierig, ihr Verhalten auf gedächtnislosen stochastischen Quellen zu untersuchen. Ein Ansatz ist folgendermaßen möglich. Für jedes Paar x, y wollen wir die asymptotische Wahrscheinlichkeit bestimmen, daß x vor y steht. Dazu betrachten wir folgende zeitdiskrete Markov-Kette mit den Zuständen $(s_1(x), s_2(x), s_1(y), s_2(y)) \in \mathbb{N}_0^4$. Die Übergangswahrscheinlichkeiten sind gegeben durch

$$(a, b, c, d) \longmapsto \left\{ \begin{array}{ccc} (a+1, b+1, c+1, d+1) & \text{mit Wahrscheinlichkeit} & 1 - p_x - p_y \\ (0, a+1, c+1, d+1) & \text{mit Wahrscheinlichkeit} & p_x \\ (a+1, b+1, 0, c+1) & \text{mit Wahrscheinlichkeit} & p_y \end{array} \right\}.$$

Die asymptotische Wahrscheinlichkeit, daß x vor y steht, wenn $\mathrm{SBR}(\alpha)$ angewendet wird, ist dann

$$b(x,y) = \sum_{\substack{(a,b,c,d) \\ (1-\alpha)a+\alpha b<(1-\alpha)c+\alpha d}} \pi(a,b,c,d),$$

wobei $\pi(a,b,c,d)$ die stationäre Wahrscheinlichkeit des Zustands (a,b,c,d) bezeichnet. Wir wissen nicht, wie man diese Werte allgemein berechnen kann.

1.4.3. Analyse der SORT-BY-DELAY-Verfahren

In diesem Abschnitt untersuchen wir die SBD(k)-Regeln. Es stellt sich heraus, daß sie die Bedingung der paarweisen Unabhängigkeit erfüllen, so daß auch ihr Verhalten auf stochastischen Quellen der Analyse zugänglich ist. Zuerst aber betrachten wir den kompetitiven Faktor der SBD(k)-Regeln.

Verhalten auf beliebigen Eingabefolgen

Satz 1.17. *Für alle $k \geq 2$ ist SBD(k) k-kompetitiv.*

Der Beweis verläuft ähnlich wie der obige Beweis zu Satz 1.13. Jetzt definieren wir für jedes Paar $x \neq y$ die Gewichtsfunktion

$$w(x,y) = \left\{ \begin{array}{ccll} 0 & : & \text{falls } \langle x,y\rangle & \text{keine Inversion ist} \\ \ell & : & \text{falls } \langle x,y\rangle & \text{Inversion ist und } x \text{ von } y \text{ in } L \text{ überholt wird,} \\ & & & \text{genau dann wenn jetzt } \ell \text{ mal auf } y \text{ zugegriffen} \\ & & & \text{wird, } \ell = 1,\ldots,k \end{array} \right\},$$

und die Potentialfunktion $\Phi = \sum_{x\neq y} w(x,y)$. Zuerst zeigen wir wieder, daß das Gewicht einer Inversion $\langle z,y\rangle$ nicht größer werden kann, wenn auf ein anderes Element x zugegriffen wird.

Lemma 1.18. *Seien y und z zwei Elemente, so daß $\langle z,y\rangle$ eine Inversion in L mit dem Gewicht $w(z,y) = w$ ist, $1 \leq w \leq k$. Sei x ein von z und y verschiedenes Element. Dann bleibt $w(z,y) = w$, nachdem eine Nachfrage nach x von SBD(k) bearbeitet wurde.*

Beweis. Wenn zum Zeitpunkt t gilt, daß $w(z,y) = w$, dann wird z von y überholt werden, wenn jetzt w mal auf y zugegriffen wird. Das bedeutet also $s_k(y,t+w-1) \leq s_k(z,t+w-1)$ im Falle $\sigma_j = y$ für $j = t,\ldots,t+w-1$. Mit Hilfe von $s_k(y,t+w-1) = w-1+s_{k-w+1}(y,t)$ und $s_k(z,t+w-1) = w-1+s_k(z,t)$ erhalten wir daraus

$$s_{k-w+1}(y,t) \leq s_k(z,t). \tag{1.11}$$

Nehmen wir an, daß $w(z,y) \geq w+1$ gelten würde, nachdem ein Zugriff $\sigma_t = x$ bearbeitet wurde. Das bedeutet, daß y nicht z überholen wird, wenn es jetzt w mal nachgefragt wird, d.h. $s_k(y,t+w) > s_k(z,t+w)$ im Falle $\sigma_j = y$ für $j = t+1,\ldots,t+w$. Aber dann gilt $s_k(y, t+w) = w-1+s_{k-w+1}(y, t+1)$ und $s_k(z, t+w) = w-1+s_k(z, t+1)$, also zusammen $s_{k-w+1}(y, t+1) > s_k(z, t+1)$. Wegen $\sigma_t \neq y$ und $\sigma_t \neq z$ gilt $s_{k-w+1}(y, t+1) = 1+s_{k-w+1}(y, t)$ und $s_k(z, t+1) = 1+s_k(z, t)$, und insgesamt schließen wir $s_{k-w+1}(y, t) > s_k(z, t)$, was einen Widerspruch zu (1.11) darstellt. Da bei einem Zugriff auf x das Gewicht $w(z, y)$ auch nicht abnehmen kann, folgt die Aussage. □

Analog zum Beweis von Satz 1.13 setzen wir hier voraus, daß auf jedes Element mindestens k mal zugegriffen wurde. Sei $\sigma_t = x$ der aktuelle Zugriff. Wir definieren wieder die Mengen $\mathcal{A}, \mathcal{B}, \mathcal{C}$: dabei ist $\mathcal{A}$ die Menge aller Elemente z, so daß $\langle z, x\rangle$ in L keine Inversion darstellt; die Menge $\mathcal{B}$ umfaßt alle z für die $\langle z, x\rangle$ in L eine Inversion ist, und $\mathcal{C}$ ist die Menge aller Elemente z, so daß $\langle x, z\rangle$ in L eine Inversion darstellt. Sei wieder $A = |\mathcal{A}|$, $B = |\mathcal{B}|$ und $C = |\mathcal{C}|$.

Wie im Beweis zu Satz 1.13 sind die tatsächlichen Kosten zum Zeitpunkt t gerade $\text{SBD}(k)_t = A + B + 1$, und die Kosten in der optimalen faktorisierten Liste betragen $\text{FOPT}_t = A + C + 1$.

Jetzt betrachten wir die Potentialänderung, die bei der Bearbeitung von $\sigma_t = x$ durch SBD(k) entsteht. Wenn ein Element $z \in \mathcal{B}$ von x überholt wird, dann war das Gewicht der Inversion $\langle z, x\rangle$ in L gleich 1, und ist jetzt eliminiert. Wenn ein Element $z \in \mathcal{B}$ nicht von x überholt wurde, dann war das Gewicht der Inversion $\langle z, x\rangle$ gleich ℓ für ein $\ell = 2, \dots, k$, und ist jetzt auf $\ell - 1$ reduziert. Insgesamt nimmt aufgrund der Elemente in $\mathcal{B}$ das Potential um genau B ab. Andererseits kann durch die Aktion von SBD(k) das Gewicht jeder Inversion $\langle x, z\rangle$ mit $z \in \mathcal{C}$ um 1 steigen, was einen Zuwachs von höchstens C ausmacht.

Als nächstes untersuchen wir die Potentialänderung, die durch Aktionen des optimalen Algorithmus FOPT für die faktorisierte Liste entstehen kann. Angenommen, eine neue Inversion $\langle x, z\rangle$ in L entsteht, weil x das Element z in L überholt hat, aber nicht in L_{xz}. Dann war $z \in \mathcal{A}$ vor dem Zugriff auf x, und wir benutzen das folgenden Lemma:

Lemma 1.19. *Wenn zum Zeitpunkt t in der Liste L das Element $\sigma_t = x$ ein Element $z \in \mathcal{A}$ überholt hat, dann ist*

$$s_1(z, t) \leq s_k(x, t).$$

Wenn dieser Zugriff eine Inversion $\langle x, z\rangle$ in L geschaffen hat, und die nächsten $k-1$ Zugriffe auf z gerichtet sind ($\sigma_j = z$ für $j = t+1, \dots, t+k-1$), dann wird x wieder von z überholt werden. Mit anderen Worten, diese Inversion hat ein Gewicht $w(x, z) \leq k-1$.

Beweis. Für das Widerspruchsargument nehmen wir an, daß $s_1(z, t) > s_k(x, t)$. Das bedeutet, daß z seit dem k-letzten Zugriff auf x nicht mehr nachgefragt wurde. Aber dann würde natürlich FOPT das Element x schon vor dem aktuellen Zugriff $\sigma_t = x$ vor z in der Liste L_{xz} gesetzt haben. Das ergibt den Widerspruch $z \notin \mathcal{A}$ zur Voraussetzung.

Sei $\sigma_j = z$ für $j = t+1, \dots, t+k-1$. Wir wollen zeigen, daß $s_k(z, t+k-1) \leq s_k(x, t+k-1)$. Aber wegen $s_k(z, t+k-1) = k-1+s_1(z, t)$ und $s_k(x, t+1) = k-1+s_k(x, t)$ folgt dies aus dem ersten Teil des Lemmas. □

Als nächstes betrachten wir den Fall, daß eine neue Inversion $\langle x, z\rangle$ in L entsteht, weil z von x in L_{xz} überholt wurde, aber nicht in L. Dann war $z \in \mathcal{A}$ vor dem Zugriff, und das Gewicht der neuen Inversion kann höchstens $k-1$ sein, da $k-1$ darauffolgende Zugriffe auf x wieder x an den Listenkopf setzen und die Inversion damit beseitigen. Insgesamt ergibt sich für die Aktionen von FOPT, daß das Gewicht der neuen Inversionen höchstens $(k-1) \cdot A$ sein kann.

Zusammengefaßt gilt für die Potentialänderung durch das Bearbeiten von σ_t, daß

$$\Phi_t - \Phi_{t-1} \leq C - B + (k-1) \cdot A \ .$$

Die amortisierten Kosten für SBR(α) beim Zugriff auf σ_t sind somit

$$\begin{aligned} a_t &= \text{SBR}(\alpha)_t \ + \ \Phi_t - \Phi_{t-1} \\ &\leq A + B + 1 \ + \ C - B + (k-1) \cdot A \\ &= k \cdot A + C + 1 \\ &= (k - (k-1) \cdot (C+1)/(A+C+1)) \cdot (A+C+1) \\ &\leq (k - (k-1)/n) \cdot \text{FOPT}_t. \end{aligned}$$

Wir schließen, daß SBD(k) k-kompetitiv ist. □

Wie man anhand der Anfragefolge

$$\sigma = (x_1^k \, x_2^k \, x_3^k \ \cdots \ x_n^k)^m$$

sieht, ist k auch eine untere Schranke für den kompetitiven Faktor von SBD(k). Denn ein nachgefragtes x_i wird immer gerade dann erst an den Listenanfang gesetzt, wenn es zu spät ist. Die Kosten von SBD(k) sind $C(\sigma) \geq kn^2(m-1)$ (beim ersten Durchgang wird direkt nach vorne geschoben), und die Kosten von OPT=MTF sind $\text{OPT}(\sigma) \leq (n + k - 1)nm$. Für das Verhältnis folgt

$$\frac{C(\sigma)}{\text{OPT}(\sigma)} \geq \frac{kn(m-1)}{(n+k-1)m} \ \longrightarrow \ k$$

für $n \to \infty$, $m \to \infty$.

Verhalten auf stochastischen Quellen

Wir zeigen zunächst, daß die SORT-BY-DELAY-Verfahren die Bedingung der paarweisen Unabhängigkeit erfüllen, und benutzen dieses Ergebnis dann, um erwartete Suchkosten zu berechnen. Zuerst ist eine alternative Definition der SBD(k)-Regeln hilfreich:

Lemma 1.20. *Eine zur Definition 1.3 äquivalente Charakterisierung der SBD(k)-Verfahren ist folgende:*

> *Wenn x nachgefragt wird, dann setze x vor das erste Element y, das vor x steht, und das seit dem $(k-1)$-letzten Zugriff auf x höchstens $k-1$ mal nachgefragt wurde. Setze x an den Listenanfang, wenn kein solches Element y existiert.*

Beweis. Man beachte, daß für alle festen k und t und Elemente $x \neq y$ stets gilt, daß $s_k(x,t) \neq s_k(y,t)$, also sind die Ränge immer eindeutig. Sei $\sigma_t = x$. Dann ist $s_k(x,t) = 1 + s_{k-1}(x,t-1)$ und $s_k(y,t) = 1 + s_k(y,t-1)$ für alle $y \neq x$.

Sei y ein Element vor x, so daß x gemäß der Definition von SBD(k) auf eine Position hinter y gesetzt wird. Das bedeutet $s_k(y,t) < s_k(x,t)$, also $s_k(y,t-1) < s_{k-1}(x,t-1)$. Wir schließen, daß x nicht vor ein Element y gesetzt werden kann, daß seit dem $(k-1)$-letzten Zugriff auf x mindestens k mal nachgefragt wurde, wie im Lemma behauptet.

Sei andererseits z ein Element, so daß x vor z eingefügt wird, also $s_k(x,t) < s_k(z,t)$, d.h. $s_{k-1}(x,t-1) < s_k(z,t-1)$. Deshalb wurde z seit dem $(k-1)$-letzten Zugriff auf x weniger als k mal nachgefragt. □

Die Eigenschaft der paarweisen Unabhängigkeit folgt aus folgendem

Lemma 1.21. *Wenn auf x zugegriffen wird, dann stehen alle Elemente, die seit dem $(k-1)$-letzten Zugriff auf x mindestens k mal nachgefragt wurden, vor allen Elementen, die seit dem $(k-1)$-letzten Zugriff auf x weniger als k mal nachgefragt wurden.*

Die Elemente, die höchstens $(k-1)$ mal nachgefragt wurden, bilden also einen zusammenhängenden Block vor x. Bei einem Zugriff überholt x diesen gesamten Block, und es ist äquivalent, ob x vor das erste Element vor x gesetzt wird, das höchstens $(k-1)$ mal nachgefragt wurde, oder hinter das letzte Element vor x, das mindestens k mal nachgefragt wurde. Das bedeutet, daß man aus den Werten $s_k(x,t)$ und $s_k(y,t)$ die relative Ordnung von x und y ableiten kann, ohne die Positionen der anderen Elemente kennen zu müssen.

Beweis. Sei v_x das Pivot-Element für x, d.h. wenn x nachgefragt wird, dann wird es direkt vor v_x gesetzt. Wir wollen zeigen, daß alle Listenelemente vor v_x mindestens k mal seit dem $(k-1)$-letzten Zugriff auf x nachgefragt wurden, und alle Elemente zwischen v_x (einschließlich) und x weniger als k mal seit diesem Zeitpunkt nachgefragt wurden. Sei der aktuelle Zugriff $\sigma_t = x$. Nach der Bearbeitung dieses Zugriffs ist

$$s_k(y,t) < s_k(x,t) \leq s_k(v_x,t) \leq s_k(z,t)$$

für alle Elemente y vor v_x und alle Elemente z hinter v_x (oder v_x selbst). Also muß vor dem Zugriff auf σ_t gegolten haben:

$$s_k(y,t-1) < s_{k-1}(x,t-1) \leq s_k(v_x,t-1) \leq s_k(z,t-1)\ ,$$

was aus $s_k(x,t) = 1 + s_{k-1}(x,t-1)$ und $s_k(w,t) = 1 + s_k(w,t-1)$ für alle Elemente $w \neq \sigma_t$ folgt. Aber daraus folgt unmittelbar die Behauptung. □

Die Aussage des Lemmas sieht man auch folgendermaßen ein: durch Projektion von σ auf σ_{xy} werden alle Elemente verschieden von x und y gelöscht. Die absoluten Werte der Rangfunktionen $s_k(z,t)$ ändern sich dadurch, aber die relative Ordnung der Ränge bleibt erhalten. Und nur die relative Ordnung der k-letzten Zugriffe bestimmt das aktuelle Aussehen der Liste.

Die Aussage des Lemmas wenden wir jetzt auf σ_{xy} an und erhalten Informationen über die relative Ordnung von x und y.

Lemma 1.22. *Wir betrachten einen beliebigen Zeitpunkt in der Zugriffsfolge, an dem schon mindestens $2k-1$ Zugriffe auf x oder y stattgefunden haben. Dann steht bei Anwendung der* SBD(k)*-Regel das Element x genau dann vor y in der Liste, wenn die Mehrheit der letzten $2k-1$ Zugriffe auf x oder y (d.h. mindestens k) auf das Element x gerichtet waren.*

Beweis. Wenn der Suffix von σ_{xy} der Länge $2k-1$ mindestens k Zugriffe auf x enthält, dann kann er höchstens $k-1$ Nachfragen für y enthalten. Daraus folgt, daß der k-letzte Zugriff auf x noch innerhalb dieses Suffixes liegt, aber der k-letzte Zugriff auf y nicht mehr, also früher aufgetreten sein muß. Deshalb wurde x beim letzten Zugriff vor y gesetzt. Wenn umgekehrt höchstens $k-1$ Zugriffe auf x in dem Suffix enthalten sind, dann folgt mit der gleichen Argumentation, daß y vor x stehen muß. □

Stochastische Quellen ohne Gedächtnis. Wir betrachten den Fall, daß die Zugriffe unabhängig und identisch verteilt mit einer Verteilung $\mathbf{p} = (p_1, \dots, p_n)$ erfolgen, dabei sei $p_x > 0$ für alle x. Aus Lemma 1.22 ergibt sich die asymptotische Wahrscheinlichkeit $b(y, x)$, daß y vor x in der von SBD(k) verwalteten Liste steht, als

$$b(y,x) = \sum_{\ell=0}^{k-1} \binom{2k-1}{\ell} \frac{p_x^{\ell} p_y^{2k-1-\ell}}{(p_x+p_y)^{2k-1}} \,. \tag{1.12}$$

Die erwarteten Suchkosten unter der Verteilung $\mathbf{p}$ erhält man dann aus

$$\begin{aligned}
E_{\mathrm{SBD}(k)}(\mathbf{p}) &= \sum_{x=1}^{n} p_x \cdot (1 + \sum_{y \neq x} b(y,x)) \\
&= 1 + \sum_{x} \sum_{y \neq x} \sum_{\ell=0}^{k-1} \binom{2k-1}{\ell} \frac{p_x^{\ell+1} p_y^{2k-1-\ell}}{(p_x+p_y)^{2k-1}} \\
&= 1/2 + \sum_{x} \sum_{y} \sum_{\ell=0}^{k-1} \binom{2k-1}{\ell} \frac{p_x^{\ell+1} p_y^{2k-1-\ell}}{(p_x+p_y)^{2k-1}} \\
&= \sum_{1 \le x \le y \le n} \sum_{\ell=0}^{k-1} \binom{2k-1}{\ell} \frac{p_x^{\ell+1} p_y^{2k-1-\ell} + p_x^{2k-1-\ell} p_y^{\ell+1}}{(p_x+p_y)^{2k-1}}.
\end{aligned}$$

Satz 1.23. *Für unabhängige, identisch verteilte Zugriffe mit der Wahrscheinlichkeitsverteilung $(p_1, \dots, p_n)$, $p_x > 0$ für alle x, gilt für die asymptotischen erwarteten Suchkosten der* SBD(k)*-Regeln*

$$E_{\mathrm{SBD}(k)}(\mathbf{p}) = \sum_{1 \le x \le y \le n} \sum_{\ell=0}^{k-1} \binom{2k-1}{\ell} \frac{p_x^{\ell+1} p_y^{2k-1-\ell} + p_x^{2k-1-\ell} p_y^{\ell+1}}{(p_x+p_y)^{2k-1}}.$$

Für $k = 1$ (Move-to-Front) wurden die erwarteten Suchkosten in [M65] hergeleitet, und für $k = 2$ (Timestamp) in [AM98]. Als nächstes wollen wir die erwarteten Suchkosten mit den optimalen Suchkosten $M(\mathbf{p})$ vergleichen, die ein optimaler statischer Gegner erzielen kann. Für $p_1 \ge p_2 \ge \cdots \ge p_n$ ist $M(\mathbf{p}) = \sum_{i=1}^{n} i p_i$.

Satz 1.24. *Die Familie* SBD(k) *ist asymptotisch optimal, d.h. für alle Verteilungen* $\mathbf{p}$ *gilt*

$$E_{\mathrm{SBD}(k)}(\mathbf{p}) \longrightarrow M(\mathbf{p}) \qquad \text{für } k \to \infty.$$

Genauer gilt, daß für alle Verteilungen das Verhältnis der Kosten

$$\begin{aligned} E_{\mathrm{SBD}(k)}(\mathbf{p})/M(\mathbf{p}) &\leq \max_{x\geq 1} \sum_{\ell=0}^{k-1} \binom{2k-1}{\ell} \frac{x^{2k-1-\ell}+x^{\ell+1}}{(1+x)^{2k-1}} \\ &\leq 1 + \frac{1}{\sqrt{\pi k}} + O(k^{-3/2}) \end{aligned} \tag{1.13}$$

beträgt. Insbesondere ergeben sich für kleine Werte k *die folgenden Schranken:*

$$\begin{aligned} E_{\mathrm{SBD}(1)}(\mathbf{p})/M(\mathbf{p}) &\leq 1.5708 \\ E_{\mathrm{SBD}(2)}(\mathbf{p})/M(\mathbf{p}) &\leq 1.3156 \\ E_{\mathrm{SBD}(3)}(\mathbf{p})/M(\mathbf{p}) &\leq 1.2176 \\ E_{\mathrm{SBD}(4)}(\mathbf{p})/M(\mathbf{p}) &\leq 1.1734 \\ E_{\mathrm{SBD}(5)}(\mathbf{p})/M(\mathbf{p}) &\leq 1.1473. \end{aligned}$$

Für $k = 1$ (MTF) ist das Verhältnis durch $\pi/2 \leq 1.5708$ beschränkt [CHS88], und für $k = 2$ (TS), wurde die Schranke 1.34 in [AM98] hergeleitet. Wir verallgemeinern die dort angewendete Technik auf beliebige $k \geq 1$, stellen aber fest, daß mit einer anderen einfacheren Technik noch bessere Schranken gezeigt werden können.

Die SBD(k)-Verfahren sind zwar nicht kompetitiv-optimal, aber sie weisen einen konstanten kompetitiven Faktor (unabhängig von der Listenlänge) auf. Außerdem erzielen sie bei unabhängigen Zugriffen Suchzeiten, die beliebig nahe an der optimalen erwarteten Suchzeit liegen. Meines besten Wissens ist SBD(k) die einzige Familie von Listenalgorithmen mit dieser Eigenschaft. Alle anderen Familien, die asymptotisch optimale erwartete Suchkosten aufweisen, z.B. FREQUENCY-COUNT, SIMPLE(k)-MTF, BATCHED(k)-MTF, haben für alle $k \geq 2$ einen kompetitiven Faktor $\Omega(n)$ bei einer Listenlänge n.

Beweis. Was die asymptotische Optimalität im Fall $k \to \infty$ betrifft, so genügt es zu zeigen, daß

$$b(y, x) \to 0 \qquad \text{für alle } x, y \text{ mit } p_x > p_y ,$$

da in einer optimalen Liste die Elemente nach absteigenden Zugriffswahrscheinlichkeiten sortiert sind. Wir setzen $p = p_x/(p_x + p_y)$ und $q = p_y/(p_x + p_y)$, und benutzen die folgende Chernoff-Schranke [HR89, Formel (12)]

$$b(y, x) = \sum_{\ell=0}^{k-1} \binom{2k-1}{\ell} p^{l} q^{2k-1-\ell} \leq \left(\frac{(2k-1)q}{k}\right)^{k} \cdot e^{k-(2k-1)q} ,$$

was möglich ist, da $q < 1/2$. Im Falle $k \to \infty$ geht die rechte Seite der Ungleichung gegen 0, woraus die Behauptung folgt.

Um das Verhältnis der Kosten von SBD(k) zu den optimalen Kosten $M(\mathbf{p})$ für kleine Werte abzuschätzen, benutzen wir zunächst ein Verfahren, das auf der Ungleichung von Hilbert basiert und für den Fall $k = 1$ (MOVE-TO-FRONT) in [CHS88] erstmals angewendet wurde.

Ungleichung von Hilbert
Seien $p, q > 1$ mit $\frac{1}{p} + \frac{1}{q} = 1$ und sei $K(x, y)$ eine nichtnegative Funktion, die homogen vom Grad -1 ist. Wenn gilt

$$\int_0^\infty K(x,1)x^{-1/p}\,dx = \int_0^\infty K(1,y)y^{-1/q}\,dy = C\ ,$$

dann ist für jede nichtnegative Funktion f

$$\int_0^\infty \left(\int_0^\infty K(x,y)f(x)\,dx\right)^p dy \ \le\ C^p \int_0^\infty f^p(x)\,dx\ .$$

Zum Beweis siehe [HLP52, Seiten 229-232].

Lemma 1.25. *Sei* f *auf* $(0, \infty)$ *integrierbar mit* $\int_0^\infty f(x)dx = 0$. *Sei* $G(x, y)$ *homogen vom Grad* 1, $H = \frac{\partial^2 G}{\partial x\,\partial y}$ *und* $H^+(x, y) = \max\{H(x, y), 0\}$. *Dann gilt*

$$\frac{\int_0^\infty \int_0^\infty G(x,y)f(x)f(y)\,dx dy}{\int_0^\infty \int_0^\infty \min(x,y)f(x)f(y)\,dx dy} \le \int_0^\infty H^+(x,1)x^{-1/2}dx\ .$$

Der Beweis dieser Ungleichung steht in [AM98] und benutzt als wesentlichen Bestandteil die Ungleichung von Hilbert.

Seien $p_1 \geq p_2 \geq \cdots \geq p_n > 0$ die Zugriffswahrscheinlichkeiten. Wegen $M(\mathbf{p}) = \sum_i ip_i = \frac{1}{2} + \frac{1}{2}\sum_{i,j}\min(p_i, p_j)$ gilt für das Verhältnis

$$\begin{aligned}
\frac{E_{\mathrm{SBD}(k)}(\mathbf{p})}{M(\mathbf{p})} &= \frac{\frac{1}{2} + \sum_{i,j}\sum_{\ell=0}^{k-1}\binom{2k-1}{\ell}\frac{p_i^{\ell+1}p_j^{2k-1-\ell}+p_i^{2k-1-\ell}p_j^{\ell+1}}{2(p_i+p_j)^{2k-1}}}{\frac{1}{2} + \frac{1}{2}\sum_{i,j}\min(p_i,p_j)} \\
&\le \frac{\sum_{i,j}\sum_{\ell=0}^{k-1}\binom{2k-1}{\ell}\frac{p_i^{\ell+1}p_j^{2k-1-\ell}+p_i^{2k-1-\ell}p_j^{\ell+1}}{(p_i+p_j)^{2k-1}}}{\sum_{i,j}\min(p_i,p_j)}\ .
\end{aligned}$$

Mit folgendem Trick kann die Aussage des obigen Lemmas für den kontinuierlichen Fall auf den diskreten Fall übertragen werden. Seien $0 < x_1 < x_2 < \cdots < x_n$ und $0 < \delta < \min_{i \neq j}|x_i - x_j|$. Sei f_δ die Funktion mit $f_\delta = 1$ in Intervallen der Länge δ zentriert in den Punkten x_i, $i = 1, \dots, n$ und 0 sonst. Dann gilt mit $G(x, y) = \sum_{\ell=0}^{k-1}\binom{2k-1}{\ell}\frac{x^{\ell+1}y^{2k-1-\ell}+x^{2k-1-\ell}y^{\ell+1}}{(x+y)^{2k-1}}$, daß

$$\frac{\sum_{i,j}\sum_{\ell=0}^{k-1}\binom{2k-1}{\ell}\frac{p_i^{\ell+1}p_j^{2k-1-\ell}+p_i^{2k-1-\ell}p_j^{\ell+1}}{(p_i+p_j)^{2k-1}}}{\sum_{i,j}\min(p_i,p_j)} \ \le\ \lim_{\delta\to 0}\frac{\int_0^\infty \int_0^\infty G(x,y)f_\delta(x)f_\delta(y)\,dx dy}{\int_0^\infty \int_0^\infty \min(x,y)f_\delta(x)f_\delta(y)\,dx dy}.$$

Jetzt kann das obige Lemma angewendet werden, und wir berechnen das jeweilige Integral numerisch mit Hilfe von MAPLE. Wir haben die folgenden Summanden der Kostenfunktion für SBD(k):

$$\begin{aligned}
k=2:&\quad G(x,y)=\tfrac{xy(x^2+6xy+y^2)}{(x+y)^3}\\
k=3:&\quad G(x,y)=\tfrac{xy(x^4+5x^3y+20x^2y^2+5xy^3+y^4)}{(x+y)^5}\\
k=4:&\quad G(x,y)=\tfrac{xy(x^6+7x^5y+21x^4y^2+70x^3y^3+21x^2y^4+7xy^5+y^6)}{(x+y)^7}\\
k=5:&\quad G(x,y)=\tfrac{xy(x^8+9x^7y+36x^6y^2+84x^5y^3+252x^4y^4+84x^3y^5+36x^2y^6+9xy^7+y^8)}{(x+y)^9}\,.
\end{aligned}$$

Beispielsweise benutzen wir für $k=4$ folgendes MAPLE-Programmstück:

```
> readlib(`evalf/int`):

> G := x*y * (x^6 + 7*x^5*y + 21*x^4*y^2 + 70*x^3*y^3 + 21*x^2*y^4 +
>               7*x*y^5 + y^6 ) / (x+y)^7;

                6       5           4   2        3   3        2   4         5    6
         x y (x  + 7 x  y + 21 x  y  + 70 x  y  + 21 x  y  + 7 x y  + y )
    G := ----------------------------------------------------------------
                                                 7
                                          (x + y)

> H := simplify( diff(G,x,y) );

                               3   3     2                2
                              x   y  (3 x  - 10 x y + 3 y )
                  H := - 140 ----------------------------
                                                  9
                                           (x + y)

> y := 1: H1 := simplify(H);
                                   3     2
                                  x  (3 x  - 10 x + 3)
                      H1 := - 140 --------------------
                                                 9
                                          (x + 1)

> s := `evalf/int`( H1*x^(-1/2), x=1/3 .. 3, Digits, _NCrule );

                               s := 1.290343897 ,
```

wobei die Nullstellen 1/3 und 3, zwischen denen der positive Teil von H1 liegt, vorher bestimmt worden waren.

Im folgenden geben wir eine bessere Abschätzung für das Kostenverhältnis an und bestimmen auch die oben angegebene Schranke für die Konvergenzgeschwindigkeit.

Dazu stellen wir die erwarteten Suchkosten $E(k) = E_{SBD(k)}(\mathbf{p})$ anders da und schätzen das Verhältnis zu $OPT = E_{OPT}(\mathbf{p})$ direkt ab. Zunächst ist

$$\begin{aligned} E(k) &= 1 + \sum_{i=1}^{n} p_i \sum_{j \neq i} b(j,i) \\ &= \sum_{i=1}^{n} \sum_{j \leq i} p_i b(j,i) + p_j b(i,j). \end{aligned}$$

Für zwei Folgen $(a_i)_{i=1}^{n}$ und $(b_j)_{j=1}^{n}$ von positiven Zahlen beweist man mit vollständiger Induktion, daß $\sum_i a_i / \sum_j b_j \leq \max_k (a_k/b_k)$ gilt. Mit $OPT = \sum_{i=1}^{n} i p_i$ haben wir

$$\begin{aligned} \frac{E(k)}{OPT} &= \frac{\sum_{i=1}^{n} \sum_{j=1}^{i} \sum_{\ell=0}^{k-1} \binom{2k-1}{\ell} \frac{p_i^{\ell+1} p_j^{2k-1-\ell} + p_j^{\ell+1} p_i^{2k-1-\ell}}{(p_i+p_j)^{2k-1}}}{\sum_{i=1}^{n} i p_i} \\ &\leq \max_{1 \leq i \leq n} \frac{1}{i} \sum_{j=1}^{i} \sum_{\ell=0}^{k-1} \binom{2k-1}{\ell} \frac{p_i^{\ell} p_j^{2k-1-\ell} + p_j^{\ell+1} p_i^{2k-2-\ell}}{(p_i+p_j)^{2k-1}} \\ &\leq \max_{1 \leq i \leq n} \frac{1}{i} \sum_{j=1}^{i} \sum_{\ell=0}^{k-1} \binom{2k-1}{\ell} \frac{x_{ij}^{2k-1-\ell} + x_{ij}^{\ell+1}}{(1+x_{ij})^{2k-1}} \\ &\leq \max_{x \geq 1} \sum_{\ell=0}^{k-1} \binom{2k-1}{\ell} \frac{x^{2k-1-\ell} + x^{\ell+1}}{(1+x)^{2k-1}}, \end{aligned}$$

wobei $x_{ij} = p_j/p_i \geq 1$ gesetzt wurde. Im letzten Schritt haben wir dabei die Summe $\sum_{j=1}^{i}$ durch i mal das Maximum des Summanden abgeschätzt. Damit haben wir (1.13) gezeigt. Für die asymptotische Abschätzung rechnen wir weiter:

$$
\begin{aligned}
&\sum_{\ell=0}^{k-1} \binom{2k-1}{\ell} (x^{2k-1-\ell} + x^{\ell+1}) \\
&= \sum_{\ell=k}^{2k-1} \binom{2k-1}{\ell} x^{\ell} + \sum_{\ell=0}^{k-1} \binom{2k-1}{\ell} x^{\ell+1} \\
&= (1+x)^{2k-1} - \sum_{\ell=0}^{k-1} \binom{2k-1}{\ell} x^{\ell} + \sum_{\ell=0}^{k-1} \binom{2k-1}{\ell} x^{\ell+1} \\
&= (1+x)^{2k-1} + \sum_{\ell=0}^{k-1} \binom{2k-1}{\ell} (x^{\ell+1} - x^{\ell}) \\
&= (1+x)^{2k-1} + \binom{2k-1}{k-1} x^k - \sum_{\ell=1}^{k-1} \left(\binom{2k-1}{\ell} - \binom{2k-1}{\ell-1} \right) x^{\ell} - 1 \\
&\leq (1+x)^{2k-1} + \binom{2k-1}{k-1} x^k - \binom{2k-1}{k-1} \\
&\leq (1+x)^{2k-1} + \binom{2k-1}{k-1} x^k.
\end{aligned}
$$

Nach Division durch $(1+x)^{2k-1}$ erhalten wir

$$\frac{E(k)}{OPT} \leq 1 + \binom{2k-1}{k-1} \cdot \max_{x \geq 1} \frac{x^k}{(1+x)^{2k-1}}.$$

Diese Funktion hat im interessierenden Bereich ihr Maximum bei $x = k/(k-1)$, so daß

$$
\begin{aligned}
\frac{E(k)}{OPT} &\leq 1 + \binom{2k-1}{k-1} \frac{k^k (k-1)^{k-1}}{(2k-1)^{2k-1}} \\
&\leq 1 + \frac{1}{\sqrt{\pi k}} + O(k^{-3/2}).
\end{aligned}
\tag{1.14}
$$

Die asymptotische Abschätzung für $k \to \infty$ folgt aus der Stirling'schen Formel für die Fakultät

$$\frac{k!}{k^k} = \frac{\sqrt{2\pi k}}{e^k} \cdot \left(1 + \frac{1}{12k} + O(k^{-2}) \right).$$

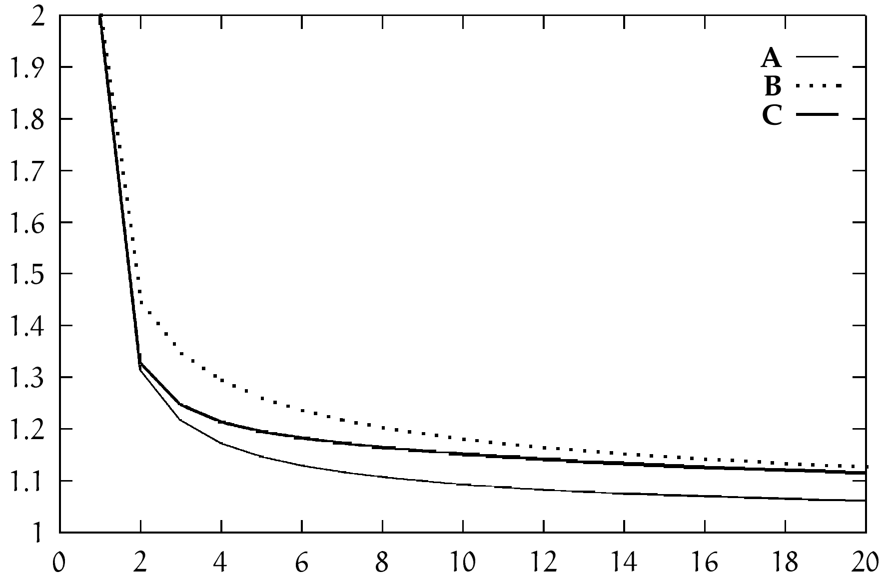
Eine etwas vorsichtigere Abschätzung als (1.14), die ebenfalls noch ein symbolisch berechenbares Maximum erlaubt (quadratische Gleichung für die Maximumsstelle), ist

$$
\begin{aligned}
\frac{E(k)}{OPT} &\leq 1 + \max_{x \geq 1} \frac{cx^k - dx^{k-1} - \binom{2k-1}{k-2}}{(1+x)^{2k-1}} \\
&\leq 1 + \max_{x \geq 1} \frac{x^{k-1}(cx-d)}{(1+x)^{2k-1}},
\end{aligned}
\tag{1.15}
$$

wobei $c = \binom{2k-1}{k-1}$ und $d = \binom{2k}{k-1}/k = \binom{2k-1}{k-1} - \binom{2k-1}{k-2}$ ist. Für kleine Werte von k ist diese Abschätzung vorteilhaft, bringt aber asymptotisch keinen Gewinn.

Die folgende Tabelle vergleicht die Schranken, die mit Hilfe der Ungleichung von Hilbert, der Berechnung des Maximums der Funktion (1.13) und deren Abschätzung (1.15) gefunden wurden. Das anschließende Diagramm stellt die Maxima **A** der Funktion (1.13) sowie die Abschätzungen (1.14), **B**, und (1.15), **C**, gegenüber. □

k	Hilbert-Ungleichung	Maximum **A**	Abschätzung **C**
1	1.5708	2	2
2	1.3391	1.3156	1.3284
3	1.3044	1.2176	1.2470
4	1.2904	1.1734	1.2143
5	1.2828	1.1473	1.1956



Wir betrachten die verallgemeinerten Zipf-Verteilungen $\mathbf{p}(\alpha)$ mit Parameter $\alpha \geq 0$, die durch

$$p_i = \frac{1}{i^\alpha \cdot H_n(\alpha)}, \qquad i = 1, \dots, n$$

definiert sind (vgl. Anhang A.1). Dabei ist der Normalisierugsfaktor $H_n(\alpha) = \sum_{j=1}^{n} j^{-\alpha}$ die n-te harmonische Zahl der Ordnung α. Diese Verteilungen sind interessant, da der Parameter α auf einfache Weise die Konzentration der Verteilung steuert: $\alpha = 0$ ergibt die Gleichverteilung, und für wachsende α nimmt die Gewichtung der kleineren Elemente immer mehr zu. Der Wert $\alpha = 1$ entspricht der Zipf-Verteilung [Z49], und mit $\alpha = 2$ erhält man die Lotka-Verteilung.

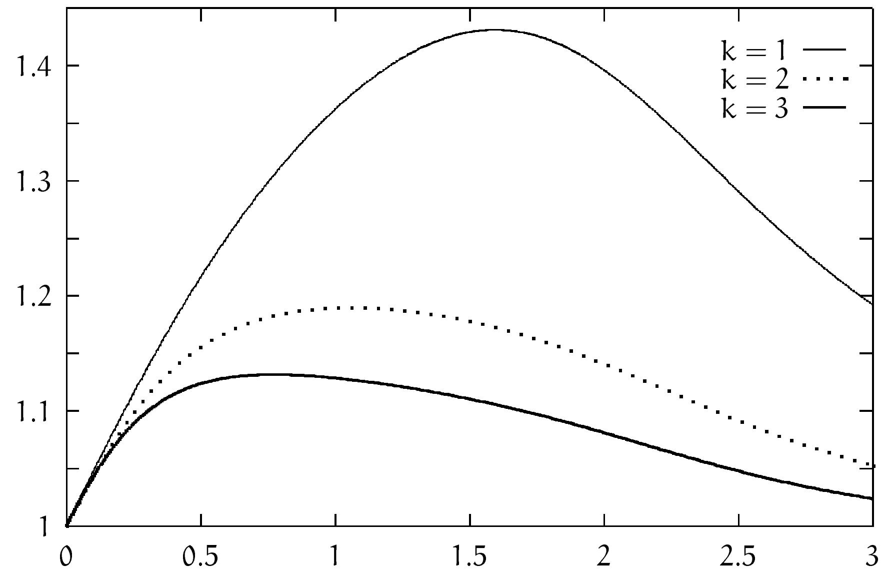
Die erwartete Suchzeit einer optimalen Listenanordnung unter der verallgemeinerten Zipf-Verteilung $\mathbf{p}(\alpha)$ ist

$$M(\mathbf{p}(\alpha)) = \sum_{j=1}^{n} jp_j = \frac{H_n(\alpha - 1)}{H_n(\alpha)}.$$

In [GMS81] wurde gezeigt, daß bei der Nach-Vorne-Schieben-Regel für $\alpha = 2$ das Verhältnis von erwarteten Kosten zu optimalen Kosten pessimal wird:

$$\frac{E_{\mathrm{MTF}}(\mathbf{p}(2))}{M(\mathbf{p}(2))} \to \frac{\pi}{2} \qquad \text{für } n \to \infty.$$

Dies zeigt insbesondere, daß die obere Schranke von $\pi/2$ scharf ist. Das folgende Diagramm gibt die Verhältnisse $E_{\mathrm{SBD}(k)}(\mathbf{p}(\alpha))/M(\mathbf{p}(\alpha))$ für $k = 1, 2, 3$ und $0 \le \alpha \le 3$ an. Bei der Berechnung wurden $n = 100$ Listenelemente zugrundegelegt. Das erklärt auch, warum bei $k = 1$ das Maximum bei 1.6 anstatt bei 2 auftritt und den Betrag 1.43 anstatt $\pi/2$ hat.



Für die Zipf-Verteilung ($\alpha = 1$) berechnen wir für kleine Werte k das Verhältnis zu den optimalen Kosten $E_{\mathrm{OPT}} = n/H_n$ asymptotisch exakt. Die Schranke $2\ln(2)$ im Fall $k = 1$ wurde in [K98] angegeben.

Satz 1.26. *Für die Zipf-Verteilung gilt asymptotisch ($n \to \infty$)*

$$\begin{aligned}
E_{\mathrm{SBD}(1)}/E_{\mathrm{OPT}} &= 2\ln(2) &\le 1.3863\\
E_{\mathrm{SBD}(2)}/E_{\mathrm{OPT}} &= 1/2 + \ln(2) &\le 1.1932\\
E_{\mathrm{SBD}(3)}/E_{\mathrm{OPT}} &= 7/16 + \ln(2) &\le 1.1307\\
E_{\mathrm{SBD}(4)}/E_{\mathrm{OPT}} &= 13/32 + \ln(2) &\le 1.1100.
\end{aligned}$$

Beweis. Wir skizzieren die Berechnung für $k = 2$, für andere Werte k verläuft sie analog. Zunächst ist

$$\begin{aligned}
E_{\text{SBD}(2)} &= \frac{1}{2} + \sum_{i=1}^{n} \sum_{j=1}^{n} p_i b(j,i) \\
&= \frac{1}{2} + \sum_{i=1}^{n} \sum_{j=1}^{n} \frac{p_i(p_j^3 + 3p_i p_j^2)}{(p_i + p_j)^3} \\
&= \frac{1}{2} + \frac{1}{H_n} \cdot \sum_{i=1}^{n} \sum_{j=1}^{n} \frac{i^2 + 3ij}{(i+j)^3} \\
&= \frac{1}{2} + \frac{1}{H_n} \cdot \left(\sum_{m=2}^{n} \sum_{i=1}^{m-1} \frac{i^2 + 3i(m-i)}{m^3} + \sum_{m=n+1}^{2n} \sum_{i=m-n}^{n} \frac{i^2 + 3i(m-i)}{m^3} \right) \\
&= \frac{1}{2} + \frac{1}{H_n} \cdot \Big((n+1/2)H_{2n} - (n+1)H_n + (2n^2 + 2n + 1/3)H_{2n}^{(2)} \\
&\qquad - (2n^2 + 2n + 2/3)H_n^{(2)} - (4n^3/3 + 2n^2 + 2n/3)(H_{2n}^{(3)} - H_n^{(3)}) \Big),
\end{aligned}$$

wobei wir $m = i + j$ gesetzt haben. Mit Hilfe der asymptotischen Entwicklung der harmonischen Zahlen

$$\begin{aligned}
H_n &= \sum_{i=1}^{n} 1/i \; = \; \ln(n) + \gamma + 1/(2n) + O(n^{-2}) \\
H_n^{(2)} &= \sum_{i=1}^{n} 1/i^2 \; = \; \pi^2/6 - 1/n + O(n^{-2}) \\
H_n^{(3)} &= \sum_{i=1}^{n} 1/i^3 \; = \; \zeta(3) + O(n^{-2})
\end{aligned}$$

folgt die Abschätzung

$$E_{\text{SBD}(2)} = (1/2 + \ln(2))n/H_n + O(1)$$

und daraus das angegebene Verhältnis zu $E_{\text{OPT}} = n/H_n$. □

Der folgende Satz 1.27 gibt eine allgemeine untere Schranke für die erwartete Suchzeit der SBD(k)-Regeln an: für beliebige $\varepsilon > 0$ gibt es eine Verteilung $\mathbf{p}$, so daß für eine Approximation der optimalen erwarteten Kosten bis auf einen Faktor $1 + \varepsilon$ mindestens

$$k \geq \ln(1/\varepsilon)/2 - 1$$

gewählt werden muß.

Satz 1.27. *Sei $\varepsilon > 0$. Es gibt eine Verteilung $\mathbf{p}$ auf $n \geq 2$ Elementen, so daß jeder Listenalgorithmus R_t, der für die Entscheidung, welches von zwei Listenelementen x, y vor das andere plaziert werden soll, nur die t letzten Zugriffe auf x oder y betrachtet, mindestens*

$$t \geq t^*(\varepsilon) := \ln(1/\varepsilon) - 3$$

wählen muß, um erwartete Kosten

$$E[R_t] \leq (1+\varepsilon) \cdot E[OPT]$$

zu erzielen.

Beweis. Wir betrachten $n = 2$ Elemente x und y mit Wahrscheinlichkeiten p_x, p_y und $p_x + p_y = 1$, die später festgelegt werden. Man beachte, daß bei unabhängigen Zugriffen die Reihenfolge keine Information enthält, so daß der Algorithmus nur die Häufigkeitsverteilung auswerten kann. Wenn $2k-1$ mal gezogen wird, dann beträgt die Wahrscheinlichkeit, genau ℓ mal ein x zu erhalten,

$$P(\ell) = \binom{2k-1}{\ell} p_x^{\ell} p_y^{2k-1-\ell} \,.$$

Unter der Voraussetzung $p_x > p_y$ beträgt die Wahrscheinlichkeit für eine falsche Statistik nach $2k-1$ Ziehungen

$$p(e) = \sum_{\ell=0}^{k-1} P(\ell) \;\geq\; \binom{2k-1}{k-1} p_x^{k-1} p_y^{k} \;\geq\; 2^{k-1} p_x^{k-1} p_y^{k} \,.$$

Denn in diesem Fall würde der Algorithmus y vor x in der Liste setzen, obwohl $\langle x, y \rangle$ die optimale Anordnung ist. (Wenn der Algorithmus in dieser Situation doch x vor y setzt, so nutzt er die zur Verfügung stehende Information schlecht und man kann damit analog argumentieren). Die erwarteten Kosten nach $t = 2k-1$ Schritten betragen somit

$$\begin{aligned}
E[R_t] &= p(e)(p_y + 2p_x) + (1 - p(e))(p_x + 2p_y) \\
&= p(e)(p_x - p_y) + p_x + 2p_y \\
&\geq (p_x + 2p_y) \cdot \Big(1 + p(e)(p_x - p_y)/(p_x + 2p_y)\Big) \\
&\geq E[OPT] \cdot \left(1 + 2^{k-1} p_x^{k-1} (1-p_x)^k \frac{2p_x - 1}{2 - p_x}\right) \\
&\geq E[OPT] \cdot (1 + 1/6^{k+1}),
\end{aligned}$$

wenn man den letzten Ausdruck in p_x optimiert und grob abschätzt:

$$\begin{aligned}
\max_{1/2<p<1} 2^{k-1} p^{k-1} (1-p)^k \cdot \frac{2p-1}{2-p} &\geq \max_{1/2<p<1} 2^{k-2} (p(1-p))^k (2p-1) \\
&= \frac{1}{4\sqrt{2k+1}} \cdot \left(\frac{k}{2k+1}\right)^k \\
&\geq \frac{1}{4 \cdot 2^k} \cdot \left(\frac{1}{3}\right)^k \\
&\geq 1/6^{k+1} \,.
\end{aligned}$$

Das bedeutet, daß jeder Listenalgorithmus nach $t = 2k-1$ Zugriffen auf x oder y die optimalen Kosten im Mittel höchstens bis auf einen Faktor $1 + 1/6^{k+1}$ annähern kann. Für $\varepsilon > 0$ folgt, daß $t \geq 2\ln(1/\varepsilon)/\ln(6) - 3 \geq \ln(1/\varepsilon) - 3$ eine notwendige Bedingung ist, die optimalen Kosten bis auf einen Faktor $1 + \varepsilon$ zu approximieren. □

Stochastische Quellen mit Gedächtnis. Im folgenden untersuchen wir das erwartete Verhalten der SBD(k)-Regeln bei Zugriffsfolgen, die durch eine positiv-rekurrente Markov-Kette erster Ordnung produziert werden.

Sei $\mathbf{P} = (p_{ij})$ die Matrix der Übergangswahrscheinlichkeiten, $\pi = (\pi_j)$ der Vektor der stationären Wahrscheinlichkeiten und (m_{ij}) die Matrix der mittleren Erst-Eintritts-Zeiten (siehe Anhang).

Satz 1.28. *Bei einer Markov-Zugriffsfolge* $\mathbf{P}$ *sind die erwarteten Kosten der* SBD(k)*-Regel pro Zugriff asymptotisch gegeben durch*

$$\begin{aligned}
E_{\mathrm{SBD}(1)}(\mathbf{P}) &= 1 + \sum_{i \neq j} 1/(m_{ij} + m_{ji}) \\
E_{\mathrm{SBD}(2)}(\mathbf{P}) &= 1 + \sum_{i \neq j} (2M^2 - (m_{ii} + 2m_{jj})M + 2m_{ii}m_{jj})/(m_{ij} + m_{ji})^3 \\
E_{\mathrm{SBD}(3)}(\mathbf{P}) &= 1 + \sum_{i \neq j} (3M^4 - 3(m_{ii} + 2m_{jj})M^3 + (m_{ii}^2 + 12m_{ii}m_{jj} + 3m_{jj}^2)M^2 \\
&\qquad - 3(2m_{ii}^2 m_{jj} + 3m_{ii}m_{jj}^2)M + 6m_{ii}^2 m_{jj}^2)/(m_{ij} + m_{ji})^5,
\end{aligned}$$

wobei $M = m_{ij} + m_{ji}$. *Man beachte, daß* $m_{ii} = 1/\pi_i$.

Der Fall $k = 1$ (MOVE-TO-FRONT) wurde in [LLS84] betrachtet, siehe auch Seite 14.

Beweis. Wir wollen die asymptotische Wahrscheinlichkeit $b(j, i|i)$ berechnen, daß j vor i steht, wenn auf i zugegriffen wird. Wenn auf i zugegriffen wird, dann verfolgen wir die Zugriffsfolge rückwärts zu den $(2k - 1)$ letzten Zugriffen für i oder j. Die Wahrscheinlichkeit, von i rückwärts zu gehen, und ein Auftreten von j vor einem Auftreten von i zu finden, ist $q_{ij} = m_{ii}/(m_{ij} + m_{ji})$, siehe [LLS84]. Dann ist die Wahrscheinlichkeit, beim Rückwärtsgehen von i aus ein anderes i zu treffen, ohne vorher einem j zu begegnen, gleich $q_{ii} = 1 - m_{ii}/(m_{ij} + m_{ji})$. Die Situation mit Startzustand j ist symmetrisch, d.h. es gilt $q_{ji} = m_{jj}/(m_{ij} + m_{ji})$ und $q_{jj} = 1 - m_{jj}/(m_{ij} + m_{ji})$.

Wenn man nun über alle Fälle summiert, die in Lemma 1.22 genannt sind, erhält man $b(j, i|i)$, woraus man dann

$$E(\mathbf{P}) = 1 + \sum_i \pi_i (1 + \sum_{j \neq i} b(j, i|i))$$

gewinnt. Zur Berechnung von $b(j, i|i)$ für ein festes Paar i, j benutzen wir folgenden Ansatz. Sei $p_i(s, t)$ die Wahrscheinlichkeit, daß eine Folge aus $\{i, j\}^s$, die mit i beginnt, genau t mal j enthält, und $p_j(s, t)$ die Wahrscheinlichkeit, daß eine Folge der Länge s, die mit j startet, genau t mal j enthält, wenn die Übergangswahrscheinlichkeiten wie oben definiert sind. Uns interessiert

$$b(j, i|i) = \sum_{t=k}^{2k-1} p_i(2k - 1, t).$$

Die $p_i(s,t)$ sind rekursiv definiert als

$$\begin{aligned} p_i(s,t) &= q_{ii}p_i(s-1,t) + q_{ij}p_j(s-1,t-1) + [s=0,t=0] \\ p_j(s,t) &= q_{ji}p_i(s-1,t) + q_{jj}p_j(s-1,t-1) + [s=0,t=0]. \end{aligned}$$

Daraus erhalten wir mit den erzeugenden Funktionen

$$\begin{aligned} f_i(x,y) &= \sum_{s=0}^{\infty}\sum_{t=0}^{\infty} p_i(s,t)x^s y^t \\ f_j(x,y) &= \sum_{s=0}^{\infty}\sum_{t=0}^{\infty} p_j(s,t)x^s y^t \end{aligned}$$

das Gleichungssystem

$$\begin{aligned} f_i(x,y) &= xq_{ii}f_i(x,y) + xyq_{ij}f_j(x,y) + 1 \\ f_j(x,y) &= xq_{ji}f_i(x,y) + xyq_{jj}f_j(x,y) + 1. \end{aligned}$$

Die Lösung ergibt sich zu

$$\begin{aligned} f_i(x,y) &= \frac{1 + xy(q_{ij} - q_{jj})}{1 - xq_{ii} - xyq_{jj} + x^2y(q_{ii}q_{jj} - q_{ij}q_{ji})} \\ f_j(x,y) &= \frac{1 + x(q_{ji} - q_{ii})}{1 - xq_{ii} - xyq_{jj} + x^2y(q_{ii}q_{jj} - q_{ij}q_{ji})}. \end{aligned}$$

Sei $K = q_{ij}q_{ji} - q_{ii}q_{jj}$. Da $y = (1-xq_{ii})/(xq_{jj}+x^2K)$ eine einfache Nullstelle der Nenner ist, erhält man zunächst

$$\begin{aligned} f_i(x,y) &= \sum_{t=0}^{\infty} \frac{(xq_{jj}+x^2K)^t}{(1-xq_{ii})^{t+1}} \cdot y^t + x(q_{ij}-q_{jj}) \cdot \sum_{t=0}^{\infty} \frac{(xq_{jj}+x^2K)^t}{(1-xq_{ii})^{t+1}} \cdot y^{t+1} \\ f_j(x,y) &= \sum_{t=0}^{\infty} \frac{(1 - x(q_{ji}-q_{ii}))(xq_{jj}+x^2K)^t}{(1-xq_{ii})^{t+1}} \cdot y^t. \end{aligned}$$

Jetzt nutzen wir aus, daß $1/(1-xq_{ii})^{t+1} = \sum_{s\geq 0}\binom{s+t}{t}(xq_{ii})^s$, siehe [GKP94] :

$$\begin{aligned} f_i(x,y) &= \sum_{t\geq 0} y^t \sum_{s\geq 0} \binom{s+t}{t} q_{ii}^s x^{s+t} \sum_{h=0}^{t} \binom{t}{h} (xK)^h q_{jj}^{t-h} \\ &\quad + \sum_{t\geq 0} y^{t+1} \sum_{s\geq 0} \binom{s+t}{t} q_{ii}^s x^{s+t+1}(q_{ij}-q_{jj}) \sum_{h=0}^{t} \binom{t}{h} (xK)^h q_{jj}^{t-h} \\ f_j(x,y) &= \sum_{t\geq 0} y^t \sum_{s\geq 0} \binom{s+t}{t} q_{ii}^s x^{s+t} \sum_{h=0}^{t} \binom{t}{h} (xK)^h q_{jj}^{t-h} \\ &\quad + \sum_{t\geq 0} y^t \sum_{s\geq 0} \binom{s+t}{t} q_{ii}^s x^{s+t+1}(q_{ji}-q_{ii}) \sum_{h=0}^{t} \binom{t}{h} (xK)^h q_{jj}^{t-h}. \end{aligned}$$

Damit folgt für die gesuchten Koeffizienten

$$\begin{aligned}
p_i(s,t) &= \sum_{h=0}^{t} \binom{t}{h}\binom{s-h}{t} K^h q_{jj}^{t-h} q_{ii}^{s-t-h} \\
&\quad + (q_{ij}-q_{jj}) \cdot \sum_{h=0}^{t-1} \binom{t-1}{h}\binom{s-h-1}{t-1} K^h q_{jj}^{t-1-h} q_{ii}^{s-t-h} \\
p_j(s,t) &= \sum_{h=0}^{t} \binom{t}{h}\binom{s-h}{t} K^h q_{jj}^{t-h} q_{ii}^{s-t-h} \\
&\quad + (q_{ji}-q_{ii}) \cdot \sum_{h=0}^{t} \binom{t}{h}\binom{s-h-1}{t} K^h q_{jj}^{t-h} q_{ii}^{s-1-t-h} .
\end{aligned}$$

Damit diese Formeln für alle s, t gültig sind, müssen wir $\binom{n}{m} = 0$ für alle $n < m$ festsetzen, insbesondere auch für $n < 0$. Schließlich ergibt sich

$$\begin{aligned}
b(j,i|i) &= 1 - \sum_{t=0}^{k-1} p_i(2k-1,t) \\
&= 1 - \sum_{t=0}^{k-1} \Bigg(\binom{2k-1-t}{t} q_{ii}^{2(k-t)-1} (q_{ij}q_{ji} - q_{ii}q_{jj})^t \\
&\quad + \sum_{h=0}^{t-1} \binom{t}{h}\binom{2k-1-h}{t} q_{ii}^{2k-1-t-h} q_{jj}^{t-h-1} (q_{ij}q_{ji} - q_{ii}q_{jj})^h \\
&\quad \cdot (q_{jj} + (q_{ij}-q_{jj})(t-h)/(2k-1-h)) \Bigg),
\end{aligned}$$

und für kleine Werte von k erhalten wir das obige Ergebnis. □

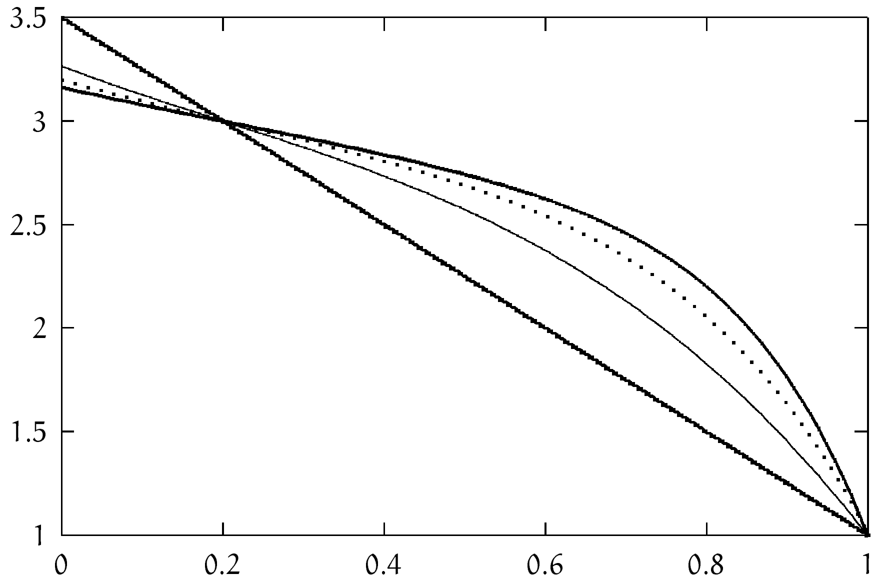
Im Spezialfall elementweiser Lokalität finden wir

Korollar 1.29. Für $0 \leq \alpha < 1$ und Übergangswahrscheinlichkeiten

$$p_{ij} = \left\{ \begin{array}{ccc} \alpha & : & i = j \\ (1-\alpha)/(n-1) & : & i \neq j \end{array} \right\}$$

gilt mit $\beta = (1-\alpha) \cdot n/(n-1)$, daß

$$\begin{aligned}
E_{\text{SBD}(1)}(\mathbf{P}) &= 1 + (1-\alpha) \cdot n/2 \\
E_{\text{SBD}(2)}(\mathbf{P}) &= 1 + (1-\alpha) \cdot n \cdot (\beta^2 - 3\beta + 4)/4 \\
E_{\text{SBD}(3)}(\mathbf{P}) &= 1 + (1-\alpha) \cdot n \cdot (3\beta^4 - 15\beta^3 + 32\beta^2 - 36\beta + 24)/16 \\
E_{\text{SBD}(4)}(\mathbf{P}) &= 1 + (1-\alpha) \cdot n \cdot (5\beta^6 - 35\beta^5 + 108\beta^4 - 190\beta^3 + 208\beta^2 \\
&\qquad - 144\beta + 64)/32.
\end{aligned}$$



Beispiel mit $n = 5$ für $E_{\text{SBD}(k)}(\mathbf{P})$ für $k = 1, \ldots, 4$ und $\alpha \in [0, 1]$.

Dieses Ergebnis ist ähnlich den in Abschnitt 1.2 vorgestellten Beobachtungen, wo im Falle der Lokalität eine verzögerte Anpassung (großes k) die erwartete Suchzeit verschlechtert. Im Falle unabhängiger Zugriffe ist es dagegen durch eine verzögerte Anpassung möglich, die asymptotischen erwarteten Suchkosten beliebig nahe an die optimalen Suchkosten heranzubringen.

1.4.4. Analyse der SORT-BY-TIME-Verfahren

Zunächst untersuchen wir die Rangfunktion

$$u_t(x) := \sum_{\substack{\sigma_{t-s}=x \\ 0 \leq s \leq t-1}} q^s$$

näher. Bei Bearbeitung der Zugriffsfolge ändert sie sich folgendermaßen:

$$u_{t+1}(x) = \left\{ \begin{array}{ccc} q \cdot u_t(x) & : & \sigma_t \neq x \\ 1 + q \cdot u_t(x) & : & \sigma_t = x \end{array} \right\}.$$

Gemäß der Definition von $\text{SBT}(q)$ ist die Liste stets nach absteigenden Werten von $u_t(.)$ sortiert. Wie bei $\text{SBR}(\alpha)$ kann es auch hier passieren, daß zwei Elemente den gleichen Rang besitzen. In diesem Fall wird das nachgefragte Element soweit nach vorne geschoben, bis nur noch Elemente mit echt größerem Rang davor stehen (oder der Listenkopf erreicht ist). Wenn zu einem Zeitpunkt t die letzten k Zugriffe auf x gerichtet waren, gilt

$$u_t(x) \geq \sum_{j=0}^{k-1} q^j = \frac{1 - q^k}{1 - q}.$$

Der Rang eines anderen Elementes y zu diesem Zeitpunkt beträgt höchstens

$$u_t(y) < \sum_{j=k}^{\infty} q^j = \frac{q^k}{1-q}.$$

Das bedeutet, daß in dieser Situation x vor y steht, wenn

$$q \leq \sqrt[k]{1/2}. \tag{1.16}$$

Daraus folgt, daß für alle $0 \leq q \leq 1/2$ das $\textsc{sbt}(t)$-Verfahren identisch mit der Nach-Vorne-Schieben-Regel ist. Für $q > 1/2$ wird es zunehmend träger, und im Fall $q = 1$ fällt es mit FREQUENCY-COUNT zusammen. Damit ein Verfahren k-kompetitiv ist, sollte es ein Element an den Listenanfang setzen, wenn k mal hintereinander darauf zugegriffen wurde. (Ein 2-kompetitiver Algorithmus ist beispielsweise auch möglich, wenn bei 2 maliger Nachfrage ein Element an den zweiten Listenplatz gesetzt wird und erst beim dritten Zugriff an den Listenkopf.) Wir zeigen

Satz 1.30. *Für $1/2 < q < 1$ ist das $\textsc{sbt}(q)$-Verfahren $\lceil 1/\log(1/q)\rceil$-kompetitiv*[1].

Der Beweis verläuft ähnlich wie der obige Beweis zu Satz 1.17. Sei im folgenden $k = \lceil 1/\log(1/q)\rceil$ fest. Wir definieren wieder für jedes Paar $x \neq y$ die Gewichtsfunktion

$$w(x,y) = \left\{ \begin{array}{lll} 0 & : \text{ falls } \langle x,y\rangle & \text{keine Inversion ist} \\ \ell & : \text{ falls } \langle x,y\rangle & \text{Inversion ist und } x \text{ von } y \text{ in } L \text{ überholt wird,} \\ & & \text{genau dann wenn jetzt } \ell \text{ mal auf } y \text{ zugegriffen} \\ & & \text{wird, } \ell = 1,\ldots,k \end{array} \right\},$$

und die Potentialfunktion $\Phi = \sum_{x\neq y} w(x,y)$. Zunächst zeigen wir wieder, daß das Gewicht einer Inversion $\langle z,y\rangle$ nicht zunehmen kann, wenn auf ein Element x verschieden von z und y zugegriffen wird.

Lemma 1.31. *Seien y und z zwei Elemente, so daß $\langle z,y\rangle$ eine Inversion mit dem Gewicht $w(z,y) = w$ ist, $1 \leq w \leq k$. Sei x ein anderes Element. Dann bleibt $w(z,y) = w$, nachdem eine Nachfrage nach x von $\textsc{sbt}(q)$ bearbeitet wurde.*

Beweis. Aufgrund der Definition der Potentialfunktion gilt zum Zeitpunkt t und $w(z,y) = w$, daß z von y überholt wird, wenn jetzt w mal auf y zugegriffen wird. Gemäß der Vorüberlegungen (1.16) bedeutet das $u_{t+w-1}(y) \geq u_{t+w-1}(z)$, falls $\sigma_j = y$ für $j = t,\ldots,t+w-1$. Wegen $u_{t+w-1}(y) = q^{w-1}u_t(y) + (1-q^{w-1})/(1-q)$ und $u_{t+w-1}(z) = q^{w-1}u_t z$ folgt daraus

$$u_t(y) + \frac{1-q^{w-1}}{(1-q)q^{w-1}} \geq u_t(z). \tag{1.17}$$

Angenommen, nach einem Zugriff $\sigma_t = x$ würde $w(z,y) \geq w+1$ gelten. Das bedeutet aber, daß y nicht z überholen wird, wenn es jetzt w mal nachgefragt wird, also $u_{t+w}(y) < u_{t+w}(z)$ bei $\sigma_j = y$ für $j = t+1,\ldots,t+w$. Dann gilt $u_{t+w}(y) =$

[1] In dieser Arbeit bezeichnet $\log(.)$ stets den Logarithmus zur Basis 2.

$q^{w-1}u_{t+1}(y) + (1-q^{w-1})/(1-q)$ und $u_{t+w}(z) = q^{w-1}u_{t+1}(z)$. Weil $\sigma_t = x$ verschieden von y und z ist, gilt weiter $u_{t+w}(y) = q^w u_t(y) + q(1-q^{w-1})/(1-q)$ und $u_{t+w}(z) = q^w u_t(z)$. Nach Division durch q^w folgern wir $u_t(z) > u_t(y) + (1-q^{w-1})/((1-q)q^{w-1})$, was einen Widerspruch zu (1.17) darstellt. Da bei einem Zugriff auf x das Gewicht der Inversion $\langle z, y\rangle$ auch nicht abnehmen kann, folgt die Behauptung des Lemmas. ☐

Wie beim Beweis von Satz 1.17 setzen wir voraus, daß auf jedes Element mindestens k mal zugegriffen wurde. Sei $\sigma_t = x$ der aktuelle Zugriff, dann definieren wir die Mengen $\mathcal{A}, \mathcal{B}, \mathcal{C}$: $\mathcal{A}$ als die Menge der Elemente z, so daß $\langle z, x\rangle$ keine Inversion darstellt, $\mathcal{B}$ die Menge der z, für die $\langle z, x\rangle$ eine Inversion ist, und $\mathcal{C}$ die Zusammenfassung aller z, so daß $\langle x, z\rangle$ eine Inversion darstellt. Sei $A = |\mathcal{A}|$, $B = |\mathcal{B}|$ und $C = |\mathcal{C}|$.

Analog zu vorher betragen die tatsächlichen Kosten von SBT(q) zum Zeitpunkt t gerade $\text{SBT}(q)_t = A + B + 1$, und die Kosten eines optimalen Algorithmus für die faktorisierte Liste sind $\text{FOPT}_t = A + C + 1$.

Wie im Beweis zu Satz 1.17 argumentiert man, daß die Potentialänderung, die durch die Aktion von SBT(q) bewirkt wird, höchstens $C - B$ beträgt. Die Potentialänderung durch den optimalen Algorithmus FOPT beschränken wir folgendermaßen. Wenn eine neue Inversion $\langle x, z\rangle$ entsteht, weil SBT(q) das Element x vor z gesetzt hat, aber FOPT nicht, dann war $z \in \mathcal{A}$ vor dem Zugriff, und wir zeigen, daß die neue Inversion höchstens das Gewicht $k-1$ haben kann.

Lemma 1.32. *Wenn in der Liste L das Element $\sigma_t = x$ ein Element $z \in \mathcal{A}$ überholt hat, dann gilt*

$$u_t(z) + \frac{1-q^{k-1}}{(1-q)q^{k-1}} \geq u_t(x).$$

Wenn durch diese Aktion eine Inversion $\langle x, z\rangle$ entstanden ist, und die nächsten $k-1$ Zugriffe auf z gerichtet sind, dann wird z wieder vor x gesetzt werden. Also hat diese Inversion ein Gewicht $w(x, z) \leq k-1$.

Beweis. Für den Beweis durch Widerspruch nehmen wir an, daß $q^{k-1}u_t(z) + (1-q^{k-1})/(1-q) < q^{k-1}u_t(x)$ ist. Das bedeutet aber, daß z seit dem k-letzten Zugriff auf x nicht mehr nachgefragt wurde Aber dann würde FOPT das Element x schon vor dem aktuellen Zugriff $\sigma_t = x$ vor z gesetzt haben (man beachte $k \geq 2$), ein Widerspruch zur Voraussetzung $z \in \mathcal{A}$ zum Zeitpunkt t.

Sei $\sigma_j = z$ für $j = t+1, \ldots, t+k-1$. Wenn zum Zeitpunkt $t+k-1$ wieder z vor x steht, muß gelten, daß $u_{t+k-1}(z) \geq u_{t+k-1}(x)$. Es ist $u_{t+k-1}(z) = q^{k-1}u_t(z) + (1-q^{k-1})/(1-q)$ und $u_{t+k-1}(x) = q^{k-1}u_t(x)$, also folgt die Behauptung aus dem ersten Teil des Lemmas. ☐

Analog zum Beweis von Satz 1.17 sieht man, daß eine neue Inversion $\langle x, z\rangle$, die entsteht, weil FOPT z vor x gesetzt hat, höchstens das Gewicht $k-1$ haben kann. Zusammen folgt, daß das Gewicht aller neuer Inversionen aufgrund der Aktionen von FOPT höchstens

$(k-1) \cdot A$ beträgt. Also folgt für die amortisierten Kosten a_t von $\text{SBT}(q)$ zum Zeitpunkt t

$$\begin{aligned} a_t &= \text{SBT}(q)_t + \Phi_t - \Phi_{t-1} \\ &\leq A + B + 1 \;+\; C - B + (k-1) \cdot A \\ &\leq (k - (k-1)/n) \cdot (A + C + 1). \end{aligned}$$

Also ist $\text{SBT}(q)$ k-kompetitiv mit $k = \lceil 1/\log(1/q) \rceil$. □

Analog zu der Argumentation für die $\text{SBD}(k)$-Regel auf Seite 37 sieht man, daß $k = \lceil 1/\log(1/q) \rceil$ auch eine untere Schranke für den kompetitiven Faktor der $\text{SBT}(q)$-Verfahren im Fall $1/2 < q < 1$ ist.

Wie man an dem Beispiel von Seite 29 sieht, erfüllen die $\text{SBT}(q)$-Regeln die paarweise Unabhängigkeit im allgemeinen nicht:

$$\sigma_1 = x, y, y, a, x \qquad \sigma_2 = x, a, y, y, x.$$

Die auf $\{x, y\}$ projizierte Folge ist in beiden Fällen gleich $\sigma_{xy} = x, y, y, x$; jedoch gilt für $q = 4/5$ und $t = 5$ im ersten Fall $u_t(x) = 1 + q^4 > q^2 + q^3 = u_t(y)$ und im zweiten Fall $u_t(x) = 1 + q^4 < q + q^2 = u_t(y)$.

Satz 1.30 zeigt insbesondere, daß $\text{SBT}(q)$ für $q = \sqrt{1/2} = 0.707\ldots$ noch 2-kompetitiv ist. Dieser Algorithmus ist verschieden von TIMESTAMP, er reagiert jedoch nicht träger. Denn nach der Zugriffsfolge $\sigma = y, a^i, x, a^j, y, a^k, x$, also zum Zeitpunkt $t = i + j + k + 4$ ($i, j, k \geq 0$ beliebig), wird TS das Element x vor y gesetzt haben. Wegen $u_t(x) = 1 + q^{j+k+2}$ und $u_t(y) = q^{k+1} + q^{i+j+k+3}$ folgt $u_t(x) \geq u_t(y)$ für alle $0 \leq q \leq 1$, also wird auch $\text{SBT}(q)$ das Element x vor y setzen.

Analog zu der Bemerkung auf Seite 34 kann man folgenden Ansatz machen, um $\text{SBT}(q)$ auf gedächtnislosen stochastischen Quellen zu analysieren. Für zwei Elemente x, y betrachten wir den Markov-Prozeß mit Zuständen $(u_t(x), u_t(y)) \in [0, 1]^2$. Die Zustandsübergänge sind

$$(a, b) \longmapsto \left\{ \begin{array}{lll} (q \cdot a, q \cdot b) & \text{mit Wahrscheinlichkeit} & 1 - p_x - p_y \\ (1 + q \cdot a, q \cdot b) & \text{mit Wahrscheinlichkeit} & p_x \\ (q \cdot a, 1 + q \cdot b) & \text{mit Wahrscheinlichkeit} & p_y \end{array} \right\}.$$

Dann erhält man die asymptotische Wahrscheinlichkeit, daß x vor y steht, wenn $\text{SBT}(q)$ angewendet wird, als

$$b(x, y) = \sum_{\substack{(a,b) \\ 0 \leq b < a \leq 1}} \pi(a, b),$$

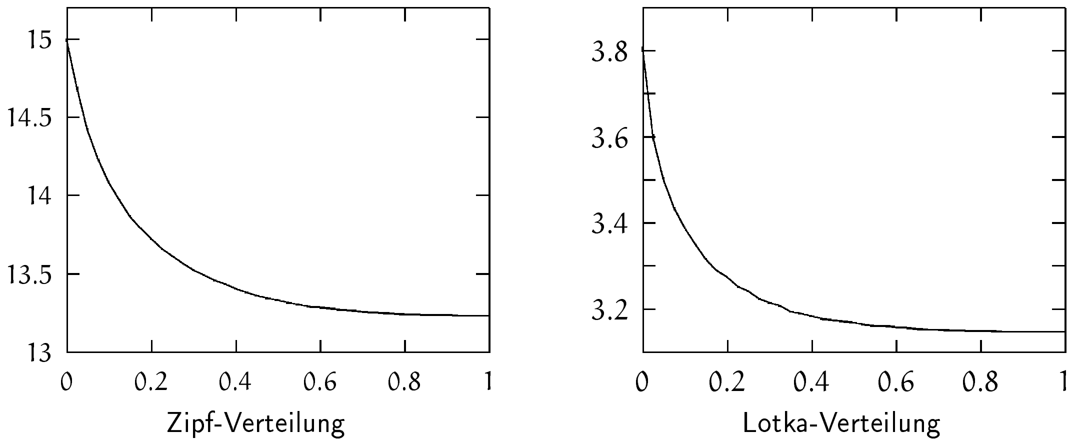
dabei bezeichnet $\pi(a, b)$ die stationäre Wahrscheinlichkeit des Zustands (a, b). Auch hier wissen wir nicht, wie man diese Werte allgemein berechnet.

1.4.5. Empirische Ergebnisse

In diesem Abschnitt werden empirische Untersuchungen der $\text{SBR}(\alpha)$- und $\text{SBD}(k)$-Verfahren bei unabhängigen identisch verteilten Zugriffsfolgen vorgestellt.

Sort-by-Rank

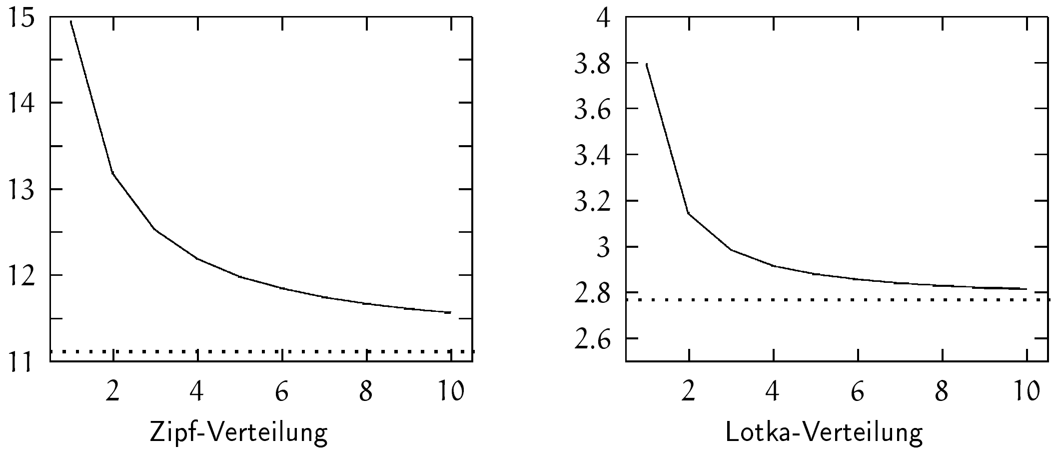
Zum Test der SBR(α)-Verfahren wurden unabhängige Zugriffe auf eine Liste mit $n = 50$ Elementen simuliert. Sie waren nach der Verteilung von Zipf ($p_i \propto 1/i$) und nach der Verteilung von Lotka ($p_i \propto 1/i^2$) erzeugt worden. Für den Bereich $0 \leq \alpha \leq 1$ sind die Ergebnisse unten dargestellt, dafür wurden in einer Schrittweite von 0.01 für jeden festen Wert von α mit der angegebenen Verteilung 100000 Zugriffe simuliert.



Man erkennt, daß die Suchkosten einen graduellen Übergang zwischen den Kosten von MTF und TS bilden. Diese Beobachtung bestärkt unsere Intuition, daß die Verfahren eine stetige Interpolation erlauben. Dies gilt speziell für den Fall stochastischer Quellen, in dem wir keine analytischen Resultate finden konnten.

Sort-by-Delay

Die folgenden Diagramme geben die Suchkosten von SBD(k) bei einer Listenlänge von $n = 50$ für $k = 1, \dots, 10$ an. Die optimalen Kosten betragen bei der Zipf-Verteilung 11.11 und bei der Lotka-Verteilung 2.77.



1.5. Konvergenzgeschwindigkeit

Wir interessieren uns dafür, wie lange eine Regel für selbstorganisierende Listen benötigt, um nahe an die asymptotischen erwarteten Kosten zu kommen, wenn man eine bestimmte Verteilung über den Startkonfigurationen voraussetzt. Dieses Problem wurde für die Nach-Vorne-Schieben-Regel und die Vertauschen-Regel bereits mehrfach untersucht, zunächst von *J. Bitner* [B79], dann ausgiebig für MTF von *J. Fill* [FH96, F96, F96a], vgl. auch [J98].

Intuitiv ist es einleuchtend, daß eine Regel, deren asymptotische erwartete Kosten die optimalen erwarteten Kosten gut approximiert, genaue statistische Informationen über die Zugriffsfolge braucht, die nur über einen längeren Zeitraum gesammelt werden können (vgl. Satz 1.27 auf Seite 47). Regeln mit guten asymptotischen Kosten sollten daher eine langsame Konvergenz aufweisen.

1.5.1. Batched(k)-MOVE-TO-FRONT

Wir betrachten nun die Familie der Batched(k)-Nach-Vorne-Schieben-Regeln. Diese haben gegenüber den Simple(k)-Regeln den Vorteil, daß bei der Analyse keine Schwierigkeiten mit überlappenden k-Tupeln entstehen. Sei $E_k(t)$ die erwartete Suchzeit nach t Zugriffen, und χ_A die charakteristische Funktion einer Bedingung A.

Satz 1.33. *Sei σ die Startkonfiguration der Liste. Bei unabhängigen, identisch verteilten Zugriffen mit der Verteilung $\mathbf{p} = (p_1, \dots, p_n)$ beträgt die erwartete Suchzeit unter der Batched(k)-MTF-Regel nach kt Zugriffen*

$$E_k(kt) = E_k(\infty) + OW_{k,\sigma}(kt)$$

mit den asymptotischen Kosten

$$E_k(\infty) = 1 + \sum_{i<j} \frac{p_i p_j^k + p_i^k p_j}{p_i^k + p_j^k}$$

und dem anfänglichen Mehraufwand (Overwork) *pro Zugriff*

$$OW_{k,\sigma}(kt) = \sum_{i<j} \left(p_i \chi_{\sigma(j)<\sigma(i)} + p_j \chi_{\sigma(i)<\sigma(j)} - \frac{p_i p_j^k + p_i^k p_j}{p_i^k + p_j^k} \right) \cdot (1 - p_i^k - p_j^k)^t.$$

Beweis. Unter einem k-Zugriff verstehen wir einen Block von k aufeinanderfolgenden Nachfragen auf ein bestimmtes Element. Sei $b_{kt}(j, i)$ die Wahrscheinlichkeit, daß nach kt Zugriffen j vor i steht. Dies ist der Fall, wenn innerhalb von kt Zugriffen noch gar nicht auf i und nicht auf j zugegriffen wurde, und j schon in der Startkonfiguration σ vor i stand, was mit Wahrscheinlichkeit

$$\chi_{\sigma(j)<\sigma(i)} \cdot (1 - p_i^k - p_j^k)^t$$

auftritt. Oder es ist der Fall, wenn bereits k-Zugriffe auf i oder j aufgetreten sind, und der letzte k-Zugriff war für j. Dies passiert mit Wahrscheinlichkeit

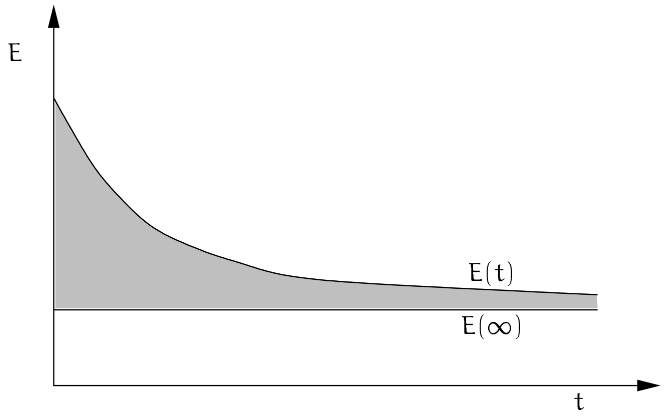
$$p_j^k \sum_{s=0}^{t-1} (1 - p_i^k - p_j^k)^s = \frac{p_j^k}{p_i^k + p_j^k}(1 - (1 - p_i^k - p_j^k)^t),$$

indem man über die Anzahl s der k-Zugriffe seit dem letzten k-Zugriff auf j konditioniert. Für die erwarteten Kosten nach kt Schritten folgt also

$$\begin{aligned}
E_k(kt) &= \sum_{i=1}^{n} p_i(1 + \sum_{j \neq i} b_{kt}(j,i)) \\
&= 1 + \sum_{i=1}^{n} \sum_{j \neq i} \left(\frac{p_i p_j^k}{p_i^k + p_j^k} + p_i \cdot \left(\chi_{\sigma(j)<\sigma(i)} - \frac{p_j^k}{p_i^k + p_j^k} \right) \cdot (1 - p_i^k - p_j^k)^t \right) \\
&= 1 + \sum_{i<j} \frac{p_i p_j^k + p_i^k p_j}{p_i^k + p_j^k} + \sum_{i<j} \Big(p_i \chi_{\sigma(j)<\sigma(i)} + p_j \chi_{\sigma(i)<\sigma(j)} \\
&\qquad - \frac{p_i p_j^k + p_i^k p_j}{p_i^k + p_j^k} \Big) \cdot (1 - p_i^k - p_j^k)^t .
\end{aligned}$$

□

Die Summe des anfänglichen Mehraufwands über die ersten t Zugriffe kann man graphisch veranschaulichen als die Fläche zwischen den tatsächlichen Kosten und den asymptotischen Kosten (schattiert).



Korollar 1.34. Sei $p_1 \geq \cdots \geq p_n > 0$. Wenn keine Annahmen über die Startkonfigurationen der Liste gemacht werden können, beträgt der anfängliche Mehraufwand höchstens

$$OW_k(kt) \leq \sum_{i<j} \frac{(p_i - p_j)p_i^k}{p_i^k + p_j^k} \cdot (1 - p_i^k - p_j^k)^t .$$

Beweis. Der schlechteste Fall tritt ein, wenn die Startkonfiguration gerade die Umkehrung der optimalen Sortierung ist, d.h. $\sigma = \langle n, n-1, \dots, 1\rangle$. Dies folgt aus einer Monotonie-Beziehung zwischen Startkonfigurationen σ und Overwork $OW_{k,\sigma}(t)$, die für den Fall MTF in [F96] gezeigt wurde und hier analog gilt:

Die sogenannte *schwache Bruhat-Ordnung* ist eine partielle Ordnung auf der symmetrischen Gruppe $\mathcal{S}_n$ der Permutationen [B84, F96]. Dabei gilt $\sigma \leq \tau$, wenn man τ aus σ durch eine Folge von paarweise benachbarten Transpositionen (ij) mit $i < j$ erhalten kann. Die identische Permutation $id = \langle 1, 2, \dots, n\rangle$ ist ein minimales Element in dieser Ordnung, und ihre Umkehrung $rev = \langle n, n-1, \dots, 1\rangle$ ist ein maximales Element. Betrachten wir einen Schritt (ij) mit $i < j$ in der Folge von Transpositionen, die σ in τ überführt. Dann gilt

$$\{(a,b) : \tau(a) < \tau(b)\} = \{(a,b) : \sigma(a) < \sigma(b)\} \cup \{(j,i)\} - \{(i,j)\}.$$

Damit beträgt die Differenz des Mehraufwands

$$OW_{k,\tau}(kt) - OW_{k,\sigma}(kt) = (p_i - p_j) \cdot (1 - p_i^k - p_j^k)^t \geq 0.$$

Wir haben also gezeigt, daß aus $\sigma \leq \tau$ folgt, daß $OW_{k,\sigma} \leq OW_{k,\tau}$ ist. Da rev ein maximales Element der schwachen Bruhat-Ordnung ist, wird der Mehraufwand durch $OW_{k,rev}$ maximiert. □

Korollar 1.35. Unter der Annahme, daß alle Startkonfigurationen der Liste gleichwahrscheinlich sind, folgt für den anfänglichen Mehraufwand

$$OW_k(kt) = \sum_{i<j} \frac{(p_i - p_j)(p_i^k - p_j^k)}{2 \cdot (p_i^k + p_j^k)} \cdot (1 - p_i^k - p_j^k)^t.$$

Beweis. Wenn $\mathrm{Prob}(\sigma) = 1/n!$ für alle Listenkonfigurationen σ gilt, dann folgt für $i \neq j$, daß $\mathrm{Prob}(\sigma(j) < \sigma(i)) = 1/2$, und damit $E[\chi_{\sigma(j)<\sigma(i)}] = 1/2$ für die Zufallsvariable $\chi_{\sigma(j)<\sigma(i)}$. □

Für den Fall $k = 1$ (MTF) wurde dieses Resultat bereits in [B79] gezeigt. Da die erwarteten Kosten nach t Schritten gegen die asymptotischen erwarteten Kosten konvergieren, gilt $OW_k(t) \to 0$ für $t \to \infty$. Wir interessieren uns dafür, wie schnell dies geschieht und leiten nun eine verteilungsunabhängige Schranke für die Konvergenzgeschwindigkeit her.

Satz 1.36. *Falls alle Startverteilungen gleich wahrscheinlich sind, gilt für beliebige* $\varepsilon > 0$, $\mathbf{p} = (p_1, \dots, p_n)$ *und* $k \geq 1$, *daß*

$$t \geq t^* := \left\lceil \frac{1}{ek} \cdot \left(\frac{n^2}{4\varepsilon}\right)^k \right\rceil \Longrightarrow E_k(kt) \leq (1+\varepsilon) \cdot E_k(\infty).$$

Beweis. Sei $\varepsilon > 0$ gegeben und o.B.d.A. $p_1 \geq \cdots \geq p_n$. Gesucht ist ein t^*, so daß für $t \geq t^*$ gilt: $OW_k(kt) \leq \varepsilon \cdot E_k(\infty)$. Wir fordern eine hinreichende Bedingung, nämlich $OW_k(kt) \leq \varepsilon$. Da 1 eine untere Schranke für die optimalen asymptotischen Kosten und diese ein untere Schranke für die erwarteten Kosten $E_k(\infty)$ sind, folgt der Satz aus

$$E_k(kt) = E_k(\infty) + OW_k(kt) \leq E_k(\infty) + \varepsilon \cdot 1 \leq E_k(\infty) + \varepsilon E_k(\infty) = (1+\varepsilon) \cdot E_k(\infty).$$

Wegen $p_i \geq p_j$ für $i < j$ und $(1-x)^t \leq \exp(-xt)$ für $0 \leq x \leq 1$ folgt

$$\begin{aligned}
OW_k(kt) &= \sum_{i<j} \frac{(p_i - p_j)(p_i^k - p_j^k)}{2(p_i^k + p_j^k)} \cdot (1 - p_i^k - p_j^k)^t \\
&\leq \sum_{i<j} \frac{p_i - p_j}{2} \cdot \frac{p_i^k}{p_i^k + p_j^k} \cdot (1 - p_i^k)^t \\
&\leq \sum_{i<j} \frac{p_i}{2} \cdot 1 \cdot (1 - p_i^k)^t \\
&\leq \binom{n}{2} \cdot \max_{1 \leq i \leq n} \left(p_i \cdot (1 - p_i^k)^t \right) /2 \\
&\leq \binom{n}{2} \cdot \max_{0 \leq p \leq 1} \left(p \cdot e^{-tp^k} \right) /2 \\
&= \binom{n}{2} \cdot \left(\frac{1}{tke} \right)^{1/k} /2,
\end{aligned}$$

da das Maximum von $p \exp(-tp^k)$ bei $p = (kt)^{-1/k}$ angenommen wird. Also ist eine hinreichende Bedingung an t:

$$t \geq t^* := \left\lceil \frac{1}{ek} \cdot \left(\frac{n^2}{4\varepsilon} \right)^k \right\rceil.$$

□

Speziell erhalten wir daraus das Ergebnis, daß $O(n^2/\varepsilon)$ Schritte bei der Nach-Vorne-Schieben-Regel hinreichend sind.

Für spezielle Verteilungen kann man bessere Schranken angeben. In den folgenden beiden Sätzen wächst t^* nur logarithmisch in $1/\varepsilon$ anstatt polynomiell.

Satz 1.37. *Sei $0 < q < 1$ und $\mathbf{p} = (p_1, \ldots, p_n)$ die geometrische Verteilung auf n Elementen mit Parameter q, d.h. $p_i = p(n) \cdot q^{i-1}$. Dabei ist $p(n) = (1-q)/(1-q^n)$ der Normalisierungsfaktor. Wenn alle Startverteilungen gleich wahrscheinlich sind, dann gilt für $\varepsilon > 0$, daß*

$$t \geq t^* := \left\lceil \frac{\ln(n/(2\varepsilon))}{((1-q)q^n)^k} \right\rceil \implies E_k(kt) \leq (1+\varepsilon) \cdot E_k(\infty).$$

Beweis. Wir schätzen zunächst so ab wie in Satz (1.36).

$$\begin{aligned} OW_k(kt) &\le \sum_{i<j} \frac{p_i - p_j}{2} \cdot \frac{p_i^k}{p_i^k + p_j^k} \cdot (1 - p_i^k)^t \\ &\le n/2 \cdot \sum_{i=1}^{n-1} p_i \cdot \exp(-tp_i^k) \\ &= n/2 \cdot \sum_{i=1}^{n-1} p(n) \cdot q^{i-1} \cdot \exp(-tp(n)^k q^{k(i-1)}) \\ &\le n/2 \cdot p(n) \cdot \sum_{i=1}^{n-1} q^{i-1} \cdot \exp(p(n)^k q^{kn})^{-t} \\ &\le n/2 \cdot \exp((p(n)q^n)^k)^{-t} \\ &\le n/2 \cdot \exp(((1-q)q^n)^k)^{-t} \end{aligned}$$

Eine hinreichende Bedingung für $OW_k(kt) \le \varepsilon$ ist also

$$t \ge \frac{\ln(n/(2\varepsilon))}{((1-q)q^n)^k}.$$

□

Mit einer analogen Rechnung finden wir

Satz 1.38. *Sei* $\alpha \ge 0$. *Für die verallgemeinerte Zipf-Verteilung mit Parameter* α *auf* n *Elementen, also* $p_i = 1/(i^\alpha H_n(\alpha))$, *gilt für* $\varepsilon > 0$ *und gleichwahrscheinliche Startverteilungen, daß*

$$t \ge t^* := \lceil (n^\alpha H_n(\alpha))^k \cdot \ln(n/(2\varepsilon)) \rceil \implies E_k(kt) \le (1+\varepsilon) \cdot E_k(\infty).$$

□

Wir betrachten die Markov-Kette auf den Listenpermutationen und interessieren uns für die Übergangswahrscheinlichkeiten zwischen den einzelnen Listenzuständen. Die folgende Formel ist grundlegend für die weitere Analyse und stellt eine Verallgemeinerung des Ergebnisses von *Fill* dar, der in [F96] den Fall der MTF-Regel betrachtet hat. Wir benötigen folgende Notation. Für einen Vektor $\mathbf{w} = (w_1, \ldots, w_n)$ von positiven Zahlen und $0 \le i \le j \le n$ sei

$$w_i^+ = \sum_{h=1}^{i} w_h, \qquad w_i^* = \prod_{h=1}^{i} w_h, \qquad w_{j,i} = \prod_{\substack{h \ne i \\ 0 \le h \le j}} (w_i^+ - w_h^+)^{-1},$$

wobei $w_0^+ = 0$, $w_0^* = 1$ und $w_{0,0} = 1$. Sei $\sigma \in \mathcal{S}_n$ eine Permutation von $\{1, \ldots, n\}$. Die Permutation hat einen *Abstieg* an Position i, falls $\sigma(i) > \sigma(i+1)$. Sei $L(\sigma)$ die Position des letzten Abstiegs von σ. Dabei ist $L(id) = 0$ für die identische Permutation und stets $0 \le L(\sigma) \le n-1$.

Satz 1.39. *Sei* $\mathbf{p} = (p_1, \dots, p_n)$ *eine Wahrscheinlichkeitsverteilung. Die Wahrscheinlichkeit, in* kt *Schritten von Listenkonfiguration* σ *nach* τ *zu gelangen, wenn die Batched*(k)*-MTF-Regel angewendet wird, ist*

$$p_{\mathbf{p}}^{(kt)}(\sigma, \tau) = \sum_{j=L(\sigma^{-1}\tau)}^{n} w_j^* \sum_{i=0}^{j} w_{j,i} \sum_{\ell=0}^{t-j} \binom{t}{\ell} s^\ell (w_i^+)^{t-\ell}. \tag{1.18}$$

Dabei ist $s = 1 - \sum_{h=1}^{n} p_h^k$ *und* $w_i = p_{\tau(i)}^k$.

Beweis. Im Falle $k = 1$ kollabiert die letzte Summe zu $(w_i^+)^t$, da $s = 0$ und wegen $0^0 = 1$ nur der Summand bei $\ell = 0$ übrig bleibt, und man erhält das Ergebnis von *Fill*. Da $p_{\mathbf{p}}^{(kt)}(\sigma, \tau) = p_{\mathbf{p}\circ\sigma}^{(kt)}(id, \sigma^{-1}\tau)$ ist, genügt es, als Startkonfiguration $\sigma = id$ zu betrachten. Sei $q^{(kt)}(\tau, j)$ die Wahrscheinlichkeit, daß während der t k-Zugriffe genau j verschiedene Elemente an den Listenkopf gesetzt werden und die Konfiguration τ ergeben. Dann gilt

$$p_{\mathbf{p}}^{(kt)}(id, \tau) = \sum_{j=0}^{n} q^{(kt)}(\tau, j). \tag{1.19}$$

Für $L(\tau) > j$ gilt $q^{(kt)}(\tau, j) = 0$, da es in diesem Fall nicht ausreicht, j Elemente an den Listenkopf zu bewegen, um den Fehlstand an Position $L(\tau)$ zu beseitigen. Falls $L(\tau) \leq j$, so müssen die j ersten Elemente in der Liste in der Reihenfolge $\tau(j), \tau(j-1), \dots, \tau(1)$ bewegt worden sein. Zwischen den k-Zugriffen, die diese Elemente bewegen, dürfen beliebige andere k-Tupel eingestreut sein, die keine Listenänderung hervorrufen. Diese Tupel treten auf mit Wahrscheinlichkeit s, und es können höchstens $t - j$ viele sein. Indem man nach den Zeitpunkten der letzten k-Zugriffe auf die Elemente $\tau(j), \dots, \tau(1)$ konditioniert, findet man

$$\begin{aligned} q^{(kt)}(\tau, j) &= \sum_{\ell=0}^{t-j} \binom{t}{\ell} s^\ell \sum_{\substack{(h_1, \dots, h_j) \\ \sum h_i = t-\ell-j \\ h_i \geq 0}} (w_j^+)^{h_m} w_j (w_{j-1}^+)^{h_{m-1}} w_{j-1} \cdots (w_2^+)^{h_2} w_2 (w_1^+)^{h_1} w_1 \\ &= w_j^* \sum_{\ell=0}^{t-j} \binom{t}{\ell} s^\ell \sum_{\substack{(h_1, \dots, h_j) \\ \sum h_i = t-\ell-j \\ h_i \geq 0}} \prod_{r=1}^{j} (w_r^+)^{h_r}. \end{aligned}$$

Die Formel (2.5) aus [F96] ergibt (mit Substitution $t \to t - \ell$)

$$\sum_{\substack{(h_1, \dots, h_j) \\ \sum h_i = t-\ell-j \\ h_i \geq 0}} \prod_{r=1}^{j} (w_r^+)^{h_r} = \sum_{i=0}^{j} (w_i^+)^{t-\ell} w_{j,i}.$$

Damit folgern wir

$$q^{(kt)}(\tau, j) = \left\{ \begin{array}{ll} w_j^* \sum_{\ell=0}^{t-j} \binom{t}{\ell} s^\ell \sum_{i=0}^{j} (w_i^+)^{t-\ell} w_{j,i} & : \quad L(\tau) \leq j \\ 0 & : \quad L(\tau) > j \end{array} \right\}$$

und Einsetzen in (1.19) gibt das Ergebnis. □

Im Fall $t \to \infty$ erhält man für beliebige Startkonfigurationen σ die stationäre Wahrscheinlichkeit der Konfiguration τ. Das Ergebnis erhält man auch durch eine direkte Übertragung der Formel von *Hendricks* [H72], da die k-Tupel, die nicht ausschließlich auf ein Element gerichtet waren und somit keine Aktion bewirken, zwar die Konvergenz verlangsamen, aber auf den Grenzwert keinen Einfluß haben. Die stationäre Wahrscheinlichkeit von τ ist die Wahrscheinlichkeit, die k-Tupel $\tau(1)^k, \ldots, \tau(n)^k$ in dieser Reihenfolge durch Ziehen ohne Zurücklegen zu finden, unter der Bedingung, nur Elemente aus $\{\tau(1)^k, \ldots, \tau(n)^k\}$ zu ziehen. Die Wahrscheinlichkeit eines k-Zugriffs auf $\tau(1)$ unter dieser Bedingung ist $p^k_{\tau(1)}/w_n^+$.

Korollar 1.40. Die stationäre Wahrscheinlichkeit der Listenkonfiguration τ unter der Batched(k)-MTF-Regel ist

$$\begin{aligned} \pi(\tau) &= w_n^* \cdot w_{n,n} \\ &= \frac{p^k_{\tau(1)}}{w_n^+} \cdot \frac{p^k_{\tau(2)}}{w_n^+ - p^k_{\tau(1)}} \cdot \frac{p^k_{\tau(3)}}{w_n^+ - p^k_{\tau(1)} - p^k_{\tau(2)}} \cdot \cdots \cdot \frac{p^k_{\tau(n)}}{w_n^+ - p^k_{\tau(1)} - p^k_{\tau(2)} - \cdots - p^k_{\tau(n-1)}}. \end{aligned}$$

Beweis. Wir schreiben (1.18) um als

$$p_{\mathbf{p}}^{(kt)}(\sigma, \tau) = \sum_{i=0}^{n} \sum_{j=\max\{i, L(\sigma^{-1}\tau)\}}^{n} w_j^* w_{j,i} \sum_{\ell=0}^{t-j} \binom{t}{\ell} s^\ell (w_i^+)^{t-\ell}. \tag{1.20}$$

Nun ist $s + w_i^+ < 1$ für alle $i < n$ und $s + w_n^+ = 1$. Also gilt auch

$$\sum_{\ell=0}^{t-j} \binom{t}{\ell} s^\ell (w_i^+)^{t-\ell} \longrightarrow \left\{ \begin{array}{ll} 1 & : \quad i = n \\ 0 & : \quad i < n \end{array} \right\}, \quad t \to \infty.$$

Das bei der Summation fehlende Stück $\sum_{\ell=t-j+1}^{t}$ hat konstant j Summanden und spielt im Fall $t \to \infty$ keine Rolle mehr. Somit bleibt beim Grenzübergang nur noch der Summand mit $i = n$ übrig und man erhält (unabhängig von der Startkonfiguration σ)

$$p_{\mathbf{p}}^{(kt)}(\sigma, \tau) \longrightarrow w_n^* w_{n,n} = \pi(\tau).$$

□

1.5.2. Sort-by-Delay(k)

Wir betrachten die erwartete Suchzeit unter der SBD(k)-Regel nach t Schritten. Hierzu modifizieren wir diese Verfahren so, daß erst nach k Zugriffen ein Element bewegt wird, bei den ersten $k-1$ Nachfragen bleibt es auf seinem ursprünglichen Platz stehen.

Satz 1.41. *Sei σ die Startkonfiguration der Liste. Bei unabhängigen identisch verteilten Zugriffen mit der Verteilung $\mathbf{p} = (p_1, \ldots, p_n)$ beträgt die erwartete Suchzeit unter der* SBD(k)*-Regel nach t Zugriffen*

$$E_k(t) = E_k(\infty) + OW_{k,\sigma}(t)$$

mit den asymptotischen Kosten

$$E_k(\infty) = 1 + 2\sum_{i<j}\sum_{\ell=0}^{k-1}\binom{2k-1}{\ell}\frac{p_i^{\ell+1}p_j^{2k-1-\ell} + p_i^{2k-1-\ell}p_j^{\ell+1}}{(p_i+p_j)^{2k-1}}$$

und dem anfänglichen Mehraufwand (Overwork) *pro Zugriff*

$$\begin{aligned} OW_{k,\sigma}(kt) \quad &= \quad \sum_{i<j} p_i \cdot \sum_{h=0}^{k-1}\Big(\chi_{\sigma(j)<\sigma(i)} \cdot \sum_{\ell=0}^{k-1}\binom{t}{\ell}\binom{t-\ell}{h} p_i^j p_j^\ell (1-p_i-p_j)^{t-\ell-h} \\ &\quad + \sum_{s=t}^{\infty}\binom{s}{k-1}\binom{s-k+1}{h} p_i^h p_j^k (1-p_i-p_j)^{s-k+1-h}\Big). \end{aligned}$$

Beweis. Wir sind an der Wahrscheinlichkeit $b_t(j,i)$ interessiert, daß nach t Schritten j vor i steht. Wenn es bis dahin höchstens $k-1$ Zugriffe auf j gab, dann steht j vor i genau dann, wenn auch höchstens $k-1$ mal i nachgefragt wurde und am Anfang schon j vor i stand. Dies tritt auf mit Wahrscheinlichkeit

$$b_t^{(1)}(j,i) = \chi_{\sigma(j)<\sigma(i)} \cdot \sum_{\ell=0}^{k-1}\sum_{h=0}^{k-1}\binom{t}{\ell}\binom{t-\ell}{h} p_j^\ell p_i^h (1-p_i-p_j)^{t-\ell-h}.$$

Wenn es schon mindestens k Zugriffe auf j gab, dann befindet sich j vor i, wenn seit dem k-letzten Zugriff auf j höchstens $k-1$ Nachfragen nach i aufgetreten sind. Konditioniert man danach, daß nach dem k-letzten Zugriff auf j noch genau s Schritte gefolgt sind, so erhält man

$$b_t^{(2)}(j,i) = p_j^k \cdot \sum_{s=k-1}^{t-1}\sum_{h=0}^{k-1}\binom{s}{k-1}\binom{s-k+1}{h} p_i^h (1-p_i-p_j)^{s-k+1-h}.$$

Für $t \to \infty$ konvergiert dieser Wert gegen die asymptotische Wahrscheinlichkeit, daß j

vor i steht, siehe Formel (1.12). Denn es gilt mit $\binom{s}{k-1}\binom{s-k+1}{h} = \binom{k-1+h}{h}\binom{s}{k-1+h}$, daß

$$
\begin{aligned}
b_\infty^{(2)}(j,i) &= \sum_{h=0}^{k-1}\binom{k-1+h}{h}p_i^h p_j^k \cdot \sum_{s=0}^{\infty}\binom{s}{k-1+h}(1-p_i-p_j)^{s-k+1-h} \\
&= \sum_{h=0}^{k-1}\binom{k-1+h}{h}\frac{p_i^h p_j^k}{(p_i+p_j)^{k+h}} \\
&= \frac{1}{(p_i+p_j)^{2k-1}} \cdot \sum_{h=0}^{k-1}\binom{k-1+h}{h}p_i^h p_j^k \sum_{\ell=0}^{k-1-h}\binom{k-1-h}{\ell}p_i^\ell p_j^{k-1-h-\ell} \\
&= \frac{1}{(p_i+p_j)^{2k-1}} \cdot \sum_{m=0}^{k-1}\sum_{h=0}^{k-1}\binom{k-1+h}{k-1}\binom{k-1-h}{k-1-m}p_i^m p_j^{2k-1-m} \\
&= \frac{1}{(p_i+p_j)^{2k-1}} \cdot \sum_{m=0}^{k-1}\binom{2k-1}{m}p_i^m p_j^{2k-1-m} \\
&= b(j,i).
\end{aligned}
$$

Dabei folgt das zweite Gleichheitszeichen mit $\sum_{s=0}^{\infty}\binom{t+s}{s}z^s = (1-z)^{-t-1}$ und das vorletzte Gleichheitszeichen aus $\sum_{h=0}^{k}\binom{k+h}{k}\binom{k-h}{k-m} = \binom{2k+1}{m}$, siehe [GKP94]. Für die erwarteten Kosten nach t Schritten gilt

$$
E_k(t) = \sum_{i=1}^{n} p_i\Big(1 + \sum_{j\neq i} b_t^{(1)}(j,i) + b_t^{(2)}(j,i)\Big)
$$

und Einsetzen ergibt das Ergebnis. Aufgrund der partiellen Summation $\sum_{s=t}^{\infty}$ scheint leider keine weitere Vereinfachung möglich zu sein. □

1.6. Gewinnung statischer Listen

Den bisher vorgestellten Algorithmen ist gemeinsam, daß sie immer fortgeführt werden und daß die Liste immer wieder umgeordnet wird. All diese Verfahren (bis auf FREQUENCY-COUNT, das bei der folgenden Diskussion zunächst nicht betrachtet wird) benutzen ein endliches Gedächtnis, und die zugrundeliegende Markov-Kette über den Zuständen der Datenstruktur ist ergodisch. Daraus folgt, daß jede mögliche Anordnung der Liste mit positiver stationärer Wahrscheinlichkeit auftritt und somit die Zugriffszeit ab und zu sehr schlecht werden kann [Me77]. Diese Verfahren werden *ergodische* Heuristiken genannt.

Man kann selbst-organisierende Listen aber auch dazu benutzen, um statische Datenstrukturen zu erhalten. In diesem Fall muß der Algorithmus so modifiziert werden, daß die zugrundeliegende Markov-Kette absorbierend wird. Irgendwann bleibt sie in einem absorbierenden Zustand hängen, und dieser beschreibt dann die statische Datenstruktur, die von diesem Zeitpunkt ab ausschließlich verwendet wird. Das Ziel dabei ist, die Wahrscheinlichkeit der Absorption in „guten“ Konfigurationen groß zu machen, d.h. die erwartete Suchzeit der statischen Datenstruktur zu minimieren. Dies kann jedoch nur sinnvoll sein, wenn man *unabhängige* und identisch verteilte Zugriffe voraussetzt. Solche Verfahren bezeichnen wir als *absorbierende* Heuristiken .

In der Literatur gibt es einige Vorschläge für absorbierende Heuristiken für adaptive Listen. Es wurde aber stets nur gezeigt, daß die Wahrscheinlichkeit der Absorption in der optimalen Konfiguration beliebig nahe an 1 liegen kann, wenn man die Regel entsprechend wählt [OHM90, ON93]. In einer anderen Arbeit [R92] wird ein experimenteller Ansatz beschrieben, bei dem gemäß des Paradigma vom *Simulated Annealing* die Heuristik immer mehr „abgekühlt“ wird, d. h. immer seltener aufgerufen wird, bis irgendwann ein statischer Zustand erreicht ist.

Wir wollen im folgenden den Zusammenhang zwischen ergodischen und absorbierenden Verfahren systematisch beschreiben. Eine genaue Analyse der Verfahren ermöglicht Aussagen über die Güte der erzielten statischen Struktur und über die nötige Konvergenzzeit. Unter anderem kann dabei ein Tradeoff beobachtet werden: wenn die Konvergenzzeit klein ist, hat das Verfahren nur wenig Gelegenheit, die Verteilung der Quelle zu erlernen, und die Güte der erzielten Struktur ist schlecht (vgl. Satz 1.27).

Im Zusammenhang mit diesen Überlegungen stehen die Ergebnisse von *Hofri* und *Shachnai* [HS91]. Sie haben folgenden *Synergie-Effekt* festgestellt: wenn für zwei Listenelemente i und j die Zugriffswahrscheinlichkeiten p_i und p_j sehr verschieden sind, dann ist es für geeignete Verfahren einfach, diese Elemente richtig zu trennen und das Element mit der größeren Wahrscheinlichkeit vor das andere zu setzen. Wenn andererseits die Wahrscheinlichkeiten p_i und p_j nahe beieinander liegen, dann ist es schwierig, diese Elemente in der richtigen Reihenfolge in der endgültigen Liste zu plazieren. Aber das macht nichts, denn eine Vertauschung wird sich in der erwarteten Suchzeit kaum Auswirken, da eben die Wahrscheinlichkeiten ungefähr gleich groß sind.

Diese Beobachtung ist die intuitive Begründung dafür, daß es möglich ist, verteilungsunabhängige Schranken für die Güte der erzielten Struktur zu zeigen.

Grundsätzlich sind zwei Vorgehensweisen möglich, um statische Strukturen zu erhalten: entweder werden absorbierende Heuristiken verwendet, oder die Operation der ergodischen Verfahren wird nach einer bestimmten Zeit (*stopping time*) eingestellt. Im zweiten Fall gelten für die erwartete Suchzeit nach einer Stopping Time von t Schritten die gleichen Aussagen, wie sie im letzten Kapitel über die Konvergenzgeschwindigkeit gemacht wurden. Deshalb beschäftigen wir uns nun mit absorbierenden Listenalgorithmen.

Absorption

Wir betrachten folgendes allgemeine Schema, um aus ergodischen Verfahren absorbierende Regeln abzuleiten:

> Gegeben sei ein ergodischer Listenalgorithmus, der nachgefragte Elemente entweder an den Listenkopf setzt oder gar nicht bewegt.

Der korrespondierende absorbierende Algorithmus bewegt jedes Element nur einmal und plaziert es damit an seine endgültige Position. Wenn der ergodische Algorithmus ein Element zum ersten Mal nach vorne schiebt, setzt das absorbierende Verfahren es direkt vor alle Elemente, die noch nicht bewegt wurden.

Folgendes Beispiel des absorbierenden MTF-Verfahrens zeigt, wie der statische Teil der Liste (grau schattiert) vom Listenkopf ausgehend nach hinten wächst, wenn die Zugriffsfolge $\sigma = 1, 3, 5, 3, 4, 2$ bearbeitet wird. Nachdem $n - 1$ Elemente plaziert wurden, steht die Struktur fest.

$\sigma_1 = 1$:	1	2	3	4	5
$\sigma_2 = 3$:	1	3	2	4	5
$\sigma_3 = 5$:	1	3	5	2	4
$\sigma_4 = 3$:	1	3	5	2	4
$\sigma_5 = 4$:	1	3	5	4	2
$\sigma_6 = 2$:	1	3	5	4	2

Literaturübersicht

Die absorbierende MTF-Regel wurde von *Bitner* [B79] unter dem Namen FirstRequestRule eingeführt. Das absorbierende Gegenstück zur Simple(k)-MTF-Regel wurde von *Oommen, Hansen* und *Munro* betrachtet [OHM90]. Dort wurde vermutet, aber nicht gezeigt, daß sie asymptotisch optimal ist. Wir geben den ensprechenden Beweis in Satz 1.44.

Die entsprechende Batched(k)-Regel wurde offenbar noch nicht untersucht. Die in [B79] eingeführten ergodischen Wait-k and Move-Regeln benutzen für jedes Element einen Zugriffszähler mit Wertebereich $0, \ldots, k$. Diese Zähler sind zu Anfang auf 0 gesetzt und werden bei jeder Anfrage des Elements erhöht. Sobald für ein Element der Zähler den Wert k erreicht, wird dieses Element an den Anfang der Liste gesetzt und der Zähler auf 0 zurückgesetzt. Dieser Regel entspricht eine absorbierende Regel, die ebenfalls in [OHM90] betrachtet wurde. Sie haben gezeigt, daß diese Regel asymptotisch optimal ist, wenn $k \to \infty$. Interessanterweise kann man von verschiedenen ergodischen Verfahren zu dieser Regel gelangen: die absorbierenden Versionen der Limited Counter Schemes [HS91a, HS91, HS98] und der Sort-by-Delay(k)-Regeln ergeben alle dieses Verfahren. Deshalb werden wir im folgenden von der absorbierenden SBD(k)-Regel sprechen und die in Abschnitt 1.4.3 gemachten Analysen übertragen.

Eine weitere absorbierende Regel, die keiner ergodischen Heuristik entspricht, verdient Beachtung. Diese von *Oommen* und *Ng* [ON93] eingeführte Heuristik ist verwandt mit der absorbierenden Simple(k)-MTF-Regel. Bei der Entscheidung, wann k mal hintereinander auf das gleiche Element zugegriffen worden ist, werden aber die Elemente, die bereits ihre endgültige Position gefunden haben, ausgeblendet. Das hat den Effekt, daß man die absorbierende Simple(k)-Regel anwendet, aber mit bedingten Zugriffswahrscheinlichkeiten. Die Bedingung wird also dargestellt durch die Menge der schon bewegten Elemente.

Im weiteren interessieren wir uns insbesondere für die Konvergenzgeschwindigkeit der verschiedenen Verfahren und den Tradeoff zwischen der erreichten Güte der Struktur und der nötigen Zeit bis zur Absorption. Zur Analyse beobachten wir folgende wichtige Aussage, die ergodische und absorbierende Heuristiken verbindet und für den Fall der absorbierenden MTF-Regel schon in [B79] gezeigt wurde. Wie bei dem oben angegebenen allgemeinen Schema setzen wir voraus, daß der ergodische Listenalgorithmus ein Element entweder gar nicht bewegt oder an den Listenkopf setzt.

Satz 1.42. *Für jede Verteilung über den Startkonfigurationen der Listen ist die Wahrscheinlichkeit, nach einer bestimmten Schrittzahl eine gewisse Listenkonfiguration zu erhalten, für die ergodische und die absorbierende Variante einer Regel gleich, wenn die Zugriffe unabhängig und identisch verteilt sind.*

Beweis. Für jede Zugriffsfolge $\sigma_1, \sigma_2, \ldots, \sigma_m$ für die ergodische Regel produziert die umgedrehte Zugriffsfolge $\sigma_m, \ldots, \sigma_1$ die gleiche Liste bei Anwendung der absorbierenden Regel. Da die Zugriffe unabhängig sind, sind die Wahrscheinlichkeiten für diese beiden Folgen gleich. $\square$

Das bedeutet, daß durch die absorbierende Regel eine Listenkonfiguration gemäß der stationären Verteilung der ergodischen Regel erzeugt wird. Die Aufgabe des Sampling der stationären Verteilung einer Markov-Kette ist auch in anderen Anwendungen von Interesse, siehe z.B. *Aldous* [A95] und *Lovász* und *Winkler* [LW95]. Es impliziert, daß die so gewonnene statische Listenkonfiguration durch weitere Anwendung der ergodischen Regel nicht weiter verbessert werden kann.

Absorbierendes Sort-by-Delay(k)

Hier gilt, daß Element j vor Element i plaziert wird, wenn k Zugriffe auf j vor k Zugriffen auf i aufgetreten sind. Die Wahrscheinlichkeit $b(j,i)$ dafür ist

$$\begin{aligned} b(j,i) &= p_j^k \sum_{t=k}^{\infty} \binom{t}{k-1} \sum_{\ell=0}^{k-1} \binom{t-k+1}{\ell} p_j^\ell (1-p_i-p_j)^{t-k+1-\ell} \\ &= \sum_{\ell=0}^{k-1} \binom{2k-1}{\ell} \frac{p_i^\ell p_j^{2k-1-\ell}}{(p_i+p_j)^{2k-1}} \,. \end{aligned}$$

Den Ansatz erhält man, indem man bezüglich der Zeitdauer t zwischen dem ersten Zugriff auf j und dem aktuellen Zugriff konditioniert. Nach dem ersten Zugriff auf j

gab es noch $k-1$ weitere Zugriffe auf j, aber höchstens $k-1$ Nachfragen für i. Wie man nachrechnet oder (gemäß Satz 1.42) ist diese Wahrscheinlichkeit gleich der asymptotischen Wahrscheinlichkeit für j vor i der ergodischen Regel, siehe Formel (1.12). Damit ergeben sich für die Qualität der absorbierenden SBD(k)-Regel die gleichen Schranken von Satz 1.24 wie für die ergodischen Regeln.

Die nächste Frage ist, wie lange es dauert, bis Absorption eintritt. Dies ist ein *Coupon Collector Problem*, siehe z.B. [BH97] für einen Überblick. Die Zufallsvariable T_1 beschreibe die Anzahl der Schritte, die nötig ist, damit jedes Element mindestens einmal nachgefragt wurde. Dann interessieren wir uns insbesondere für den Erwartungswert $E[T_1]$. Bei der absorbierenden SBD(k)-Regel wird ein Element plaziert, wenn es k mal gezogen wurde. Für $k = 1$ erhält man das klassische Coupon Collector Problem und für $k > 1$ eine Variante, die *Double Dixie Cup Problem* genannt wird. Der Name stammt von einem Spiel, bei dem jedes von k Kindern versucht, einen kompletten Satz von Bildern aus Eisbechern (*dixie cups*) zu sammeln, und die Kinder untereinander tauschen. Das bedeutet, es genügt, wenn alle Kinder zusammen jedes Bild mindestens k mal finden, durch anschließendes Tauschen erhält dann jedes Kind von jedem Bild mindestens ein Exemplar. Das Problem wurde für den Fall der Gleichverteilung erstmals von *Newman/Shepp* [NS60] untersucht und die asymptotische Formel für die erwartete Zeit $E[T_k]$ bis zu k Sätzen von Elementen

$$E[T_k] \sim n\ln(n) + (k-1)n\ln(\ln(n)) \qquad k \text{ fest}\,,\ n \to \infty$$

hergeleitet ($\ln n$ bezeichnet den natürlichen Logarithmus). An diesem Ergebnis erkennt man, daß die Wartezeit für das erste Exemplar im Mittel $\ln n$ ist, während jedes weitere Exemplar nach nur $\ln\ln n$ Ziehungen gefunden wird. Eine weitergehende Analyse wurde in [B63] durchgeführt, aber sie gilt nur für Verteilungen, bei denen das Verhältnis zweier Wahrscheinlichkeiten durch eine Konstante beschränkt ist. Die geometrische Verteilung oder Zipf-Verteilung fallen nicht darunter. In [BH97, B63, FGT92] findet man die Darstellung

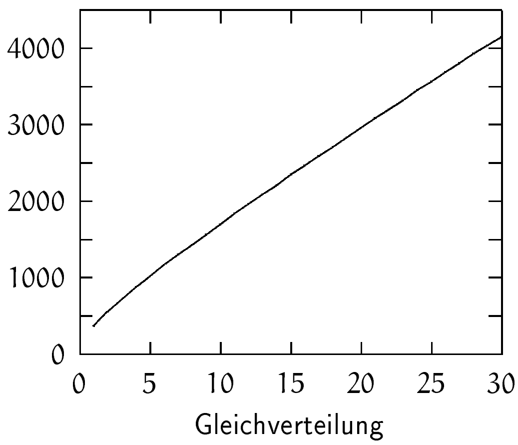
$$E[T_k] = \int_0^\infty 1 - \prod_{i=1}^n (1 - e^{-p_i t} e_k(p_i t))\ dt,$$

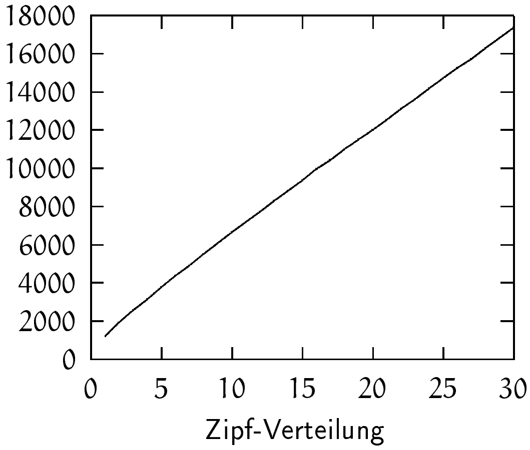
wobei $e_k(x) = \sum_{j=0}^{k-1} x^j/j!$ die abgeschnittene Exponentialreihe ist. Diese Integraldarstellung wurde im Fall $k = 1$ in [BP96] untersucht und unter anderem folgendes asymptotische Resultat für die verallgemeinerte Zipf-Verteilung $p_i = 1/(H_n(\alpha)i^\alpha)$ gezeigt

$$E[T_1] \sim \left\{ \begin{array}{ccc} \frac{n}{1-\alpha}\ln(n) & : & 0 < \alpha < 1 \\ n(\ln n)^2 & : & \alpha = 1 \\ H_n(\alpha)n^\alpha \ln n & : & \alpha > 1 \end{array} \right\}.$$

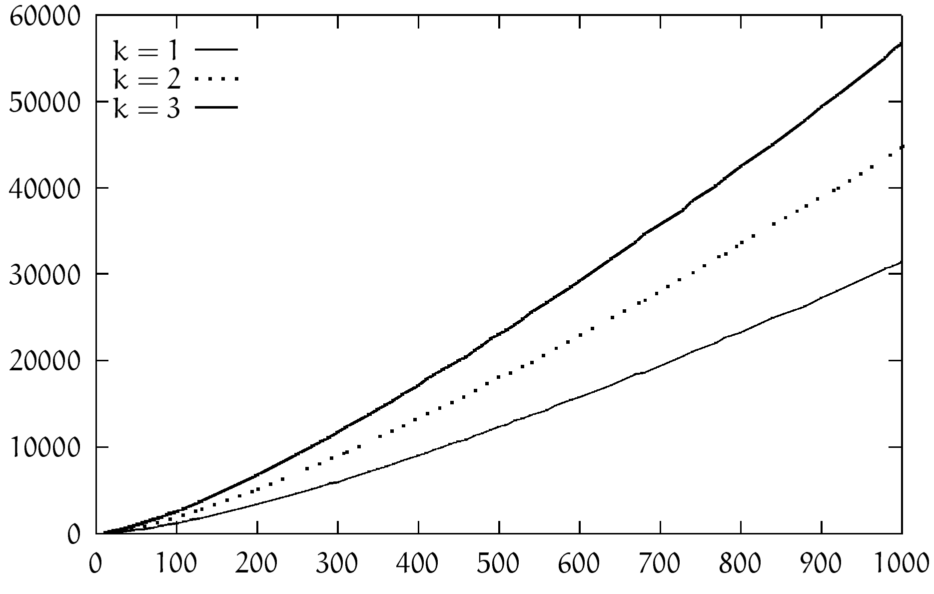
Die Analyse läßt sich jedoch nicht auf beliebige $k > 1$ erweitern, da man nur den führenden Term der asymptotischen Entwicklung ($n\ln n$ bei Gleichverteilung) erhält, nicht aber die entscheidenden Ausdrücke niedrigerer Ordnung ($(k-1)n\ln\ln n$ bei der Gleichverteilung).

Die Diagramme zeigen die Ergebnisse von numerischen Simulationen mit den SBD(k)-Verfahren. Für jeden Punkt wurde der Mittelwert aus 100 Experimenten gebildet. Die ersten beiden Bilder zeigen die Wartezeit T_k für $k = 1, \dots, 30$ bei $n = 100$ Listenelementen und bei Gleichverteilung und Zipf-Verteilung der Zugriffe. Man erkennt die lineare Abhängigkeit von k nicht nur bei der Gleichverteilung.

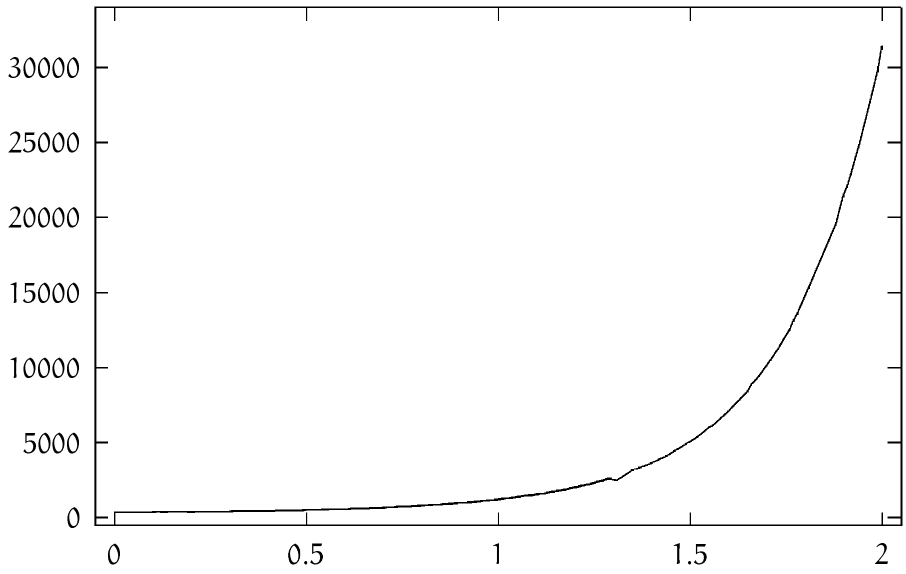




Das zweite Bild stellt das Wachstum der Wartezeiten bei $k = 1, 2, 3$, Zipf-Verteilung und Listenlängen zwischen 10 und 1000 gegenüber.



Das folgende Diagramm vergleicht die Wartezeiten beim klassischen Coupon Collector Problem ($k = 1$) mit $n = 100$ Elementen auf der verallgemeinerten Zipf-Verteilung mit Parameter α zwischen 0 (Gleichverteilung) und 2.



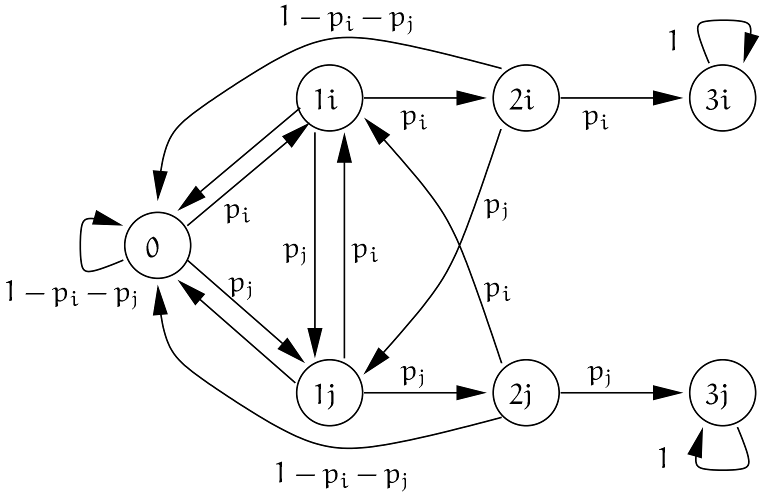
Absorbierendes Simple(k)-Move-to-Front

Diese Regel arbeitet folgendermaßen: wenn ein Listenelement zum ersten Mal k mal hintereinander angefragt wurde, wird es auf seine endgültige Position gesetzt. Dabei wird es so weit nach vorne geschoben, daß davor nur noch Elemente stehen, die ebenfalls schon ihre endgültige Position erreicht haben. Dieses Verfahren wurde bereits in [OHM90] betrachtet, der Nachweis der Konvergenz zu einer optimalen Konfiguration bei $k \to \infty$ aber als offenes Problem bezeichnet. Diese Aussage ist aber eine unmittelbare Folgerung von Satz 1.42 und der entsprechenden Aussage für die ergodische Regel, vgl. (1.2). Wir geben auch einen direkten Beweis an:

Lemma 1.43. *Bei der absorbierenden Simple(k)-MTF-Regel steht in der endgültigen Konfiguration i vor j mit Wahrscheinlichkeit*

$$b_k(i,j) = \frac{p_i^k(1-p_i)(1-p_j^k)}{p_i^k(1-p_i)+p_j^k(1-p_j)-(2-p_i-p_j)(p_ip_j)^k}.$$

Beweis. Wir betrachten die zugehörige absorbierende Markov-Kette, die hier am Beispiel $k=3$ dargestellt ist.



Wir suchen die Wahrscheinlichkeit $\mathrm{Prob}(Z)$ der Absorption in Zustand (k, i), wenn man in einem Zustand Z startet. Dazu betrachten wir für jeden Zustand Z die bedingten Absorptionswahrscheinlichkeiten nach einem Schritt in der Markov-Kette und erhalten

$$\begin{aligned}
\mathrm{Prob}(k,i) &= 1\\
\mathrm{Prob}(k,j) &= 0\\
\mathrm{Prob}(0) &= p_i\mathrm{Prob}(1,i) + p_j\mathrm{Prob}(1,j) + (1-p_i-p_j)\mathrm{Prob}(0)\\
\mathrm{Prob}(\ell,i) &= p_i\mathrm{Prob}(\ell+1,i) + p_j\mathrm{Prob}(1,j) + (1-p_i-p_j)\mathrm{Prob}(0) \quad (\ell = 1,\dots,k-1)\\
\mathrm{Prob}(\ell,j) &= p_i\mathrm{Prob}(1,i) + p_j\mathrm{Prob}(\ell+1,j) + (1-p_i-p_j)\mathrm{Prob}(0) \quad (\ell = 1,\dots,k-1).
\end{aligned}$$

Daraus folgt zunächst

$$\begin{aligned}
\mathrm{Prob}(0) &= p_i\mathrm{Prob}(1,i) + p_j\mathrm{Prob}(1,j) + (1-p_i-p_j)\mathrm{Prob}(0)\\
\mathrm{Prob}(1,i) &= p_i^{k-1} + (p_j\mathrm{Prob}(1,j) + (1-p_i-p_j)\mathrm{Prob}(0))(1-p_i^{k-1})/(1-p_i)\\
\mathrm{Prob}(1,j) &= (p_i\mathrm{Prob}(1,i) + (1-p_i-p_j)\mathrm{Prob}(0))(1-p_j^{k-1})/(1-p_j)\ .
\end{aligned}$$

Wenn man dieses übriggebliebene Gleichungssystem in den drei Variablen $\mathrm{Prob}(0)$, $\mathrm{Prob}(1,i)$, $\mathrm{Prob}(1,j)$ löst, erhält man schließlich

$$b_k(i,j) = \mathrm{Prob}(0) = \frac{p_i^k(1-p_i)(1-p_j^k)}{p_i^k(1-p_i) + p_j^k(1-p_j) - (2-p_i-p_j)(p_ip_j)^k}\ .$$

□

Satz 1.44. *Für $k \to \infty$ konvergiert die Wahrscheinlichkeit der Absorption in einer optimalen Listenkonfiguration unter der absorbierenden Simple(k)-MTF-Regel gegen 1.*

Absorbierendes Batched(k)-Move-to-Front

Hier findet man analog zu oben, daß die Wahrscheinlichkeit für j vor i in der endgültigen Konfiguration durch

$$b'_k(j,i) = \frac{p_j^k}{p_i^k + p_j^k}$$

gegeben ist. Damit gelten die gleichen Schranken wie für die asymptotischen erwarteten Kosten, siehe (1.2). Jetzt sind wir in der Lage, auch die erwartete Zeit bis zur Absorption berechnen zu können. Dazu betrachten wir die Menge aller k-Tupel über dem Alphabet $\mathcal{A}$, wobei $[i_1 \dots i_k]$ mit Wahrscheinlichkeit $p_{i_1} \cdots p_{i_k}$ auftritt. Gesucht ist die Wartezeit T_k , bis die Teilmenge der n Tupel $D = \{[1 \dots 1], \dots, [n \dots n]\}$ gezogen wurde. Für diese Fragestellung gilt nach [BH97]

$$E[T_k] = \int_0^\infty 1 - \prod_{i \in D} (1 - e^{-p_i t})\,dt.$$

Durch eine Modifikation der Rechnung aus [BP96] erhält man für die Gleichverteilung

$$E[T_k] \sim n^k \ln n$$

und für die verallgemeinerte Zipf-Verteilung

$$E[T_k] \sim H_n(\alpha)^k n^{\alpha k} \ln n,$$

speziell für $\alpha = 1$ also $E[T_k] \sim n^k (\ln n)^{k+1}$. Damit folgt, daß die Zeit bis zur Absorption exponentiell mit k wächst, was angesichts der Batched(k)-Regel zu erwarten war.

1.7. Randomisierte Listenalgorithmen

In diesem Abschnitt werden wir einige Listenalgorithmen betrachten, die zu ihren Entscheidungen nicht nur Informationen über vergangene Zugriffe, sondern auch zufällige Bits benutzen dürfen. Bei der kompetitiven Analyse ist insbesondere das Modell des blinden Gegners (*oblivious adversary*) interessant. Ein solcher Gegenspieler darf die Zufallsentscheidungen des Online-Algorithmus nicht sehen, sondern muß die gesamte Anfragefolge im voraus angeben.

Bei Modellen mit stärkeren Gegnern, die die Konstruktion der Zugriffsfolge von den zufälligen Entscheidungen abhängig machen dürfen (*adaptive adversaries*), nützt die Randomisierung nichts. Wenn der Gegner die Anfragen in Abhängigkeit der Zufallsentscheidungen stellen darf, und darüber hinaus erst am Ende die gesamte Folge abarbeiten muß (*adaptive offline adversary*), so existiert auch stets ein deterministischer Listenalgorithmus mit gleichem kompetitiven Faktor wie der randomisierte Algorithmus [BBKTW94]. Gegenüber einem derart starken Gegner bringt Randomisierung also keinen Vorteil.

Im folgenden werden wir nur noch das Modell des blinden Gegners betrachten. Sei σ eine beliebige Zugriffsfolge, $\text{OPT}(\sigma)$ die Kosten des optimalen Offline-Gegners und $E[\text{ALG}(\sigma)]$ die erwarteten Kosten des randomisierten Online-Algorithmus, wobei der Erwartungswert über die zufälligen Entscheidungen des Algorithmus gebildet wird. Dann hat ALG einen kompetitiven Faktor c, falls eine Konstante a existiert, so daß für alle Folgen σ gilt

$$E[\text{ALG}(\sigma)] \leq c \cdot \text{OPT}(\sigma) + a.$$

Der beste bisher bekannte randomisierte Listenalgorithmus COMBINATION von *S. Albers et.al.* [ASW95] besitzt den kompetitiven Faktor 1.6, der damit deutlich unter der unteren Schranke von 2 für deterministische Listenalgorithmen liegt. Für randomisierte Verfahren ist eine untere Schranke von 1.5 bekannt [T93].

Bei kompetitiver Analyse besteht der Vorteil randomisierter Verfahren darin, daß sie ihr Verhalten gegenüber dem Gegner verschleiern. Der Gegner kann keine Zugriffsfolge konstruieren, die für den Algorithmus eine Eingabe im schlechtesten Fall darstellt. Manche Verfahren (z.B. TIMESTAMP(p) [A98]) sind eine randomisierte Mischung zweier deterministischer Algorithmen, die unterschiedliche Worst-Case-Eingaben besitzen. Da der Gegner nicht weiß, welches Verfahren beim aktuellen Zugriff verwendet wird, ist es ihm auch nicht möglich, die schwächste Stelle eines deterministischen Verfahrens zu treffen [BoE97]. Andere Verfahren (z.B. BIT [RWS94]) benutzen nur zur Initialisierung einige zufällige Bits zur Auswahl eines deterministischen Algorithmus, der anschließend ausschließlich benutzt wird, und sind schon dadurch gegenüber einem blinden Gegner im Vorteil.

Bei unabhängigen, identisch verteilten Zugriffen gemäß einer festen Wahrscheinlichkeitsverteilung bringt Randomisierung keinen Gewinn.

1.7.1. RMTF(p)

Wir untersuchen eine einfache randomisierte Variante des Nach-Vorne-Schieben-Algorithmus und geben einen vereinfachten Beweis für folgenden Satz aus [G97] an.

> RMTF(p): Sei $0 < p \leq 1$. Wenn auf das Listenelement x zugegriffen wurde, dann setze x mit Wahrscheinlichkeit p an den Listenkopf. Mit Wahrscheinlichkeit $1 - p$ tue nichts.

Satz 1.45. *Der kompetitive Faktor von* RMTF(p) *beträgt* $\max\{2, 1/p\}$.

Beweis. Sei eine beliebige Zugriffsfolge $\sigma = (\sigma_1, \ldots, \sigma_m)$ gegeben und sei $\sigma_t = x$ die aktuelle Anfrage.

Wie vorher sei $\langle y, x\rangle$ eine Inversion, wenn in der von RMTF(p) verwalteten Liste y vor x steht, in der optimalen Liste aber x vor y. Sei k die Position von x in OPT's Liste, und r die Anzahl der Inversionen $\langle y, x\rangle$. Dann sind die Kosten von RMTF(p) in diesem Schritt $\text{RMTF}(p)_t \leq k + r$ und die Kosten des optimalen Algorithmus $\text{OPT}_t = k$.

Wir benutzen amortisierte Analyse, um mit Hilfe einer Potentialfunktion die zusätzlichen Kosten aufgrund der Inversionen ausgleichen zu können. Mit Wahrscheinlichkeit p wird das nachgefragte Element x nach vorne gesetzt, und dadurch werden alle Inversionen $\langle y, x\rangle$ verschwinden. Allgemeiner werden mit Wahrscheinlichkeit $p(1-p)^{(j-1)}$ erst nach j-maliger Nachfrage nach x diese Inversionen verschwinden. Eine Inversion verursacht also erwartete Kosten von $\sum_{j=1}^{\infty} jp(1-p)^{(j-1)} = 1/p$ und muß dadurch mit dem Faktor $1/p$ gewichtet werden. Die Potentialfunktion lautet damit

$$\Phi = \sum_{\substack{y \neq z \\ \langle y,z\rangle \text{ Inversion}}} 1/p \, .$$

Beim Zugriff auf x wird OPT sein x auf eine Position k' rücken, $1 \leq k' \leq k$. Der Online-Algorithmus RMTF(p) wird mit Wahrscheinlichkeit p das Element x an den Listenkopf setzen. Dann verschwinden die r Inversionen vom Typ $\langle y, x\rangle$, aber k' neue Inversionen der Art $\langle x, z\rangle$ kommen hinzu, da der optimale Algorithmus sein x nur bis zur Position k' bewegt. Wenn mit Wahrscheinlichkeit $1-p$ das Element x an seinem Platz verbleibt, dann bleiben die r Inversionen erhalten, und es können bis zu $k-k'$ Inversionen der Art $\langle z, x\rangle$ neu entstehen, indem OPT sein x an z vorbei nach vorne bewegt. Die erwartete Potentialänderung beträgt somit höchstens

$$\frac{1}{p} \cdot \left[p(k'-r) + (1-p)(k-k')\right] \;=\; -r + \frac{(1-p)k + (2p-1)k'}{p},$$

und die amortisierten erwarteten Kosten a_t in Schritt t sind

$$a_t \;=\; \text{RMTF}(p)_t + \Delta\Phi \;\leq\; k + r - r + \frac{(1-p)k + (2p-1)k'}{p}.$$

1. Fall: $p \leq 1/2$

   In diesem Fall ist $(1-p)k + (2p-1)k' \leq (1-p)k$ für $0 \leq k' \leq k-1$. Für die amortisierten Kosten folgt

   $$a_t \;\leq\; \left(1 + \frac{1-p}{p}\right) k \;=\; k/p,$$

   und für den kompetitiven Faktor $a_t \leq 1/p \cdot \text{OPT}_t$.

2. Fall: $p > 1/2$

   In diesem Fall ist $(1-p)k+(2p-1)k' \leq pk$ für $0 \leq k' \leq k-1$. Für die amortisierten Kosten folgt

   $$a_t \;\leq\; \left(1 + \frac{p}{p}\right) k,$$

   und für den kompetitiven Faktor $a_t \leq 2 \cdot \text{OPT}_t$.

Insgesamt folgt für jeden Schritt t, daß $a_t \leq \max\{2, 1/p\} \cdot \mathrm{OPT}_t$, und damit auch für die gesamten amortisierten Kosten. □

Diese Schranke ist scharf für $p \leq 1/2$, wie man folgendermaßen sieht (vgl. [BE98] für $p = 1/2$). Wir betrachten die Zugriffsfolge

$$\sigma = x_1^k, x_2^k, \ldots, x_n^k.$$

Der MTF-Algorithmus erzeugt Kosten von $\mathrm{MTF}(\sigma) = n(k-1) + n(n+1)/2$, wenn er in Konfiguration $\langle x_1, \ldots, x_n \rangle$ gestartet wird. Das RMTF(p)-Verfahren zieht ein Element im Mittel erst nach $1/p$ Zugriffen nach vorne und produziert daher bei Start in einer beliebigen Konfiguration die erwarteten Kosten von mindestens $E[\mathrm{RMTF}(\sigma)] \geq n(k - 1/p) + n(n+1)/(2p)$. Für das Verhältnis folgt

$$\frac{E[\mathrm{RMTF}(\sigma)]}{\mathrm{OPT}(\sigma)} \geq \frac{1/p(n^2 - n) + 2nk}{n^2 + n + 2n(k-1)} = \frac{1}{p} + O(1/n)$$

für feste k und $n \to \infty$. Also kann das Verhältnis beliebig nahe an $1/p$ liegen. Dieses Resultat widerlegt die Vermutung aus [AMN93], daß RMTF(1/2) ebenso 1.75-kompetitiv ist wie der BIT-Algorithmus. Der Grund dafür ist, daß BIT ein Element garantiert am Listenkopf plaziert hat, wenn es zweimal hintereinander nachgefragt wurde. Bei RMTF(1/2) muß *im Mittel* zweimal hintereinander zugegriffen werden, damit das Element nach vorne geschoben wird, es kann aber beliebig lange dauern.

1.7.2. BIT

Als nächstes analysieren wir das Verhalten des randomisierten Algorithmus BIT auf stochastischen Quellen.

> BIT: Mit jedem Listenelement j ist ein Bit $b(j) \in \{0, 1\}$ assoziiert, das bei jedem Zugriff auf j gekippt wird. Wenn $b(j) = 0$ ist, wird j an den Listenanfang gesetzt, ansonsten passiert nichts. Die n Bits werden zufällig unabhängig und gleichverteilt initialisiert.

Wie erwähnt erzielt BIT gegenüber dem blinden Gegner einen kompetitiven Faktor von 1.75 [RWS94].

Satz 1.46. *Bei unabhängigen Zugriffen gemäß der Verteilung* $\mathbf{p} = (p_1, \ldots, p_n)$ *erzeugt* BIT *die asymptotischen erwarteten Kosten pro Zugriff*

$$E(\mathbf{p}) \;=\; 1 + \sum_{j<i} \frac{7p_i^3 p_j + 18 p_i^2 p_j^2 + 7 p_i p_j^3}{4(p_i + p_j)^3} \;\leq\; \frac{7}{4} \cdot \mathrm{OPT}(\mathbf{p}).$$

Beweis. Wir betrachten ein Paar $i \neq j$ von Elementen und untersuchen für $(b(i), b(j))$ die vier Fälle $(0,0)$, $(0,1)$, $(1,0)$ und $(1,1)$, die alle unabhängig voneinander mit Wahrscheinlichkeit $1/4$ auftreten. Dabei bedeutet $b(i) = 0$, daß i beim letzten Zugriff an den

Listenkopf gesetzt wurde, und $b(i) = 1$, daß i bei der vorletzten Nachfrage nach vorne bewegt wurde. Für die asymptotische Wahrscheinlichkeit, daß j vor i steht, folgt dann

$$\begin{aligned}
b(j,i \mid b(i)=0, b(j)=0) &= \frac{p_j}{p_i+p_j} \\
b(j,i \mid b(i)=0, b(j)=1) &= \frac{p_j^2}{(p_i+p_j)^2} \\
b(j,i \mid b(i)=1, b(j)=0) &= \frac{p_j}{p_i+p_j} + \frac{p_i p_j}{(p_i+p_j)^2} \\
b(j,i \mid b(i)=1, b(j)=1) &= \frac{2p_i p_j^2}{(p_i+p_j)^3} + \frac{p_j^2}{(p_i+p_j)^2} \, .
\end{aligned}$$

Die folgende Tabelle erläutert die Zugriffsfolgen, die zu obigen Formeln führen. Dabei stehen drei Punkte (. . .) für eine Folge beliebiger Länge von Elementen verschieden von i und j; eine solche Folge hat die Wahrscheinlichkeit $\sum_{t=0}^{\infty}(1-p_i-p_j)^t = 1/(p_i+p_j)$.

$b(i)$	$b(j)$	Zugriffsfolge, die $\langle j,i \rangle$ erzeugt
0	0	$j \ldots$
0	1	$j \ldots j \ldots$
1	0	$j \ldots$
		$j \ldots i \ldots$
1	1	$j \ldots j \ldots i \ldots$
		$j \ldots i \ldots j \ldots$
		$j \ldots j \ldots .$

Summation und Division durch 4 gibt

$$b(j,i) = \frac{4p_j^3 + 9p_j^2 p_i + 3p_j p_i^2}{4(p_i+p_j)^3} \, .$$

Also folgt für die asymptotischen erwarteten Kosten

$$\begin{aligned}
E(\mathbf{p}) &= \sum_{i=1}^{n} p_i \cdot (1 + \sum_{j \neq i} b(j,i)) \\
&= 1 + \sum_{j<i} \frac{7p_i^3 p_j + 18 p_i^2 p_j^2 + 7 p_i p_j^3}{4(p_i+p_j)^3} \\
&= \sum_{i=1}^{n} \sum_{j=1}^{n} \frac{7p_i^3 p_j + 18 p_i^2 p_j^2 + 7 p_i p_j^3}{4(p_i+p_j)^3} \, .
\end{aligned}$$

Für die Abschätzung zu den optimalen erwarteten Kosten folgt

$$\begin{aligned}
\frac{E(\mathbf{p})}{OPT(\mathbf{p})} &\leq \max_{1\leq i\leq n} \frac{1}{i}\sum_{j=1}^{i} \frac{7p_i^2p_j + 18p_ip_j^2 + 7p_j^3}{4(p_i+p_j)^3} \\
&\leq \max_{x\geq 1} \frac{7x^3+18x^2+7x}{4(1+x)^3} \\
&\leq 7/4.
\end{aligned}$$

□

Da der Faktor 7/4 im kompetitiven Modell, also für beliebige Zugriffsfolgen gilt, ist er im stochastischen Modell naheliegend. Für bestimmte Verteilungen können wir exakte Analysen vornehmen und bessere Schranken finden.

Satz 1.47. *Die erwarteten Kosten von* BIT *unter der Zipf-Verteilung betragen asymptotisch* ($n \to \infty$)

$$\begin{aligned}
E_{\mathrm{BIT}}(\mathbf{p}) &= \frac{1+14\ln(2)}{8}\cdot\frac{n}{H_n} - \frac{7}{2} + o(1) \\
&\leq 1.3381 \cdot E_{\mathrm{OPT}}.
\end{aligned}$$

Beweis. Wir setzen die Definition $p_i = 1/(iH_n)$ ein und erhalten

$$\begin{aligned}
E_{\mathrm{BIT}}(\mathbf{p}) &= 1/2 + \sum_{i=1}^{n}\sum_{j=1}^{n} \frac{4p_ip_j^3 + 9p_i^2p_j^2 + 3p_i^3p_j}{4(p_i+p_j)^3} \\
&= 1/2 + \frac{1}{4H_n}\cdot\sum_{i=1}^{n}\sum_{j=1}^{n}\frac{4i^2+9ij+3j^2}{(i+j)^3} \\
&= 1/2 + \frac{1}{4H_n}\cdot\Bigg(\sum_{k=2}^{n}\sum_{i=1}^{k-1}\frac{4i^2+9i(k-i)+3(k-i)^2}{k^3} \\
&\qquad + \sum_{k=n+1}^{2n}\sum_{i=k-n}^{n}\frac{4i^2+9i(k-i)+3(k-i)^2}{k^3}\Bigg).
\end{aligned}$$

Dabei haben wir $k = i + j$ substituiert. Durch Expansion der Zähler in $3k^2 + 3ki - 2i^2$ und Summation über i folgt zunächst

$$\begin{aligned}
E_{\mathrm{BIT}}(\mathbf{p}) &= 1/2 + \frac{1}{4H_n}\cdot\Bigg(\sum_{k=2}^{n}\Big(\frac{23}{6} - \frac{7}{2k} - \frac{1}{3k^2}\Big) \\
&\qquad + \sum_{k=n+1}^{2n}\Big(-\frac{23}{6} + \frac{7+14n}{2k} + \frac{1+6n+6n^2}{3k^2} - \frac{2n+6n^2+4n^3}{3k^3}\Big)\Bigg).
\end{aligned}$$

Indem man die asymptotische Entwicklung für die harmonischen Zahlen verwendet, siehe Seite 47, erhält man mit $E_{\mathrm{OPT}}(\mathbf{p}) = n/H_n$ die Abschätzung

$$\begin{aligned} E_{\mathrm{BIT}}(\mathbf{p}) &= \left(\frac{1}{8} + \frac{7\ln(2)}{4}\right) \cdot \frac{n}{H_n} - \frac{7}{2} + o(1) \\ &= \left(\frac{1}{8} + \frac{7\ln(2)}{4} + o(1)\right) \cdot E_{\mathrm{OPT}}(\mathbf{p}), \end{aligned}$$

wobei $o(1) \to 0$ für $n \to \infty$. $\square$

Der BIT-Algorithmus verwendet nur n zufällige Bits, unabhängig von der Länge der Zugriffsfolge. Darin unterscheidet er sich von anderen randomisierten Verfahren wie RMTF(p), die bei jeder Anfrage mindestens ein Zufallsbit benötigen. Online-Algorithmen wie BIT, deren Bedarf an zufälligen Bits nicht von der Anzahl der Zugriffe abhängt, werden *schwach randomisiert (barely random)* genannt [RWS94].

Wir interessieren uns dafür, wieviel Zufall wirklich nötig ist, und ob ein Tradeoff zwischen der Ressource Zufall und dem erzielbaren kompetitiven Faktor möglich ist. Man vergleiche hierzu eine Arbeit von *P. Raghavan* und *M. Snir* [RS89], in der für Caching-Algorithmen ein Tradeoff zwischen der Cachegröße und der Anzahl der verwendeten zufälligen Bits, um einen gegebenen kompetitiven Faktor zu erreichen, gezeigt wurde.

Wir gehen anders vor und zeigen anhand der Analyse des BIT-Algorithmus, daß $\lceil \log(n+1) \rceil$ Zufallsbits hinreichend sind für einen kompetitiven Faktor von 1.75. Zunächst betrachten wir eine Variante, die nur ein zufalliges Bit benutzen darf.

> BIT(1): Jedem Listenelement x ist ein Bit $b(x)$ zugeordnet. Zu Beginn werden alle $b(x)$ mit dem gleichen zufälligen Bit initialisiert, danach läuft der Algorithmus ab wie BIT.

Im $(i-1)$-Kostenmodell ist dieses Verfahren nicht besser als deterministische Algorithmen.

Lemma 1.48. *Es gibt Zugriffsfolgen σ, so daß im $(i-1)$-Kostenmodell das Verhältnis von $E[C_{\mathrm{BIT}(1)}(\sigma)]$ zu $C_{\mathrm{OPT}}(\sigma)$ beliebig nahe an 2 liegt.*

Beweis. Wir betrachten eine zweielementige Liste über $\{x, y\}$. Durch zwei initiale Zugriffe auf y kann sichergestellt werden, daß y vor x steht. Wenn die ersten drei Zugriffe der Folge $\sigma' = y, y, y, x, y, y$ abgearbeitet wurden, sind die Bits von x und y verschieden. Mit Wahrscheinlichkeit $1/2$ setzt $\mathrm{BIT}(1)$ dann das x vor y, und beim nächsten Zugriff wird y nicht nach vorne geschoben, so daß Kosten 3 entstehen. Mit Wahrscheinlichkeit $1/2$ wird x nicht bewegt, dies ist auch die Strategie des optimalen Algorithmus, die Kosten 1 verursacht. Durch m-malige Wiederholung von σ' folgt für $\sigma = y, y, (\sigma')^m$, daß $E[C_{\mathrm{BIT}(1)}(\sigma)] \geq 2m$ und $C_{\mathrm{OPT}}(\sigma) \leq 1 + m$, also

$$\frac{E[C_{\mathrm{BIT}(1)}(\sigma)]}{C_{\mathrm{OPT}}(\sigma)} \geq \frac{2m}{m+1} \longrightarrow 2 \qquad (m \to \infty).$$

$\square$

Als nächstes geben wir an, wie man aus einem Vektor $(Z_1, \dots, Z_k)$ von zufälligen Bits einen Bitvektor $(X_1, \dots, X_n)$ erhält, für den gilt

(a) Die eindimensionalen Randverteilungen sind die Gleichverteilung, d.h. $\mathrm{Prob}(X_i = 0) = \mathrm{Prob}(X_i = 1) = 1/2$ für alle $i = 1, \dots, n$.

(b) Je zwei Variable X_i, X_j sind unabhängig voneinander, d.h. $\mathrm{Prob}(X_i = a \mid X_j = b) = 1/2$ für $i \neq j$ und $a, b \in \mathbb{B}$.

Wie Satz 1.50 zeigen wird, sind diese Bedingungen hinreichend dafür, daß der kompetitive Faktor von BIT weiterhin 1.75 beträgt. Man beachte, daß die Forderungen schwach sind. Beispielsweise können je drei Variable X_i, X_j, X_ℓ voneinander abhängig sein, und Vektoren $(x_1, \dots, x_n) \in \mathbb{B}^n$ dürfen auch die Wahrscheinlichkeit 0 besitzen.

Lemma 1.49. *Für* $n \in \mathbb{N}$ *und* $k \geq \lceil \log(n+1) \rceil$ *gibt es eine Abbildung* $f : (Z_1, \dots, Z_k) \longmapsto (X_1, \dots, X_n)$*, so daß* $(X_1, \dots, X_n)$ *den obigen Bedingungen genügt.*

Beweis. Seien die $K = 2^k - 1$ nichtleeren Teilmengen A_j von $\{1, \dots, k\}$ durchnumeriert mit $1, \dots, K$. Dann setzen wir

$$X_j = \bigoplus_{i \in A_j} Z_i \qquad \text{für } j = 1, \dots, n,$$

wobei $\bigoplus$ die Addition modulo 2 bedeutet. Wenn die Z_i unabhängig und gleichverteilt sind, folgt $\mathrm{Prob}(X_j = 0) = \mathrm{Prob}(X_j = 1) = 1/2$. Denn sei $|A_j| = 1$, dann gilt $\mathrm{Prob}(X_j = 0) = \mathrm{Prob}(Z_i)$ für ein i. Wenn $|A_j| > 1$ ist, so ist $A_j = A_j' \cup \{\ell\}$ für $\ell \notin A_j$. Damit folgt

$$\mathrm{Prob}(X_j = 0) = \mathrm{Prob}(X_j' = 0) \cdot \mathrm{Prob}(Z_\ell = 0) + \mathrm{Prob}(X_j' = 1) \cdot \mathrm{Prob}(Z_\ell = 1) = 1/2,$$

da $\mathrm{Prob}(X_j' = 0) = \mathrm{Prob}(X_j' = 1) = 1/2$ nach Induktionsvoraussetzung. Für die Forderung (b) sei $A_\ell = A_i \cap A_j$ für ein ℓ und $A_i' = A_i - A_\ell$ sowie $A_j' = A_j - A_\ell$. Falls $A_\ell = \emptyset$, so folgt sofort, daß X_i und X_j unabhängig sind. Ansonsten beobachtet man, daß die zugehörigen Zufallsvariablen X_ℓ, X_i', X_j' unabhängig sind und man erhält

$$\begin{aligned}
\mathrm{Prob}(X_i = a \wedge X_j = b) &= \mathrm{Prob}(X_\ell = 0) \cdot \mathrm{Prob}(X_i' = a) \cdot \mathrm{Prob}(X_j' = b) \\
&\quad + \mathrm{Prob}(X_\ell = 1) \cdot \mathrm{Prob}(X_i' = \overline{a}) \cdot \mathrm{Prob}(X_j' = \overline{b}) \\
&= 1/8 + 1/8 \ = \ 1/4.
\end{aligned}$$

Somit folgt (b) aus

$$\mathrm{Prob}(X_i = a \mid X_j = b) = \frac{\mathrm{Prob}(X_i = a \wedge X_j = b)}{\mathrm{Prob}(X_j = b)} = \frac{1/4}{1/2} = \frac{1}{2}\,.$$

□

Satz 1.50. *Wenn für den Bitvektor* $(X_1, \dots, X_n)$ *zur Initialisierung des* BIT*-Algorithmus gilt, daß*

(a) *Prob*$(X_i = 0) =$ *Prob*$(X_i = 1) = 1/2$ *für alle* $i = 1, \ldots, n$ *und*

(b) *Prob*$(X_i = a \mid X_j = b) \leq p$ *für alle* $i \neq j$ *und* $a, b \in \mathbb{B}$,

dann ist BIT *auf beliebigen Eingabefolgen* $(3 + p)/2$*-kompetitiv.*

Beweis. Wir führen eine phasenweise Analyse im $(i-1)$-Kostenmodell durch, wie sie von *S. Albers* [ASW95, A98] vorgeschlagen wurde. Da BIT die Bedingung der paarweisen Unabhängigkeit erfüllt, genügt es, Folgen $\sigma \in \{x, y\}^*$ zu betrachten. Diese Folgen werden in Phasen eingeteilt, wobei eine Phase genau dann endet, wenn mindestens zweimal hintereinander auf das gleiche Element zugegriffen wurde, und der nächste Zugriff auf ein anderes Element gerichtet ist. Dann setzt nämlich BIT das mehrfach nachgefragte Element garantiert nach vorne, und ohne Beschränkung der Allgemeinheit macht ein optimaler Algorithmus auf $\{x, y\}$ dies auch [RW90]. Wenn die letzte Phase mit mehrfachen Zugriffen auf y endete, dann sind drei Typen von Phasen möglich: (a) x^k für $k \geq 2$, (b) $(xy)^k x^\ell$ für $k \geq 1, \ell \geq 2$ und (c) $(xy)^k y^\ell$ für $k \geq 1, \ell \geq 1$.

Wir untersuchen nun diese drei Typen von Phasen jeweils für die vier möglichen Fälle von $b(x), b(y)$ zu Beginn der Phase. Zur anschließenden Berechnung der erwarteten Kosten benutzen wir die Voraussetzungen, daß der Wert eines Bits, o.B.d.A. $b(x)$, gleichverteilt ist, aber das andere Bit $b(y)$ vom Gegner soweit kontrolliert werden kann, wie es die Annahme (b) über die Abhängigkeit von $b(x)$ und $b(y)$ zuläßt.

(a) Phase x^k mit $k \geq 2$. Die Kosten eines optimalen Offline-Verfahrens im $(i-1)$-Kostenmodell betragen 1. Für BIT finden wir

$b(x)$	$b(y)$	Kosten
0	0	1
0	1	1
1	0	2
1	1	2

Die mittleren Kosten betragen also $3/2$, und für das kompetitive Verhältnis folgt ebenfalls $3/2$.

(b) Phase $(xy)^k x^\ell$ mit $k \geq 1, \ell \geq 2$. Ein optimaler Offline-Algorithmus erzielt Kosten $k + 1$. Wegen dem Alternieren der Bits bei BIT betrachten wir die Fälle $k = 2m$ und $k = 2m + 1$ separat.

$b(x)$	$b(y)$	Kosten bei $k = 2m, m \geq 1$	Kosten bei $k = 2m + 1, m \geq 0$
0	0	$3m + 1$	$3m + 4$
0	1	$3m + 1$	$3m + 2$
1	0	$3m$	$3m + 2$
1	1	$3m + 2$	$3m + 2$

Falls k gerade ist, sind die erwarteten Kosten von BIT höchstens $3m + 1/2 + p$ und der kompetitive Faktor höchstens $3/2$. Wenn k ungerade ist, betragen die erwarteten Kosten maximal $3m + 2 + p$, und für den kompetitiven Faktor folgt ebenfalls $\leq 3/2$.

(c) Phase $(xy)^k y^\ell$ mit $k \geq 1, \ell \geq 1$. In dieser Phase erzeugt ein optimaler Offline-Algorithmus Kosten von k. Für BIT finden wir

$b(x)$	$b(y)$	Kosten bei $k = 2m, m \geq 1$	Kosten bei $k = 2m+1, m \geq 0$
0	0	$3m$	$3m+2$
0	1	$3m$	$3m+3$
1	0	$3m+1$	$3m+1$
1	1	$3m$	$3m+1$

Wenn k gerade ist, entstehen erwartete Kosten von höchstens $3m + p/2$, also ein kompetitives Verhältnis von nicht mehr als $3/2 + p/4$. Falls k ungerade ist, folgen mittlere Kosten von höchstens $3m + 3/2 + p/2$. Für $m = 0$ wird hier der kompetitive Faktor maximal: $3/2 + p/2$.

Der kompetitive Faktor des Algorithmus ist das Maximum der kompetitiven Verhältnisse in den einzelnen Phasen und beträgt $(3 + p)/2$. □

Korollar 1.51. Man kann den BIT-Algorithmus auf einer n-elementigen Liste mit nur $\lceil \log(n+1) \rceil$ zufälligen Bits initialisieren, so daß der kompetitive Faktor noch 1.75 bleibt.

Eine interessante offene Frage ist es, eine untere Schranke für den Tradeoff zwischen kompetitivem Faktor und Anzahl der Zufallsbits zu bestimmen. Welcher kompetitive Faktor ist möglich, wenn man z.B. $O(\log \log n)$ zufällige Bits zur Verfügung hat? Durch ausgefeiltere Pseudo-Zufalls-Generatoren als das Schema von Lemma 1.49 ist eventuell eine weitere Derandomisierung von BIT möglich, vgl. [Ni96].

Nach der zufälligen Initialisierung arbeitet BIT deterministisch nach einem sehr einfachen Prinzip. In [G97] wurde versucht, durch eine höhere interne Komplexität die Strategie gegenüber dem Offline-Gegner besser zu verschleiern und dadurch den kompetitiven Faktor zu senken: anstatt das Bit $b(x)$ bei jedem Zugriff auf x zu kippen, wird $b(x)$ durch einen Schritt einer Markov-Kette auf $\{0, 1\}$ bestimmt. Jedoch hat sich gezeigt, daß dadurch keine Verbesserung möglich war.

Die Bestimmung des minimalen kompetitiven Faktors bei randomisierten Listenalgorithmen bleibt ein wichtiges offenes Problem [AM97, ASW95, T93]. Der beste bisher bekannte Algorithmus COMB mit Faktor 1.6 stellt eine randomisierte Mischung des deterministischen TIMESTAMP-Verfahrens und des randomisierten BIT-Algorithmus dar. Die Analyse zeigt, daß BIT hierbei noch eine recht rasche Umordnung der Liste vornimmt. Eventuell liefert die Kombination von TS mit einem „trägeren“ Verfahren als BIT eine Verbesserung. Ein Kandidat dafür ist RANDOMRESET [RWS94], das einen kompetitiven Faktor von $\sqrt{3} \leq 1.733$ aufweist. Die Entscheidungen über die Bewegung einzelner Elemente sind dabei paarweise unabhängig. Allerdings besteht bei diesem Algorithmus die Möglichkeit, daß erst nach 3 aufeinanderfolgenden Zugriffen ein

Element nach vorne bewegt wird. Um eine phasenweise Analyse durchzuführen, muß die Anfragefolge $\sigma \in \{x, y\}^m$ in Abschnitte eingeteilt werden, die genau dann enden, wenn mindestens dreimal hintereinander auf das gleiche Element zugegriffen wird. Während man eine Phasenaufteilung in Abschnitte, die bei mindestens zweimaligen Zugriff auf ein Element enden, übersichtlich durchführen kann (siehe oben), wird eine solche Einteilung bei dreimaligem Zugriff sehr unübersichtlich, und die Kosten der verschiedenen Algorithmen auf den jeweiligen Phasen lassen sich nicht einfach darstellen.

1.8. Das gewichtete Listenproblem

In diesem Abschnitt untersuchen wir eine Verallgemeinerung des Listenproblems, bei dem die Elemente verschiedene Längen bzw. Gewichte besitzen. Eine Instanz dieses Problems besteht also aus den Listenelementen $\mathcal{A} = \{1, \dots, n\}$ und zugehörigen Gewichten $\mathbf{w} = (w_1, \dots, w_n)$ mit $w_j > 0$. Das Lesen eines Elementes j kostet nun w_j, und bei einem Zugriff auf ein bestimmtes Symbol müssen alle Elemente vor und einschließlich dem gesuchten Symbol gelesen werden.

Mit diesem Modell kann man beispielsweise das sequentielle Lesen von Datensätzen von einem Band modellieren, bei dem die Datensätze verschiedene Längen besitzen.

Im Falle von Anfragefolgen einer gedächtnislosen stochastischen Quelle mit Zugriffswahrscheinlichkeiten $\mathbf{p} = (p_1, \dots, p_n)$ wurde das gewichtete Listenproblem in [S56] untersucht, siehe [K98, Seiten 403*f.*]. Bei einer Listenordnung $\langle 1, 2, \dots, n\rangle$ entstehen die erwarteten Kosten pro Zugriff

$$p_1 w_1 + p_2(w_1 + w_2) + p_3(w_1 + w_2 + w_3) + \cdots + p_n(w_1 + \cdots + w_n).$$

Satz 1.52. *(Smith 1956)*
Die Anordnung $\langle 1, 2, \dots, n\rangle$ ist genau dann optimal, wenn

$$\frac{p_1}{w_1} \geq \frac{p_2}{w_2} \geq \cdots \geq \frac{p_n}{w_n}.$$

Unter dem kompetitiven Kostenmodell wurde das gewichtete Listenproblem in [AL94, AMN93] behandelt. Dabei wurden zwei Erweiterungen der Nach-Vorne-Schieben-Regel eingeführt: RANDOM-MOVE-TO-FRONT (RMTF) bewegt Listenelement j nach einem Zugriff nur mit einer Wahrscheinlichkeit proportional zu $1/w_j$ nach vorne, ansonsten passiert nichts. COUNTER-MOVE-TO-FRONT (CMTF) ist die derandomisierte Version von RMTF. Hier wird für jedes Element j ein reellwertiger Zähler $c_j \in [0, 1)$ benutzt, der bei einem Zugriff auf j um einen Wert zwischen 0 und 1 und proportional zu $1/w_j$ inkrementiert wird. Immer wenn dies einen Wert $c_j \geq 1$ ergeben würde, wird davon 1 abgezogen und das Element nach vorne geschoben. Diese Derandomisierung ist sinnvoll, wenn die Zugriffe unabhängig sind, oder wenn man vom Modell des statischen Gegners (*lazy adversary*) ausgeht, wie es in [AMN93] gemacht wird, siehe auch [BM85] für dieses Modell. Sei $w_{max} = \max\{w_j : 1 \leq j \leq n\}$ und w_{min} entsprechend, sowie $W = \sum_{i=1}^{n} w_i$.

Satz 1.53. *(d'Amore et.al. 1993, 1994)*

(a) Gegenüber einem statischen Gegner sind RMTF *und* CMTF *2-kompetitiv.*

*(b) Bei beliebigen deterministischen Gegnern ist der Standard-*MTF*-Algorithmus* $(1 + w_{max}/w_{min})$*-kompetitiv, falls* $w_{max}/w_{min} \leq 2$.

Im letzten Fall muß das Kostenmodell so erweitert werden, daß ein bezahlter Austausch (*paid exchange*) zwischen zwei Elementen x und y, bei denen x vor y steht, gerade $w(x)$ kostet. Wenn man nur Einheitskosten annehmen würde, dann könnte der Gegner vor einem Zugriff das benötigte Element mit Hilfe von bezahlten Austauschen an den Listenkopf bewegen und dadurch weniger Kosten verursachen als beim regulären Durchsuchen der Liste.

Wir bemerken, daß die Bedingung $w_{max}/w_{min} \leq 2$ nicht erforderlich ist. Wir geben eine untere Schranke an, die zeigt, daß der kompetitive Faktor von Satz 1.53 (b) höchstens um einen Faktor $(1 + w_{max}/W) \cdot w_{max}/w_{min}$ über dem Optimum liegt. Dazu benötigen wir

Lemma 1.54. *Seien* $a_1, \dots, a_n$ *und* $b_1, \dots, b_n$*, alle positiv, mit*

$$\frac{a_1}{b_1} \geq \frac{a_2}{b_2} \geq \dots \geq \frac{a_n}{b_n}. \tag{1.21}$$

Sei $A = \sum_{i=1}^{n} a_i$ *und* $B = \sum_{i=1}^{n} b_i$ *sowie* $b_{max} = \max\{b_i : 1 \leq i \leq n\}$ *und* $b_{min} = \min\{b_i : 1 \leq i \leq n\}$*. Dann gilt*

$$\sum_{i=1}^{n} a_i \cdot \sum_{j=1}^{i} b_j \leq \frac{b_{max}}{b_{min} + b_{max}} \cdot A \cdot (B + b_{max}).$$

Beweis. (Induktion nach n)
Für $n = 1$ gilt

$$a_1 \cdot b_1 \leq \frac{1}{2} a_1 (b_1 + b_1).$$

Für $n > 1$ nehmen wir an, daß die Aussage für $n - 1$ gilt. Dabei betrachten wir den Fall, daß b_{max} und b_{min} in $b_1, \dots, b_{n-1}$ liegen. Die Fälle $b_{max} = b_n$ oder $b_{min} = b_n$ behandelt man leicht separat.

$$\begin{aligned}
\sum_{i=1}^{n} a_i \cdot \sum_{j=1}^{i} b_j &\leq \frac{b_{max}}{b_{min} + b_{max}} \cdot (A - a_n) \cdot (B - b_n + b_{max}) + a_n \cdot B \\
&= \frac{b_{max}[AB + (A - a_n)(b_{max} - b_n)] + a_n b_{min} B}{b_{min} + b_{max}} \\
&\leq \frac{b_{max}}{b_{min} + b_{max}} \cdot \big(A(B + b_{max}) - Ab_n - a_n(b_{max} - b_n) + a_n B\big) \\
&\leq \frac{b_{max}}{b_{min} + b_{max}} \cdot \big(A(B + b_{max}) - Ab_n + a_n B\big) \\
&\leq \frac{b_{max}}{b_{min} + b_{max}} \cdot A(B + b_{max}) .
\end{aligned}$$

Die letzte Ungleichung folgt mit Hilfe von (1.21) aus

$$\frac{A}{B} = \frac{\sum_{i=1}^{n} a_i}{\sum_{j=1}^{n} b_j} \geq \min_{1 \leq k \leq n} \left\{ \frac{a_k}{b_k} \right\} = \frac{a_n}{b_n} ,$$

was man mit Induktion nach n sieht. □

Satz 1.55. *Für jeden deterministischen Online-Algorithmus* ALG *und jede Gewichtszuordnung* $(w_1, \ldots, w_n)$ *existieren Zugriffsfolgen* σ*, so daß der kompetitive Faktor mindestens*

$$\left(1 + \frac{w_{min}}{w_{max}}\right) \cdot \frac{W}{W + w_{max}}$$

beträgt.

Beweis. Wir konstruieren die Zugriffsfolge σ der Länge m so, daß stets auf das letzte Element in ALG's Liste zugegriffen wird. Die Kosten betragen dann $\text{ALG}(\sigma) = mW = m \sum_{i=1}^{n} w_i$. Sei m_j die Häufigkeit des Symbols j in σ. Ein möglicher Offline-Algorithmus sortiert seine Liste nach absteigenden m_j/w_j, was Kosten von höchstens $\binom{n}{2} w_{max}$ erzeugt. Sei diese Ordnung durch $\langle 1, 2, \ldots, n \rangle$ gegeben. Dann wird der Algorithmus die Anfragefolge σ mit dieser statischen Liste abarbeiten. Der optimale Offline-Algorithmus OPT hat keine höheren Kosten als dieses Verfahren. Dabei entstehen Kosten von

$$m_1 w_1 + m_2(w_1 + w_2) + \cdots + m_n W \;\leq\; \frac{w_{max}}{w_{min} + w_{max}} \cdot m \cdot (W + w_{max}),$$

wobei mit Hilfe von Lemma 1.54 abgeschätzt wurde. Für das Verhältnis der Kosten folgt

$$\frac{\text{ALG}(\sigma)}{\text{OPT}(\sigma)} \geq \frac{mW}{m(W + w_{max})w_{max}/(w_{min} + w_{max}) + \binom{n}{2} w_{max}},$$

was für feste n und hinreichend große m beliebig nahe an

$$\left(1 + \frac{w_{min}}{w_{max}}\right) \cdot \frac{W}{W + w_{max}}$$

liegt. □

In Hinblick auf Satz 1.52 und die SORT-BY-DELAY(k)-Regeln aus Abschnitt 1.4.3 erscheint auf den ersten Blick eine verallgemeinerte Familie WEIGHTED-SBD(k) möglich, bei der die Liste zum Zeitpunkt t die Form $\langle x_1, \ldots, x_n \rangle$ hat, falls

$$w_{x_1} s_k(x_1, t) \leq w_{x_2} s_k(x_2, t) \leq \cdots \leq w_{x_n} s_k(x_n, t).$$

Da beim Übergang $t \to t + 1$ aber alle Rangwerte (außer dem des nachgefragten Elementes σ_{t+1}) um 1 erhöht werden, können sich aufgrund der Vorfaktoren w_x auch die

Reihenfolgen von beliebigen Elementen ändern. Die Kosten dieser bezahlten Austausche sind nicht zu kontrollieren. Außerdem erfüllen diese Regeln nicht die Bedingung der paarweise Unabhängigkeit, da es nicht nur auf die relative Ordnung, sondern auf die absoluten Werte der Ränge $w_x s_k(x, t)$ ankommt.

Wir untersuchen den randomisierten Algorithmus BIT (siehe Seite 76) in dem Modell des gewichteten Listenproblems. Im Standardmodell ist BIT 1.75-kompetitiv [RWS94]. Wir zeigen

Satz 1.56. *Gegenüber einem blinden Offline-Gegner* (oblivious adversary) OPT *gilt für alle Gewichte* $\mathbf{w} = (w_1, \ldots, w_n)$ *und alle Zugriffsfolgen* σ *der Länge* $|\sigma| = m$, *daß*

$$E[\text{BIT}(\sigma)] \leq \max\left\{1 + \frac{3}{4}\frac{w_{max}}{w_{min}}, \frac{3}{2}\frac{w_{max}}{w_{min}}\right\} \cdot \text{OPT}(\sigma).$$

Beweis. Eine wichtige Beobachtung aus [RWS94] ist, daß zu jedem Zeitpunkt t für jedes Element x das entsprechende Bit $b(x)$ gleichverteilt und unabhängig von den anderen Bits oder der Position von x ist.

Wir modifizieren nun die Potentialfunktion aus [RWS94], so daß die Gewichte berücksichtigt werden. Ein Paar $\langle y, x\rangle$ heißt Inversion, wenn y vor x in der von BIT verwalteten Liste steht und x vor y in OPT's Liste. Diese Inversion ist vom Typ $1 + b(x)$ und hat das Gewicht w_y. Das Potential der Datenstruktur zum Zeitpunkt t definieren wir als

$$\Phi_t = \sum_{\substack{\langle y,x\rangle \\ \text{Inversion}}} (1 + b(x)) \cdot w_y.$$

Man beachte, daß Φ eine Zufallsvariable ist. Zu Beginn starten BIT und OPT mit der gleichen Liste, also $\Phi_0 = 0$. Sei der aktuelle Zugriff $\sigma_t = x$. Sei $\mathcal{A}$ die Menge der Elemente, die bei BIT und bei OPT vor x stehen, und $\mathcal{B}$ die Menge der y, so daß $\langle y, x\rangle$ eine Inversion ist. Für eine Menge $\mathcal{X}$ von Listenelementen sei $w(\mathcal{X}) = \sum_{x\in\mathcal{X}} w_x$ ihr Gewicht. Für die tatsächlichen Kosten in diesem Schritt gilt dann

$$\text{OPT}_t \geq w_x + w(\mathcal{A}), \qquad \text{BIT}_t = w_x + w(\mathcal{A}) + w(\mathcal{B}).$$

Für die amortisierten Kosten müssen wir die Potentialänderung berechnen. Dazu betrachten wir erst, welche Inversionen durch die Aktion von BIT verschwinden. Falls $b(x) = 0$, dann wird x nach vorne geschoben, und alle $\langle y, x\rangle$ mit $y \in \mathcal{B}$ verschwinden. Im Falle $b(x) = 1$ wird x nicht bewegt, aber alle Inversionen $\langle y, x\rangle$ mit $y \in \mathcal{B}$ ändern ihren Typ von 2 auf 1. Insgesamt folgt, daß das Potential um $w(\mathcal{B})$ abnimmt. Durch die Aktion von BIT und OPT können auch neue Inversionen entstehen, nämlich dann, wenn entweder BIT oder OPT das Element x an einem $z \in \mathcal{A}$ vorbei nach vorne ziehen, aber nicht beide. Sei $\mathcal{A}' \subseteq \mathcal{A}$ die Menge der Elemente, die nach der Abarbeitung des Zugriffs auf x in der Liste von OPT noch vor x stehen. Sei $b(x) = 0$, d.h. x wird von BIT nach vorne bewegt. Dann entsteht für jedes $z \in \mathcal{A}'$ eine neue Inversion $\langle x, z\rangle$ vom Typ $1 + b(z)$, und das erwartete Gesamtgewicht dieser Inversionen ist $\leq \frac{3}{2} \cdot |\mathcal{A}'| \cdot w(x)$. Im

Falle $b(x) = 1$ bleibt x auf seiner Position und durch die Aktion von OPT wird für jedes $z \in \mathcal{A} - \mathcal{A}'$ eine Inversion $\langle z, x\rangle$ entstehen. Diese Inversionen haben alle den Typ 1, da das Bit von x gekippt wird. Das erwartete Gesamtgewicht dieser Inversionen ist also $\le w(\mathcal{A}-\mathcal{A}')$. Da das Bit $b(x)$ gleichverteilt ist, folgt für den mittleren Potentialzuwachs aufgrund neuer Inversionen

$$\begin{aligned}
&\le \frac{1}{2}\cdot\frac{3}{2}\cdot|\mathcal{A}'|\cdot w_x + \frac{1}{2}\cdot w(\mathcal{A}-\mathcal{A}') \\
&\le \frac{3}{4}\cdot\frac{w_{max}}{w_{min}}\cdot w(\mathcal{A}') + \frac{1}{2}\cdot w(\mathcal{A}-\mathcal{A}') \\
&\le \frac{3}{4}\cdot\frac{w_{max}}{w_{min}}\cdot w(\mathcal{A}).
\end{aligned}$$

Dabei haben wir benutzt, daß für beliebige Elemente x, y die Beziehung $w_x \le w_y w_{max}/w_{min}$ gilt. Insgesamt folgt für die mittleren amortisierten Kosten des Zugriffs auf x

$$\begin{aligned}
E[a_t] &= w(\mathcal{A}) + w(\mathcal{B}) + w_x + \Delta\Phi \\
&\le w(\mathcal{A}) + w(\mathcal{B}) + w_x - w(\mathcal{B}) + \frac{3}{4}\frac{w_{max}}{w_{min}}\cdot w(\mathcal{A}) \\
&= \left(1+\frac{3}{4}\frac{w_{max}}{w_{min}}\right)\cdot\big(w(\mathcal{A})+w_x\big) - \frac{3}{4}\frac{w_{max}}{w_{min}}\cdot w_x \\
&\le \left(1+\frac{3}{4}\frac{w_{max}}{w_{min}}\right)\cdot \mathrm{OPT}_t - \frac{3}{4}\cdot w_{max}.
\end{aligned}$$

Wir müssen auch noch den Fall eines bezahlten Austauschs zwischen zwei Elementen x und y betrachten. Dieser Austausch kostet w_x für OPT. Falls dabei eine Inversion $\langle y, x\rangle$ entsteht, so wird das mittlere Potential um

$$\frac{3}{2}w_y \le \frac{3}{2}\frac{w_{max}}{w_{min}}\cdot w_x$$

erhöht. Insgesamt können wir das Verhältnis zu den optimalen Kosten abschätzen durch

$$\frac{E[\mathrm{BIT}(\sigma)]}{\mathrm{OPT}(\sigma)} \le \max\left\{1+\frac{3}{4}\frac{w_{max}}{w_{min}}, \frac{3}{2}\frac{w_{max}}{w_{min}}\right\}.$$

□

Als nächstes betrachten wir stochastische Quellen ohne Gedächtnis. Die Simple(k)- bzw. Batched(k)-MTF-Verfahren können übertragen werden, indem wir das obige Schema von RANDOM-MTF oder COUNTER-MTF anwenden. Wenn nur bei k-fachem Zugriff das nachgefragte Element j bewegt wird, darf dies nur ungefähr alle w_j^k Fälle geschehen. Wir konzentrieren uns auf Batched(k)-MTF in Kombination mit der randomisierten Variante. Sei $\alpha_j = A/w_j$ die Wahrscheinlichkeit, daß RANDOM-MTF eine Aktion ausführt, wenn auf j zugegriffen wird. Dabei muß $0 < A \le \min\{w_j\}$ sein, damit $0 < \alpha_j \le 1$. Indem man A maximal wählt, wird die Konvergenz des Verfahrens optimiert.

Satz 1.57. *Sei σ eine Startkonfiguration. Die erwartete Suchzeit der Batched(k)-RMTF-Regel nach kt Zugriffen beträgt*

$$E_k(kt) = E_k(\infty) + OW_{k,\sigma}(kt).$$

Dabei betragen die asymptotischen Kosten

$$E(\infty) = 1 + \sum_{i<j} \frac{p_i(\alpha_j p_j)^k + p_j(\alpha_i p_i)^k}{(\alpha_i p_i)^k + (\alpha_j p_j)^k}$$

und der anfängliche Mehraufwand pro Zugriff

$$\begin{aligned} OW_{k,\sigma}(kt) &= \sum_{i<j} \left(p_i \chi_{\sigma(j)<\sigma(i)} + p_j \chi_{\sigma(i)<\sigma(j)} - \frac{p_i(\alpha_j p_j)^k + p_j(\alpha_i p_i)^k}{(\alpha_i p_i)^k + (\alpha_j p_j)^k} \right) \\ &\qquad \cdot (1 - (\alpha_i p_i)^k - (\alpha_j p_j)^k)^t. \end{aligned}$$

Beweis. Für die Wahrscheinlichkeit $b_{kt}(j,i)$, daß j nach kt Zugriffen vor i steht, beachten wir, daß ein Element ℓ nur mit Wahrscheinlichkeit $(\alpha_\ell p_\ell)^k$ bewegt wird. Die Wahrscheinlichkeit, daß nach kt Schritten weder i noch j an den Listenkopf gesetzt wurden, ist $(1 - (\alpha_i p_i)^k - (\alpha_j p_j)^k)^t$. Wenn innerhalb der letzten kt Zugriffe die letzte Bewegung von j nach der letzten Bewegung von i aufgetreten ist, so hat dies die Wahrscheinlichkeit $(\alpha_j p_j)^k / ((\alpha_i p_i)^k + (\alpha_j p_j)^k) \cdot (1 - (1 - (\alpha_i p_i)^k - (\alpha_j p_j)^k)^t)$. Zusammen erhält man (vgl. Satz 1.33)

$$\begin{aligned} b_{kt}(j,i) &= \frac{(\alpha_j p_j)^k}{(\alpha_i p_i)^k + (\alpha_j p_j)^k} \\ &\quad + \left(\chi_{\sigma(j)<\sigma(i)} - \frac{(\alpha_j p_j)^k}{(\alpha_i p_i)^k + (\alpha_j p_j)^k} \right) \cdot (1 - (\alpha_i p_i)^k - (\alpha_j p_j)^k)^t. \end{aligned}$$

□

Satz 1.58. *Wenn alle Startverteilungen σ gleich wahrscheinlich sind, gilt für beliebige $\varepsilon > 0$, daß*

$$t \geq t^* := \left\lceil \frac{1}{ek} \cdot \left(\frac{n^2}{4\varepsilon\alpha_{min}} \right)^k \right\rceil \implies E_k(kt) \leq (1+\varepsilon) \cdot E_k(\infty).$$

Beweis. Eine hinreichende Bedingung für die Aussage ist $OW_k(kt) \leq \varepsilon$, vgl. Satz 1.36.

$$\begin{aligned} OW_k(kt) &= \sum_{i<j} \frac{(p_i - p_j)((\alpha_i p_i)^k - (\alpha_j p_j)^k)}{2((\alpha_i p_i)^k + (\alpha_j p_j)^k)} \cdot (1 - (\alpha_i p_i)^k - (\alpha_j p_j)^k)^t \\ &\leq \binom{n}{2} \cdot \max_{1\leq i\leq n} \left(p_i \cdot (1 - (\alpha_i p_i)^k)^t \right)/2 \\ &\leq \binom{n}{2} \cdot \max_{0\leq p\leq 1} \left(p \cdot e^{-t(\alpha_{min} p)^k} \right)/2 \\ &= \binom{n}{2} \cdot \frac{1}{2\alpha_{min}} \cdot \left(\frac{1}{tke} \right)^{1/k}. \end{aligned}$$

Damit folgt die hinreichende Bedingung

$$t \geq t^* := \left\lceil \frac{1}{ek} \cdot \left(\frac{n^2}{4\varepsilon\alpha_{min}} \right)^k \right\rceil .$$

□

Insbesondere folgt für die Nach-Vorne-Schieben-Regel, daß $O(\frac{n^2}{\varepsilon\alpha_{min}})$ Zugriffe hinreichend sind. Indem man $A = w_{min}$ maximal wählt, wird die Konvergenzzeit minimiert: $O(\frac{n^2 w_{max}}{\varepsilon w_{min}})$ Schritte sind hinreichend.

1.9. Bidirektionale Suche in Listen

In diesem Abschnitt betrachten wir Suchverfahren für doppelt verkettete lineare Listen, bei denen die Suche nach einem Listenelement von jeder Seite der Liste beginnen kann. Die Suchkosten sollen wieder gleich der Anzahl der Vergleiche sein, die nötig sind, auf das entsprechende Element zuzugreifen.

Solche Mechanismen werden beispielsweise in Betriebssystemen eingesetzt, um Speicherblöcke oder Zugriffe auf Plattenblöcke zu verwalten, siehe [K98]. Ein anderes Modell, das im folgenden unsere Intuition leitet, ist die Realisierung einer Datenbank, die von zwei Personen L und R benutzt wird. Bei Zugriffen von L wird die Liste von der linken Seite durchsucht, und bei Zugriffen von R von der rechten Seite. In dieser Situation versuchen adaptive Listenalgorithmen, die Zugriffskosten für L und R gleichzeitig zu minimieren.

Wir kennen nur die Arbeiten [MRB80, NO89, VO93], in denen Heuristiken für doppelt verkettete Listen analysiert werden. Die naheliegende Methode, um gute Verfahren zu erhalten, wurde *directed mapping* genannt [NO89], sie kann auf jeden Algorithmus für einfach verkettete Listen angewendet werden. Wenn ein Element vom linken Ende der Liste aus gesucht wird, wird das Verfahren wie bisher angewendet und das nachgefragte Element ein Stück nach links bewegt. Wenn die Suche vom rechten Ende der Liste aus beginnt, dann wird die „Reflektion“ des Verfahrens angewendet und das Element ein Stück nach rechts bewegt. Auf diese Weise ist sichergestellt, daß der Aufwand für die Umordnung proportional zu den Suchkosten bleibt.

Zum Beispiel ergibt sich aus der gewöhnlichen Nach-Vorne-Schieben-Regel MTF die Nach-Vorne-Schieben-Regel für doppelt verkettete Listen DMTF, bei der das nachgefragte Element an die Seite der Liste bewegt wird, von dem die Suche begann.

1.9.1. Mittleres Verhalten der DMTF-Regel

Als erstes betrachten wir unabhängige identisch verteilte Zugriffe; dies ist die einzige Situation, die bisher in der Literatur betrachtet wurde.

Unabhängige Zugriffe

Sei $\mathbf{p} = (p_{1L}, \ldots, p_{nL}, p_{1R}, \ldots, p_{nR})$ eine Zugriffsverteilung für eine doppelt verkettete Liste, dabei ist $p_{iL} \geq 0$, $p_{iR} \geq 0$ und $\sum_{i=1}^{n}(p_{iL} + p_{iR}) = 1$. Dabei soll p_{iL} die Wahrscheinlichkeit bezeichnen, mit der Element i von links her nachgefragt wird, analog p_{iR} die Wahrscheinlichkeit für ein Zugriff von rechts. Wenn Element i von links her nachgefragt wird, sagen wir auch, daß auf das Element iL zugegriffen wird, usw.

Sei ohne Beschränkung der Allgemeinheit

$$p_{1L} - p_{1R} \geq p_{2L} - p_{2R} \geq \cdots \geq p_{nL} - p_{nR}.$$

Dann ist die optimale Listenordnung $\langle 1, 2, \ldots, n \rangle$ und hat die erwarteten Kosten

$$\begin{aligned} M(\mathbf{p}) &= \sum_{i=1}^{n} p_{iL} \cdot i + p_{iR} \cdot (n + 1 - i) \\ &= \sum_{i=1}^{n} (p_{iL} - p_{iR}) \cdot i + (n + 1) \cdot \sum_{i=1}^{n} p_{iR}. \end{aligned}$$

Bezeichne $b(j, i)$ die asymptotische Wahrscheinlichkeit, daß Element j links von Element i steht. Es gilt

$$b(j, i) = \frac{p_{jL} + p_{iR}}{p_{jL} + p_{iR} + p_{iL} + p_{jR}},$$

und die erwarteten Kosten von DMTF ergeben sich zu

$$\begin{aligned} E_{\mathrm{DMTF}}(\mathbf{p}) &= 1 + \sum_{i} \sum_{j \neq i} p_{iL} b(j, i) + p_{iR} b(i, j) \\ &= 1 + \sum_{i} \sum_{j \neq i} \frac{p_{iL}(p_{jL} + p_{iR}) + p_{iR}(p_{iL} + p_{jR})}{p_{jL} + p_{iR} + p_{iL} + p_{jR}}, \end{aligned} \tag{1.22}$$

und es folgt $E_{\mathrm{DMTF}}(\mathbf{p}) \leq 2 \cdot M(\mathbf{p})$ für alle Zugriffsverteilungen $\mathbf{p}$, siehe [MRB80].

Wir betrachten nun dieses Zugriffsmodell unter einem etwas anderen Blickwinkel. Seien $p(L) = \sum_{i=1}^{n} p_{iL}$ und $p(R) = 1 - p(L)$ die Wahrscheinlichkeiten für einen Zugriff von L oder von R. Weiter seien zwei Verteilungen auf $\{1, \ldots, n\}$ definiert durch $l_i = p_{iL}/p(L)$ und $r_i = p_{iR}/p(R)$. Dann kann die ursprüngliche Zugriffsverteilung ausgedrückt werden durch

$$p_{iL} = p(L) \cdot l_i, \qquad p_{iR} = p(R) \cdot r_i,$$

wobei $p(L)$ die Balance zwischen Zugriffen von L und R beschreibt. Wenn man die Verteilungen $(l_i)_{i=1}^{n}$ und $(r_i)_{i=1}^{n}$ festhält und einen Balance-Parameter $0 \leq \alpha \leq 1$ variiert, also die Verteilung

$$p_{iL} = \alpha \cdot l_i, \qquad p_{iR} = (1 - \alpha) \cdot r_i$$

betrachtet, sei $E_{DMTF}(\alpha)$ die erwartete Suchzeit unter der DMTF-Regel. Wir bezeichnen zwei Verteilungen $(l_i)_{i=1}^n$ und $(r_i)_{i=1}^n$ als *anti-monoton*, wenn für alle $i < j$ gilt, daß $l_i > l_j$ und $r_i > r_j$. Das steht im Gegensatz zu *doppelt monotonen* (*dually monotonic*) Paaren von Verteilungen, die durch $l_i > l_j$ und $r_i < r_j$ definiert sind [VO93]. Für doppelt monotone Verteilungen ist die optimale Listenanordnung $\langle 1, 2, \ldots, n\rangle$. Für anti-monotone Verteilungen ist die Listenordnung, die die erwartete Zugriffszeit minimiert, nicht unmittelbar klar. Dieses Zugriffsmodell beschreibt die (manchmal realistische) Situation, daß die Präferenzen von L und R die gleiche Reihenfolge besitzen.

Der folgende Satz besagt, daß bei anti-monotonen Zugriffsverteilungen und beliebiger Balance die Kosten bei der simultanen Optimierung der Listenordnung für L und R niemals geringer sind als die gewichteten Kosten von zwei separaten MTF-Listen für die Zugriffe von links bzw. rechts. Anschaulich gesprochen stören sich die Benutzer L und R bei der Listenoptimierung gegenseitig, so daß das Gesamtergebnis schlechter ausfällt als bei getrennten Verfahren.

Satz 1.59. *Seien ein Paar von anti-monotonen Verteilungen* $(l_i)_{i=1}^n$ *und* $(r_i)_{i=1}^n$ *und ein Balance-Parameter* $0 \leq \alpha \leq 1$ *gegeben, dann gilt*

$$E_{DMTF}(\alpha) \geq \alpha \cdot E_{DMTF}(1) + (1-\alpha) \cdot E_{DMTF}(0).$$

Beweis.

$$\begin{aligned} E_{DMTF}(\alpha) &= 1 + \sum_i \sum_{j \neq i} p_{iL} b(j,i) + p_{iR} b(i,j) \\ &= 1 + \sum_{i<j} p_{iL} b(j,i) + p_{iR} b(i,j) + p_{jL} b(i,j) + p_{jR} b(j,i) \\ &= 1 + \sum_{i<j} \underbrace{\frac{\alpha^2 l_i l_j + 2\alpha(1-\alpha) l_i r_i + (1-\alpha)^2 r_i r_j}{\alpha(l_i + l_j) + (1-\alpha)(r_i + r_j)}}_{g(\alpha)} . \end{aligned}$$

Das Resultat folgt aus der analogen Behauptung für beliebige Paare $i < j$, also $g(\alpha) \geq \alpha g(1) + (1-\alpha) g(0)$, die für alle $0 \leq \alpha \leq 1$ gilt, was man direkt nachrechnet. □

Ein Sonderfall davon entsteht, wenn beide Verteilungen gleich sind, $l_i = r_i$. In diesem Fall ist die erwartete Suchzeit symmetrisch im Balance-Paramter, d.h. $E_{DMTF(\alpha)} = E_{DMTF(1-\alpha)}$ für $0 \leq \alpha \leq 1$. Wir betrachten den Fall einer Verteilung nach Zipf für L und R, d.h. $l_i = r_i = c/i$ für $i = 1, \ldots, n$, wobei $c = 1/H_n$ und $H_n = \sum_{j=1}^n 1/j$ die $n$te harmonische Zahl ist.

Satz 1.60. *Für anti-monotone Zipf-Verteilungen und Balance-Parameter* $0 \leq \alpha \leq 1$ *gilt*

$$\begin{aligned} E_{DMTF} &= 2\alpha(1-\alpha)n + (2\alpha-1)^2 \Big((2n+1)H_{2n+1} - 2(n+1)H_{n+1} + 1\Big)/H_n + 1/2 \\ &= 2\alpha(1-\alpha)n + 2\ln(2)(2\alpha-1)^2 \cdot n/H_n - 1/2 + 4\alpha - 4\alpha^2 + o(1) \end{aligned}$$

Man beachte, daß für einfach verkettete Listen und MTF unter der Zipf-Verteilung die erwartete Suchzeit

$$\begin{aligned} E_{\mathrm{MTF}} &= \frac{1}{2} + c\sum_i \sum_j \frac{1}{i+j} \\ &= 2\ln(2)n/H_n - 1/2 + o(1) \end{aligned}$$

mit $o(1) \to 0$ für $n \to \infty$ gilt, siehe [K98, Seite 402], was aus der asymptotischen Entwicklung der Summe

$$\begin{aligned} \sum_{i=1}^{n}\sum_{j=1}^{n} 1/(i+j) &= (2n+1)H_{2n+1} - 2(n+1)H_{n+1} + 1 \\ &= 2\ln(2)\cdot n - \ln(n) - 1/2 + \ln(2) - \gamma - O(1/n) \end{aligned}$$

folgt und in obiger Formel für $\alpha = 0$ oder $\alpha = 1$ enthalten ist. Dabei ist $\gamma = 0.577\ldots$ die Eulersche Konstante. Die asymptotischen erwarteten Suchkosten berechnen wir wie folgt.

Beweis. Man beobachtet zunächst, daß $\frac{j}{i(i+j)} = \frac{1}{i} - \frac{1}{i+j}$ gilt. Damit folgt aus (1.22), daß

$$\begin{aligned} E_{\mathrm{DMTF}} &= \frac{1}{2} + c\sum_{i,j} \frac{\frac{\alpha}{i}\cdot(\alpha i + (1-\alpha)j) + \frac{1-\alpha}{i}\cdot(\alpha j + (1-\alpha)i)}{\alpha i + (1-\alpha)j + \alpha j + (1-\alpha)i} \\ &= \frac{1}{2} + c(\alpha^2 + (1-\alpha)^2)\sum_{i,j}\frac{1}{i+j} + 2c\alpha(1-\alpha)\sum_{i,j}\frac{j}{i(i+j)} \\ &= \frac{1}{2} + 2\alpha(1-\alpha)\cdot n + c(2\alpha-1)^2\sum_{i,j}\frac{1}{i+j} \end{aligned}$$

und durch Einsetzen der asymptotischen Entwicklung für $\sum_{i,j} 1/(i+j)$ und $c = 1/H_n$ erhält man das Ergebnis. □

Abhängige Zugriffe

Nun betrachten wir die Situation einer Zugriffsfolge, die durch eine positiv-rekurrente Markov-Kette **P** erster Ordnung modelliert werden kann. Der Zustandsraum der Kette ist $\{1L, \ldots, nL, 1R, \ldots, nR\}$, und die Übergänge werden durch Wahrscheinlichkeiten $p_{iD,jD'}$ beschrieben, wobei $i, j \in \{1, \ldots, n\}$ und $D, D' \in \{L, R\}$. Die eindeutigen stationären Wahrscheinlichkeiten seien π_{iD}. Weiter seien $m_{iD,jD'}$ die mittleren Ersteintrittszeiten und $\hat{p}_{iD,jD'} = p_{jD',iD}\cdot\pi_{jD'}/\pi_{iD}$ die Übergangswahrscheinlichkeiten der rückwärtslaufenden Kette. Die folgende Beobachtung ist grundlegend für die Analyse.

Lemma 1.61. *Wenn auf* iL *zugegriffen wird, dann steht* j *links von* i, *wenn wir von* iL *ausgehend die Zugriffsfolge zurückverfolgen und auf* jL *oder* iR *treffen, bevor* iL *oder* jR *aufgetreten ist. Wenn auf* iR *zugegriffen wird, dann steht* i *links von* j, *wenn wir von* iR *ausgehend die Zugriffsfolge zurückverfolgen und auf* iL *oder* jR *treffen, bevor* iR *oder* jL *aufgetreten ist.*

Sei $b(j,i|iL)$ die asymptotische Wahrscheinlichkeit, daß j links von i steht, wenn gerade auf iL zugegriffen wird, und $b(i,j|iR)$ die asymptotische Wahrscheinlichkeit, daß i links von j steht, wenn gerade auf iR zugegriffen wird. Diese Wahrscheinlichkeiten können mit Hilfe von Tabu-Mengen berechnet werden. Dazu bezeichne ${}_{iL,jR,iR}\hat{f}^*_{iL,jL}$ die Wahrscheinlichkeit, von iL aus rückwärts zu laufen und in einer beliebigen Anzahl von Schritten das Element jL zu erreichen, ohne die Tabu-Zustände iL, jR oder iR als Zwischenzustände zu benutzen. Hierbei wird auch iR ausgeschlossen, um den ersten Eintritt in jL oder iR als disjunkte Ereignisse betrachten zu können. Diese Wahrscheinlichkeiten können gemäß Formel (A.3) angegeben werden, und wir erhalten

Lemma 1.62.

$$\begin{aligned} b(j,i|iL) &= {}_{iL,jR,iR}\hat{f}^*_{iL,jL} + {}_{iL,jR,jL}\hat{f}^*_{iL,iR} \\ b(i,j|iR) &= {}_{iR,jL,iL}\hat{f}^*_{iR,jR} + {}_{iR,jL,jR}\hat{f}^*_{iR,iL}\,. \end{aligned}$$

Mit Hilfe von $b(i,j|iR) = 1 - b(j,i|iR)$ ergibt sich

Satz 1.63. *Die asymptotische erwartete Suchzeit bei Zugriffsfolgen gemäß der Markov-Kette* $\mathbf{P}$ *ist*

$$E_{DMTF}(\mathbf{P}) = 1 + \sum_{i=1}^{n} \sum_{j \neq i} \pi_{iL} b(j,i|iL) + \pi_{iR} b(i,j|iR).$$

Wir geben ein Beispiel. Seien $(l_1,\dots,l_n)$ und $(r_1,\dots,r_n)$ zwei Wahrscheinlichkeitsverteilungen auf $\{1,\dots,n\}$. Wir betrachten das oben vorgestellte Szenario zweier Benutzer L und R, deren Zugriffe unabhängig voneinander sind, aber deren Zugriffe in Blöcken ankommen. Dabei sei die Blockgröße K geometrisch verteilt mit Parameter $0 < \beta < 1$, also $\text{Prob}(K = k) = (1-\beta)\beta^{k-1}$ für $k \geq 1$. Die erwartete Blockgröße ist also $1/(1-\beta)$.

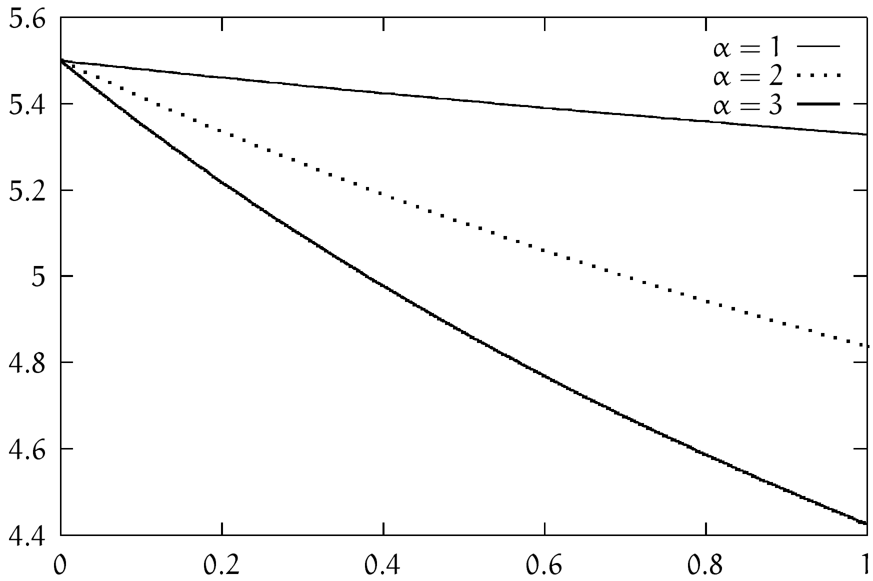
Mit diesen Annahmen finden wir für die Übergangsmatrix der Markov-Kette

$$\begin{aligned} p_{iL,jL} &= \beta l_j & \qquad p_{iL,jR} &= (1-\beta) r_j \\ p_{iR,jR} &= \beta r_j & \qquad p_{iR,jL} &= (1-\beta) l_j. \end{aligned}$$

Lemma 1.64. *In dem oben definierten Beispiel gilt* $\pi_{iL} = l_i/2$, $\pi_{iR} = r_i/2$, *und*

$$\begin{aligned} b(j,i|iL) &= \frac{(r_i + r_j) l_j \beta + l_j + r_i}{(l_i r_i + l_i r_j + l_j r_i + l_j r_j)\beta + l_i + l_j + r_i + r_j} \\ b(i,j|iR) &= \frac{(l_i + l_j) r_j \beta + l_i + r_j}{(l_i r_i + l_i r_j + l_j r_i + l_j r_j)\beta + l_i + l_j + r_i + r_j}\,. \end{aligned}$$

Das Lemma folgt durch Auswerten von Lemma 1.62 und eine längere, aber einfache Rechnung (Berechnen mehrerer 3×3-Matrizen). Wir illustrieren es für die Verteilungen $l_i = r_i = 1/(i^\alpha H_n^{(\alpha)})$ und $\alpha = 1, 2, 3$ am Beispiel von $n = 10$ Elementen.



Verlauf von $E_{\text{DMTF}}(\mathbf{P})$ für $\beta \in [0, 1]$.

Im Fall $\beta = 0$ werden von L und R nur einzelne Zugriffe gemacht, deren Reihenfolge gleichverteilt ist. Somit ist keinerlei Optimierung möglich, und die Suchzeit beträgt $(n + 1)/2$. Mit zunehmender Blockgröße (wachsendem β) hat der Algorithmus mehr Zeit, die Listenordnung jeweils für L oder R zu optimieren; dies funktioniert umso besser, je stärker die Zugriffsverteilung von der Gleichverteilung entfernt ist (größere α). Im Grenzfall $\beta = 1$ erfolgen die Zugriffe nur von einer Seite, und man erhält das gleiche Ergebnis wie bei einer einfach verketteten Liste.

1.9.2. Kompetitives Verhalten der DMTF-Regel

In diesem Abschnitt zeigen wir, daß die Kosten der DMTF-Regel höchstens um den Faktor 2 höher sind als die Kosten beliebiger Offline-Gegner.

Satz 1.65. *Für beliebige Zugriffsfolgen σ gilt:*

$$\text{DMTF}(\sigma) \leq 2 \cdot \text{OPT}(\sigma).$$

Beweis. Für zwei Listenelemente $x \neq y$ sagen wir $\langle x, y \rangle$, wenn in der von DMTF verwalteten Liste x links von y steht. $\langle x, y \rangle$ heißt Inversion, wenn in der Liste des optimalen Gegners y links von x steht.

Sei eine Zugriffsfolge $\sigma = (\sigma_1, \dots, \sigma_m)$ gegeben, und sei der aktuelle Zugriff $\sigma_t \in \{xL, xR\}$. Für die amortisierte Analyse benutzen wir die Anzahl aller Inversionen als Standard-Potentialfunktion. Sei weiter $\mathcal{A}$ die Menge der Elemente y, so daß $\langle y, x \rangle$ keine Inversion ist, $\mathcal{B}$ die Menge der y, so daß $\langle y, x \rangle$ eine Inversion ist. Sei $\mathcal{C}$ die Menge der Elemente y, so daß $\langle x, y \rangle$ eine Inversion ist, und $\mathcal{D}$ die Menge der y, so daß $\langle x, y \rangle$ keine Inversion ist. Schließlich sei $|\mathcal{A}| = A$, $|\mathcal{B}| = B$, $|\mathcal{C}| = C$ und $|\mathcal{D}| = D$.

Sei nun der aktuelle Zugriff $\sigma_t = xL$. Die Kosten des optimalen Algorithmus sind $\text{OPT}_t = A + C + 1$, und der Online-Algorithmus weist Kosten $\text{DMTF}_t = A + B + 1$ auf.

Aufgrund der Aktion von DMTF verringert sich das Potential um B, da die Inversionen $\langle y, x\rangle$ mit $y \in B$ verschwinden. Je nachdem, wie weit OPT sein x nach links bewegt, verwandeln sich einige oder alle der $\langle z, x\rangle$ mit $z \in \mathcal{A}$ zu Inversionen $\langle x, z\rangle$ mit $z \in \mathcal{C}$. Die Potentialänderung beträgt also höchstens $A - B$, und für die amortisierten Kosten a_t von DMTF folgt

$$a_t \;=\; \text{DMTF}_t + \Delta\Phi \;=\; A + B + 1 + A - B \;\leq\; 2 \cdot \text{OPT}_t - 1.$$

Analog sei der aktuelle Zugriff $\sigma_t = xR$. Die Kosten des optimalen Algoritmus sind $\text{OPT}_t = B + D + 1$, und der Online-Algorithmus weist Kosten $\text{DMTF}_t = C + D + 1$ auf. Aufgrund der Aktion von DMTF verringert sich das Potential um C, da die Inversionen $\langle x, y\rangle$ mit $y \in C$ verschwinden. Durch die Aktion von OPT können bis zu D neue Inversionen entstehen. Die Potentialänderung beträgt also höchstens $D - C$, und für die amortisierten Kosten a_t von DMTF gilt

$$a_t \;=\; \text{DMTF}_t + \Delta\Phi \;=\; C + D + 1 + D - C \;\leq\; 2 \cdot \text{OPT}_t - 1.$$

Insgesamt folgt, daß DMTF ein 2-kompetitiver Listenalgorithmus ist. □

Da das Modell der bidirektionalen Suche die Suche in einer Richtung als Spezialfall enthält, ist 2 auch eine untere Schranke für den kompetitiven Faktor von DMTF, und damit ist DMTF optimal im kompetitiven Modell.

1.10. Suche in mehreren Listen

Wir betrachten folgendes Modell, das in [CW90] eingeführt wurde. Die n Elemente der Menge $\mathcal{A}$ sollen auf k Listen $L_1, \dots, L_k$ verteilt werden. Beim Zugriff auf ein Element i wird in der Liste L_h mit $i \in L_h$ gesucht, die Zugriffskosten sind gleich der Position, auf der i gefunden wurde. Die Auswahl der richtigen Liste L_h verursacht keine Kosten.

Dieses Modell beschreibt beispielsweise Hashing mit Verkettung, wenn die Auswertung der Hashfunktion keine Kosten erzeugt, sondern der Suchaufwand in der linearen Liste bewertet wird. Ein praktisches Beispiel für diese Situation ist ein Warenlager, in dem Kisten übereinander stehen. Bei dem Finden des richtigen Kistenstapels entstehen keine Kosten, aber der Zugriff auf die k-oberste Kiste verursacht Kosten k. Die Aufgabe ist es nun, die Kisten so zu Stapeln zusammenzustellen bzw. dynamisch umzuschichten, daß die Zugriffskosten minimiert werden.

In [CW90] wird ausschließlich die Klasse $\mathcal{K}$ der Regeln betrachtet, die das gefundene Element an den Kopf einer beliebigen Liste setzen (MOVE-TO-FRONT *for multiple lists*). Eine solche Regel ist also eine Abbildung des aktuellen Zustands der Listen und des aktuellen Zugriffs auf eine Liste. Man beachte, daß die Längen der Listen bei diesem Vorgang variieren können. Die Autoren führen als Teilmenge von $\mathcal{K}$ die Klasse der *partition policies* ein, die nur am Anfang eine Zuordnung der Elemente auf die einzelnen Listen vornehmen, und anschließend diese Aufteilung nicht mehr ändern. Unter dem Modell von stochastisch unabhängigen, identisch verteilten Zugriffen zeigen sie eine hinreichende Bedingung dafür, daß eine optimale Strategie eine *partition policy* ist und geben auch ein Beispiel, bei der eine andere Strategie geringere Kosten hat als die beste *partition policy*.

Unabhängige stochastische Zugriffe

Im folgenden lassen wir weitere Regeln zu. Zunächst charakterisieren wir optimale Anordnungen. Seien $p_1 \geq \cdots \geq p_n$ die Zugriffswahrscheinlichkeiten der Elemente. Weiter sei $n = m \cdot k$ für ein m. Eine optimale Aufteilung auf die Listen ist dann

$$\begin{array}{rcllllllllllll} L_1 & = & \langle & 1 & , & k+1 & , & 2k+1 & , & \ldots & , & n-k+1 & \rangle \\ L_2 & = & \langle & 2 & , & k+2 & , & 2k+2 & , & \ldots & , & n-k+2 & \rangle \\ & & & & & & & \vdots & & & & & \\ L_k & = & \langle & k & , & k+k & , & 2k+k & , & \ldots & , & n & \rangle. \end{array}$$

Denn es ist optimal, die k Elemente mit den größten Wahrscheinlichkeiten auf die jeweils ersten Listenplätze zu setzen und entsprechend fortzufahren. Keine Vertauschung von zwei Elementen kann dann die erwarteten Suchkosten verkleinern.

$$\mathrm{E}[\text{MOPT}](\mathbf{p}) = \sum_{h=1}^{k} \sum_{i=1}^{n/k} i \cdot p_{k \cdot (i-1)+h},$$

wobei MOPT für MEHRFACHLISTEN-OPT steht. Die bekannten Heuristiken für adaptive Listen können wir direkt auf diesen Fall übertragen, wenn wir die Kosten für die Umordnung zunächst nicht betrachten. Dazu lesen wir die obigen Aufteilung spaltenweise, und versuchen, die Ordnung $\langle 1, 2, \ldots, n \rangle$ möglichst gut zu approximieren. Genauergesagt betrachten wir die bijektive Zuordnung

$$f : (L_1, \ldots, L_k) \longleftrightarrow L$$

mit

$$\begin{array}{rcl} L_h & = & \langle \sigma_h(1), \ldots, \sigma_h(n/k) \rangle \qquad \text{für} \quad h = 1, \ldots, k \\ L & = & \langle \sigma_1(1), \sigma_2(1), \ldots, \sigma_k(1), \sigma_1(2), \sigma_2(2), \ldots, \sigma_k(2), \ldots, \sigma_1(n/k), \ldots, \sigma_k(n/k) \rangle. \end{array}$$

Durch diese Aufteilung hat jedes L_h stets n/k Elemente. Eine beliebige Regel R für die Liste L wird nun auf $(L_1, \ldots, L_k)$ angewendet durch

$$(L_1, \ldots, L_k) \xrightarrow{f} L \xrightarrow{R} L' \xrightarrow{f^{-1}} (L'_1, \ldots, L'_k).$$

Diese Regel soll MEHRFACHLISTEN-R (MR) heißen. Bereits für MMTF ist keine einfache Formulierung der erwarteten Suchkosten möglich, da bei einem Zugriff auf i in einer Liste L_h ein Element j genau dann vor i steht, wenn zwischen j und i sich $mk-1$ Elemente befinden für ein $m \in \mathbb{N}$. Wir zeigen aber

Satz 1.66. *Gilt für eine Familie $R(\ell)$ von Listenalgorithmen, daß*

$$E_{R(\ell)}(\mathbf{p}) \to E_{OPT}(\mathbf{p}) \qquad (\ell \to \infty)$$

für eine Zugriffsverteilung $\mathbf{p}$, so folgt auch

$$E_{MR(\ell)}(\mathbf{p}) \to E_{MOPT}(\mathbf{p}) \qquad (\ell \to \infty).$$

Beweis. Für $p_j < p_i$ muß für die asymptotische Wahrscheinlichkeit $b_\ell(j,i)$, daß j vor i steht, wenn $R(\ell)$ angewendet wird, gelten: $b_\ell(j,i) \to 0$ für $\ell \to \infty$. Damit folgt, daß die Wahrscheinlichkeit dafür, daß wie bei der optimalen Anordnung höchstens $t = |\{j : p_j \geq p_i\}|$ Elemente vor i stehen, gegen 1 strebt. Damit verursacht ein Zugriff auf i im Grenzwert $\ell \to \infty$ die erwarteten Kosten von höchstens $\lceil t/k \rceil$, wenn $\mathrm{MR}(\ell)$ angewendet wird. □

Bei diesen Verfahren ist jedoch zu bedenken, daß weitere Elemente neben dem nachgefragten Element bewegt werden müssen. Die Kosten dieser Umordnungen haben wir nicht berücksichtigt. Wir geben nun Listenalgorithmen an, die der obigen Klasse $\mathcal{K}$ angehören, also keine weiteren Umordnungskosten erzeugen, und adaptiv arbeiten, d.h. ohne Kenntnis der Wahrscheinlichkeiten eine Umsortierungsstrategie implementieren.

> MRAND: Setze das nachgefragte Element an den Kopf einer zufällig ausgewählten Liste.

Satz 1.67. *Bei unabhängigen Zugriffen betragen die erwarteten Suchkosten von* MRAND *unter obigem Kostenmodell (gewertet wird nur die lineare Suche, nicht die Auswahl der Liste)*

$$\begin{aligned} E_{MRAND}(\mathbf{p}) &= 1 + \frac{2}{k} \cdot \sum_{i<j} \frac{p_i p_j}{p_i + p_j} \\ &\leq 2 \cdot E_{MOPT}(\mathbf{p}) + 1 - \frac{2}{k} \,. \end{aligned}$$

Beweis. Das Element j steht genau dann vor i, wenn es beim letzten Zugriff an den Kopf der Liste gesetzt wurde, die i enthält, und danach i oder j nicht mehr angefragt wurden,

$$b(j,i) = \frac{p_j}{k} \cdot \sum_{t \geq 0} (1 - p_i - p_j)^t = \frac{1}{k} \cdot \frac{p_j}{p_i + p_j}.$$

Daraus folgt

$$\begin{aligned} E_{MRAND}(\mathbf{p}) &= 1 + \sum_i \sum_{j \neq i} p_i b(j,i) \; = \; 1 + \frac{2}{k} \cdot \sum_{i<j} \frac{p_i p_j}{p_i + p_j} \\ &\leq 1 + 2 \cdot \sum_{i<j} \frac{p_j}{k} \; = \; 1 + 2 \cdot \sum_j p_j \cdot \frac{j-1}{k} \\ &= 1 - \frac{2}{k} + 2 \cdot \sum_j p_j \cdot \frac{j}{k} \; \leq \; 1 - \frac{2}{k} + 2 \cdot \sum_j p_j \cdot \left\lceil \frac{j}{k} \right\rceil \\ &= 1 - \frac{2}{k} + 2 \cdot E_{MOPT}(\mathbf{p}). \end{aligned}$$

□

Der nächste Algorithmus, den wir betrachten, arbeitet deterministisch und kann als eine derandomisierte Version von MRAND aufgefaßt werden. Anstatt einer zufälligen Auswahl der Liste, in die die nachgefragten Elemente gesetzt werden, werden sie zyklisch auf die Listen verteilt.

> ARRAY: Das Verfahren startet in einer beliebigen Konfiguration der Listen, gleichzeitig bilden die L_h selbst die Einträge einer Liste $\langle L_{\sigma(1)}, \ldots, L_{\sigma(k)} \rangle$, die mit MTF verwaltet wird. Nachdem auf ein Element i zugegriffen wurde, wird i an den Kopf der zur Zeit letzten Liste $L_{\sigma(k)}$ gesetzt und diese gemäß MTF an den Anfang der Liste der L_h bewegt.

Die Intuition dabei ist, daß die letzte Liste $L_{\sigma(k)}$ die wenigsten Zugriffe erhalten hat, und deshalb durch Hinzufügen des aktuellen Elements populärer gemacht werden soll. Dadurch sollen die Nachfragen möglichst gut über alle Listen verteilt werden. Die Listenlängen können dabei schwanken, und auch leere Listen sind möglich. Wir zeigen, daß bei unabhängigen identisch verteilten Zugriffen die deterministische ARRAY-Regel genauso gut arbeitet wie MRAND.

Satz 1.68. *Die erwarteten Suchkosten von* ARRAY *unter obigem Kostenmodell betragen*

$$\begin{aligned} E_{\text{ARRAY}}(\mathbf{p}) &= 1 + \sum_{j \neq i} \frac{p_i p_j}{p_i + p_j} \cdot \frac{p_i (1-p_i)^{k-1}}{1-(1-p_i)^k} \\ &\leq 2 \cdot E_{\text{MOPT}}(\mathbf{p}) + 1 - \frac{2}{k} \,. \end{aligned}$$

Beweis. In der Folge der Listen hat die letzte Liste $L_{\sigma(k)}$ die Eigenschaft, daß seit mindestens $k-1$ Zugriffen kein Element mehr daraus nachgefragt wurde. Nur in diese Liste werden neue Elemente gesetzt, und wenn sie dann an den Anfang aller Listen bewegt wird, bleibt die Ordnung der Elemente erhalten. Das bedeutet, daß Element j vor i steht, wenn seit dem letzten Zugriff auf i genau $k-1+\ell k$ andere Elemente nachgefragt wurden (eventuell auch j), und dann zum letzten Mal auf j zugegriffen wird ($\ell \geq 0$). Wenn man danach konditioniert, daß der letzte Zugriff auf j genau t Schritte zurückliegt, erhält man

$$\begin{aligned} b(j,i) &= p_i \cdot (1-p_i)^{k-1} \cdot \sum_{\ell=0}^{\infty} (1-p_i)^{k\ell} \cdot p_j \cdot \sum_{t=0}^{\infty} (1-p_i-p_j)^t \\ &= \frac{p_i p_j}{p_i + p_j} \cdot \frac{(1-p_i)^{k-1}}{1-(1-p_i)^k} \,. \end{aligned}$$

Für die asymptotischen erwarteten Zugriffskosten folgt

$$\begin{aligned}
E_{\text{ARRAY}}(\mathbf{p}) &= \sum_i p_i\Big(1+\sum_{j\neq i} b(j,i)\Big)\\
&= 1+\sum_{j\neq i}\frac{p_ip_j}{p_i+p_j}\cdot\frac{p_i(1-p_i)^{k-1}}{1-(1-p_i)^k}\\
&\leq 1+\sum_{j\neq i}\frac{p_ip_j}{p_i+p_j}\cdot\frac{1}{k}\\
&= 1+\frac{2}{k}\cdot\sum_{j<i}\frac{p_ip_j}{p_i+p_j}\,.
\end{aligned}$$

Die Abschätzung folgt dabei aus

$$\max_{0\leq p\leq 1}\frac{p(1-p)^{k-1}}{1-(1-p)^k}\;\leq\;\frac{1}{k}\,,$$

was man durch eine Kurvendiskussion zeigt: die Funktion ist monoton fallend und nimmt ihr Maximum $1/k$ bei $p=0$ an (hebbare Singularität). □

Die beiden Verfahren ARRAY und MRAND erzielen somit den optimalen Speedup des MTF-Verfahrens in diesem Modell: das Verhältnis zu den optimalen Kosten beträgt konstant 2 unabhängig von der Anzahl k der Listen.

Kompetitives Modell

In einem Kostenmodell mit Gegner muß zunächst geklärt werden, wieviel beliebige Vertauschungen von Listenelementen kosten. Ein bezahlter Austausch innerhalb einer Liste wird wie im Standardmodell mit Einheitskosten bewertet. Zwischen verschiedenen Listen ist ein Austausch der Elemente an den Listenköpfen mit Kosten 1 möglich.

Satz 1.69. *Es gibt Zugriffsfolgen σ, so daß jeder deterministische Algorithmus* A *gegenüber einem statischen Gegner* OPT *um einen Faktor*

$$\frac{C_A(\sigma)}{C_{OPT}(\sigma)}\geq\frac{2}{1+\lceil n/k\rceil^{-1}}$$

schlechter ist.

Beweis. Sei A ein deterministisches Verfahren für k Listen. Dann gibt es stets eine Liste der Länge mindestens $\lceil n/k\rceil$. Der Gegner konstruiert eine Zugriffsfolge σ der Länge m, indem er immer auf das letzte Element in einer solchen Liste zugreift und dadurch Gesamtkosten $C_A(\sigma)\geq mn/k$ erzeugt. Diese Folge kann offline bearbeitet werden, indem die Liste zunächst wie bei MOPT geordnet wird (siehe oben) und dann statisch bleibt, was Kosten $\binom{n}{2}$ für die Sortierung und dann höchstens $m(\lceil n/k \rceil + 1)/2$ für die Zugriffe verursacht. Also folgt

$$\frac{C_A(\sigma)}{C_{\mathrm{OPT}}(\sigma)} \geq \frac{m\lceil n/k \rceil}{\binom{n}{2} + m(\lceil n/k \rceil + 1)/2} \longrightarrow \frac{2}{1 + \lceil n/k \rceil^{-1}} \qquad (m \to \infty).$$

□

Für feste k muß also mit wachsendem n der kompetitive Faktor beliebig nahe an 2 liegen. Wenn aber $n = \alpha k$ für ein festes α ist, so besteht die Möglichkeit, daß das kompetitive Verhältnis eines deterministischen Algorithmus für die Suche in mehreren Listen unter 2 liegt.

2. Anwendungen

Dieses Kapitel beschreibt Anwendungen der vorgestellten adaptiven Verfahren. Im ersten Abschnitt wird die Anwendung von adaptiven Listenalgorithmen zur Gewinnung statistischer Modelle für die Datenkompression beschrieben. Anschließend werden zwei Verfahren zur effizienten Berechnung von konvexen Hüllen untersucht. Wir diskutieren, wie man die Kosten adaptiver Suchalgorithmen als Maß für die Lokalität in Datenströmen einsetzen kann. Schließlich gehen wir auf die Unterschiede zwischen adaptiven Listen und Bäumen ein.

2.1. Datenkompression

Adaptive Listenalgorithmen stellen eine einfach zu implementierende und in der Komplexität billige Methode dar, um statistische Informationen aus Datenströmen zu gewinnen. Es gibt bereits einige Untersuchungen mit Listenalgorithmen, die für Kompressionsverfahren eingesetzt werden. Die folgenden Ergebnisse verallgemeinern die Aussagen für MOVE-TO-FRONT [BSTW86, E87] und TIMESTAMP [AM98]. Man vergleiche auch [GRVW95, J88] für die Anwendung von adaptiven Bäumen für die Datenkompression sowie [BCW90] für einen allgemeinen Überblick über verlustfreie Algorithmen zur Textkompression.

Eine alternative Methode, um mit geringem Ressourcen-Aufwand (Speicherplatz) Statistiken von Datenströmen zu erhalten, besteht in der Verwendung von approximativen Zählern. Solche Algorithmen sparen Speicherplatz, indem sie die Genauigkeit verringern und randomisiert arbeiten. Beispielsweise können alle Zahlen im Bereich $2^i, \dots, 2^{i+1}$ durch i repräsentiert werden, beim Inkrementieren wird der Übergang $i \mapsto i+1$ nur mit Wahrscheinlichkeit $1/2^i$ gemacht. Die Schätzung von statistischen Informationen mit solchen Verfahren wird in [AMS96, F85, HK96] behandelt.

Der Grundgedanke der Kompression mit Hilfe von Listenalgorithmen ist, daß die Liste ein statistisches Modell des Datenstroms liefert. Die Position jedes Atoms (Buchstabe oder Wort des Datenstroms) in der Liste gibt statistische Auskünfte über vergangene Auftreten des Atoms. In Abhängigkeit des verwendeten Listenalgorithmus ist das Modell mehr oder weniger adaptiv.

Zu Beginn ist die Liste in zufälliger Ordnung oder (wie bei uns) bereits nach absteigenden Zugriffshäufigkeiten der Listenelemente sortiert (was einen ersten Durchlauf über die Liste oder Vorwissen über die Daten erfordert). Die letztere Variante wurde in anderen Arbeiten betrachtet [AM98, BSTW86, E87], und wir behalten sie für die empirischen Untersuchungen bei, um die Ergebnisse vergleichbar zu machen. Als Atome betrachten wir die Bytes der Datei, die zur Kompression folgendermaßen sequentiell kodiert werden. Ein ankommendes Zeichen wird in der Liste gesucht. Wenn es auf Position i gefunden wurde, wird ein Kodewort für i ausgegeben und der Listenalgorithmus angewendet.

Zur Dekompression ist es notwendig, daß die Liste des Dekodierers mit der gleichen Ordnung wie die des Kodierers initialisiert wird. Wenn der Wert i zu dekodieren ist, dann wird das Zeichen auf Position i ausgegeben und der jeweilige Listenalgorithmus aktiviert. Da der Listenalgorithmus deterministisch ist, ist sichergestellt, daß bei Kodierer und Dekodierer zu jedem Zeitpunkt die gleiche Listenordnung vorliegt.

Wir müssen definieren, wie eine natürliche Zahl i kodiert werden soll. Die sogenannten Elias-Kodes [E75] sind eine Sammlung präfixfreier Kodierungen für diesen Zweck. Sei $n = \lfloor \log(i) \rfloor$. Die Binärdarstellung von $i = \sum_{j=0}^{n} a_j 2^j$ ist $\mathtt{bin}(i) = \langle 1, a_{n-1}, \dots, a_1, a_0 \rangle$. Im einfachsten Fall wird $i \in \mathbb{N}$ durch seine Binärdarstellung $\mathtt{bin}(i)$ repräsentiert, und die Länge dieser Darstellung unär kodiert, damit der Kode präfixfrei wird. Die Kodierung lautet

$$c_1(i) = 0^{\lfloor \log(i) \rfloor} \cdot \mathtt{bin}(i)$$

und hat die Länge $|c_1(i)| = 1 + 2\lfloor \log(i) \rfloor$, da $|\mathtt{bin}(i)| = 1 + \lfloor \log(i) \rfloor$.

Man kann dieses Schema iterieren und auch die Länge von $\mathtt{bin}(i)$ binär repräsentieren, und die Länge dieser Darstellung dann unär. Damit wird i dargestellt als

$$\begin{aligned} c_2(i) &= c_1(1 + \lfloor \log(i) \rfloor) \cdot \mathtt{bin}(i) \\ &= 0^{\lfloor \log(1 + \lfloor \log(i) \rfloor) \rfloor} \cdot \mathtt{bin}(1 + \lfloor \log(i) \rfloor) \cdot \mathtt{bin}(i) \end{aligned}$$

und hat die Länge $|c_2(i)| = 2 + \lfloor \log(i) \rfloor + 2\lfloor \log(1 + \lfloor \log(i) \rfloor) \rfloor$. Durch eine leichte Modifikation kann man noch eine Stelle sparen, da das führende Bit von $\mathtt{bin}(i)$ stets 1 ist. Mit $\mathtt{bin}(i) = 1 \cdot \mathtt{bin}'(i)$ erhält man

$$\begin{aligned} c_3(i) &= c_1(1 + \lfloor \log(i) \rfloor) \cdot \mathtt{bin}'(i) \\ &= 0^{\lfloor \log(1 + \lfloor \log(i) \rfloor) \rfloor} \cdot \mathtt{bin}(1 + \lfloor \log(i) \rfloor) \cdot \mathtt{bin}'(i). \end{aligned}$$

Diese Kodierung c_3 und die Längenfunktion

$$f(i) = 1 + \log(i) + 2\log(1 + \log(i)) \;\geq\; |c_3(i)|$$

verwenden wir im folgenden.

Es gibt Ansätze, wie man die Kompressionseigenschaften der adaptiven Verfahren noch etwas verbessern kann, wenn man für verschiedene Zahlbereiche von i verschiedene Kodierungen verwendet, und in einem zweistufigen Verfahren erst die Auswahl der Kodierung und dann die eigentliche Darstellung von i kodiert [F96, R98]. Im zweiten Schritt werden bedingte Wahrscheinlichkeiten benutzt, die bei geschickter Auswahl der Verfahren eine geringfügig bessere Kodierung möglich machen.

2.1.1. Theoretische Ergebnisse

Der folgende Satz besagt, daß die erwartete Kodelänge eines Zeichens nur wenig über der unteren Schranke der Entropie $H(\mathbf{p})$ liegt, und auch nur wenig über der oberen Schranke $H(\mathbf{p})+1$ für optimale erwartete Kodewortlängen, wie sie beispielsweise ein statischer Huffman-Kode erzeugt.

Satz 2.1. *Für beliebige Verteilungen $\mathbf{p} = (p_1, \dots, p_n)$ gilt für die erwartete Länge $E(\mathbf{p})$ der Kodierung eines Zeichens mit der* SBD(k)*-Regel*

$$E(\mathbf{p}) \leq 1 + \overline{H}(\mathbf{p}) + 2\log(1 + \overline{H}(\mathbf{p})),$$

wobei $\overline{H}(\mathbf{p}) = H(\mathbf{p}) - a(\mathbf{p})$ ist, und $a(\mathbf{p}) \geq 0$ eine von der Verteilung abhängige Konstante.

Beweis. Sei ohne Beschränkung der Allgemeinheit $p_1 \geq \cdots \geq p_n$. Sei $f(i) = 1 + \log(i) + 2\log(1+\log(i))$. Wir benutzen die Ungleichung von Jensen, die besagt, daß für beliebige konkave Funktionen f und positive reelle Konstanten $p_1, \dots, p_n$ gilt, daß $\sum_{i=1}^n p_i f(x_i) \leq f(\sum_{i=1}^n p_i x_i)$. Die asymptotische mittlere Position von Element i ist $1 + \sum_{j \neq i} b(j,i)$, wobei $b(j,i)$ gemäß (1.12) auf Seite 39 gegeben ist.

$$\begin{aligned}
&\sum_{i=1}^n p_i \log(1 + \sum_{j\neq i} b(j,i)) \\
&= \sum_{i=1}^n p_i \log\left(\frac{1}{2} + \sum_{j=1}^n \sum_{\ell=0}^{k-1} \binom{2k-1}{\ell} \frac{p_i^\ell p_j^{2k-1-\ell}}{(p_i+p_j)^{2k-1}}\right) \\
&= \sum_{i=1}^n p_i \log\left(\frac{1}{2} + \sum_{j=1}^n \frac{p_j}{p_i+p_j} + \sum_{j=1}^n \sum_{\ell=0}^{k-2} \binom{2k-2}{\ell} \frac{p_i^{\ell+1} p_j^{2k-2-\ell} - p_i^{2k-2-\ell} p_j^{\ell+1}}{(p_i+p_j)^{2k-1}}\right) \\
&\leq \sum_{i=1}^n p_i \log\left(\frac{1}{p_i} + \sum_{j=1}^n \sum_{\ell=0}^{k-2} \binom{2k-2}{\ell} \frac{p_i^{\ell+1} p_j^{2k-2-\ell} - p_i^{2k-2-\ell} p_j^{\ell+1}}{(p_i+p_j)^{2k-1}}\right) \\
&= \sum_{i=1}^n p_i \log\left(\frac{1}{p_i}\right) + \sum_{i=1}^n p_i \log\left(1 + \sum_{j=1}^n \sum_{\ell=0}^{k-2} \binom{2k-2}{\ell} \frac{p_i^{\ell+2} p_j^{2k-2-\ell} - p_i^{2k-1-\ell} p_j^{\ell+1}}{(p_i+p_j)^{2k-1}}\right) \\
&\leq \sum_{i=1}^n p_i \log\left(\frac{1}{p_i}\right) + \log\Bigg(1 - \sum_{1\leq i\leq j\leq n} \frac{p_i p_j}{(p_i+p_j)^{2k-1}} \\
&\qquad \cdot \sum_{\ell=0}^{k-2} \binom{2k-2}{\ell} \left(p_i^{2k-1-\ell} p_j^{\ell} - p_i^{\ell+2} p_j^{2k-3-\ell} - p_i^{2k-2-\ell} p_j^{\ell+1} + p_i^{\ell+1} p_j^{2k-2-\ell}\right)\Bigg) \\
&= H(\mathbf{p}) - a(\mathbf{p}) =: \overline{H}(\mathbf{p})\,.
\end{aligned}$$

Dabei folgt das erste Gleichheitszeichen durch Einsetzen von (1.12) und der Tatsache $b(i,i) = 1/2$, das zweite Gleichheitszeichen aus $\binom{2k-2}{\ell} + \binom{2k-2}{\ell+1} = \binom{2k-1}{\ell+1}$ und die folgende Ungleichung aus $1/2 + \sum_{j=1}^n p_j/(p_i+p_j) \leq 1/p_i$. Die Aussage $a(\mathbf{p}) \geq 0$ folgt durch eine Analyse der inneren Summe in der Funktion $a(\mathbf{p})$. Für die innere Klammer gilt nämlich wegen $p_i \geq p_j$

$$\begin{aligned}
&p_i^{2k-1-\ell}p_j^{\ell} - p_i^{\ell+2}p_j^{2k-3-\ell} - p_i^{2k-2-\ell}p_j^{\ell+1} + p_i^{\ell+1}p_j^{2k-2-\ell} \\
&= p_i^{\ell+1}p_j^{\ell} \cdot \left(p_i^{2k-2-2\ell} - p_i p_j^{2k-3-2\ell} - p_i^{2k-3-2\ell}p_j + p_j^{2k-2-2\ell}\right) \\
&= p_i^{\ell+1}p_j^{\ell} \cdot (p_i - p_j)\left(p_i^{2k-3-2\ell} - p_j^{2k-3-2\ell}\right) \\
&\geq 0.
\end{aligned}$$

Insgesamt folgt

$$\begin{aligned}
E(\mathbf{p}) &\leq \sum_{i=1}^{n} p_i f\Big(1 + \sum_{j\neq i} b(j,i)\Big) \\
&\leq 1 + \sum_{i=1}^{n} p_i \log\Big(1 + \sum_{j\neq i} b(j,i)\Big) + 2\sum_{i=1}^{n} \log\Big(1 + p_i \log(1 + \sum_{j\neq i} b(j,i))\Big) \\
&\leq 1 + \overline{H}(\mathbf{p}) + 2\log(1 + \overline{H}(\mathbf{p}))\ .
\end{aligned}$$

□

Auch wenn keine Voraussetzungen über den Datenstrom gemacht werden, können wir die mittlere Kodelänge gegen die empirische Entropie der Folge abschätzen. Sei $\sigma = \sigma_1, \ldots, \sigma_m$ eine beliebige Folge von Listenelementen, und sei m_i die Häufigkeit des Vorkommens von i in σ. Die relative Häufigkeit von i ist damit m_i/m, und die empirische Entropie ist

$$H^*(\sigma) = \sum_{i=1}^{n} \frac{m_i}{m} \log\Big(\frac{m}{m_i}\Big).$$

Sei $E(\sigma)$ die mittlere Kodewortlänge eines Zeichens.

Satz 2.2. *Sei σ eine beliebige Zugriffsfolge, dann gilt unter Verwendung der* SBD(k)*-Regel und für* $k > 1$,

$$E(\sigma) \leq 1 + \log(k-1) + H^*(\sigma) + 2\log(1 + \log(k-1) + H^*(\sigma)).$$

Beweis. Für $k = 1$ wurde in [BSTW86] gezeigt, daß $E(\sigma) \leq 1 + H^*(\sigma) + 2\log(1 + H^*(\sigma))$. Sei nun $k > 1$. Wir benutzen wieder die Funktion $f(i) = 1 + \log(i) + 2\log(1 + \log(i))$ als obere Schranke der Kodewortlängen des Elias-Kodes c_3. Sei i ein beliebiges Element der Liste, und $1 \leq t_1 < t_2 < \cdots < t_{m_i}$ die Zeitpunkte mit $\sigma_{t_s} = i$, $s = 1, \ldots, m_i$. Ohne Beschränkung der Allgemeinheit werden die ersten $k-1$ Auftreten von i durch $f(t_1), \ldots, f(t_{k-1})$ kodiert. Sei nun $s \geq k$ und sei r_s die Position von i in der Liste, direkt nachdem σ_{t_s} übertragen wurde. Wir zeigen, daß das s-te Vorkommen von i mit $f(r_{s-k+1} + t_s - t_{s-k+1} - r_s)$ Bits kodiert werden kann. Sei d_s die Anzahl der Elemente

$j \neq i$, die in dem Intervall $[t_{s-k+1}+1, t_s]$ mindestens k mal nachgefragt werden. Es gilt also $r_s = d_s + 1$. Da während dieses Intervalls auf höchstens $t_s - t_{s-k+1} - 1 - (k-1)d_s \leq t_s - t_{s-k+1} - 1 - d_s$ verschiedene Elemente zugegriffen wurde, können also höchstens so viele Elemente vor i gezogen worden sein, und das s-te Auftreten von i kann mit $f(r_{s-k+1} + t_s - t_{s-k+1} - 1 - d_s) = f(r_{s-k+1} + t_s - t_{s-k+1} - r_s)$ Bits kodiert werden. Zusammen erhält man für die Länge der Kodierung aller m_i Vorkommen von i

$$
\begin{aligned}
& f(t_1) + \cdots + f(t_{k-1}) + \sum_{s=k}^{m_i} f(r_{s-k+1} + t_s - t_{s-k+1} - r_s) \\
\leq\ & m_i \cdot f\left(\frac{1}{m_i}\Big(t_1 + \cdots + t_{k-1} + \sum_{s=k}^{m_i}(r_{s-k+1} + t_s - t_{s-k+1} - r_s)\Big)\right) \\
=\ & m_i \cdot f\left(\frac{1}{m_i}\Big(t_{m_i-k+2} + \cdots + t_{m_i} + r_1 + \cdots + r_{k-1} - r_{m_i-k+2} - \cdots - r_{m_i}\Big)\right) \\
\leq\ & m_i \cdot f\left(\frac{(k-1)m}{m_i}\right).
\end{aligned}
$$

Dies gilt, da $t_s \leq m$ für alle s und $r_1 = \cdots = r_{k-1} = 1$ (nach Definition des Algorithmus wird bei den ersten $k-1$ Zugriffen das Element stets an den Listenanfang gesetzt) sowie $r_s \geq 1$ für alle s. Aufsummieren über alle i und Division durch m ergibt

$$
\begin{aligned}
E(\sigma) \ &\leq\ \sum_{i=1}^{n} \frac{m_i}{m} \cdot f\left(\frac{(k-1)m}{m_i}\right) \\
&\leq\ 1 + \log(k-1) + H^*(\sigma) + 2(1 + \log(k-1) + H^*(\sigma)).
\end{aligned}
$$

□

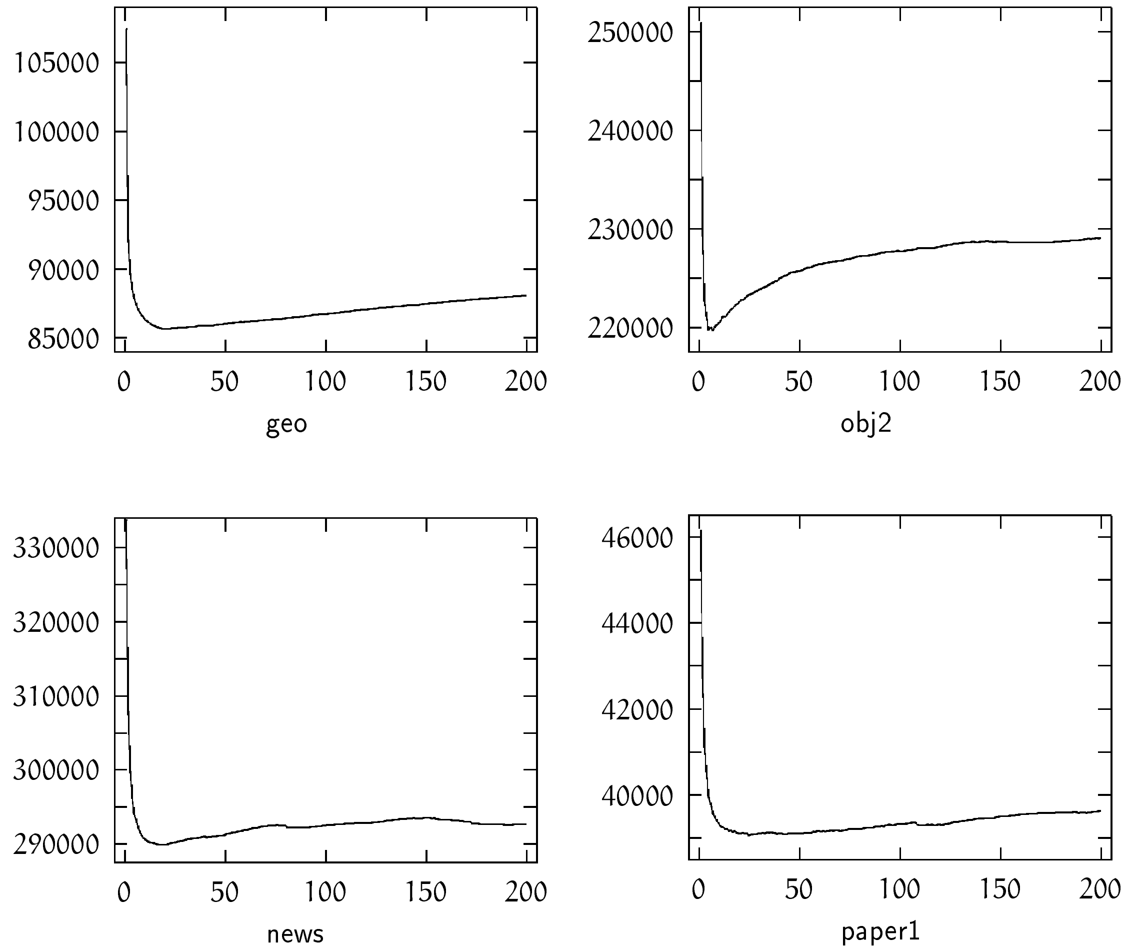
Dieses Resultat stellt eine Parallele zu Satz 1.17 dar. Da bei der Kompression nicht die kumulierten Zugriffskosten betrachtet werden, sondern die Summe der logarithmierten Kosten, ist die mittlere Kodewortlänge nicht um einen Faktor k, sondern um einen Summand $\log(k-1)$ größer als bei kompetitiv-optimalen Verfahren.

2.1.2. Empirische Ergebnisse

SBD(k)-Algorithmen

Mit den SBD(k)-Verfahren haben wir Experimente auf realen Datenströmen ausgeführt. Eine beliebte Grundlage hierzu ist der *Calgary/Canterbury Compression Corpus* [WB], der als eine Standard-Sammlung von Dateien zum Test von Kompressionsverfahren dient.

In unserem Fall enthält die Liste die 256 möglichen Bytes, die in den Dateien vorkommen können. Weiter ist zu beachten, daß bei dem Kompressionsexperiment nicht die Zugriffskosten gemessen werden, sondern die Kodewortlängen, was den logarithmierten Zugriffskosten entspricht. Dies machen wir, um einen direkten Vergleich mit den Kompressionsexperimenten anderer Arbeiten zu ermöglichen.



In den obigen Abbildungen sind für vier Beispiele die Längen der komprimierten Dateien dargestellt, wenn SBD(k) benutzt wird, $1 \leq k \leq 200$. Der Wert k^* soll diejenige Regel unter den 200 getesteten Fällen beschreiben, die die beste Kompressionsrate erreicht. Mit anderen Worten, der Listenalgorithmus SBD(k^*) ist am besten an die Eingabefolge angepaßt. Da sich die SBD(k)-Verfahren im Grad ihrer Adaptivität unterscheiden, kann man auch folgern, daß der Parameter k^* den Grad der Lokalität in der Eingabefolge beschreibt. Manchmal ist der Wert k^* jedoch nicht deutlich ausgeprägt, und die Kompressionsrate ist in einem ganzen Intervall um k^* herum fast gleich gut (z.B. bei den Dateien news und paper1). Deshalb kann der Wert k^* kein präzises Maß für Lokalität darstellen.

In der folgenden Tabelle sind die Ergebnisse zusammengestellt und verglichen mit der Größe der ursprünglichen Datei, und mit den Resultaten bei MOVE-TO-FRONT-Kompression ($k = 1$), und FREQUENCY-COUNT-Kompression. Letztere benutzt als statistisches Modell für die Daten eine statische Liste, die nach absteigenden Zugriffshäufigkeiten sortiert ist und nie geändert wird.

File	Original Bytes	FC Bytes	MTF Bytes	k^*	SORT-BY-DELAY(k^*) Bytes	% Orig	% FC	% MTF
bib	111261	88641	106468	73	89145	80.1	100.6	83.7
book1	768771	516198	644418	153	516966	67.2	100.1	80.2
book2	610856	434798	515255	23	432940	70.9	99.6	84.0
geo	102400	85533	107422	19	85687	83.7	100.2	79.8
news	377109	292800	333729	18	289925	76.9	99.0	86.9
obj1	21504	19304	19351	3	17802	82.8	92.2	92.0
obj2	246814	237165	250952	7	219668	89.0	92.6	87.5
paper1	53161	39617	46140	25	39066	73.5	98.6	84.7
paper2	82199	56356	69437	54	56539	68.8	100.3	81.4
paper3	46526	32497	39373	35	32697	70.3	100.6	83.0
paper4	13286	9229	11273	30	9327	70.2	101.1	82.7
paper5	11954	8741	10195	18	8759	73.3	100.2	85.9
paper6	38105	28487	32073	13	27320	71.7	95.9	85.2
pic	513216	111607	119125	7	109733	21.4	98.3	92.1
progc	39611	30779	39150	13	30414	76.8	98.8	77.7
progl	71646	51260	55178	14	48776	68.1	95.2	88.4
progp	49379	35066	40041	9	34903	70.7	99.5	87.2
trans	93695	82297	82055	5	76895	82.1	93.4	93.7

Die Ergebnisse von SBD(k^*) sind deutlich besser als die von MTF, mit typischen Verbesserungen um 15 %. Es fällt auf, daß SBD(k^*) ungefähr das gleiche Ergebnis wie FREQUENCY-COUNT erzielt. Das gute Abschneiden von FC auf den Dateien des Calgary Compression Corpus wurde auch schon in [BE97] festgestellt. Da bei SBD(k^*) der Algorithmus mit optimalem k gewählt wird, was nur durch einen Vergleich der Kompressionsraten mit allen k, $1 \leq k \leq 200$, gefunden wurde, stellen die Werte der SBD(k^*) eher eine untere Schranke für online erreichbare Kompressionsraten mit diesen Algorithmen dar.

Adaptive Wahl des Parameters

Deshalb untersuchen wir jetzt, wie man geeignete Parameter k für SBD(k) online bestimmen kann und welche Auswirkungen auf die Kompression sich ergeben. Dazu wird die Eingabefolge in Blöcke der festen Länge L geteilt und für jeden gelesenen Block nachträglich festgestellt, welches Verfahren in diesem Block die beste Kompression ergeben hätte. Aus dem auf diese Weise bestimmten Parameter $\overline{k}$ und dem zuletzt verwendeten Parameter k wird ein neuer Wert ermittelt, der das Verfahren zur Kodierung des nächsten Blocks angibt. Man beachte, daß Kodierung und Dekodierung online (d.h. ohne Pufferung der Daten) ablaufen können. Jedoch müssen Kodierer und Dekodierer recht umfangreiche Statistiken über die Daten des letzten Blocks führen. Es hat sich in den Experimenten herausgestellt, daß es wesentlich besser ist, nur die Statistik des letzten Blocks zu betrachten, als alle Daten vom Anfang der Datei bis zur aktuellen Stelle einzubeziehen; auf diese Weise kann man sich wechselnden Eigenschaften des Datenstroms besser anpassen. Nach jedem Block muß der als nächstes verwendete Parameter dem Dekodierer mitgeteilt werden. Diesen haben wir in den Experimenten in einem Byte dargestellt, was einen Mehraufwand von $1/L$ der Größe der ursprünglichen Datei bedeutet.

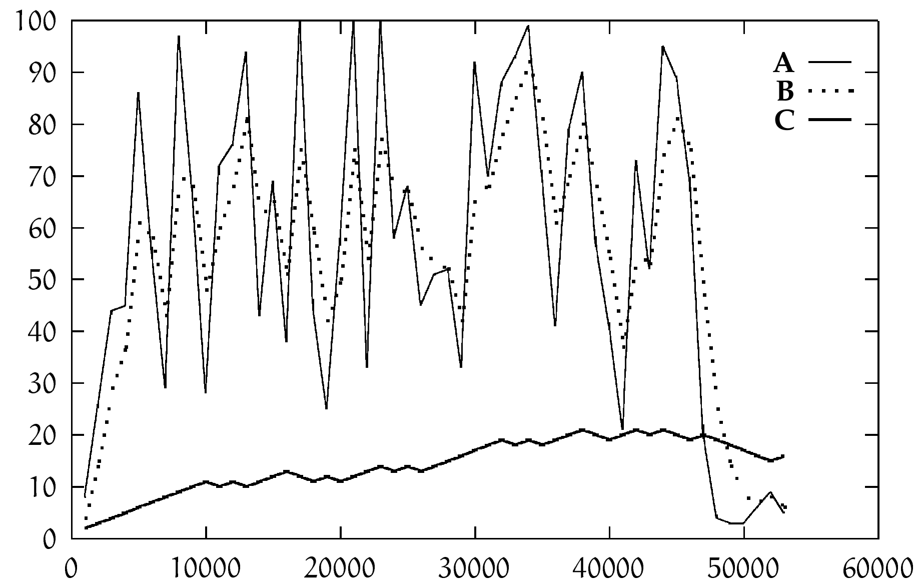
Zur Wahl des Parameters haben wir drei Varianten untersucht.

A. Der für den letzten Block beste Parameter $\overline{k}$ im Bereich $1 \leq \overline{k} \leq 100$ wird als neuer Wert übernommen. Dies kann zu heftigen Schwankungen des Parameters führen, erzielte aber eine gute Kompression.

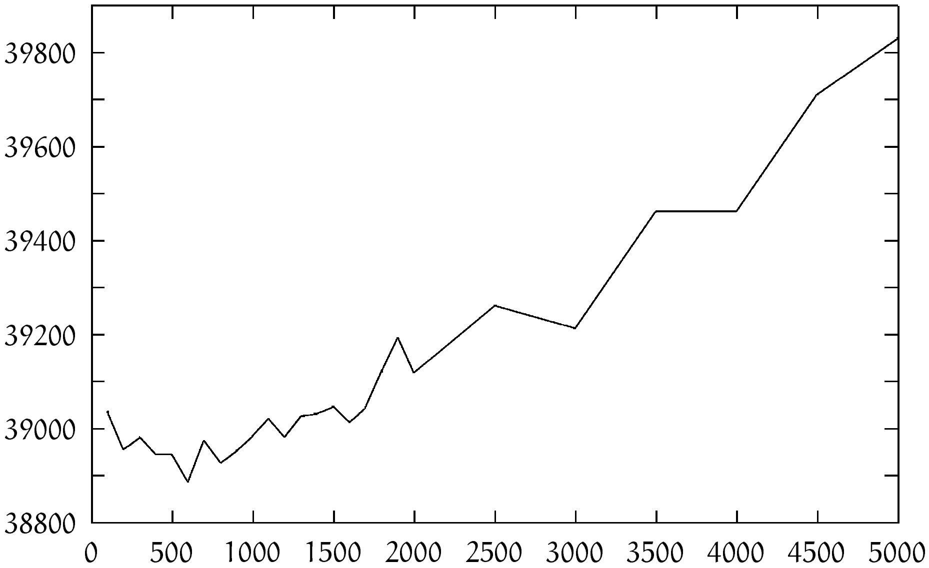
B. Der im letzten Block beste Wert $\overline{k}$ zwischen 1 und 100 und der aktuelle Wert k werden gemittelt, und $\lfloor (k + \overline{k})/2 \rfloor$ als neuer Wert benutzt. Auf diese Weise werden die starken Parameterschwankungen bei **A** gedämpft.

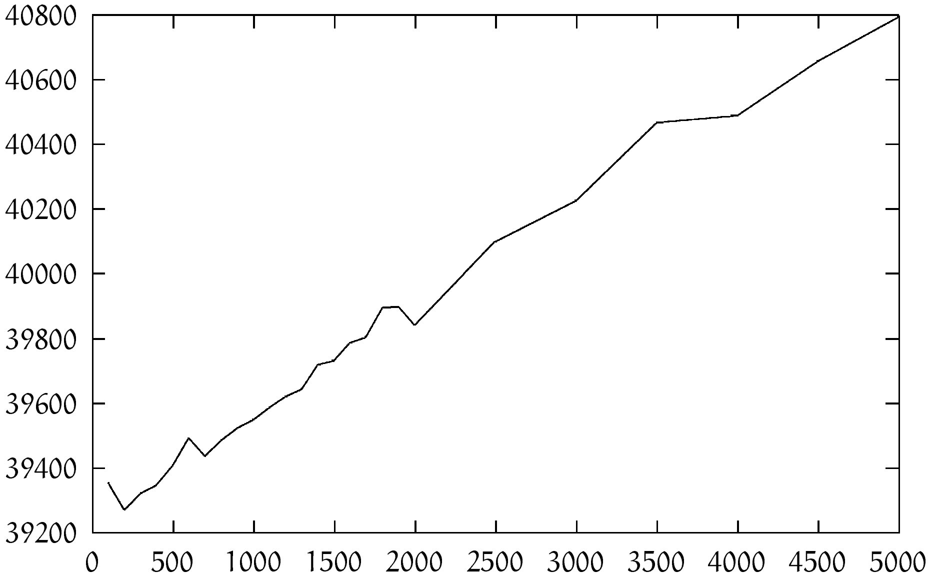
C. Zwischen zwei Blöcken darf sich der Parameter höchstens um 1 ändern. Wenn $k > 1$ ist und $k-1$ besser gewesen wäre, dann wird mit SBD$(k-1)$ weitergemacht. Falls $k < 100$ und $k+1$ besser gewesen wäre, wird der nächste Block mit SBD$(k+1)$ kodiert. Falls sowohl $k+1$ und $k-1$ die bessere Kompression ergeben hätten, dann wird als zweites Kriterium der Betrag der Kompression betrachtet und die bessere Alternative ausgewählt.

Der Startparameter ist stets $k = 1$, d.h. der erste Block wird mit MTF kodiert. Das folgende Diagramm zeigt den Verlauf des Parameters k bei der Kodierung der Datei `paper1` mit den obigen Verfahren und Blocklänge $L = 1000$.



Die nächsten Grafiken vergleichen die Kodierungen **A** bzw. **C** der Datei paper1 für L zwischen 100 und 5000. Gute Werte liegen zwischen 200 und 1000, und im folgenden halten wir $L = 1000$ fest.





Die folgende Tabelle vergleicht die erzielten Kompressionsraten der Verfahren **A**, **B** und **C** bei einer Blocklänge von $L = 1000$. Man erkennt, daß die online komprimierten Dateien ungefähr die gleiche Größe besitzen wie die mit SBD(k^*) kodierten Dateien, wobei k^* offline bestimmt wurde. Bei Verfahren **A** ist das Ergebnis stets um weniger als 1% größer als bei SBD(k^*). Das bedeutet, daß auf den Dateien des Calgary Compression

Corpus eine Online-Kodierung mit adaptiver Wahl der SBD(k)-Verfahren möglich ist, ohne nennenswerte Kompressionsverluste gegenüber dem optimalen Offline-SBD(k)-Verfahren hinnehmen zu müssen. Zum Teil ist sogar eine noch bessere Kompression möglich (Dateien paper1 und paper2 und Verfahren **A**), da die adaptiven Verfahren sich möglichen Änderungen des Lokalitätsgrades innerhalb des Datenstroms anpassen können. Die Prozentangaben beziehen sich auf das Kompressionsergebnis bei SBD(k^*).

File	Original Bytes	k^*	SBD(k^*) Bytes	**A** Bytes	**A** %	**B** Bytes	**B** %	**C** Bytes	**C** %
bib	111261	73	89145	89195	100.1	89210	100.1	89889	100.8
book1	768771	153	516966	517958	100.2	517732	100.1	518428	100.3
book2	610856	23	432940	433178	100.1	433214	100.1	434165	100.3
geo	102400	19	85687	85897	100.2	85835	100.2	86028	100.4
news	377109	18	289925	290472	100.2	290757	100.3	290361	100.2
obj1	21504	3	17802	17859	100.3	18324	102.9	17927	100.7
obj2	246814	7	219668	220670	100.5	221024	100.6	220075	100.2
paper1	53161	25	39066	38985	99.8	39069	100.0	39550	101.2
paper2	82199	54	56539	55646	98.4	56636	100.2	57114	101.0
paper3	46526	35	32697	32762	100.2	32777	100.2	33156	101.4
paper4	13286	30	9327	9407	100.9	9426	101.1	9712	104.1
paper5	11954	18	8759	8778	100.2	8790	100.4	9082	103.7
paper6	38105	13	27320	27362	100.2	27427	100.4	27679	101.3
pic	513216	7	109733	110067	100.3	110491	100.7	111550	101.7
progc	39611	13	30414	30619	100.7	30612	100.7	30715	101.0
progl	71646	14	48776	48814	100.1	48819	100.1	48903	100.3
progp	49379	9	34903	34939	100.1	35042	100.4	35107	100.6
trans	93695	5	76895	77474	100.8	77474	100.8	77266	100.5

SBR(α) und die Burrows-Wheeler-Transformation

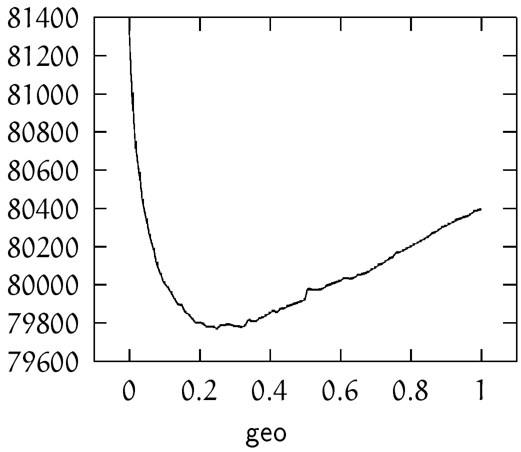
Die SBR(α)-Verfahren, die den Raum zwischen MOVE-TO-FRONT und TIMESTAMP ausfüllen, reagieren schneller und agressiver als die SBD(k)-Regeln für $k \geq 3$. Deshalb kann ihr Einsatz nur bei Datenströmen mit sehr hoher Lokalität sinnvoll sein. Solche Datenströme erscheinen beispielsweise als Ergebnis einer Transformation, die 1994 von *M. Burrows* und *D. Wheeler* veröffentlicht wurde [BW94].

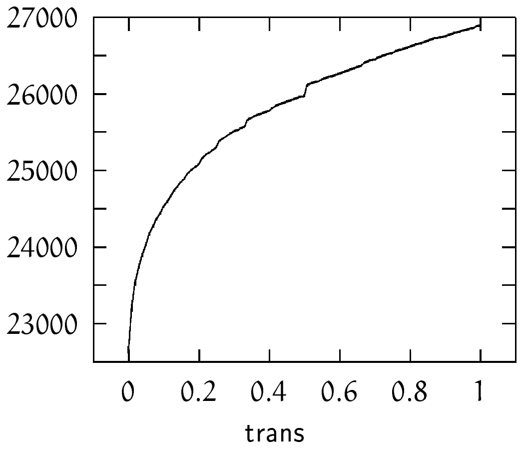
Diese Transformation teilt den Eingabestrom in lange Blöcke (z.B. 100000 Bytes). Jeder Block wird durch ein Sortierverfahren auf einen gleichlangen Block abgebildet, und mit sehr wenig Zusatzinformationen ist diese Abbildung umkehrbar. Die Ausgabe der Transformation besitzt eine hohe Lokalität und ist damit für eine Kompression mit dem MTF-Verfahren bestens geeignet. Weitere Untersuchungen zu diesem interessanten Verfahren findet man in [F96, R98, Sa98].

Unser Ziel ist es, die SBR(α)-Regeln zu einer Feineinstellung der Kompression zu benutzen. Wir verwenden dazu die Beispielimplementierung aus [Ne96] und setzen die Blocklänge auf 100000 fest. Für $0 \le \alpha \le 1$ in Schrittweiten von 0.01 haben wir den Parameter α^* bestimmt, der die beste Kompression ergibt. Nur bei 5 der 18 Dateien des Calgary Corpus stellte sich ein Wert $\alpha > 0$ besser als MTF heraus. Angesichts der nur geringen Verbesserungen und der Einfachheit von MTF schließen wir, daß MTF der für praktische Anwendungen beste Listenalgorithmus in dieser Situation ist. Es ist aber denkbar, daß andere Kodierungen der Listenpositionen nach der Transformation durch BWT und den Listenalgorithmus anstatt der unmittelbaren Darstellung durch den Elias-Kode c_3 in Kombination mit SBR(α) andere Ergebnisse erzielen.

Nach der tabellarischen Auflistung der Ergebnisse ist für geo und trans die erzielte Kompression in Abhängigkeit von α als Diagramm dargestellt.

File	Original	BWT+MTF		α^*	BWT+SBR(α^*)		
	Bytes	Bytes	% Orig		Bytes	% Orig	% BWT+MTF
bib	111261	35440	31.9	0	35440	31.9	100.0
book1	768771	305707	39.8	0.21	299532	39.0	98.0
book2	610856	208610	34.2	0	208610	34.2	100.0
geo	102400	81371	79.5	0.25	79770	77.9	98.0
news	377109	150260	39.9	0	150260	39.9	100.0
obj1	21504	12852	59.8	0.01	12759	59.3	99.2
obj2	246814	96151	39.0	0	96151	39.0	100.0
paper1	53161	18465	34.7	0	18465	34.7	100.0
paper2	82199	28677	34.9	0.03	28530	34.7	99.4
paper3	46526	17582	37.8	0	17582	37.8	100.0
paper4	13286	5496	41.4	0	5496	41.4	100.0
paper5	11954	5070	42.4	0	5070	42.4	100.0
paper6	38105	13442	35.3	0	13442	35.5	100.0
pic	513216	102491	20.0	0.23	99340	19.4	97.0
progc	39611	13898	35.1	0	13898	35.1	100.0
progl	71646	18870	26.3	0	18870	26.3	100.0
progp	49379	12856	26.0	0	12856	26.0	100.0
trans	93695	22616	24.1	0	22616	24.1	100.0





Zum Vergleich mit Standard-Kompressionsprogrammen, die auf Wörterbüchern und dem Ziv-Lempel-Verfahren basieren, zeigt die nächste Tabelle die Ergebnisse des Kompressionsexperimentes mit dem Programm gzip Version 1.2.4, sowohl mit dem Standardverfahren als auch mit der Option -9, die das größtmögliche Wörterbuch und damit die beste Kompression erlaubt. Man erkennt, daß die Kompression mit der Burrows-Wheeler-Transformation in einigen Fällen (book1, paper1, ..., paper4) besser abschneidet, in anderen Fällen (pic) aber fast doppelt so groß werden kann.

File	Original	gzip		gzip -9		BWT+MTF	
	Bytes	Bytes	% Orig	Bytes	% Orig	Bytes	% gzip -9
bib	111261	35063	31.5	34900	31.4	35440	101.5
book1	768771	313376	40.8	312281	40.6	305707	97.9
book2	610856	206687	33.8	206158	33.8	208610	101.2
geo	102400	68493	66.9	68414	66.8	81371	118.9
news	377109	144840	38.4	144400	38.3	150260	104.1
obj1	21504	10323	48.0	10320	48.0	12852	124.5
obj2	246814	81631	33.1	81087	32.9	96151	118.6
paper1	53161	18577	34.9	18543	34.9	18465	99.6
paper2	82199	29753	36.2	29667	36.1	28677	96.7
paper3	46526	18097	38.9	18074	38.8	17582	97.3
paper4	13286	5536	41.7	5534	41.7	5496	99.3
paper5	11954	4995	41.8	4995	41.8	5070	101.5
paper6	38105	13232	34.7	13213	34.7	13442	101.7
pic	513216	56442	11.0	52381	10.2	102491	195.7
progc	39611	13275	33.5	13261	33.5	13898	104.8
progl	71646	16273	22.7	16164	22.6	18870	116.7
progp	49379	11246	22.8	11186	22.7	12856	114.9
trans	93695	18985	20.3	18862	20.1	22616	119.9

2.2. Konvexe Hüllen

In diesem Abschnitt untersuchen wir die Anwendung von adaptiven Algorithmen auf Fragestellungen der algorithmischen Geometrie. Für die Berechnung von maximalen Punkten und konvexen Hüllen sind bereits einige Ergebnisse bekannt [BCL90, D99, G94, H97, S97, S99, XZL98]. Die Vorteile von adaptiven Verfahren bei räumlichen Abhängigkeiten (Anpassung an unbekannte Wahrscheinlichkeitsverteilungen) und zeitlichen Abhängigkeiten (Ausnutzung von Lokalität) wurden auch in [GOR97] festgestellt und das Konzept der persistenten Datenstrukturen, die ihre Vergangenheit speichern, auf Splay-Bäume angewendet, um eine adaptive Punktlokalisierung zu erreichen.

Wir analysieren einen Algorithmus zur effizienten Berechnung der konvexen Hülle von n Punkten in der Ebene. Das Verfahren wurde von *G. Hotz* entwickelt [H97] und in [HS97, S99] weiter untersucht. Im folgenden geben wir einen einfachen Beweis der erwarteten linearen Laufzeit für eine allgemeine Klasse von Zugriffsverteilungen und entwickeln eine dynamische Variante des Verfahrens, die die Anzahl der Punkte n nicht im voraus zu kennen braucht. Ähnliche Ansätze wie unser Basisalgorithmus wurden in [BCL90] vorgestellt, jedoch sind sie auf gleichverteilte Punkte innerhalb eines Quadrates beschränkt, vgl. auch [BS78].

Sei die Menge $\mathcal{A} = \{P_1, \dots, P_n\}$ von Punkten des $\mathbb{R}^2$ die Eingabe, außerdem sei $conv(\mathcal{B})$ die Menge der Eckpunkte der konvexen Hülle einer endlichen Punktmenge $\mathcal{B}$. Die Idee des Verfahrens ist es, eine Teilmenge $\mathcal{X} \subset \mathcal{A}$ zu berechnen, für die gilt:

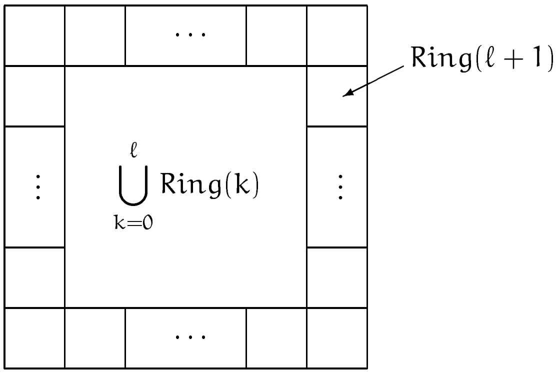
(a) $\mathcal{X}$ kann in linearer Zeit $O(n)$ gefunden werden;

(b) es gibt ein Kriterium B_n, das eine hinreichende Bedingung dafür ist, daß $conv(\mathcal{A}) = conv(\mathcal{X})$ ist, dieses Kriterium kann in linearer Zeit $O(n)$ ausgewertet werden und ist mit hoher Wahrscheinlichkeit erfüllt;

(c) $\mathcal{X}$ ist klein: $|\mathcal{X}| = O(n^\delta)$ für ein $\delta < 1$, so daß die konvexe Hülle von $\mathcal{X}$ in linearer Zeit $O(n)$ berechnet werden kann.

Mit anderen Worten, die Berechnung von $\mathcal{X}$ kann man als Filterungs-Schritt auffassen, in dem versucht wird, die meisten der Punkte, die im Inneren der konvexen Hülle liegen, auszusortieren. Somit fließt nur eine kleine Obermenge der die konvexe Hülle bestimmenden Punkte in die tatsächliche Berechnung ein. Aus Sicht der Informationstheorie besteht die Idee darin, die Ausführungspfade des Programms so zu transformieren, daß die mittlere Tiefe der Berechnung linear in n ist [H97].

Wir gehen von einer geometrischen Verteilung in Quadratringen über dem gesamten $\mathbb{R}^2$ aus. Für einen festen Parameter $0 < p < 1$ sei für einen Anfragepunkt P die Wahrscheinlichkeit, in $Ring(m)$ zu landen, Prob($P \in Ring(m)) = (1-p)p^m$ für $m \geq 0$. Auf den Ringen kann die Verteilung variieren, wir setzen nur voraus, daß die Wahrscheinlichkeit jedes Eckfeldes von $Ring(m)$ mindestens $q \geq \frac{(1-p)p^m}{8m}$ beträgt, was zum Beispiel die Gleichverteilung auf den Ringen erfüllt, falls alle Felder gleich groß sind.



Bei bekannten n und p läuft der Algorithmus folgendermaßen ab: wähle Ring $\ell = \lfloor \alpha \cdot \log(n) \rfloor$, wobei $\alpha = 1/(2\log(1/p))$ und $\log(.)$ der Logarithmus zur Basis 2 ist. Sei $K = \bigcup_{j=0}^{\ell-1} \mathrm{Ring}(j)$ das Quadrat innerhalb von $\mathrm{Ring}(\ell)$. Für jeden Punkt P wird der Test $P \in K$? in konstanter Zeit durchgeführt, und die Punkte $P \notin K$ und $P \in K$ separat in zwei Listen $\mathcal{X}$ und $\mathcal{Y}$ gespeichert. Außerdem gibt es für jedes der vier Eckfelder von $\mathrm{Ring}(\ell)$ ein Bit, das gesetzt wird, wenn mindestens ein Punkt in dem Eckfeld liegt. Dies kann ebenfalls für jeden Punkt in konstanter Zeit ausgeführt werden. Falls die vier Eckfelder von $\mathrm{Ring}(\ell)$ besetzt sind (Ereignis B), so wird die konvexe Hülle von $\mathcal{X}$ berechnet, ansonsten die konvexe Hülle aller Punkte $\mathcal{X} \cup \mathcal{Y}$.

Die Korrektheit des Verfahrens folgt daraus, daß die konvexe Hülle der Punkte das Quadrat K beinhaltet, wenn in jedem Eckfeld von $\mathrm{Ring}(\ell)$ ein Punkt liegt. Die Einteilung der n Punkte in die Mengen $\mathcal{X}$ und $\mathcal{Y}$ läuft in linearer Zeit $O(n)$ ab. Wir zeigen im folgenden, daß die erwarteten Kosten $E[C]$ zur Berechnung der konvexen Hülle ebenfalls linear in n sind.

Die Wahrscheinlichkeit, daß alle 4 Eckfelder von $\mathrm{Ring}(\ell)$ besetzt sind, ist [HS97]

$$\begin{aligned} p(B) &= \sum_{\substack{1 \le h_1,\dots,h_4 \le n \\ h_1+\dots+h_4 \le n}} \binom{n}{h_1\ h_2\ h_3\ h_4} q^{h_1+\dots+h_4}(1-q)^{n-(h_1+\dots+h_4)} \\ &= 1 - 4(1-q)^n + 6(1-2q)^n - 4(1-3q)^n + (1-4q)^n \\ &\ge 1 - 4(1-q)^n , \end{aligned}$$

wobei $q = q(\ell) \ge (1-p)p^\ell/(8\ell)$ ist. Im folgenden wählen wir n stets so groß, daß $1 - 4(1-q)^n > 0$ ist. Für die erwarteten Kosten $E[C]$ betrachten wir die bedingten Erwartungswerte:

$$\begin{aligned} E[C] &= p(B) \cdot E[C|B] + (1 - p(B)) \cdot E[C|\overline{B}] \\ &\le 1 \cdot E[C|B] + 4(1-q)^n \cdot n \log n, \end{aligned}$$

wenn man den Planesweep-Algorithmus zur Berechnung der konvexen Hülle in Zeit $O(n \log n)$ oder ein anderes Verfahren mit dieser Laufzeit verwendet. Dabei ist $E[C|B] = \sum_{k=0}^{n} p(k|B) \cdot k \log k$, und $p(k|B)$ die Wahrscheinlickkeit, daß genau k Punkte in $\mathrm{Ring}(\ell)$

oder außerhalb (d.h. in $\mathbb{R}^2 - K = \bigcup_{j=\ell}^{\infty} \mathtt{Ring}(j)$) liegen, wenn alle vier Ecken von $\mathtt{Ring}(\ell)$ besetzt sind. Zur Vereinfachung der Schreibweise lassen wir die O-Notation zunächst weg, sie wird am Ende wieder eingefügt.

Für die Wahrscheinlichkeit, daß genau k Punkte in $\mathtt{Ring}(\ell)$ oder außerhalb liegen (d.h. in $\mathcal{X}$), gilt $p(k) = \binom{n}{k}(p^\ell)^k(1-p^\ell)^{n-k}$, also folgt für die bedingte Wahrscheinlichkeit

$$p(k|B) = \frac{p(k \cap B)}{p(B)} \leq \frac{p(k)}{p(B)} \leq \frac{\binom{n}{k}(p^\ell)^k(1-p^\ell)^{n-k}}{1-4(1-q)^n}$$

und für den bedingten Erwartungswert

$$\begin{aligned}
E[C|B] &\leq \sum_{k=1}^{n} \frac{\binom{n}{k}(p^\ell)^k(1-p^\ell)^{n-k}}{1-4(1-q)^n} \cdot k \log k \\
&\leq \frac{\log n}{1-4(1-q)^n} \cdot \sum_{k=1}^{n} \binom{n}{k}(p^\ell)^k(1-p^\ell)^{n-k} \cdot k \\
&= \frac{\log n}{1-4(1-q)^n} \cdot np^\ell.
\end{aligned}$$

Zusammen erhalten wir für die erwarteten Kosten

$$E[C] \leq \underbrace{\left(\frac{p^\ell}{1-4(1-q)^n} + 4(1-q)^n\right)}_{=:D} \cdot n \log n.$$

Wir zeigen nun, daß $D = O(1/\log n)$ gilt, woraus $E[C] = O(n)$ folgt. Dazu beachte man $q \geq (1-p)p^\ell/(8\ell)$ und $\ell = \lfloor \alpha \log n \rfloor$. Außerdem gilt $(1-q)^n \leq \exp(-nq)$. Also ist (mit $p^\ell \geq p^{\alpha \log n} = n^{\alpha \log p}$):

$$\begin{aligned}
(1-q)^n &\leq \exp(-(1-p)n^{1+\alpha \log p}/(8\alpha \log n)) \\
&\leq \exp(-\Omega(\log n)) \\
&\leq O(1/n),
\end{aligned}$$

da nach Voraussetzung $\alpha < -1/\log(p)$, damit $1 + \alpha \log p > 0$ ist. Daraus folgt dann auch

$$\frac{p^\ell}{1-4(1-q)^n} = O(p^\ell)$$

und insgesamt

$$\begin{aligned}
D &\leq O(p^\ell) + O(1/n) \\
&= O(n^{\alpha \log p} + 1/n) \\
&= O(n^{\alpha \log p}) && \text{da } \alpha \log p > -1 \text{ nach Wahl von } \alpha \\
&= o(1/\log n) && \text{da } \alpha \log p < 0 \text{ für alle } p < 1.
\end{aligned}$$

Damit folgt eine sublineare erwartete Laufzeit $E[C] \leq o(n)$ für die Konstruktion der konvexen Hülle von $\mathcal{A}$, wenn $\mathcal{X}$ gegeben ist. Da aber für jeden Punkt P der Eingabe zunächst $P \in K$? getestet werden muß, erhält man eine lineare Gesamtlaufzeit.

Satz 2.3. *Bei bekannten n und p berechnet obiges Verfahren die konvexe Hülle von n Punkten in der Ebene gemäß der obigen Verteilung in erwarteter Laufzeit $O(n)$ im Registermaschinenmodell.* □

Als nächtes geben wir eine dynamische Variante des Verfahrens an, die auch bei unbekannter Punktanzahl n funktioniert. Natürlich kann die gesamte Eingabe in einer linearen Liste zwischengespeichert werden, und der obige Filterungs-Algorithmus erst gestartet werden, nachdem alle Punkte erschienen sind. In manchen Situationen kann es aber wünschenswert sein, ein dynamisches Verfahren zu besitzen, daß nach n Punkten $\mathcal{A}_n$ eine Menge $\mathcal{X}_n \subset \mathcal{A}_n$ und ein Kriterium B_n mit den obigen Eigenschaften liefert, und bei Hinzufügen eines weiteren Punktes P_{n+1} in konstanter Zeit $\mathcal{X}_{n+1}$ und B_{n+1} berechnen kann. Dazu modifizieren wir den obigen Algorithmus so, daß er inkrementell arbeitet und phasenweise den kritischen Ring ℓ nach außen schiebt.

Wir setzen $n_i = 2^i$. Der Algorithmus startet in Phase 1. Phase $i > 1$ beginnt, nachdem n_{i-1} Punkte aufgetreten sind und endet, wenn n_i Punkte erschienen sind. Für Phase i betrachten wir den Ring mit Index $\ell_i = \lfloor \alpha \cdot \log(n_i) \rfloor = \lfloor \alpha \cdot i \rfloor$, wobei wie vorher $\alpha = 1/(2\log(1/p))$ ist. Jeder neue Punkt P wird in eine von zwei Listen $\mathcal{M}_i, \mathcal{N}_i$ eingefügt, je nachdem ob $P \in \bigcup_{j=\ell_i}^{\infty} \mathrm{Ring}(j)$ oder $P \in \bigcup_{j=0}^{\ell_i - 1} \mathrm{Ring}(j)$ liegt; dies ist in konstanter Zeit pro Punkt möglich. Gleichzeitig werden entsprechende Bits gesetzt, wenn die Eckfelder von $\mathrm{Ring}(\ell_i)$ besetzt werden, dieses Kriterium sei C_i.

Wenn n Punkte erschienen sind, befindet man sich in Phase $i(n) = \lfloor \log(n) \rfloor$. Dann definieren wir die dynamisch berechnete Menge

$$\mathcal{X}_n = \bigcup_{j=0}^{i(n)} \mathcal{M}_j$$

und das zugehörige Kriterium

$$B_n = C_{i(n)-1} \vee C_{i(n)} .$$

Wenn diese Bedingung erfüllt ist, d.h. alle Eckfelder von $\mathrm{Ring}(\ell_{i(n)-1})$ oder von $\mathrm{Ring}(\ell_{i(n)})$ besetzt sind, berechnet man die konvexe Hülle von $\mathcal{X}_n$, ansonsten die konvexe Hülle aller Punkte $\mathcal{A}_n$. Wie oben zeigen wir, daß der zweite Fall nur mit geringer Wahrscheinlichkeit auftritt und daß $\mathcal{X}_n$ klein ist, so daß die erwartete Laufzeit linear in n bleibt.

Die Hauptidee des Verfahrens ist, daß wir bei einem Phasenwechsel die bisher gesammelten Informationen nicht wegwerfen, und deshalb das Kriterium B_n auch auf die letzte *abgeschlossene* Phase $i(n) - 1$ beziehen können. Wenn wir nur $B_n = C_{i(n)}$ setzen würden, könnte ein Gegner gerade so viele Punkte n produzieren, daß ein Phasenwechsel auftritt, und das neue Kriterium $C_{i(n)}$ nie erfüllt ist.

Sei die aktuelle Phase $i = i(n)$. Wie oben sieht man, daß

$$p(B_n) \geq p(C_{i-1}) \geq 1 - 4(1-q)^{n_{i-1}} ,$$

wobei $q = q(\ell_{i-1}) = (1-p)p^{\ell_{i-1}}/(8\ell_{i-1})$ ist. Sei $p(k)$ die Wahrscheinlichkeit, daß genau k Punkte in $\mathcal{X}_n$ liegen. Diese Wahrscheinlichkeit ist explizit nur umständlich auszudrücken, aber wir sind lediglich am Erwartungswert der Verteilung interessiert, den wir mit $E[X_n]$ bezeichnen. Für die bedingte Wahrscheinlichkeit, genau k Punkte in $\mathcal{X}_n$ zu finden, wenn B_n erfüllt ist, gilt wie oben

$$p(k|B_n) \leq \frac{p(k)}{p(B_n)} .$$

Damit finden wir für den bedingten Erwartungswert der Kosten

$$\begin{aligned}
E[C|B_n] &\leq \sum_{k=1}^{n} \frac{p(k)\cdot k\log k}{p(B_n)} \\
&\leq \sum_{k=1}^{n} \frac{p(k)\cdot k\log k}{1-4(1-q)^{n_{i-1}}} \\
&\leq \frac{\log n}{1-4(1-q)^{n_{i-1}}} \cdot \sum_{k=1}^{n} p(k)\cdot k \\
&= \frac{\log n}{1-4(1-q)^{n_{i-1}}} \cdot E[X_n] .
\end{aligned}$$

Zur Berechnung von $E[X_n]$ beobachten wir, daß die einzelnen Phasen des Algorithmus eine feste Länge besitzen. Also können wir den Erwartungswert ausdrücken als die Summe der Erwartungswerte in den einzelnen Phasen. In Phase j sind genau n_{j-1} Punkte aufgetreten.

$$\begin{aligned}
E[X_n] &= (n-n_{i-1})p^{\ell_i} + \sum_{j=1}^{i-1} n_{j-1}p^{\ell_j} \\
&\leq np^{\ell_i} + \sum_{j=1}^{i-1} 2^{j-1}p^{\lfloor \alpha j\rfloor} \\
&\leq np^{\ell_i} + \sum_{j=1}^{i-1} 2^{j-1}p^{\alpha j-1} \\
&= np^{\ell_i} + \frac{(2p^{\alpha})^{i-1}-1}{p^{1-\alpha}(2p^{\alpha}-1)} \\
&\leq np^{\ell_i} + \frac{np^{\alpha i-1}}{2p^{\alpha}-1} \\
&\leq \underbrace{\left(1+\frac{1/p}{2\sqrt{1/2}-1}\right)}_{=:d} \cdot np^{\ell_i} ,
\end{aligned}$$

wobei wir für die vorletzte Ungleichung $2^{i-1} \leq n$ benutzt haben und für die letzte Ungleichung $p^{\alpha} = \sqrt{1/2}$ beachten. Außerdem haben wir $\ell_j \geq \alpha j - 1 \geq \ell_j - 1$ angewendet.

Für die erwarteten Kosten folgt wie vorher

$$\begin{aligned} E[C] &= p(B_n) \cdot E[C|B_n] + (1 - p(B_n)) \cdot E[C|\overline{B_n}] \\ &\leq E[C|B_n] + 4(1-q)^n \cdot n \log n \\ &\leq \underbrace{\left(\frac{dp^{\ell_i}}{1 - 4(1-q)^n} + 4(1-q)^n \right)}_{=:D_n} \cdot n \log n \; . \end{aligned}$$

Es bleibt zu zeigen, daß $D_n = O(1/\log n)$ ist. Da d aber eine von n unabhängige Konstante ist, kann die oben gemachte Berechnung übernommen werden. Die Konstanten verschwinden in der O-Notation, und wir erhalten

Satz 2.4. *Bei bekanntem p berechnet obiges Verfahren inkrementell eine Obermenge $\mathcal{X}_n$ der konvexen Hülle. Damit kann die konvexe Hülle von n Punkten in der Ebene, deren Verteilung dem obigen Schema entspricht, in erwarteter Laufzeit $O(n)$ gefunden werden.* ☐

Eine interessante offene Frage ist, sich auch von der Abhängigkeit von p zu befreien. Wir wählen den kritischen Ring ℓ so, daß $np^{\ell} = O(n^{\delta})$ für ein $\delta < 1$. Wenn n und p unbekannt sind, dann muß die Phaseneinteilung, mit der der kritische Ring nach außen geschoben wird, von der Anzahl der bisher in den äußeren Bereich gefallenen Punkte abhängen. Da die Phasen dann variable Längen aufweisen, erscheint eine Analyse als schwierig.

2.3. Lokalität

Von *D. Knuth* wurde angemerkt, daß die Annahme unabhängiger Zugriffe bei der Analyse von adaptiven Algorithmen oft zu pessimistischen Ergebnissen führt. Praktische Experimente mit Zugriffen auf Symboltabellen in Compilern belegen: *successive searches are not independent (small groups of keys tend to occur in bunches)* [K98].

In [ACK87] wird notiert: *the locality property is exhibited when a program tends to use small subsets of its pages for relatively long periods of time*. Dort wird auch ein Markov-Modell für Zugriffe mit Lokalität angegeben, daß unserem Modell der elementweisen Lokalität entspricht, siehe Anhang A.2.2.

Eine ausführliche Diskussion und ein Markov-Modell zur Simulation des empirisch beobachteten Lokalitätsverhaltens bei Seitenzugriffen im virtuellen Speicher wurde in [ST72] präsentiert. Im *Working Set Model* von *P. Denning et.al.* [DS72] wird versucht, dieses Verhalten auszunutzen und durch *Prefetching* die Verwaltung des virtuellen Speichers zu verbessern. Neuere Untersuchungen beinhalten das Phänomen der Lokalität bei der Online-Navigation in Graphen, zum Beispiel Seitenzugriffe im World Wide Web [ABCO96].

Man kann das träge Übergangsverhalten, was bei Markov-Ketten mit Lokalität auftritt, an dem Beispiel eines mehrsprachigen Wörterbuchs mit den Sektionen *Deutsch-Englisch*, *Deutsch-Französisch*, *Deutsch-Spanisch*, usw. veranschaulichen [H93]. Bei der

Benutzung dieses Wörterbuchs zur Übersetzung von deutschen Texten in andere Sprachen wird man typischerweise lange Zeit nur die Einträge einer Fremdsprache benötigen, bevor man auf eine andere Fremdsprache umschaltet.

2.3.1. Lokalität bei Markov-Ketten

Wir sind weniger daran interessiert, ein real beobachtetes Lokalitätsverhalten zu simulieren, sondern suchen nach Kriterien, Lokalität bei Markov-Ketten festzustellen bzw. zu messen.

Wir betrachten folgendes Beispiel von $n \times n$-Übergangsmatrizen mit den Einträgen $\beta = (1-\alpha)/(n-1)$ und $0 < \beta \leq \alpha < 1$.

$$\mathbf{P}_1 = \begin{pmatrix} \alpha & \beta & \beta & \cdots & \beta \\ \beta & \alpha & \ddots & & \\ \vdots & \ddots & \ddots & & \\ & & & \ddots & \\ \beta & \cdots & \beta & \beta & \alpha \end{pmatrix} \qquad \mathbf{P}_2 = \begin{pmatrix} \beta & \alpha & \beta & \cdots & \beta \\ \beta & \beta & \alpha & & \\ \vdots & \ddots & \ddots & \ddots & \\ & & & \beta & \alpha \\ \alpha & \beta & \cdots & \beta & \beta \end{pmatrix}.$$

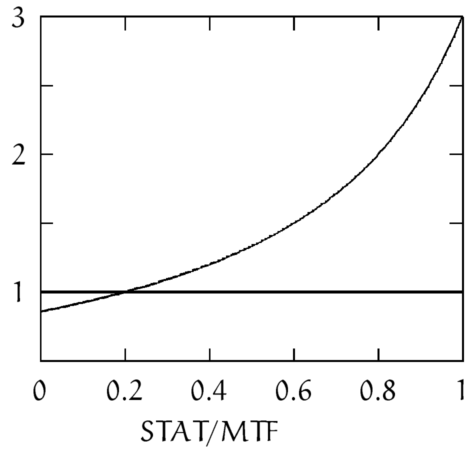
Im ersten Fall liegt elementweise Lokalität vor (vgl. Anhang A.2.2) und im zweiten Fall ist die Zugriffsfolge nahezu zyklisch und weist keine Lokalität auf. Die stationäre Verteilung beider Markov-Ketten ist die Gleichverteilung, deshalb ist ihre Entropie auch identisch: $H(\mathbf{P}_1) = H(\mathbf{P}_2)$. An diesem Beispiel erkennt man, daß die Entropie einer Markov-Kette kein ausreichendes Maß für die Lokalität bildet. Man kann lediglich festhalten, daß eine geringe Entropie eine notwendige Bedingung für das Vorliegen von Lokalität darstellt.

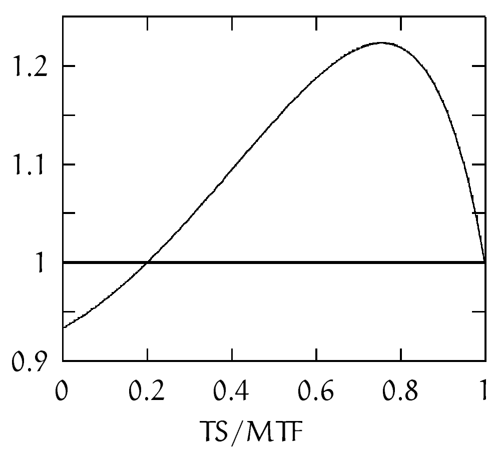
Intuitiv betrachtet bedeutet Lokalität, daß eine Markov-Kette sich nur langsam mischt und ihren Startzustand nur langsam vergißt. Entsprechende Begriffe, die das Mischungsverhalten von Markov-Ketten beschreiben, sind aus [ALW96, LW95, LW96] bekannt, vgl. auch [K94, R95]. Jedoch ist unklar, wie man diese zu dem Verhalten von adaptiven Algorithmen in Bezug setzen kann.

Wir verfolgen einen pragmatischen Ansatz, indem wir die Kosten von Listenalgorithmen auf Markov-Ketten als Maß für die Lokalität betrachten. Die Zugriffskosten des MTF-Algorithmus entsprechen gerade den sonst betrachteten Stack-Distanzen [DS72, ST72]. Dies ist die Anzahl der verschiedenen Elemente, die seit dem letzten Zugriff auf ein bestimmtes Element nachgefragt wurden. Aus diesem Grund erscheint MTF als „Benchmark"-Algorithmus geeignet. Wir vergleichen die Kosten von MTF auf einer Markov-Kette $\mathbf{P}$,

$$E(\mathbf{P}) = 1 + 2 \sum_{i<j} \frac{1}{m_{ij} + m_{ji}},$$

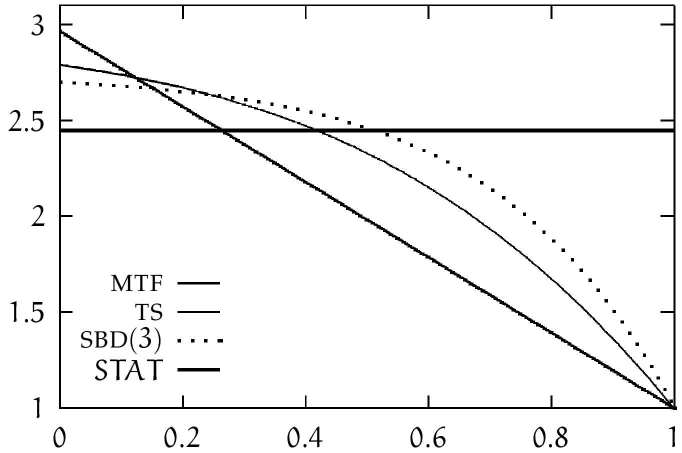
siehe Seite 14, mit den Kosten STAT der optimalen *stationären* Liste, die nach absteigenden stationären Wahrscheinlichkeiten π_i geordnet ist. Im Falle von Lokalität sollte sich MTF besser verhalten als die optimale stationäre Liste, ansonsten schlechter. Wir betrachten das Verhältnis der Kosten auf einer Liste mit $n = 5$ Elementen und einer Zugriffsfolge mit elementweiser Lokalität (siehe oben). Da die stationäre Verteilung die Gleichverteilung ist, gilt $\mathrm{STAT} = (n+1)/2 = 3$. Das linke Diagramm zeigt das Kostenverhältnis in Abhängigkeit von α.



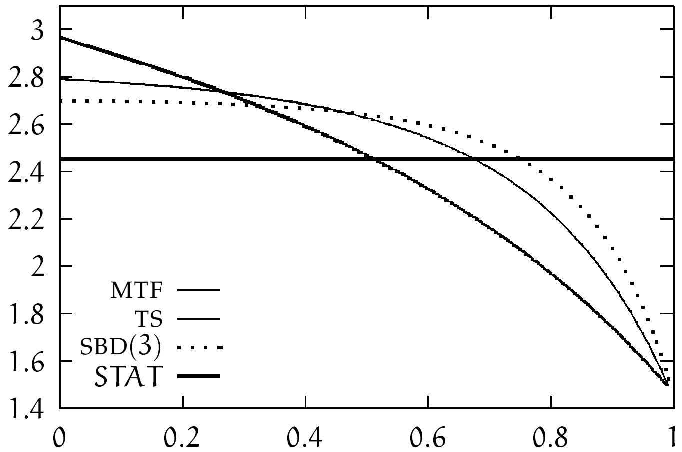


Um dieses Verhältnis auch *online* approximieren zu können, ersetzen wir die optimale stationäre Liste durch ein trägeres Verfahren als MTF, z.B. TS. Das entsprechende Kostenverhältnis ist rechts angegeben. Man vergleiche dies mit Korollar 1.29 auf Seite 51, wo die Kosten der Verfahren dargestellt sind. Man erkennt in obigem Vergleich von STAT/MTF und TS/MTF, daß für $\alpha \leq 0.7$ der Verlauf qualitativ übereinstimmt. Für größere α überwiegt jedoch auch bei dem trägeren Algorithmus TS der Einfluß der Lokalität, und das Verhältnis konvergiert gegen 1. Wenn man anstatt von TS andere Verfahren verwendet, die langsamer reagieren, z.B. SBD(k) für $k \geq 3$, so kann man zwar die Grenze 0.7, bis zu der die Kurven qualitativ ähnlich sind, nach rechts verschieben, den prinzipiellen Verlauf aber nicht ändern.

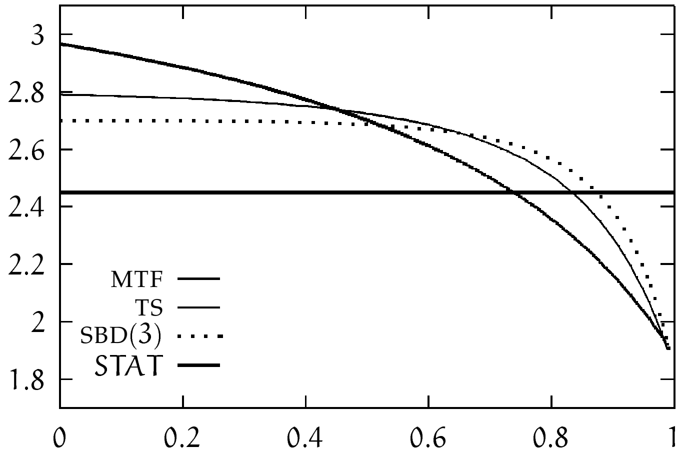
Die nächsten drei Diagramme zeigen jeweils die erwarteten Kosten von MTF, TS und SBD(3) bei blockweiser Lokalität, siehe Anhang A.2.2. Dazu betrachten wir $n = 6$ Listenelemente und die Zipf-Verteilung als zugrundeliegende Verteilung. Im ersten Fall liegen 6 Blöcke der Größe 1 vor, also elementweise Lokalität. Beim zweiten Diagramm gibt es drei Blöcke der Größe 2 und im dritten Bild zwei Blöcke der Größe 3. Außerdem sind die stationären Kosten ≈ 2.45 durch eine horizontale Linie gekennzeichnet.



$E(\mathbf{P}(\lambda))$ für $\lambda \in [0,1]$ und Blockgröße 1



$E(\mathbf{P}(\lambda))$ für $\lambda \in [0,1]$ und Blockgröße 2



$E(\mathbf{P}(\lambda))$ für $\lambda \in [0,1]$ und Blockgröße 3

An diesen Diagrammen sieht man, daß mit wachsender Blockgröße der Vorteil von reaktionsschnellen Verfahren wie MTF erst bei immer stärkerer Lokalität (größeres λ) eintritt.

Man erkennt, daß es einen kritischen Wert λ^* gibt, so daß für alle $\lambda < \lambda^*$ die stationäre Strategie die geringsten Kosten aufweist, und für alle $\lambda > \lambda^*$ die radikalste Umsortierung, nämlich MTF die besten Ergebnisse erzielt. Mit anderen Worten, es gibt kein Intervall für λ, bei dem ein Verfahren zwischen STAT und MTF optimale Kosten erreicht. Es gibt keinen stetigen Übergang zwischen den optimalen Strategien. Diese Beobachtung können wir leider nur empirisch belegen.

Weitere Ansätze zur Klassifikation von Markov-Ketten werden unter anderem auch in [EFT97, P92] gemacht, ohne jedoch auf Lokalitätseigenschaften einzugehen.

2.4. Adaptive Suchbäume

Das Äquivalent zur MTF-Regel für lineare Listen wurde von *B. Allen* und *I. Munro* untersucht [AM78]:

> **Zur-Wurzel-Rotieren-Regel**, MOVE-TO-ROOT, MTR: Wenn x nachgefragt wurde, so bringe x durch eine Folge von Rotationen an die Wurzel des Suchbaums.

Analog dazu ist es prinzipiell möglich, die in Kapitel 1 vorgestellten Algorithmen für Listen auf Suchbäume zu übertragen. Jedoch zielen diese Verfahren darauf ab, eine Ordnung der Listenelemente gemäß absteigenden Häufigkeiten zu erhalten. Bäume, deren Wurzelelement die höchste Zugriffswahrscheinlichkeit besitzt, und bei dem die Zugriffswahrscheinlichkeiten auf jedem Pfad monoton abnehmen (Heap-Ordnung), sind im allgemeinen aber keine optimalen Bäume. Diese sogenannten monotonen Bäume (*monotonic trees*) können erwartete Suchkosten aufweisen, die bis zu einem Faktor $n/(4 \log n)$ über den Kosten eines optimalen Baums mit n Knoten liegen, vgl. *K. Mehlhorn* [M75], und die Diskussion in [B79, GMS81]. Deshalb haben wir uns in Kapitel 1 auf selbst-organisiernde Listenalgorithmen beschränkt.

Im kompetitiven Analysemodell können die Splay-Bäume [ST85] logarithmische Zugriffskosten garantieren. Man vergleiche [AW98] für weitere kompetitive Analysen von adaptiven Baumstrukturen und [SS96] für einen Ansatz, die MTR-Regel bei Markov-Zugriffen zu analysieren.

3. Kodes und Suchbäume mit geometrischen Kosten

In diesem Kapitel stellen wir eine Verallgemeinerung des Kostenmaßes für Kodes und Suchbäume vor. Anstatt wie üblich jede Kante in einem Baum mit Einheitskosten zu bewerten, sollen nun die Längen der Kanten exponentiell mit der Tiefe wachsen. Zur Analyse ist ein verallgemeinerter Entropie-Begriff hilfreich, die sogenannte *Rényi-Entropie* [R61]. Obwohl Kodierungstheoreme mit neudefinierten Kostenmaßen für diese Entropie seit Mitte der 60er Jahre veröffentlicht wurden [C65, N75], ist uns keine direkte Anwendung auf Kodes und Suchbäume bekannt. Das mag daran liegen, daß diese Kostenmaße keine unmittelbare Interpretation in einem Baum erlauben. Es zeigt sich aber, daß dies durch eine entsprechende Modifikation möglich ist und man mittels des Kostenparameters die Struktur der Bäume beeinflussen und insbesondere deren Höhe kontrollieren kann.

3.1. Kodebäume für nicht geordnete Alphabete

Gegeben sei ein Symbolalphabet $\mathcal{A} = \{a_1, \dots, a_n\}$ mit einer zugehörigen Wahrscheinlichkeitsverteilung $\mathbf{p} = (p_1, \dots, p_n)$. Das Auftreten eines Symbols a_j sei unabhängig und identisch verteilt mit Wahrscheinlichkeit $\mathrm{Prob}(a_j) = p_j$.

Ein binärer Kode c für $\mathcal{A}$ in dem Kodealphabet $\mathcal{S} = \{0, 1\}$ ist eine Abbildung $c : \mathcal{A} \to \mathcal{S}^*$. Mit Kode $\mathbf{c}$ bezeichnen wir auch die Bildmenge der Abbildung c. Die Einschränkung auf binäre Kodes erfolgt nur zur einfacheren Darstellung und kann leicht aufgegeben werden; am Ende dieses Abschnitts werden die notwendigen Modifikationen für M-näre Kodes skizziert. Im folgenden bezeichnet $\log(x)$ stets den Logarithmus zur Basis 2.

Ein Kode $\{c(a_1), \dots, c(a_n)\}$ heißt präfixfrei, wenn kein Kodewort $c(a_j)$ ein Präfix eines anderen Kodewortes $c(a_i)$ ist, $j \neq i$. Präfixfreie Kodes sind eindeutig dekodierbar, siehe [H97]. Weiter sei $\ell_j = |c(a_j)|$ die Länge des Kodewortes für a_j.

Lemma 3.1. *(a) Kraft 1949, (b) McMillan 1959*

(a) Gilt für Zahlen $\ell_1, \cdots, \ell_n \in \mathbb{N}$ die Kraft'sche Ungleichung

$$\sum_{j=1}^{n} 2^{-\ell_j} \leq 1, \tag{3.1}$$

so existiert ein präfixfreier Binärkode mit diesen Kodewortlängen.

(b) *Für jeden eindeutig dekodierbaren Binärkode mit Kodewortlängen $\ell_1, \ldots, \ell_n$ gilt die Kraft'sche Ungleichung (3.1).*

Wir definieren ein neues Kostenmaß folgendermaßen.

Definition 3.1. Sei $d > 0$. Die Kosten des Kodewortes $c(a_i)$ mit Länge ℓ_i seien $\sum_{j=0}^{\ell_i-1} d^j$. Für einen Kode $\mathbf{c}$ sind die *erwarteten Kodewortkosten der Ordnung* d dann definiert durch

$$E_d(\mathbf{c}, \mathbf{p}) = \sum_{i=1}^{n} p_i(1 + d + d^2 + \cdots + d^{\ell_i - 1}).$$

□

Diese Definition verallgemeinert die mittlere Kodewortlänge und enthält als Spezialfall $d = 1$ das übliche Kostenmaß. Wenn die zugrundeliegende Zugriffsverteilung klar ist, schreiben wir für $E_d(\mathbf{c}, \mathbf{p})$ auch kürzer $E_d(\mathbf{c})$.

Eine andere Verallgemeinerung des Einheitskostenmaßes wurde in [AM80, CG96, GR98, GY96] untersucht. Dort haben die Symbole des Kodealphabetes $\mathcal{S}$ verschiedene Längen, wie es beispielsweise bei der Übertragung von Morsezeichen auftritt. Der wesentliche Unterschied zu unserem Modell ist, daß die Kosten nur von dem jeweiligen Kodesymbol abhängen, unabhängig von der Position des Symbols im Kodewort. Bei uns sind die Kosten unabhängig vom Kodesymbol, werden jedoch durch die Position innerhalb des Kodewortes bestimmt.

3.1.1. Schranken für die mittleren Kodewortkosten

Definition 3.2. *(Rényi 1961)*
Sei $\mathbf{p} = (p_1, \ldots, p_n)$ eine Wahrscheinlichkeitsverteilung und $\alpha > 0$, $\alpha \neq 1$. Man bezeichnet

$$H_\alpha(\mathbf{p}) = \frac{1}{1-\alpha} \log \left(\sum_{i=1}^{n} p_i^\alpha \right)$$

als *Rényi-Entropie der Ordnung* α. □

Mit der Regel von l'Hospital findet man die Entropie nach Shannon als Spezialfall $\alpha = 1$ der Rényi-Entropie:

$$H_1(\mathbf{p}) := \lim_{\alpha \to 1} H_\alpha(\mathbf{p}) = -\sum_{i=1}^{n} p_i \log p_i.$$

Folgende Kostenfunktion für einen Kode $\mathbf{c}$ mit Kodewortlängen ℓ_j wird in [C65] eingeführt und der anschliessende Satz gezeigt.

$$N_t(\mathbf{c}) = \frac{1}{t} \log \left(\sum_{i=1}^{n} p_i 2^{t\ell_i} \right).$$

Satz 3.2. *(Campbell 1965) Sei $\alpha > 0$, $t = 1/\alpha - 1$ und $\mathbf{p}$ eine beliebige Wahrscheinlichkeitsverteilung.*

(a) Für jeden Kode $\mathbf{c}$, dessen Kodewortlängen (3.1) erfüllen, gilt

$$H_\alpha(\mathbf{p}) \leq N_t(\mathbf{c}).$$

(b) Es gibt einen Kode $\mathbf{c}$, für dessen mittlere Kodewortlänge $N_t(\mathbf{c})$ gilt:

$$N_t(\mathbf{c}) \leq H_\alpha(\mathbf{p}) + 1.$$

Dieses Resultat können wir auf unsere Kostenfunktion $E_d(\mathbf{c})$ übertragen. Dazu sei $d > 1$. Wir setzen $2^t = d$ und damit $\alpha = 1/(1 + \log d) < 1$.

$$d^{N_t(\mathbf{c})} = \sum_{i=1}^{n} p_i d^{\ell_i} = (d-1) \cdot E_d(\mathbf{c}) + 1.$$

Außerdem gilt $\log(d)/(1-\alpha) = 1 + \log d$ und

$$d^{H_\alpha(\mathbf{p})} = \left(\sum_{i=1}^{n} p_i^{\frac{1}{1+\log d}} \right)^{1+\log d} = L_\alpha(\mathbf{p}) = L_{1/(1+\log d)}(\mathbf{p}),$$

wenn

$$L_p(\mathbf{x}) = \left(x_1^p + \cdots + x_n^p\right)^{1/p}$$

die L_p-Norm des Vektors $\mathbf{x} = (x_1, \ldots, x_n)$ bezeichnet.

Korollar 3.3. Sei $d > 1$ und $\mathbf{p}$ eine beliebige Wahrscheinlichkeitsverteilung.

(a) Für jeden Kode $\mathbf{c}$, dessen Kodewortlängen (3.1) erfüllen, gilt

$$\frac{L_{1/(1+\log d)}(\mathbf{p}) - 1}{d-1} \leq E_d(\mathbf{c}, \mathbf{p}).$$

(b) Es gibt einen Kode $\mathbf{c}$, für dessen mittlere Kodewortlänge $E_d(\mathbf{c}, \mathbf{p})$ gilt:

$$E_d(\mathbf{c}, \mathbf{p}) \leq \frac{dL_{1/(1+\log d)}(\mathbf{p}) - 1}{d-1}.$$

□

Durch Grenzwertbildung $d \to 1$ erhält man das Kodierungstheorem von Shannon zur Quellenkodierung.

Anmerkung. Wir hätten unsere Kostenfunktion auch als $c(a_j) = d^{\ell_j}$ definieren können, wodurch das Korollar die schönere Form

$$L_{1/(1+\log d)}(\mathbf{p}) \leq E_d(\mathbf{c}, \mathbf{p}) \leq d \cdot L_{1/(1+\log d)}(\mathbf{p})$$

angenommen hätte. Jedoch läßt sich diese Kostenfunktion nicht unmittelbar in Kantenkosten im Baum übersetzen, und dieses Modell enthält nicht das Standard-Kostenmaß als Spezialfall.

3.1.2. Verallgemeinerter Huffman-Algorithmus

Als nächstes geben wir einen Algorithmus an, der für beliebige $d \geq 1$ einen optimalen Kodebaum gemäß der Kostenfunktion $E_d(\mathbf{c})$ findet. Er basiert auf dem Verfahren von *D. Huffman* [H52].

Verallgemeinerter Huffman-Algorithmus

Zu jedem Knoten z wird seine Wahrscheinlichkeit $p(z)$ und sein Gewicht $w(z)$ gespeichert. Eine Warteschlange mit Prioritäten Q ist nach aufsteigenden Gewichten $w(z)$ sortiert. Sie wird mit den Alphabetsymbolen a_j initialisiert, denen die Gewichte $w(.)$ zugeordnet sind mit $w(a_j) = p_j$. Sie bilden die Blätter des entstehenden Baumes. Solange noch mehr als ein Element in Q vorhanden ist, werden zwei Elemente x, y mit kleinsten Gewichten ausgewählt und zu einem neuen Element z verschmolzen. Dazu wird ein neuer Knoten im Baum erzeugt, dessen linker und rechter Nachfolger die Knoten x und y sind. Sein Gewicht ergibt sich zu $w(z) = d \cdot (w(x) + w(y))$. Wenn nur noch ein Element in Q vorhanden ist, so repräsentiert es die Wurzel des Kodebaums. In Pseudokode lautet der Algorithmus:

```
Q = 𝒜            // (Initialisierung siehe oben)
for i from 1 to n − 1 do
    z = new node()
    x = Q.extract_min()
    y = Q.extract_min()
    left(z) = x
    right(z) = y
    w(z) = d · (w(x) + w(y))
    Q.insert(z)
od
root = Q.extract_min()
```

Im Fall $d = 1$ erhält man den originalen Huffman-Algorithmus. Die Analyse im Fall $d > 1$ folgt dem klassischen Beweis für die Optimalität des Huffman-Kodes.

Lemma 3.4. *Sei* $d > 1$ *und* $\mathcal{A} = \{a_1, \ldots, a_m\}$ *das zu kodierende Alphabet mit Gewichten* $w(a_1) \geq \cdots \geq w(a_m) > 0$ *und* $W = \sum_{i=1}^{m} w(a_i)$. *Dann gibt es einen präfixfreien Binärkode* $\mathbf{c}$ *für* $\mathcal{A}$ *und die Zugriffsverteilung* $\mathbf{p} = (w(a_1)/W, \ldots, w(a_m)/W)$, *der die erwartete Kodewortlänge* $E_d(\mathbf{c}, \mathbf{p})$ *für diese Verteilung minimiert und für den gilt:*

(a) $\ell_1 \leq \cdots \leq \ell_m$,

(b) $\ell_{m-1} = \ell_m$, *und* $c(a_{m-1})$ *und* $c(a_m)$ *unterscheiden sich nur in der letzten Stelle.*

Beweis. Durch die Normierung der Gewichte ist der Standardbeweis für die Huffman-Kodierung auch hier gültig, siehe z.B. [M96, Seite 64].

(a) Angenommen, die Kodewortlängen eines optimalen Kodes $\mathbf{c}$ für $\mathcal{A}$ erfüllen nicht $\ell_1 \leq \cdots \leq \ell_m$. Das bedeutet, es gibt $1 \leq i < j \leq m$ mit $\ell_i > \ell_j$. Falls $w(a_i) > w(a_j)$ ist, definieren wir einen Kode $\mathbf{c}'$, der aus $\mathbf{c}$ entsteht, indem die Kodewörter für a_i und a_j ausgetauscht werden. Dann gilt

$$\begin{aligned} W \cdot (d-1) \cdot (E_d(\mathbf{c}) - E_d(\mathbf{c}')) &= w(a_i)\ell_i + w(a_j)\ell_j - w(a_j)\ell_i - w(a_i)\ell_j \\ &= (w(a_i) - w(a_j))(\ell_i - \ell_j) \ > \ 0 \end{aligned}$$

im Widerspruch zur Optimalität von $\mathbf{c}$. Falls $w(a_i) = w(a_j)$ und $\ell_i > \ell_j$ ist, so können die Kodewörter für a_i und a_j ausgetauscht werden, ohne die mittlere Kodewortlänge zu verändern. Durch endlich viele solcher Austauschschritte erhält man einen optimalen Kode, der die Bedingung (a) erfüllt.

(b) Sei $\mathbf{c}$ ein optimaler Kode, der (a) erfüllt. Falls $\ell_m > \ell_{m-1}$, so kann man die letzten $(\ell_m - \ell_{m-1})$ Komponenten des Kodewortes $c(a_m)$ streichen und erhält einen Kode mit kleinerer mittlerer Kodewortlänge im Widerspruch zur Optimalität von $\mathbf{c}$. Wenn sich die gleichlangen Kodewörter $c(a_m)$ und $c(a_{m-1})$ bereits in den ersten $\ell_m - 1$ Stellen unterschieden, dann erhielte man durch Streichen der letzten Komponenten dieser Kodewörter einen immer noch präfixfreien Kode mit kleinerer mittlerer Kodewortlänge, was wegen der optimalen Kosten von $\mathbf{c}$ nicht sein kann.

□

Das Lemma besagt, daß man sich bei der Suche nach optimalen Kodes auf eine bestimmte Klasse von Kodes beschränken kann, die garantiert einen optimalen Kode enthält. Das nächste Lemma sagt aus, daß ein Schleifendurchlauf des Algorithmus die Optimalität erhält.

Lemma 3.5. *Sei $d > 1$ und $\mathcal{A} = \{a_1, \dots, a_m\}$ das zu kodierende Alphabet mit Gewichten $w(a_1) \geq \cdots \geq w(a_m) > 0$ und $W = \sum_{i=1}^{m} w(a_i)$. Die Zugriffsverteilung sei $\mathbf{p} = (w(a_1)/W, \dots, w(a_m)/W)$. Sei $\mathcal{B} = \{b_1, \dots, b_{m-1}\}$ mit $w(b_{m-1}) = d(w(a_m) + w(a_{m-1}))$ sowie $w(b_j) = w(a_j)$ für $j = 1, \dots, m-2$, also der Inhalt von Q nach einem weiteren Schleifendurchlauf des Algorithmus. Außerdem sei $W' = \sum_{i=1}^{m-1} w(b_i)$, und die Zugriffsverteilung $\mathbf{p}' = (w(b_1)/W', \dots, w(b_{m-1})/W')$.*

Sei $\mathbf{c}'$ ein optimaler präfixfreier Binärkode für $\mathcal{B}$ mit Kodewortlängen $|c'(b_j)| = \ell_j'$ Dann ist $\mathbf{c}$ mit $c(a_j) = c'(b_j)$ für $j = 1, \dots, m-2$ sowie $c(a_{m-1}) = c'(b_{m-1}) \cdot 0$ und $c(a_m) = c'(b_{m-1}) \cdot 1$ und Kodewortlängen $|c(a_j)| = \ell_j$ ein optimaler präfixfreier Kode für $\mathcal{A}$.

Beweis. Für die erwartete Kodewortlänge von $\mathbf{c}$ gilt

$$\begin{aligned}
(d-1)E_d(\mathbf{c},\mathbf{p})+1 &= \sum_{j=1}^{m-2} d^{\ell_j} w(a_j)/W + d^{\ell_{m-1}} w(a_{m-1})/W + d^{\ell_m} w(a_m)/W \\
&= \sum_{j=1}^{m-2} d^{\ell'_j} w(b_j)/W + d^{\ell'_{m-1}+1}(w(b_{m-1})+w(b_m))/W \\
&= \sum_{j=1}^{m-2} d^{\ell'_j} w(b_j)/W + d^{\ell'_{m-1}} w(b_{m-1})/W \\
&= ((d-1)E_d(\mathbf{c}',\mathbf{p}')+1)\cdot W'/W,
\end{aligned}$$

also

$$E_d(\mathbf{c},\mathbf{p}) = E_d(\mathbf{c}',\mathbf{p}')\cdot W/W' + (W'-W)/((d-1)W).$$

Wenn $\mathbf{c}$ kein optimaler Kode für $\mathcal{A}$ wäre, dann existierte ein optimaler Kode $\mathbf{f}$, der die Bedingungen von Lemma 3.4 erfüllt und für den gilt: $E_d(\mathbf{f}) < E_d(\mathbf{c})$. Dann können wir aber gemäß Lemma 3.4 die beiden längsten Kodewörter zusammenfassen und erhalten einen Kode $\mathbf{f}'$ mit Kosten $E_d(\mathbf{f},\mathbf{p}) = E_d(\mathbf{f}',\mathbf{p}')\cdot W/W' + (W'-W)/((d-1)W)$ analog zu obiger Rechnung, da W und W' nur von den Wahrscheinlichkeiten und nicht von den Kodes abhängen. Dann folgt aber $E_d(\mathbf{f}',\mathbf{p}') < E_d(\mathbf{c}',\mathbf{p}')$ im Widerspruch zur Optimalität von $\mathbf{c}'$. □

Durch Iteration des Lemmas und die Tatsache, daß zu Beginn des Verfahrens, d.h. im voll expandierten Baum $W = 1$ ist, folgt der

Satz 3.6. *Für $d \geq 1$ konstruiert der verallgemeinerte Huffman-Algorithmus zu einem gegebenen Alphabet $\mathcal{A}$ und Wahrscheinlichkeitsverteilung $\mathbf{p}$ einen präfixfreien Kode $\mathbf{c}$, der die mittlere Kodewortlänge $E_d(\mathbf{c},\mathbf{p})$ minimiert.* □

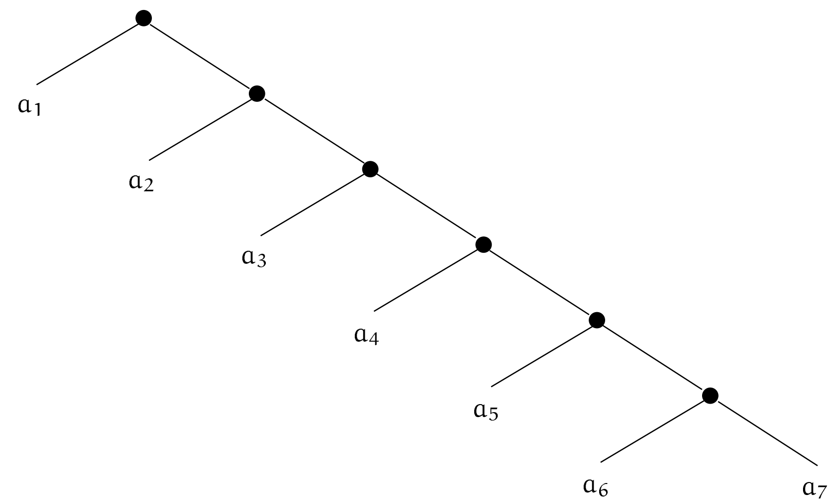
Die Laufzeit des Verfahrens ist in $O(n\log n)$, wenn man ein Registermaschinenmodell mit Einheitskosten für die arithmetischen Operationen annimmt. Zunächst müssen die Wahrscheinlichkeiten sortiert werden. Im weiteren Verlauf wird dann $n-1$ mal ein Gewicht in eine sortierte Liste eingefügt, was mit einer geeigneten Datenstruktur ebenfalls in Zeit $O(n\log n)$ möglich ist.

Wenn die Zugriffswahrscheinlichkeiten bereits sortiert vorliegen, läßt sich der Algorithmus mit einem bekannten Trick auch für eine Laufzeit $O(n)$ implementieren. Dazu beobachtet man, daß die in die Liste einzufügenden Gewichte w_j aufsteigend sortiert sind. Anstatt einer Warteschlange mit Prioritäten kann man zwei FIFO-Schlangen benutzen, eine für die vorsortierten Wahrscheinlichkeiten und eine für die im Laufe des Verfahrens erzeugten Gewichte. Die Funktion *extract_min* läßt sich durch einen Vergleich der vordersten Elemente der beiden Schlangen in konstanter Zeit realisieren.

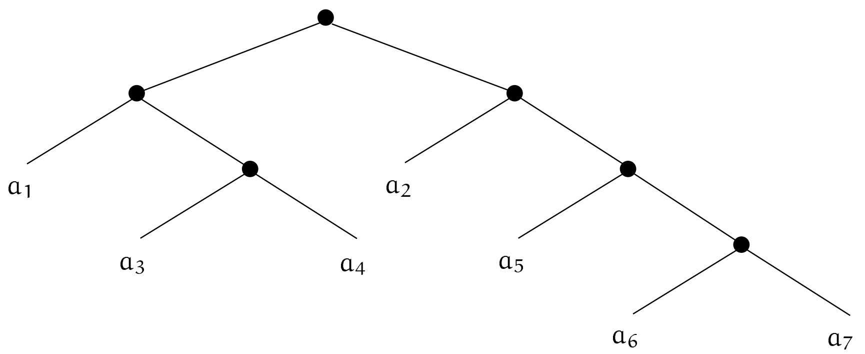
Bei Kodierung in ein M-näres Alphabet $\mathcal{S} = \{0,1,\ldots,M-1\}$ läßt sich das Verfahren entsprechend erweitern. Man muß nur darauf achten, daß die Wurzel des Kodebaums tatsächlich M Nachfolger besitzt. Dies ist garantiert, wenn für die Kardinalität n des Symbolalphabetes $\mathcal{A}$ gilt, daß $n = M + k(M-1)$ für ein $k \in \mathbb{N}_0$. Ansonsten kann man $\mathcal{A}$ durch weitere künstliche Elemente mit Gewicht 0 ergänzen, um diese Bedingung sicherzustellen, und dann obigen Algorithmus anwenden. Zur Verallgemeinerung des Huffman-Verfahrens auf M-näre Alphabete siehe auch [Mc77, Seiten 245*ff.*]. Als nächstes illustrieren wir das Verfahren und die Schranken für die optimale mittlere Kodewortlänge von Korollar 3.3.

Beispiel 3.1. Für die Verteilung $\mathbf{p} = (1/2, 1/4, 1/8, 1/16, 1/32, 1/64, 1/64)$ über dem Alphabet $\mathcal{A} = \{a_1, a_2, a_3, a_4, a_5, a_6, a_7\}$ geben wir optimale Bäume und deren erwartete Kosten bei verschiedenen d an.

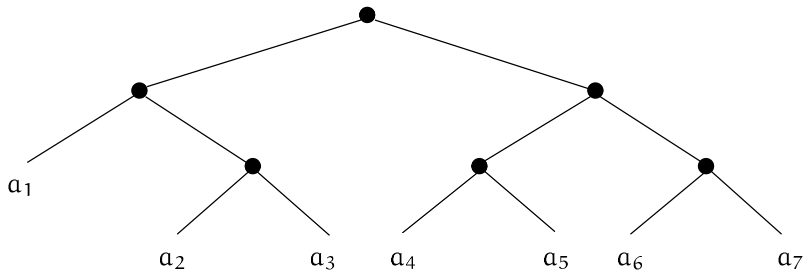
$$d = 1, E = 126/64 :$$



$$d = 2, E = 17/4 :$$



$d = 4, E = 13:$



□

Aus [C65] folgt, daß optimale Kodewortlängen ℓ_j durch

$$-\log\left(\frac{p_j^\alpha}{\sum_{i=1}^n p_i^\alpha}\right) \le \ell_j < -\log\left(\frac{p_j^\alpha}{\sum_{i=1}^n p_i^\alpha}\right) + 1$$

gegeben sind, wobei $\alpha = 1/(1 + \log d)$ gilt. Auf diese Weise könnte man einen optimalen Baum erhalten, indem man erst die Zugriffswahrscheinlichkeiten transformiert und dann den Standard-Huffman-Algorithmus auf der transformierten Eingabe startet. Dieses Verfahren erfordert aber die Auswertung der transzendentalen Funktion $\log d$, während der verallgemeinerte Huffman-Algorithmus mit Additionen und Multiplikationen auskommt.

Die maximale Kodewortlänge wird also durch $p_{min} = \min\{p_j : j = 1, \ldots, n\}$ definiert. Um Bäume mit vorgegebener beschränkter Höhe zu erhalten, kann man daher mit Hilfe obiger Abschätzung numerisch einen Wert für d berechnen, der eine maximale Tiefe des optimalen Baums im Kostenmaß $E_d(\mathbf{c})$ garantiert.

Wir merken an, daß der sogenannte Package-Merge-Algorithmus [LH90] einen im Standard-Kostenmaß optimalen präfixfreien Kode mit maximaler Kodewortlänge L in Zeit $O(nL)$ findet. Da wir eine andere Kostenfunktion verwenden, sind die Ergebnisse nicht vergleichbar.

3.2. Suchbäume

In diesem Abschnitt betrachten wir Suchbäume über einem angeordneten Alphabet. Die Bäume sind knotenorientiert, d.h. die Elemente der Menge stehen in den inneren Knoten des Baums, und die Tiefe eines Elementes ist die Länge des Pfades von der Wurzel zu diesem Element. Insbesondere ist die Tiefe der Wurzel gleich 0. Im Standardmodell werden für einen Zugriff auf a_j in Tiefe t_j die Kosten $t_j + 1$ berechnet, denn dies ist gerade die Anzahl der Elemente, mit denen das angefragte Symbol verglichen werden muß. Für unser erweitertes Kostenmaß mit Faktor $d \ge 1$ wird ein solcher Zugriff mit $\sum_{i=0}^{t_j} d^i$ bewertet. Eine erfolglose Suche endet in einem Blatt. Wenn das Blatt, welches das Intervall (a_j, a_{j+1}) repräsentiert, in Tiefe s_j liegt, dann entstehen Kosten von $\sum_{i=0}^{s_j-1} d^i$. Hierbei ist $a_0 := -\infty$ und $a_{n+1} := \infty$.

Definition 3.3. Sei $\mathcal{A} = \{a_1 < \cdots < a_n\}$ eine geordnete Menge und

$$\mathbf{p} = (q_0, p_1, q_1, p_2, \ldots, q_{n-1}, p_n, q_n)$$

eine zugehörige Zugriffsverteilung. Dabei beschreibt p_j die Wahrscheinlichkeit von Element a_j und q_j die Wahrscheinlichkeit von Intervall (a_j, a_{j+1}). Weiter bezeichne $\mathbf{T} = (s_0, t_1, s_1, \ldots, t_n, s_n)$ die Tiefen der Knoten und Blätter in einem binären Suchbaum für $\mathcal{A}$. Für $d \geq 1$ sind die erwarteten Zugriffskosten (mittlere Suchkosten) dann

$$E_d(\mathbf{T}, \mathbf{p}) = \sum_{i=1}^{n} p_i \cdot (1 + d + d^2 + \cdots + d^{t_i}) + \sum_{i=0}^{n} q_i \cdot (1 + d + d^2 + \cdots + d^{s_i - 1}).$$

□

Wenn die Verteilung $\mathbf{p}$ klar ist, schreiben wir auch kürzer $E_d(\mathbf{T})$. Sei $P = \sum_{i=1}^{n} p_i$ die Wahrscheinlichkeit einer erfolgreichen Suche und $Q = \sum_{i=0}^{n} q_i$ die Wahrscheinlichkeit einer erfolglosen Suche.

Anwendungen

Wenn $d > 1$ ist, modelliert das erweiterte Kostenmodell die Situation, daß die Suchkosten mit jedem Suchschritt exponentiell wachsen. Dadurch werden lange Suchpfade sehr teuer und billige Bäume sind gut balanciert.

Eine mögliche Anwendung liegt bei der Suche in hierarchischen Strukturen. Wenn der Aufwand für einen Test oder Vergleich auf jeder Hierarchiestufe um einen Faktor d zunimmt, so beschreibt unser Modell gerade die erwarteten Gesamtkosten. Bei bekannten Wahrscheinlichkeiten ergibt sich die Möglichkeit, die Testreihenfolge bzw. die hierarchische Zerlegung des Problems zu optimieren.

Als eine andere Anwendung ist die Herleitung allgemeiner unterer Schranken für die Laufzeit von Programmen mit Multiplikationen denkbar. Bei jeder Multiplikation wird die Länge der Operanden höchstens verdoppelt. Die Kosten (über alle Ausführungspfade des Programms gemittelt) lassen sich dadurch in unserem Modell mit $d = 2$ beschreiben.

3.2.1. Alphabetische Kodes

Ein Kode $\mathbf{c}$ für eine geordnete Menge $\mathcal{A} = \{a_1 < \cdots < a_n\}$ über einem angeordneten Kodealphabet $\mathcal{S} = \{0 < 1\}$ heißt *ordnungserhaltend* oder *alphabetisch*, wenn die Kodewörter die Ordnung auf $\mathcal{A}$ respektieren, d.h. $c(a_1) < \cdots < c(a_n)$, wobei „<“ in diesem Fall die lexikographische Ordnung auf $\mathcal{S}^*$ bedeutet.

Eine hinreichende Forderung für die Existenz eines ordnungserhaltenden Kodes mit vorgegebenen Kodewortlängen $\ell_1, \ldots, \ell_n$ ist die Kraft'sche Ungleichung

$$\sum_{j=1}^{n} 2^{-\ell_j} \leq \frac{1}{2},$$

siehe [H97]. Durch eine Modifikation der Beweise von Satz 3.2 und Korollar 3.3 folgt

Korollar 3.7. Sei $\mathcal{A} = \{a_1 < \cdots < a_n\}$, $d > 1$ und $\mathbf{q}$ eine Verteilung auf $\mathcal{A}$. Dann gibt es einen ordnungserhaltenden Kode $\mathbf{c}$ mit mittlerer Kodewortlänge

$$E_d(\mathbf{c}, \mathbf{q}) \leq \frac{d^2 L_{1/(1+\log d)}(\mathbf{q}) - 1}{d-1} .$$

□

Die untere Schranke aus Korollar 3.3 bleibt gültig, da sie für alle eindeutig dekodierbaren Kodes gilt, insbesondere auch für ordnungserhaltende Kodes. Dadurch erhält man eine Abschätzung für die Suchkosten bei erfolgloser Suche, die in einem Blatt endet, d.h. bei einer Wahrscheinlichkeitsverteilung $\mathbf{p}$ mit $P = 0$ und $Q = 1$. Dann folgt aus Korollar 3.7 die Existenz eines Suchbaums mit

$$E_d(\mathbf{T}, \mathbf{p}) \leq \frac{d^2 L_{1/(1+\log d)}(\mathbf{p}) - 1}{d-1} .$$

Wenn wir erfolgreiche und erfolglose Suchen gleichzeitig betrachten, so finden wir die Kosten

$$E_d(\mathbf{T}, \mathbf{p}) \leq \frac{d L_{1/(1+\log d)}(\mathbf{p}) \cdot \frac{P^{(\alpha)} + dQ^{(\alpha)}}{P^{(\alpha)} + Q^{(\alpha)}} - 1}{d-1} ,$$

dabei ist $P^{(\alpha)} = \sum_{i=1}^{n} p_i^{\alpha}$ und $Q^{(\alpha)} = \sum_{i=0}^{n} q_i^{\alpha}$. Zum Beweis betrachten wir eine Konstruktion des Suchbaums analog zum Shannon-Fano-Kode, vgl. [K98, Seite 445]. Seien $f_0 = \frac{1}{2} q_0$, $f_1 = q_0 + p_1 + \frac{1}{2} q_1$, usw. die Partialsummen des Wahrscheinlichkeitsvektors $\mathbf{p}$, jeweils bis zur Mitte der Intervalle (a_j, a_{j+1}) summiert. Der gewünschte Kode entsteht, indem man die f_i in Binärdarstellung schreibt und die Nachkommastellen als Kodewort interpretiert. Dabei werden die Längen der Kodeworte soweit minimiert, daß der entstehende Kode noch präfixfrei ist. Dann kann man zeigen, daß für die Tiefen t_i der inneren Knoten und die Tiefen s_i der Blätter gilt:

$$p_i \leq 2^{-t_i}, \qquad q_i \leq 2^{-s_i + 2} ,$$

woraus im Standardfall $d = 1$ die bekannte Abschätzung $E \leq H(\mathbf{p}) + 1 + Q$ folgt.

Für unser verallgemeinertes Kostenmaß betrachten wir die transformierten Wahrscheinlichkeiten

$$\overline{p}_i = \frac{p_i^{\alpha}}{P^{(\alpha)} + Q^{(\alpha)}} , \qquad \overline{q}_i = \frac{q_i^{\alpha}}{P^{(\alpha)} + Q^{(\alpha)}} ,$$

hierbei ist wieder $\alpha = 1/(1 + \log d)$ sowie $P^{(\alpha)} = \sum_{i=1}^{n} p_i^{\alpha}$ und $Q^{(\alpha)} = \sum_{i=0}^{n} q_i^{\alpha}$. Dann finden wir

$$\begin{aligned}
(d-1)E_d(\mathbf{T})+1 &= \sum_{i=1}^{n} p_i d^{t_i+1} + \sum_{i=0}^{n} q_i d^{s_i} \\
&\leq d \cdot \sum_{i=1}^{n} p_i d^{\log(1/\overline{p}_i)} + \sum_{i=0}^{n} q_i d^{\log(1/\overline{q}_i)+2} \\
&= d \cdot \sum_{i=1}^{n} p_i (1/\overline{p}_i)^{\log d} + d^2 \cdot \sum_{i=0}^{n} q_i (1/\overline{q}_i)^{\log d} \\
&= d \cdot \sum_{i=1}^{n} p_i \left(\frac{P^{(\alpha)}+Q^{(\alpha)}}{p_i^{\alpha}}\right)^{\log d} + d^2 \cdot \sum_{i=0}^{n} q_i \left(\frac{P^{(\alpha)}+Q^{(\alpha)}}{q_i^{\alpha}}\right)^{\log d} \\
&= d \cdot \sum_{i=1}^{n} p_i \left(P^{(\alpha)}+Q^{(\alpha)}\right)^{\log d} p_i^{\alpha-1} + d^2 \cdot \sum_{i=0}^{n} q_i \left(P^{(\alpha)}+Q^{(\alpha)}\right)^{\log d} q_i^{\alpha-1} \\
&= d \cdot \left(P^{(\alpha)}+Q^{(\alpha)}\right)^{\log d} \cdot (P^{(\alpha)}+dQ^{(\alpha)}) \\
&= d \cdot \left(P^{(\alpha)}+Q^{(\alpha)}\right)^{1+\log d} \cdot \frac{P^{(\alpha)}+dQ^{(\alpha)}}{P^{(\alpha)}+Q^{(\alpha)}} \\
&= d \cdot L_{1/(1+\log d)}(\mathbf{p}) \cdot \frac{P^{(\alpha)}+dQ^{(\alpha)}}{P^{(\alpha)}+Q^{(\alpha)}}
\end{aligned}$$

Dabei haben wir die Beziehung $\alpha \log d = 1 - \alpha$ benutzt.

3.2.2. Eine untere Schranke für die mittlere Suchzeit

Einen binären Suchbaum für eine Menge $\mathcal{A}$ kann man als Kode über einem dreielementigen Alphabet $\mathcal{S} = \{0, 1, \mathtt{Stop}\}$ auffassen. Bei der Suche wird in jedem Knoten entschieden, ob das gesuchte Element gefunden wurde oder kleiner bzw. größer als der Knoteneintrag ist. Dementsprechend wird die Suche beendet oder im linken bzw. rechten Unterbaum des Knotens fortgesetzt.

Die Entropie der Quelle bezüglich eines 3-elementigen Kodealphabets, $H_\alpha(\mathbf{p})/\log(3)$, stellt somit eine untere Schranke für die mittlere Länge eines den Suchbaum repräsentierenden Kodes dar. Für die mittleren Kosten eines Baums folgt daraus die untere Schranke

$$(d-1) \cdot E_d(\mathbf{T}, \mathbf{p}) + 1 \geq d^{\frac{H_\alpha(\mathbf{p})}{\log 3}} = \left(L_{1/(1+\log d)}(\mathbf{p})\right)^{1/\log 3} .$$

Lemma 3.8. *Die erwarteten Kosten $E_d(\mathbf{T}, \mathbf{p})$ eines Suchbaums betragen mindestens*

$$E_d(\mathbf{T}, \mathbf{p}) \geq \frac{(L_{1/(1+\log d)}(\mathbf{p}))^{1/\log 3} - 1}{d-1} .$$

Im Grenzfall $d \to 1$ folgt hieraus die untere Schranke $H(\mathbf{p})/\log 3$ für die Tiefe von Suchbäumen im Standardkostenmaß. Eine interessante offene Frage ist es, analog zu [B75] diese Schranke für $d > 1$ zu verbessern. Die bekannte Abschätzung

$$-\sum_{i=1}^{n} p_i \log(p_i) \;\leq\; -\sum_{i=1}^{n} p_i \log(q_i)$$

für beliebige Verteilungen $\mathbf{p}$ und $\mathbf{q}$ läßt sich jedoch nicht ohne weiteres auf die Rényi-Entropie bzw. auf $L_{1/(1+\log d)}(\mathbf{p})$ übertragen, vgl. [N75] für Ansätze in dieser Richtung.

3.2.3. Optimale Suchbäume

Optimale Suchbäume für beliebige $d \geq 1$ erhält man mit Hilfe dynamischer Programmierung [K98, M88]. Die Korrektheit des folgenden Lemmas folgt unmittelbar aus dem Paradigma der dynamischen Programmierung, daß eine optimale Lösung aus optimalen Teillösungen zusammengesetzt sein muß.

Lemma 3.9. *Sei* $d \geq 1$. *Für* $1 \leq i \leq j \leq n$ *bezeichne* $w_{i,j} = q_{i-1}+p_i+q_i+p_{i+1}+\cdots+p_j+q_j$ *die Wahrscheinlichkeit der Elemente* $\mathcal{A}_{ij} = \{a_i, \ldots, a_j\}$ *und der benachbarten Blätter und* $E_d(i,j)$ *die mittleren Suchkosten in einem optimalen Baum für* $\mathcal{A}_{ij}$.

$$\begin{aligned} E_d(i,i) &= 1 \\ E_d(i,j) &= 1 + d \cdot \min_{i \leq m \leq j} \Big(w_{i,m-1} E_d(i,m-1) + w_{m+1,j} E_d(m+1,j) \Big) / w_{i,j} \qquad \text{für } i < j. \end{aligned}$$

Wie im Fall $d = 1$ kann die Suche nach einem minimierenden Wurzelknoten a_m eingeschränkt werden, so daß das Verfahren in Zeit $O(n^2)$ läuft. Denn es genügt, daß die Gewichte $w_{i,j}$ die sogenannte Vierecksungleichung erfüllen, siehe [M88, Seite 151].

$$w_{i,j} + w_{i',j'} \;\leq\; w_{i',j} + w_{i,j'} \qquad \text{für alle } 1 \leq i \leq i' \leq j \leq j' \leq n.$$

Somit ist die zur Laufzeitverbesserung nötige Konstruktion unabhängig von dem Vorfaktor d. Wir fassen zusammen:

Satz 3.10. *Für* $d \geq 1$ *kann ein optimaler Baum* $\mathbf{T}_{\text{opt}}$ *für* $\mathbf{p} = (q_0, p_1, \ldots, p_n, q_n)$ *in Zeit* $O(n^2)$ *konstruiert werden. Im Fall* $d > 1$ *gilt*

$$\frac{(L_{1/(1+\log d)}(\mathbf{p}))^{1/\log 3} - 1}{d-1} \;\leq\; E_d(\mathbf{T}_{\text{opt}}, \mathbf{p}) \;\leq\; \frac{d L_{1/(1+\log d)}(\mathbf{p}) \cdot \frac{P^{(\alpha)}+dQ^{(\alpha)}}{P^{(\alpha)}+Q^{(\alpha)}} - 1}{d-1}.$$

Im Grenzwert $d \to 1$ *erhält man daraus das bekannte Resultat*

$$\frac{H(\mathbf{p})}{\log 3} \;\leq\; E(\mathbf{T}_{\text{opt}}, \mathbf{p}) \;\leq\; H(\mathbf{p}) + 1 + Q.$$

Dabei ist $Q = \sum_{i=0}^{n} q_i$ *die Wahrscheinlichkeit einer erfolglosen Suche.* □

Wenn wir speziell die Gleichverteilung und nur erfolgreiche Suchen betrachten, d.h. $p_i = 1/n$ für $i = 1, \ldots, n$, so folgt

$$E(\mathbf{T}_{\text{opt}}, \mathbf{p}) \;\leq\; \frac{d\, n^{\log d} - 1}{d - 1}\;.$$

Im Grenzfall $d \to 1$ erhält man das bekannte Ergebnis $E \leq 1 + \log n$ und für $d = 2$ wachsen die erwarteten Suchkosten linear in n.

Es wäre interessant, das Verhalten der MOVE-TO-ROOT-Regel [AM78] für selbstorganisierende Bäume im Fall $d > 1$ zu analysieren. Da die Funktion $t \mapsto d^t$ für $d > 1$ jedoch konkav ist, kann man den Erwartungswert nicht ohne weiteres nach oben abschätzen. Die Ungleichung von Jensen würde nur eine untere Schranke liefern: wenn X die Anzahl der Knoten auf dem Pfad beschreibt, so interessieren wir uns für $E[d^X] \geq d^{E[X]}$. Zu einer Berechnung von $E[d^X]$ sind daher genauere Informationen über die Verteilung von X nötig als lediglich der Erwartungswert.

3.3. Zufällig gewachsene Bäume

In diesem Abschnitt untersuchen wir die mittleren Kosten von zufälligen Bäumen in unserem verallgemeinerten Kostenmaß. Sei $\mathcal{A} = \{a_1 < \cdots < a_n\}$ die Menge der Baumelemente. Der zu einer Permutation $\sigma = (a_{\sigma(1)}, \ldots, a_{\sigma(n)})$ gehörende Baum $T(\sigma)$ entsteht durch sukzessives Einfügen der $a_{\sigma(i)}$ in einen anfangs leeren Baum. Das erste Element $a_{\sigma(1)}$ bildet die Wurzel und die folgenden Elemente werden gemäß der Ordnung auf $\mathcal{A}$ in den linken oder rechten Teilbaum eingefügt.

Ein Baum für $\mathcal{A}$ heißt *zufällig*, wenn alle $n!$ Permutationen σ gleich wahrscheinlich sind. Insbesondere folgt daraus, daß jedes der a_i mit gleicher Wahrscheinlichkeit $1/n$ an der Wurzel des Baums steht. Über die Untersuchung derartiger zufälliger Bäume existiert eine reichhaltige Literatur, z.B. [K98, M92, SF96, Sz98]. Das Konzept der zufällig gewachsenen Bäume kann man auch für eine effiziente Randomisierung einsetzen [MR98].

Sei $t_j(n)$ die Tiefe des j-ten Elementes in einem Baum mit n Knoten. Wenn der Baum zufällig gewachsen ist, so sind die $t_j(n)$ Zufallsvariablen. Wir untersuchen die mittlere Tiefe und die externe Pfadlänge von zufälligen Bäumen in unserem verallgemeinerten Kostenmaß. Die externe Pfadlänge $L_n = \sum_{j=1}^{n} t_j(n)$ ist die Summe über die Längen der Pfade von der Wurzel zu allen Knoten. Die mittlere Tiefe $T_n = \sum_{j=1}^{n} t_j(n)/n$ ist die Länge des Pfades von der Wurzel zu einem zufällig ausgewählten Knoten. Somit gilt $L_n = n \cdot T_n$. In unserem verallgemeinerten Kostenmaß sprechen wir analog von den mittleren Kosten T_n und den externen Pfadkosten L_n.

Ein leerer Baum hat definitionsgemäß $L_0 = T_0 = 0$. Ein einziger Knoten steht an der Wurzel und hat Kosten $L_1 = T_1 = 0$. Wenn der Baum n Knoten besitzt, so gilt für die externen Pfadkosten

$$L_n = n - 1 + d(L_k + L_{n-1-k}),$$

wobei $k = 0, \dots, n-1$ zufällig gewählt wird und die Größe des linken Unterbaums angibt. Wenn wir zu erzeugenden Funktionen übergehen, folgt

$$\begin{aligned} L_n(z) &= \sum_{j=0}^{\infty} \mathrm{Prob}(L_n = j) z^j \;=\; E[z^{L_n}] \;=\; E[z^{n-1+d(L_k+L_{n-1-k})}] \\ &= z^{n-1} \cdot \frac{1}{n} \cdot \sum_{k=0}^{n-1} L_k(z^d) \cdot L_{n-1-k}(z^d) \;. \end{aligned} \tag{3.2}$$

Für $d = 1$ wurden analoge Rechnungen u.a. in [SF96] ausgeführt. Eine kompakte Darstellung der erzeugenden Funktion $L_n(z)$ ist mit Hilfe der bivariaten Funktion

$$L(z,u) = \sum_{n=0}^{\infty} L_n(z) u^n$$

möglich, aus (3.2) erhält man

$$\frac{\partial}{\partial u} L(z,u) \;=\; u^2 \cdot L(z^d, zu)^2 \;.$$

Diese Darstellung kann für asymptotische Analysen ausgenutzt werden, [JS99, SF96], jedoch vereinfachen sich dadurch die nachfolgenden Rechnungen nicht. Wir benutzen nun (vgl. [GKP94]), daß $E[L_n] = L_n'(1)$ und $Var[L_n] = L_n''(1) + L_n'(1) - (L_n'(1))^2$. Zunächst folgt für die Ableitungen

$$\begin{aligned} L_n'(z) = {} & \frac{(n-1)z^{n-2}}{n} \cdot \sum_{k=0}^{n-1} L_k(z^d) L_{n-1-k}(z^d) \\ & + \frac{dz^{n+d-2}}{n} \cdot \sum_{k=0}^{n-1} \Big(L_k'(z^d) L_{n-1-k}(z^d) + L_k(z^d) L_{n-1-k}'(z^d) \Big) \\ L_n''(z) = {} & \frac{(n-1)(n-2)z^{n-3}}{n} \cdot \sum_{k=0}^{n-1} L_k(z^d) L_{n-1-k}(z^d) \\ & + \frac{d(2n+d-3)z^{n+d-3}}{n} \cdot \sum_{k=0}^{n-1} \Big(L_k'(z^d) L_{n-1-k}(z^d) + L_k(z^d) L_{n-1-k}'(z^d) \Big) \\ & + \frac{d^2 z^{n+2d-3}}{n} \cdot \sum_{k=0}^{n-1} \Big(L_k''(z^d) L_{n-1-k}(z^d) + 2L_k'(z^d) L_{n-1-k}'(z^d) + L_k(z^d) L_{n-1-k}''(z^d) \Big) \;. \end{aligned}$$

Sei $\ell_n = L_n'(1)$ und $s_n = L_n''(1)$. Mit $L_n(1) = 1$ erhalten wir die Rekursionsgleichungen

$$\begin{aligned}
\ell_0 &= 0 \\
\ell_n &= n-1+\frac{d}{n}\cdot\sum_{k=0}^{n-1}(\ell_k+\ell_{n-1-k}) \\
s_0 &= 0 \\
s_n &= (n-1)(n-2)+\frac{d(2n+d-3)}{n}\sum_{k=0}^{n-1}(\ell_k+\ell_{n-1-k}) \\
&\quad +\frac{d^2}{n}\sum_{k=0}^{n-1}(s_k+2\ell_k\ell_{n-1-k}+s_{n-1-k})\ .
\end{aligned}$$

Für ℓ_n folgt durch Multiplikation mit n

$$n\ell_n = n(n-1)+2d\sum_{k=0}^{n-1}\ell_k.$$

Wir benutzen die erzeugende Funktion $F(z) = \sum_{n=0}^{\infty}\ell_n z^n$ und erhalten

$$zF'(z) = \frac{2z^2}{(1-z)^3}+\frac{2dzF(z)}{1-z}\ .$$

Die homogene Differentialgleichung mit getrennten Variablen

$$G'(z) = \frac{2d}{1-z}\cdot G(z)$$

besitzt die Lösung

$$G(z) = (1-z)^{-2d}.$$

Mit dem Ansatz der Variation der Konstanten nach *Lagrange* (vgl. [SF96, Seite 99*f.*], [Z96, Seite 456]) finden wir

$$\begin{aligned}
(F(z)/G(z))' &= \frac{F'(z)G(z)-F(z)G'(z)}{G(z)^2} &= (1-z)^{2d}\left(F'(z)-\frac{2dF(z)}{1-z}\right) \\
&& = \frac{2z}{(1-z)^{3-2d}}\ ,
\end{aligned}$$

und Integration liefert schließlich

$$F(z) \ = \ G(z)\cdot\int_0^z\frac{2x}{(1-x)^{3-2d}}\,dx \ = \ \frac{2}{(1-z)^{2d}}\cdot\int_0^z\frac{x}{(1-x)^{3-2d}}\,dx,$$

dabei ist die untere Integrationsgrenze 0 so gewählt, daß $\ell_0 = 0$ ist. Mit partieller Integration finden wir

$$\int_0^z x(1-x)^\alpha\,dx \ = \ \frac{1-(1-z)^{\alpha+1}(1+(1+\alpha)z)}{(\alpha+1)(\alpha+2)}\ ,$$

und mit $\alpha = 2d - 3$ folgt dann für $d > 1$

$$\begin{aligned}
F(z) &= \frac{2}{(1-z)^{2d}} \cdot \frac{1-(1-z)^{2d-2}(1+2(d-1)z)}{(2d-1)(2d-2)} \\
&= \frac{2}{(2d-1)(2d-2)} \cdot \left(\frac{1}{(1-z)^{2d}} - \frac{1}{(1-z)^2} - \frac{2(d-1)z}{(1-z)^2} \right) \\
&= \frac{1}{(d-1)(2d-1)} \cdot \left(\sum_{n=0}^{\infty} \binom{2d-1+n}{n} z^n \right. \\
&\qquad \left. - \sum_{n=0}^{\infty} (n+1) z^n - 2(d-1) \sum_{n=0}^{\infty} (n+1) z^{n+1} \right) .
\end{aligned}$$

Also gilt für die Koeffizienten von $F(z)$

$$\ell_n = [z^n]F(z) = \frac{\binom{2d-1+n}{n} - (2d-1)n - 1}{(d-1)(2d-1)} ,$$

und mit der asymptotischen Abschätzung für $n \to \infty$

$$\binom{2d-1+n}{n} = \frac{(2d-1+n) \cdot (2d-1+n-1) \cdots (2d)}{n!} = \frac{n^{2d-1}}{\Gamma(2d)} + O(n^{2d-2})$$

folgt

Satz 3.11. *Für die erwarteten externen Pfadkosten eines zufällig gewachsenen Baums gilt im verallgemeinerten Kostenmaß für $d > 1$, daß*

$$\begin{aligned}
E[L_n] &= \frac{\binom{2d-1+n}{n} - (2d-1)n - 1}{(d-1)(2d-1)} \\
&= \frac{n^{2d-1}}{(d-1)(2d-1)\Gamma(2d)} + O\left(n^{\max\{1,2d-2\}}\right) .
\end{aligned}$$

□

Für das Standardmodell $d = 1$ ist bekannt [SF96], daß

$$\begin{aligned}
\ell_n &= [z^n]\frac{-2}{(1-z)^2} \cdot \left(z + \ln(1-z)\right) \\
&= 2(n+1)H_{n+1} - 4n - 2 \ = \ 2n\ln(n) + O(n) .
\end{aligned}$$

Wegen $\Gamma(n+1) = n!$ für $n \in \mathbb{N}$ finden wir speziell für $d = 3/2$

$$\ell_n = [z^n]\frac{z^2}{(1-z)^3} = \binom{n}{2} = n^2/2 + O(n) ,$$

und für $d = 2$ ergibt sich

$$\ell_n = [z^n]\frac{z^2(3-2z)}{3(1-z)^4} = \binom{n+1}{3} - \frac{2}{3}\binom{n}{3} = n^3/18 + O(n^2) .$$

Es ist interessant, dieses Ergebnis mit den Suchkosten in optimalen Bäumen zu vergleichen. Aus Satz 3.11 folgen erwartete Suchkosten von $O(n^{2d-2})$, wenn man von gleichverteilten Zugriffen ausgeht. Ein optimaler Baum für n Elemente bei Gleichverteilung erzeugt dagegen nur Suchkosten von $O(n^{\log d})$, vgl. die Bemerkung nach Satz 3.10. Für das Standardmodell $d = 1$ ist bekannt, daß die Suchkosten im zufälligen Baum genauso wie im optimalen Baum $O(\log n)$ betragen, so daß der zufällige Baum nicht schlechter ist als ein optimaler Baum. In unserem verallgemeinerten Kostenmaß erkennen wir dagegen, daß die zufälligen Bäume mit wachsendem d deutlich schlechter werden. Daraus folgern wir, daß bei $d > 1$ die explizite Konstruktion eines optimalen Baums einen sinnvollen Aufwand darstellt.

Für die Varianz von L_n rechnen wir weiter. Aus der obigen Rekursionsgleichung für $s_n = L_n''(1)$ erhalten wir analog zu ℓ_n die Beziehung

$$ns_n = a_n + 2d^2 \sum_{k=0}^{n-1} s_k,$$

die mit Hilfe der erzeugenden Funktion $S(z) = \sum_{n=0}^{\infty} s_n z^n$ gelöst wird; wir erhalten

$$S(z) = \frac{1}{(1-z)^{2d^2}} \cdot \int_0^z \frac{A(x)(1-x)^{2d^2}}{x}\, dx\ .$$

Dabei ist $A(z) = \sum_{n=0}^{\infty} a_n z^n$ gegeben durch

$$\begin{aligned} a_n &= n(n-1)(n-2) + d(2n+d-3)\sum_{k=0}^{n-1}(\ell_k + \ell_{n-1-k}) + 2d^2 \sum_{k=0}^{n-1} \ell_k \ell_{n-1-k} \\ &= n(n-1)(n-2) + (2n+d-3)n(\ell_n - n + 1) + 2d^2 \sum_{k=0}^{n-1} \ell_k \ell_{n-1-k} \\ &= n(n-1)(n-2) + 2n(n-1)\ell_n + (d-1)n\ell_n \\ &\quad - n(n-1)(2(n-2)+d+1) + 2d^2 \sum_{k=0}^{n-1} \ell_k \ell_{n-1-k}\ , \end{aligned}$$

woraus wir ableiten, daß

$$A(z) \;=\; \frac{6z^3}{(1-z)^4} + 2z^2F''(z) + (d-1)zF'(z) - \frac{12z^3}{(1-z)^4} - \frac{2(d+1)z^2}{(1-z)^3} + 2d^2 zF(z)^2\ .$$

Damit kann obiges Integral berechnet und können die Koeffizienten $s_n = [z^n]S(z)$ extrahiert werden. Aus $Var[L_n] = s_n + \ell_n - \ell_n^2$ folgt die gesuchte Varianz der mittleren externen Pfadkosten von zufällig gewachsenen Bäumen. Im Fall $d = 1$ ist bekannt [K98], daß

$$Var[L_n] = 7n^2 - 4(n+1)H_n^{(2)} - 2(n+1)H_n + 13n = (7 - 2\pi^2/3)n^2 + O(n \ln n)\ .$$

Mit Hilfe von MAPLE finden wir für $d = 2$:

$$\mathrm{Var}[L_n] = \frac{5n^7}{21168} + \frac{13n^6}{5670} + \frac{11n^5}{840} + \frac{11n^4}{162} + \frac{137n^3}{1008} - \frac{67n^2}{405} - \frac{1429n}{26460},$$

und allgemein für das asymptotische Wachstum von $\mathrm{Var}[L_n] = \Theta(n^m)$:

d	1	2	3	4	5	6	7	8	9	10
m	2	7	17	31	49	71	97	127	161	199

.

Man erkennt, daß das Wachstum der Varianz und damit auch der Standardabweichung $\sqrt{\mathrm{Var}[L_n]}$ für größere d asymptotisch wesentlich stärker wächst als der Erwartungswert $E[L_n]$. Das bedeutet, daß die Verteilung stark streut und nicht besonders deutlich am Erwartungswert konzentriert ist. Intuitiv ist dies einleuchtend, da bereits eine kleine Abweichung von einem optimal balancierten Baum durch das exponentielle Wachstum der Kosten starke Auswirkungen nach sich zieht.

Dies ist ein weiteres Argument daür, daß bei $d > 1$ im verallgemeinerten Kostenmaß die explizite Konstruktion eines optimalen Baums ein sinnvoller Mehraufwand sein kann.

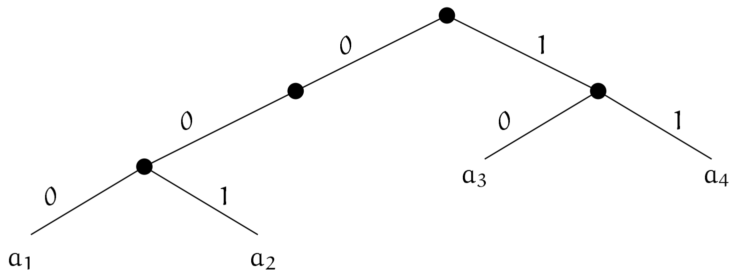
3.4. Tries

In diesem Kapitel skizzieren wir, wie sich unser verallgemeinertes Kostenmaß auf die Analyse von digitalen Suchbäumen [CFV98, K98, M92, SF96] auswirkt. Der Name *Trie* wurde in Anlehnung an *Tree* aus *Information Re**trie**val* abgeleitet [F60]. Bei dieser Speicherungsart wird vorausgesetzt, daß die Daten durch (potentiell unendlich lange) Binärworte gegeben sind. In Tiefe k des Baums wird entsprechend des k-ten Bits des Schlüssels verzweigt. Es werden also nicht zwei gesamte Schlüssel verglichen, sondern nur korrespondierende Stellen innerhalb der Schlüssel. Sobald ein Präfix des Schlüssels hinreichend lang ist, so daß er nicht mehr als Präfix eines anderen Schlüssels erscheint, wird der Eintrag an diese Stelle des Baums gesetzt. Dies setzt voraus, daß alle Schlüssel bekannt sind, wenn der Baum aufgebaut wird.

Sei eine Menge $\mathcal{A} = \{a_1, \ldots, a_n\}$ von Schlüsseln gegeben. Dabei ist $a_j = a_j(1), a_j(2), a_j(3), \ldots$ eine Folge über $\mathcal{S} = \{0, 1\}$. Ein *digitaler Suchbaum* oder *Trie* ist folgendermaßen rekursiv definiert. Für $|\mathcal{A}| = 0$ ist der Trie leer und für $|\mathcal{A}| = 1$ besteht er aus einem einzigen Knoten. Falls $|\mathcal{A}| > 1$, so wird das Alphabet gemäß dem ersten Symbol der Elemente in zwei Teilmengen $\mathcal{A}_0$ und $\mathcal{A}_1$ partitioniert, also $\mathcal{A}_i = \{a \in \mathcal{A} : a = i, a(2), a(3), \ldots\}$ für $i \in \mathcal{S}$. Die Wurzel erhält die Kanten mit Beschriftung 0 bzw. 1 Die Verallgemeinerung auf M-näre Grundalphabete ist naheliegend.

Eine etwas effizientere Variante der Tries sind die Patricia-Tries [M68]. Dabei wird zunächst ein Trie aufgebaut und dann alle Knoten mit Verzweigungsgrad 1 gelöscht. Wir beschränken uns hier auf Standard-Tries.

Beispiel 3.2. $a_1 = 00010\ldots$, $a_2 = 00110\ldots$, $a_3 = 10101\ldots$, $a_4 = 11010\ldots$.



□

Analog zu den zufälligen Suchbäumen sei $t_j(n)$ die Tiefe des j-ten Blattes eines Tries ($j = 1, \dots, n$). Die mittlere Tiefe ist definiert als $T_n = \sum_{j=1}^{n} t_j(n)/n$. Die externe Pfadlänge ist $L_n = \sum_{j=1}^{n} t_j(n)$. Wir interessieren uns für Erwartungswert und Varianz von T_n und L_n im verallgemeinerten Kostenmaß. Für die Erwartungswerte gilt $E[L_n] = nE[T_n]$. Bei der Varianz besteht kein einfacher Zusammenhang zwischen $Var[L_n]$ und $Var[T_n]$, da die $t_j(n)$ nicht unabhängig sind, vergleiche hierzu auch [KPS89].

Wir setzen voraus, daß alle Schlüssel unabhängig voneinander und gleich verteilt gemäß einem bestimmten Wahrscheinlichkeitsmodell erzeugt werden. Bei dem bisher meistens untersuchten Konzept der unabhängigen Symbole gilt für alle Schlüssel $a_j = a_j(1)a_j(2)\dots$, daß Prob$(a_j(k) = 0) = p$ und Prob$(a_j(k) = 1) = q = 1 - p$ für ein $0 < p < 1$ und alle k, j. Diese Annahme über die Schlüsselverteilung wird als *Bernoulli-Modell* bezeichnet, im Falle $p = 1/2$ auch als *symmetrisches Bernoulli-Modell*. Die Verallgemeinerung zum *Markov-Modell* ist naheliegend und wurde in [JS91] betrachtet. Der dort gemachte Ansatz läßt sich aber nicht für $d > 1$ verwenden. Am Ende dieses Abschnitts skizzieren wir die Berechnung der erwarteten Pfadlänge im Markov-Modell mit Hilfe eines rekursiven Gleichungssystems.

Im Bernoulli-Modell gilt für die erwarteten externen Pfadkosten ℓ_n in einem Trie mit n Elementen die Rekursionsgleichung

$$\ell_n = n + d \cdot \sum_{k=0}^{n} \binom{n}{k} p^k q^{n-k} \cdot (\ell_k + \ell_{n-k}) - [n = 1],$$

was auch $\ell_0 = \ell_1 = 0$ beinhaltet. Rekursionsgleichungen dieser Art und das asymptotische Verhalten ihrer Lösung wurden in [Sz88, Sz98] untersucht. Wir verallgemeinern diesen Ansatz für beliebige $d \geq 1$.

Für die exponentielle Erzeugendenfunktion $L(z) = \sum_{n=0}^{\infty} \ell_n z^n/n!$ finden wir

$$L(z) = z\exp(z) + d\,L(zp)\exp(zq) + d\,L(zq)\exp(zp) - z, \tag{3.3}$$

indem wir beachten, daß für zwei beliebige exponentielle Erzeugendenfunktionen $F(z) = \sum_{n=0}^{\infty} f_n z^n/n!$ und $G(z) = \sum_{n=0}^{\infty} g_n z^n/n!$ gilt, daß

$$F(z) \cdot G(z) \;=\; \sum_{n=0}^{\infty} \sum_{k=0}^{n} \binom{n}{k} f_k g_{n-k}\, z^n/n!\,.$$

Mit $F(z) = L(z)$ und $G(z) = \exp(z)$ folgt (3.3). Durch Multiplikation von (3.3) mit $\exp(-z)$ erhält man die sogenannte Poisson-Transformation $\overline{L}(z) = L(z)\exp(-z)$, und mit $p + q = 1$ und $L(zp)\exp(zq)\exp(-z) = \overline{L}(zp)\exp(zp + zq - z) = \overline{L}(zp)$ folgt

$$\overline{L}(z) = z + d\,\overline{L}(zp) + d\,\overline{L}(zq) - z\exp(-z). \tag{3.4}$$

Sei $\overline{L}(z) = \sum_{n=0}^{\infty} \overline{\ell}_n z^n/n!$. Dann gilt, daß $\ell_n = \sum_{k=0}^{n} \binom{n}{k} \overline{\ell}_k$, siehe z.B. [R79]. Die $\overline{\ell}_k$ erhält man durch Koeffizientenvergleich: für die Koeffizienten von z^n in (3.4) folgt nach Multiplikation mit $n!$

$$\overline{\ell}_n \;=\; [n = 1] + d\,\overline{\ell}_n p^n + d\,\overline{\ell}_n q^n - n(-1)^{n+1}\;,$$

und schließlich

$$\ell_n \;=\; \sum_{k=0}^{n} \binom{n}{k} \frac{[k = 1] + k(-1)^k}{1 - d(p^k + q^k)} \;=\; \sum_{k=2}^{n} \binom{n}{k} (-1)^k \frac{k}{1 - d(p^k + q^k)}\;.$$

Die asymptotische Entwicklung alternierender Summen vom Typ

$$\sum_{k=0}^{n} \binom{n}{k} (-1)^k f_k$$

für $n \to \infty$ wurde von *Ph. Flajolet*, *W. Szpankowski* und anderen untersucht [FS95, Sz88, Sz98]. Für $d = 1$ gilt

$$\ell_n = \frac{n\log(n)}{H(p)} + O(n),$$

wobei $H(p) = -p\log(p) - q\log(q)$ die Entropie der Verteilung $(p, 1-p)$ ist. Im Falle $d > 1$ lassen sich diese Resultate allerdings nicht anwenden, da die in der Rechnung auftretende Gleichung $p^{-z} + q^{-z} = 1/d$ im allgemeinen keine einfach darstellbaren (komplexen) Lösungen besitzt.

Zum Abschluß skizzieren wir, wie man die erwarteten externen Pfadkosten ℓ_n im Markov-Modell herleitet. An Stelle $k > 1$ des Schlüssels hängt die Wahrscheinlichkeit des Symbols $a(k)$ von dem direkt vorausgehenden Bit an der Stelle $k - 1$ ab und folgt einer Verteilung $\mathrm{Prob}(a(k) \mid a(k-1))$. Die Schlüssel sind also Realisierungen einer Markov-Kette mit Übergangsmatrix

$$\mathbf{P} = \begin{pmatrix} p_{00} & p_{01} \\ p_{10} & p_{11} \end{pmatrix}.$$

Wir setzen voraus, daß $0 < p_{ii} < 1$ für $i = 0, 1$, damit ist die Markov-Kette positiv-rekurrent und besitzt eine stationäre Verteilung $\pi = (\pi_0, \pi_1)$. Wir nehmen an, daß die Kette in der stationären Verteilung gestartet wird, d.h. die ersten Schlüsselzeichen $a_j(1)$ werden gemäß π erzeugt.

Seien ℓ_n^0, ℓ_n^1 die erwarteten Pfadkosten, wenn der Zustand im Vorgängerknoten 0 bzw. 1 ist. Dann muß gelten:

$$\begin{aligned}
\ell_n^0 &= n + d \cdot \sum_{k=0}^{n} \binom{n}{k} p_{00}^k p_{01}^{n-k} \cdot (\ell_k^0 + \ell_{n-k}^1) - [n=1] \\
\ell_n^1 &= n + d \cdot \sum_{k=0}^{n} \binom{n}{k} p_{10}^k p_{11}^{n-k} \cdot (\ell_k^0 + \ell_{n-k}^1) - [n=1],
\end{aligned}$$

was wiederum $\ell_0^0 = \ell_1^0 = 0$ und $\ell_0^1 = \ell_1^1 = 0$ beinhaltet. Die gesuchte mittlere Tiefe ist schließlich $\ell_n = \pi_0 \ell_n^0 + \pi_1 \ell_n^1$. Analog zu oben finden wir die exponentiellen Erzeugendenfunktionen

$$\begin{aligned}
L^0(z) &= z\exp(z) + d\,L^0(zp_{00})\exp(zp_{01}) + d\,L^1(zp_{10})\exp(zp_{11}) - z \\
L^1(z) &= z\exp(z) + d\,L^0(zp_{10})\exp(zp_{11}) + d\,L^1(zp_{11})\exp(zp_{10}) - z
\end{aligned}$$

und deren Poisson-Transformationen

$$\begin{aligned}
\overline{L}^0(z) &= z + \overline{L}^0(zp_{00}) + \overline{L}^1(zp_{01}) - z\exp(-z) \\
\overline{L}^1(z) &= z + \overline{L}^0(zp_{10}) + \overline{L}^1(zp_{11}) - z\exp(-z)\,.
\end{aligned}$$

Koeffizientenvergleich führt uns auf das Gleichungssystem

$$\begin{aligned}
\overline{\ell}_n^0 &= [n=1] + d\overline{\ell}_n^0 p_{00}^n + d\overline{\ell}_n^1 p_{01}^n + n(-1)^n \\
\overline{\ell}_n^1 &= [n=1] + d\overline{\ell}_n^0 p_{10}^n + d\overline{\ell}_n^1 p_{11}^n + n(-1)^n
\end{aligned}$$

mit Lösung

$$\begin{aligned}
\overline{\ell}_n^0 &= \frac{(1 - dp_{11}^n + dp_{01}^n)([n=1] + n(-1)^n)}{1 - dp_{00}^n - dp_{11}^n + d^2(p_{00}^n p_{11}^n - p_{01}^n p_{10}^n)} \\
\overline{\ell}_n^1 &= \frac{(1 - dp_{00}^n + dp_{10}^n)([n=1] + n(-1)^n)}{1 - dp_{00}^n - dp_{11}^n + d^2(p_{00}^n p_{11}^n - p_{01}^n p_{10}^n)}\,.
\end{aligned}$$

Die Rücktransformation $\ell_n = \sum_{k=0}^{n} \binom{n}{k} \overline{\ell}_k$ ergibt

$$\begin{aligned}
\ell_n^0 &= \sum_{k=2}^{n} \binom{n}{k} (-1)^k \frac{k(1 - dp_{11}^k + dp_{01}^k)}{1 - dp_{00}^k - dp_{11}^k + d^2(p_{00}^k p_{11}^k - p_{01}^k p_{10}^k)} \\
\ell_n^1 &= \sum_{k=2}^{n} \binom{n}{k} (-1)^k \frac{k(1 - dp_{00}^k + dp_{10}^k)}{1 - dp_{00}^k - dp_{11}^k + d^2(p_{00}^k p_{11}^k - p_{01}^k p_{10}^k)}\,.
\end{aligned}$$

Wie oben angedeutet, folgt für die erwarteten externen Pfadkosten im Markov-Modell bei dem verallgemeinerten Kostenmaß:

$$\ell_n = \pi_0\,\ell_n^0 + \pi_1\,\ell_n^1\,.$$

Auch hier ist im Falle $d = 1$ folgende asymptotische Entwicklung für $n \to \infty$ möglich:

$$\ell_n = \frac{n \log(n)}{H(\mathbf{P})} + O(n),$$

wobei $H(\mathbf{P})$ die Entropie der Markov-Kette ist, die die Schlüssel erzeugt [JS91].

Wir fassen unsere Ergebnisse für das verallgemeinerte Kostenmaß zusammen:

Satz 3.12. *Für alle $d \geq 1$ gilt für die mittleren externen Pfadkosten eines zufälligen Tries im Bernoulli-Modell*

$$E[L_n] = \sum_{k=2}^{n} \binom{n}{k} (-1)^k \frac{k}{1 - d(p^k + q^k)} ,$$

und im Markov-Modell

$$E[L_n] = \sum_{k=2}^{n} \binom{n}{k} (-1)^k \frac{k + d(\pi_0(p_{01}^k - p_{11}^k) + \pi_1(p_{10}^k - p_{00}^k))}{1 - dp_{00}^k - dp_{11}^k + d^2(p_{00}^k p_{11}^k - p_{01}^k p_{10}^k)} ,$$

falls diese Summen existieren. □

4. Zusammenfassung und Ausblick

4.1. Zusammenfassung

In der vorliegenden Arbeit haben wir adaptive Suchverfahren und ihre Anwendungen unter verschiedenen Kostenmodellen analysiert.

Die drei neu gefundenen Klassen von Listenalgorithmen SORT-BY-RANK(α), SORT-BY-DELAY(k) und SORT-BY-TIME(q) stellen die umfassendsten bisher bekannten Strategien für das Listenproblem dar. Sie basieren auf der hier vorgestellten Idee, *Rangfunktionen* zu verwenden, durch die die Ordnung der Liste bestimmt wird. Dabei zeigen die SORT-BY-RANK(α)-Verfahren, daß man durch „Interpolation“ kompetitiv-optimaler Verfahren die Optimalität erhalten kann, siehe Satz 1.13. Die neuen Algorithmen arbeiten auch bei stochastischen Zugriffen gut, und wir können exakte und asymptotische Schranken für das Kostenverhältnis zu optimalen Verfahren angeben. Dabei betrachten wir unabhängige und abhängige Anfragen und stellen fest, daß bei unabhängigen Zugriffen die Verfahren mit größerem Gedächtnis besser werden, während sie bei trägen abhängigen Zugriffen schlechter werden.

Weiterhin geben wir in Satz 1.27 eine allgemeine untere Schranke an, die die Fähigkeit eines Listenalgorithmen zur Approximation gedächtnisloser Quellen zu der Größe seines Gedächtnisses in Beziehung setzt.

Unter anderem zeigen die untersuchten Listenalgorithmen bei der Datenkompression ihren Nutzen, was wir in Kapitel 2.1 anhand von theoretischen und empirischen Ergebnissen belegen.

Für das randomisierte Listenproblem zeigen wir in Satz 1.50, wie man den bekannten BIT-Algorithmus derandomisieren kann, so daß er nur noch wenige zufällige Bits benötigt, ohne daß seine Qualität beeinträchtigt wird. Mit diesem Verfahren erhalten wir einen Algorithmus mit dem geringsten zur Zeit bekannten Verbrauch der Ressource Zufall, der noch die Eigenschaft besitzt, die untere Schranke für das kompetitive Verhältnis deterministischer Verfahren zu unterbieten.

Außerdem betrachten wir die Konvergenzgeschwindigkeit verschiedener Listenalgorithmen und analysieren absorbierende Verfahren. Für mehrere Varianten des Problems wie gewichtete Zugriffskosten, bidirektionale Suche und lineare Suche in Feldern schlagen wir neue Algorithmen vor und analysieren sie in verschiedenen Kostenmodellen. Adaptive Verfahren können auch bei der effizienten Behandlung von geometrischen Problemen erfolgreich sein. Wir analysieren und verallgemeinern den von *G. Hotz* vorgeschlagenen Algorithmus zur Berechnung der konvexen Hülle.

Neben der Untersuchung von adaptiven Verfahren beschäftigen wir uns mit verallgemeinerten Kosten in Suchprozessen. Wir definieren eine Kostenfunktion, die exponentiell mit der Länge der Suche (Tiefe im Baum) wächst. Ein damit verwandtes Konzept ist der Begriff der *Rényi-Entropie*, wofür bereits Kodierungstheoreme bekannt sind. Wir untersuchen die algorithmische Seite des Problems, die in der bekannten Literatur noch nicht betrachtet wurde. Neben einem verallgemeinerten Huffman-Algorithmus für optimale Kodes in diesem Modell präsentieren wir untere und obere Schranken für die Kosten von Suchbäumen. Als Grenzfall ist das üblich verwendete Einheitskostenmaß in unserer Betrachtung enthalten. Außerdem analysieren wir in dem neuen Modell die Kosten von Suchbäumen und digitalen Bäumen, die zufällig gewachsen sind, d.h. durch stochastische Prozesse definiert werden. Hierbei zeigt sich unter anderem, daß durch die exponentiell wachsenden Kosten bereits kleine Abweichungen von einem optimalen Baum stark ansteigende Suchkosten verursachen. Mit anderen Worten, in diesem Kostenmodell stellt die explizite Konstruktion von optimalen Bäumen einen sinnvollen Aufwand dar.

4.2. Ausblick

Eine Reihe interessanter offener Fragen, die im Laufe dieser Arbeit entstanden sind, schließt sich an.

4.2.1. Tradeoff zwischen den Kostenmodellen?

Wir haben die Kosten von verschiedenen Algorithmen im kompetitiven Modell und im Modell der mittleren Analyse beurteilt. Ein direkter Vergleich dieser Modelle ist uns für kein Gebiet der Online-Algorithmen bekannt.

Bei unabhängigen identisch verteilten Zugriffen gibt das Verhältnis von asymptotischen erwarteten Kosten eines Verfahrens zu optimalen erwarteten Kosten an, wie gut sich das Verfahren anpassen kann, wenn die Zugriffe eine gewisse einfache Struktur (Unabhängigkeit) besitzen. Je länger der Algorithmus die Zugriffsfolge beobachten darf, desto mehr Informationen kann er aus ihr ziehen.

Der kompetitive Faktor dagegen beschreibt, wie flexibel sich ein Algorithmus an neue Situationen anpassen kann. Die Informationen aus vergangenen Zugriffen helfen dabei nur wenig, dagegen ist es wichtig, auf beliebige Anfragen des Offline-Gegners schnell reagieren zu können.

Die SBD(k)-Verfahren ermöglichen einen Tradeoff zwischen diesen beiden Kostenmodellen. Mit wachsendem k wird das kompetitive Verhalten schlechter, die Anpassung an unabhängige identisch verteilte Zugriffe aber besser. Für $k = 1$ und $k = 2$ sind die Algorithmen kompetitiv-optimal, und für $k \to \infty$ werden sie auf jeder gedächtnislosen Quelle beliebig gut.

Ist es möglich, daß eine Familie von Listenalgorithmen existiert, die gleichzeitig in beiden Kostenmodellen beliebig gut arbeitet? Wir vermuten das Gegenteil, nämlich daß eine Funktion $f(c) > 1$ existiert, so daß es für jeden c-kompetitiven Listenalgorithmus ALG eine Wahrscheinlichkeitsverteilung $\mathbf{p}$ gibt, bei der das Verhältnis $E_{\mathrm{ALG}}(\mathbf{p})/E_{\mathrm{OPT}}(\mathbf{p}) \geq f(c)$ wird.

Da die beiden Kostenmodelle keinen direkten Vergleich erlauben, könnte es möglich sein, durch Vergleich der Kosten bei unabhängigen, identisch verteilten Zugriffen mit den Kosten auf Zugriffsquellen mit Gedächtnis diese Vermutung zu zeigen. Markov-Quellen können zwar keine Worst-Case-Situation eines Modells mit Gegner erzeugen, aber trotzdem hinreichend schwierige Zugriffsfolgen liefern. Dieser Ansatz wird aber dadurch erschwert, daß eine einfache Darstellung der Kosten auf Markov-Ketten nur für Spezialfälle bekannt ist.

4.2.2. Beschränkte Gegner

Bei der kompetitiven Analyse wird die Qualität eines Online-Algorithmus gegenüber einem allmächtigen Offline-Gegner untersucht. Dieses Modell des Gegners ist aber so stark, daß Unterschiede zwischen verschiedenen Algorithmen verdeckt werden. Beispielsweise haben wir in Abschnitt 1.4 mehrere Klassen von Verfahren vorgestellt, die alle 2-kompetitiv sind, aber dennoch Differenzen aufweisen, die erst bei einer mittleren Analyse sichtbar werden.

Ein Schwerpunkt dieser Arbeit liegt deshalb auf Markov-Zugriffsfolgen, die allgemeiner als unabhängige Anfragefolgen sind und noch eine exakte Analyse erlauben. Leider ist es nur in Ausnahmefällen möglich [Ch93], optimale Listenalgorithmen und deren Kosten auf Markov-Ketten analytisch zu bestimmen. Bei gegebener Markov-Kette kann eine konkrete Berechnung mittels dynamischer Programmierung durchgeführt werden [H60], oder bei anderen Online-Problemen kann diese Information direkt in die Strategie einfließen [KPR92].

Es erscheint interessant, weitere Modelle mit eingeschränkten Gegnern zu entwickeln [BIRS95, KP94, Y98], oder Ressourcenbeschränkungen ins Spiel zu bringen [L94, L95]. Beispielsweise kann man die Frage stellen, wie gut ein Offline-Listenalgorithmus sein kann, der zwar die gesamte Eingabefolge im voraus sieht, aber nur eine Rechenzeit verwenden darf, die polynomiell in der Länge der Liste wächst. Man beachte, daß das beste bekannte Offline-Verfahren zur Berechnung einer optimalen Strategie exponentielle Zeit benötigt [RW90]. Es ist unbekannt, ob das zugehörige Entscheidungsproblem NP-hart ist; eine 15/8-Approximation ist möglich, bei statischen Gegnern auch eine 4/3-Approximation [A98a]. Das Auffinden besserer Approximationsverfahren stellt eine weitere interessante offene Frage dar.

A. Grundlagen über Markov-Ketten

A.1. Wahrscheinlichkeitsverteilungen

Sei $(\Omega, \mathcal{F}, p)$ ein diskreter Wahrscheinlichkeitsraum. Das bedeutet, daß Ω eine nichtleere abzählbare Menge ist, $\mathcal{F} = \mathcal{P}(\Omega)$ die Potenzmenge von Ω, die eine σ-Algebra in Ω darstellt, und $p : \Omega \to \mathbb{R}$ ein Wahrscheinlichkeitsmaß auf Ω [F68, MP90]. Ein Ereignis $A \in \mathcal{F}$ hat dann die Wahrscheinlichkeit

$$\text{Prob}(A) := \sum_{\omega \in A} p(\omega).$$

Eine reellwertige Zufallsvariable X auf $(\Omega, \mathcal{F}, p)$ ist eine Funktion

$$X : \Omega \to \mathbb{R}$$

mit der Eigenschaft, daß $\{X \in B\} = X^{-1}(B) \in \mathcal{F}$ für B aus der Borel-σ-Algebra der Teilmengen von $\mathbb{R}$ (Meßbarkeit von X). Die Funktion

$$\text{Prob}(X = B) = p(X^{-1}(B))$$

bezeichnet dann das Bildmaß von p unter X.

Für abzählbares Ω und eine diskrete Zufallsvariable X sind die k-ten Momente definiert als

$$E[X^k] = \sum_{x \in X(\Omega)} x^k \cdot \text{Prob}(X = x).$$

Speziell heißt $E[X]$ der Erwartungswert und $Var[X] = E[X \cdot (X - E[X])] = E[X^2] - E[X]^2$ die Varianz von X. Im folgenden sei stets $\Omega = \{1, 2, \ldots, n\}$ und $\mathcal{F} = \mathcal{P}(\Omega)$, und wir betrachten verschiedene Verteilungen.

(a) *Gleichverteilung*: $p_i = 1/n$ für alle $i = 1, \ldots, n$. Der Erwartungswert beträgt $(n+1)/2$, die Varianz $(n+1)(n-1)/12$.

(b) *Binomialverteilung*: $p_i = \binom{n}{i} p^i (1-p)^{n-i}$ für alle $i = 1, \ldots, n$ und einen Parameter $0 \le p \le 1$. Der Erwartungswert ist np und die Varianz $np(1-p)$.

(c) *Verteilung nach Zipf*: $p_i = 1/(iH_n)$ für alle $i = 1, \ldots, n$. Hierbei ist $H_n = \sum_{j=1}^{n} 1/j = \ln(n) + \gamma + O(1/n)$ die n-te harmonische Zahl und $\gamma = 0.577\ldots$ die Eulersche Konstante. Der Erwartungswert liegt bei n/H_n, die Varianz bei $\frac{1}{2} n(n+1)/H_n - n^2 H_n^2$.

(d) *Verteilung nach Lotka*: $p_i = 1/(i^2 H_n(2))$ für alle $i = 1, \ldots, n$. Hierbei ist $H_n(2) = \sum_{j=1}^{n} 1/j^2 = \pi^2/6 - O(1/n)$ die n-te harmonische Zahl zweiter Ordnung. Der Erwartungswert hat den Wert $H_n/H_n(2)$.

(e) *Verallgemeinerte Zipf-Verteilung*: $p_i = 1/(i^\alpha H_n(\alpha))$ für alle $i = 1, \ldots, n$ und einen Parameter $\alpha \geq 0$. Hierbei ist $H_n(\alpha) = \sum_{j=1}^{n} j^{-\alpha}$ die n-te verallgemeinerte harmonische Zahl der Ordnung α. Der Erwartungswert beträgt $H_n(\alpha - 1)/H_n(\alpha)$.

(f) *Geometrische Verteilung*: $p_i = p(n)q^{i-1}$ für $i = 1, \ldots, n$ und $0 < q < 1$ Dabei ist $p(n) = (1-q)/(1-q^n)$ der Normalisierungsfaktor. Der Erwartungswert ist $\frac{1-q^n-n(q^n-q^{n+1})}{(1-q)(1-q^n)}$.

A.2. Markov-Ketten

A.2.1. Definitionen und Grundlagen

Im folgenden stellen wir wichtige Aussagen über Markov-Ketten zusammen, vgl. auch [C67, F68, KS60]. Sei $\mathcal{A} \neq \emptyset$ die abzählbare Menge der möglichen *Zustände* und $\mathbf{P} = (p_{ij})_{i,j \in \mathcal{A}}$ eine stochastische Matrix. Das bedeutet, $p_{ij} \geq 0$ für alle $i, j \in \mathcal{A}$ und $\sum_{j \in \mathcal{A}} p_{ij} = 1$ für alle $i \in \mathcal{A}$.

Eine diskrete Markov-Kette ist eine Folge $\{X_t\}_{t \in \mathbb{N}_0}$ von Zufallsvariablen auf einem Wahrscheinlichkeitsraum $(\Omega, \mathcal{F}, p)$ mit gemeinsamem Wertebereich $\mathcal{A}$, für die gilt, daß

$$\mathrm{Prob}(X_t = a_t \mid X_{t-1} = a_{t-1}, \ldots, X_0 = a_0) = \mathrm{Prob}(X_t = a_t \mid X_{t-1} = a_{t-1})$$

für alle $t \in \mathbb{N}$. Die charakteristische Eigenschaft von Markov-Ketten ist, daß die Wahrscheinlichkeitsverteilung von X_t nur von der Realisierung der direkt vorausgehenden Zufallsvariablen X_{t-1} abhängt und unabhängig von den vorhergenden Variablen $X_{t-2}, X_{t-3}, \ldots$ ist. Falls $\mathrm{Prob}(X_t = j \mid X_{t-1} = i) = p_{ij}$ unabhängig von t ist, so heißt die Markov-Kette *homogen*. Im folgenden betrachten wir nur noch homogene Markov-Ketten, sie sind durch Übergangswahrscheinlichkeiten p_{ij} und eine Startverteilung $\mathrm{Prob}(X_0)$ vollständig definiert.

Wenn man die Übergänge iteriert, erhält man die Wahrscheinlichkeit $p_{ij}^{(s)}$, in s Schritten von i nach j zu gelangen:

$$p_{ij}^{(s)} = \mathrm{Prob}(X_{t+s} = j \mid X_t = i) = \sum_{k \in \mathcal{A}} p_{ik} p_{kj}^{(s-1)}.$$

Dabei setzt man $p_{ij}^{(0)} = \delta_{ij}$. Sei $f_{ij}^{(s)}$ die Wahrscheinlichkeit, von Zustand i startend in s Übergängen zum ersten Mal nach j zu gelangen (Erst-Eintritts-Wahrscheinlichkeit, *first entrance probability*),

$$f_{ij}^{(s)} = \mathrm{Prob}(X_{t+s} = j \mid X_t = i, X_{t+1} \neq j, \ldots, X_{t+s-1} \neq j).$$

Diese Werte können iterativ berechnet werden, indem man den Übergang von i nach j in s Schritten zerlegt in den Weg bis zum ersten Eintritt in j nach σ Schritten und dann den Weg von j nach j in weiteren $s-\sigma$ Schritten:

$$p_{ij}^{(s)} = \sum_{\sigma=1}^{s} f_{ij}^{(\sigma)} p_{jj}^{(s-\sigma)}.$$

Für $s = 1, 2, \ldots$ erhält man sukzessive die f_{ij}^{s}. Sei $f_{ij}^{*} = \sum_{s=1}^{\infty} f_{ij}^{(s)}$ die Wahrscheinlichkeit, jemals von Zustand i nach Zustand j zu gelangen. Falls der Eintritt in j sicher ist, d.h. $f_{ij}^{*} = 1$, so definiert $\{f_{ij}^{(s)}\}_{s \geq 1}$ eine Wahrscheinlichkeitsverteilung auf $\mathbb{N}$, die Erst-Eintritts-Verteilung (*first entrance time distribution*). Ihr Erwartungswert $m_{ij} = \sum_{s=1}^{\infty} s f_{ij}^{(s)}$ beschreibt die erwartete Anzahl von Schritten bis zum ersten Eintritt in j, wenn die Kette in i startet (mittlere Erst-Eintritts-Zeit, *mean first passage time*).

Ein weiteres Konzept sind *Tabu-Zustände*, die während einer Folge von Übergängen nicht betreten werden dürfen. Sei $H \subset \mathcal{A}$. Dann betrachtet man die Wahrscheinlichkeit

$${}_{H}p_{ij}^{(s)} = \mathrm{Prob}(X_{t+s} = j \mid X_t = i,\ X_\tau \notin H, t < \tau < t+s),$$

daß man in s Schritten von i nach j gelangt, ohne Zustände aus der *Tabu-Menge* H zu benutzen. Analog zu oben ist

$${}_{H}f_{ij}^{(s)} = \mathrm{Prob}(X_{t+s} = j \mid X_t = i,\ X_\tau \notin H, X_\tau \neq j, t < \tau < t+s)$$

die Wahrscheinlichkeit, in s Schritten zum ersten Mal j zu erreichen, wenn man in i startet und keine Zustände aus H betritt. Es gilt also

$${}_{H}f_{ij}^{(s)} = {}_{j,H}p_{ij}^{(s)}\ ,$$

wenn man abkürzend j, H für $\{j\} \cup H$ schreibt. Mit folgender Beobachtung lassen sich Übergangswahrscheinlichkeiten bei Anwesenheit von Tabu-Mengen bestimmen. Wenn $k \notin H$ ist, dann kann man in s Schritten von i nach j gelangen, indem man entweder den Zustand k gar nicht benutzt, oder ihn zum ersten Mal im Schritt σ betritt, und dann in $s - \sigma$ Schritten von k nach j geht:

$${}_{H}p_{ij}^{(s)} = {}_{k,H}p_{ij}^{(s)} + \sum_{\sigma=1}^{s} {}_{k,H}p_{ik}^{(\sigma)}\ {}_{H}p_{kj}^{(s-\sigma)}.$$

Sei für einen Zustand $i \in \mathcal{A}$ seine Periode definiert als $d(i) = \mathrm{ggT}\{t \in \mathbb{N} \mid p_{ii}^{(t)} > 0\}$. Mit folgenden Begriffen können Zustände klassifiziert werden: Ein Zustand $i \in \mathcal{A}$ heißt

- *periodisch*, wenn $d(i) > 1$ ist.
- *aperiodisch*, falls $d(i) = 1$ ist.
- *rekurrent (persistent)*, wenn die Rückkehr von i nach i sicher ist, d.h. $f_{ii}^{*} = 1$.

- *transient (unwesentlich)*, wenn $f^*_{ii} < 1$.
- *null-rekurrent*, wenn i rekurrent und die erwartete Zeit bis zur Wiederkehr unendlich ist ($m_{ii} = \infty$).
- *positiv-rekurrent (regulär)*, wenn i rekurrent und die erwartete Zeit bis zur Wiederkehr endlich ist ($m_{ii} < \infty$).
- *absorbierend*, wenn Zustand i nicht mehr verlassen werden kann ($p_{ii} = 1$).

Wir bezeichnen eine Markov-Kette $\{X_t\}_{t \in \mathbb{N}_0}$ als *absorbierend*, wenn sie mindestens einen absorbierenden Zustand besitzt und als *positiv-rekurrent*, wenn jeder Zustand positiv-rekurrent ist. Eine homogene Markov-Kette heißt *irreduzibel*, wenn es für jedes Paar $i, j \in \mathcal{A}$ von Zuständen ein $t \geq 0$ und einen Pfad der Länge t gibt, der i und j verbindet: $p^{(t)}_{ij} > 0$. Wenn eine Markov-Kette irreduzibel ist und einen endlichen Zustandsraum $\mathcal{A}$ besitzt, und jeder Zustand aperiodisch ist, so ist sie bereits positiv-rekurrent. Ein notwendiges und hinreichendes Kriterium dafür ist, daß eine Schrittanzahl t existiert, so daß $p^{(t)}_{ij} > 0$ für alle $i, j \in \mathcal{A}$.

Eine Wahrscheinlichkeitsverteilung π auf $\mathcal{A}$ heißt *stationär* oder *invariant* für eine Markov-Kette $\{X_t\}_{t \in \mathbb{N}_0}$ mit Übergangsmatrix $\mathbf{P}$, wenn gilt

$$\forall j \in \mathcal{A}: \ \pi_j = \sum_{i \in \mathcal{A}} \pi_i p_{ij}.$$

Eine positiv-rekurrente Markov-Kette besitzt eine eindeutige stationäre Verteilung π, bei der alle Komponenten positiv sind: $\pi_j > 0$ für $j \in \mathcal{A}$. In diesem Fall konvergiert die Verteilung nach t Schritten gegen die stationäre Verteilung, unabhängig von der Verteilung des Startzustandes X_0.

$$\text{Prob}(X_t = j) = \sum_{i \in \mathcal{A}} \text{Prob}(X_0 = i) \cdot p^{(t)}_{ij} \longrightarrow \pi_j \qquad \text{für } \ t \to \infty.$$

Eine Markov-Kette heißt *ergodisch*, wenn sie irreduzibel ist und die Grenzwerte $\pi_j = \lim_{t \to \infty} 1/t \cdot \sum_{s=0}^{t-1} p^{(s)}_{ij}$ existieren und unabhängig von $i \in \mathcal{A}$ sind, d.h. wenn die $p^{(s)}_{ij}$ Cesaro-summierbar sind [KS60]. Insbesondere ist jede positiv-rekurrente Markov-Kette bereits ergodisch, aber auch Markov-Ketten mit periodischen Zuständen können ergodisch sein.

Beschreibung durch Matrizen

Wenn man nur am asymptotischen Verhalten interessiert ist, spielt die Startverteilung keine Rolle mehr, und im folgenden identifizieren wir häufig die Markov-Kette $\{X_t\}$ mit ihrer Übergangsmatrix $\mathbf{P}$. Weiter beschränken wir uns auf den endlichen Zustandsraum $\mathcal{A} = \{1, \dots, n\}$. Eine nützliche Matrix für die Untersuchung von ergodischen Markov-Ketten ist die Fundamentalmatrix $\mathbf{Z}$. Sie ist definiert als

$$\mathbf{Z} = (\mathbf{I} - \mathbf{P} + \mathbf{A})^{-1} = \mathbf{I} + \sum_{t=1}^{\infty} (\mathbf{P} - \mathbf{A})^t = \mathbf{I} + \sum_{t=1}^{\infty} (\mathbf{P}^t - \mathbf{A}).$$

Dabei ist **I** die $n \times n$ Einheitsmatrix und **A** die Matrix der unabhängigen stationären Verteilung, bei der alle Zeilen durch π gegeben sind. Es gilt, daß $\mathbf{A} = \mathbf{A}^2 = \mathbf{PA} = \mathbf{AP}$ und deshalb auch $(\mathbf{P}-\mathbf{A})^t = \mathbf{P}^t - \mathbf{A}$. Für die weitere Diskussion setzen wir voraus, daß **P** eine positiv-rekurrente Markov-Kette ist.

Mit Hilfe der Fundamentalmatrix **Z** kann man die erwartete Erst-Eintritts-Zeit von i nach j ausdrücken [KS60] als

$$m_{ij} = (\delta_{ij} - z_{ij} + z_{jj})/\pi_j. \tag{A.1}$$

Dabei ist δ_{ij} das Kronecker-Symbol. Die m_{ij} erfüllen nämlich folgende Bedingung, wenn man die bedingten Erwartungswerte nach einem Schritt betrachtet:

$$m_{ij} = p_{ij} + \sum_{k \neq j} p_{ik}(m_{kj} + 1) = 1 - p_{ij} m_{jj} + \sum_{k} p_{ik} m_{kj}. \tag{A.2}$$

Indem man diese Gleichung mit π_i multipliziert und über i summiert, sieht man, daß $m_{ii} = 1/\pi_i$ ist. Weiter kann man zeigen, daß die m_{ij} eindeutig sind, und durch Einsetzen von (A.1) in (A.2) zeigt man die Darstellung (A.1).

Sei **P** eine positiv-rekurrente Markov-Kette und $H \subset \mathcal{A}$ eine Tabu-Menge, und seien $i, j \in \mathcal{A}$, $j \notin H$. Die Erst-Eintritts-Wahrscheinlichkeit, von i nach j zu gelangen, ohne unterwegs einen Zustand aus H zu betreten, ist nach [K61]

$${}_H f^*_{ij} = \frac{\det((\delta_{ab} - 1)m_{ab} + m_{ib} + m_{aj} - m_{ij})_{a,b \in H}}{\det((\delta_{ab} - 1)m_{ab} + m_{jb} + m_{aj})_{a,b \in H}}. \tag{A.3}$$

Für einelementige Tabu-Mengen $H = \{i\}$ erhält man speziell

$${}_i f^*_{kj} = \frac{m_{ki} + m_{ij} - m_{kj}}{m_{ij} + m_{ji}}. \tag{A.4}$$

Für eine ergodische Markov-Kette **P** mit stationärer Verteilung π kann man die rückwärtslaufende Kette $\hat{\mathbf{P}}$ (*time-reversed chain*) betrachten. Dabei ist die Wahrscheinlichkeit, vom Zustand i einen Schritt rückwärts nach j zu machen, gleich der Wahrscheinlichkeit, sich in j zu befinden und von dort einen Schritt vorwärts nach i zu machen unter der Bedingung, sich anschließend in i zu befinden:

$$\hat{p}_{ij} = \frac{\pi_j p_{ji}}{\pi_i}\,.$$

Eine Matrix **P** heißt doppelt stochastisch, wenn nicht nur die Zeilensummen, sondern auch die Spaltensummen gleich 1 sind. Wenn **P** die Übergangsmatrix einer ergodischen Markov-Kette ist, dann ist die stationäre Verteilung die Gleichverteilung auf $\mathcal{A}$: $\pi = (1/n, \dots, 1/n)$. In diesem Fall ist die Übergangsmatrix $\hat{\mathbf{P}}$ der rückwärtslaufenden Kette gerade die transponierte Matrix $\mathbf{P}^T$ der ursprünglichen Kette. Falls insbesondere **P** symmetrisch ist, so heißt die Markov-Kette *zeit-umkehrbar*, da $p_{ij} = \hat{p}_{ij}$ für alle $i, j \in \mathcal{A}$. Für rückwärtslaufende Markov-Ketten gilt allgemein für $i, j \in \mathcal{A}$:

$$\hat{m}_{ij} + \hat{m}_{ji} = m_{ij} + m_{ji}. \tag{A.5}$$

Aus der Definition von **Z** folgt nämlich $\hat{z}_{ij} = z_{ji}\pi_j/\pi_i$, woraus man $\hat{m}_{ij} - m_{ij} = (z_{ij} - \hat{z}_{ij})/\pi_j = z_{ij}/\pi_j - z_{ji}/\pi_i$ folgert.

A.2.2. Beispiele

Markov-Kette mit zwei Zuständen

Wir betrachten den allgemeinen Fall einer Markov-Kette über $\{0, 1\}$ mit der Übergangsmatrix

$$\mathbf{P} = \begin{pmatrix} \alpha & 1-\alpha \\ 1-\beta & \beta \end{pmatrix}.$$

Im Fall $0 \leq \alpha < 1$, $0 \leq \beta < 1$ und $\alpha + \beta > 0$ ist diese Markov-Kette irreduzibel und aperiodisch, und damit insbesondere positiv-rekurrent. Für die Übergangswahrscheinlichkeiten in t Schritten findet man

$$\mathbf{P}^t = \frac{1}{\alpha+\beta-2} \begin{pmatrix} \beta-1 & \alpha-1 \\ \beta-1 & \alpha-1 \end{pmatrix} + \frac{(\alpha+\beta-1)^t}{\alpha+\beta-2} \begin{pmatrix} \alpha-1 & 1-\alpha \\ 1-\beta & \beta-1 \end{pmatrix},$$

und man erkennt die stationäre Verteilung $\pi = (\frac{\beta-1}{\alpha+\beta-2}, \frac{\alpha-1}{\alpha+\beta-2})$. Aufgrund der Fundamentalmatrix

$$\mathbf{Z} = \mathbf{I} + \sum_{t=1}^{\infty} (\mathbf{P}^t - \mathbf{A}) = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} + \frac{1-\alpha-\beta}{(\alpha+\beta-2)^2} \begin{pmatrix} \alpha-1 & 1-\alpha \\ 1-\beta & \beta-1 \end{pmatrix}$$

findet man die mittleren Erst-Eintritts-Zeiten

$$\mathbf{M} = \begin{pmatrix} \frac{2-\alpha-\beta}{1-\beta} & \frac{1}{1-\alpha} \\ \frac{1}{1-\beta} & \frac{2-\alpha-\beta}{1-\alpha} \end{pmatrix}.$$

Elementweise Lokalität

Sei $\mathbf{p} = (p_1, \ldots, p_n)$ eine Wahrscheinlichkeitsverteilung über $\{1, \ldots, n\}$ mit $p_i > 0$ für alle $i = 1, \ldots, n$, und $\mathbf{A}$ die Übergangsmatrix unabhängiger Zugriffe mit Verteilung $\mathbf{p}$, d.h. alle Zeilen von $\mathbf{A}$ sind gleich $\mathbf{p}$. Sei weiter $0 \leq \lambda < 1$ ein Parameter. Wir betrachten die Übergangsmatrix

$$\mathbf{P} = \lambda \mathbf{I} + (1-\lambda)\mathbf{A}.$$

Wegen $\lambda < 1$ ist $\mathbf{P}$ die Übergangsmatrix einer positiv-rekurrenten Markov-Kette, und die stationäre Verteilung ist gerade die Verteilung $\mathbf{p}$. Für die Übergangswahrscheinlichkeiten in t Schritten findet man

$$\mathbf{P}^t = \mathbf{A} + \lambda^t(\mathbf{I} - \mathbf{A}).$$

Aus der Fundamentalmatrix $\mathbf{Z} = (\mathbf{I} - \lambda\mathbf{A})/(1-\lambda)$ erhält man die mittleren Erst-Eintritts-Zeiten

$$m_{ij} = \frac{1 - \delta_{ij}\lambda}{(1-\lambda)p_j}.$$

Als Spezialfall betrachten wir die Matrix $\mathbf{P}$ mit

$$p_{ij} = \left\{ \begin{array}{ccc} \alpha & : & i = j \\ \beta & : & \text{sonst} \end{array} \right\},$$

die man mit $\mathbf{p} = (1/n, \ldots, 1/n)$ und $\lambda = (n\alpha - 1)/(n-1)$ erhält. Daraus folgt

$$m_{ij} = \left\{ \begin{array}{ccc} n & : & i = j \\ (n-1)/(1-\alpha) & : & \text{sonst} \end{array} \right\}.$$

Man beachte, daß diese Markov-Kette zeit-umkehrbar ist.

Blockweise Lokalität

Sei wieder $\mathbf{p} = (p_1, \ldots, p_n)$ eine Wahrscheinlichkeitsverteilung über $\mathcal{A} = \{1, \ldots, n\}$ mit $p_i > 0$ für alle $i = 1, \ldots, n$, und $\mathbf{A}$ die Übergangsmatrix unabhängiger Zugriffe mit Verteilung $\mathbf{p}$. Sei $0 \leq \lambda < 1$ ein Parameter und $0 = k_0 < k_1 < \cdots < k_m = n$. Wir partitionieren den Zustandsraum $\mathcal{A}$ in m Teilmengen,

$$\mathcal{A} = \mathcal{B}_1 \cup \mathcal{B}_2 \cup \ldots \cup \mathcal{B}_m,$$

dabei ist $\mathcal{B}_\ell = \{k_{\ell-1} + 1, \ldots, k_\ell\}$. Sei $s_\ell = \sum_{j \in \mathcal{B}_\ell} p_j$ die Summe der Wahrscheinlichkeiten jeder Teilmenge. Die Matrix $\mathbf{J}$ definieren wir als

$$\mathbf{J} = \left(\begin{array}{c|c|c|c} \frac{1}{s_1}\mathbf{A}|_{\mathcal{B}_1 \times \mathcal{B}_1} & & & \\ \hline & \frac{1}{s_2}\mathbf{A}|_{\mathcal{B}_2 \times \mathcal{B}_2} & & \\ \hline & & \ddots & \\ \hline & & & \frac{1}{s_m}\mathbf{A}|_{\mathcal{B}_m \times \mathcal{B}_m} \end{array} \right),$$

wobei $\mathbf{A}|_{\mathcal{B}_\ell \times \mathcal{B}_\ell}$ die Einschränkung von $\mathbf{A}$ auf die Untermatrix mit Zeilen und Spalten aus $\mathcal{B}_\ell$ bedeutet. Durch den Faktor $1/s_\ell$ wird erreicht, daß jede Zeilensumme von $\mathbf{J}$ gerade 1 ist. Sei nun

$$\mathbf{P} = \lambda\mathbf{J} + (1 - \lambda)\mathbf{A}.$$

Da $\lambda < 1$, ist $\mathbf{P}$ die Übergangsmatrix einer positiv-rekurrenten Markov-Kette, und die stationäre Verteilung ist $\mathbf{p}$. Für die Übergangswahrscheinlichkeiten in t Schritten findet man

$$\mathbf{P}^t = \mathbf{A} + \lambda^t(\mathbf{J} - \mathbf{A}).$$

Aus der Fundamentalmatrix $\mathbf{Z} = \mathbf{I} + (\mathbf{J} - \mathbf{A})\lambda/(1-\lambda)$ erhält man die mittleren Erst-Eintritts-Zeiten

$$m_{ij} = \left\{ \begin{array}{ccc} 1/p_j & : & i, j \in \mathcal{B}_\ell \\ \frac{(1-\lambda)s_\ell + \lambda p_j}{(1-\lambda)s_\ell p_j} & : & j \in \mathcal{B}_\ell, i \notin \mathcal{B}_\ell \end{array} \right\}.$$

Wir betrachten den Spezialfall, daß alle Teilmengen die gleiche Größe $|\mathcal{B}_\ell| = k$ besitzen und setzen

$$p_{ij} = \left\{ \begin{array}{ccc} \alpha & : & \text{i und j liegen in der gleichen Teilmenge} \\ \beta & : & \text{sonst} \end{array} \right\}.$$

Dadurch ist die stationäre Verteilung die Gleichverteilung, $s_\ell = k/n$, und mit dem Parameter $\lambda = k(n\alpha - 1)/(n-k)$ können wir die obigen Ergebnisse benutzen:

$$m_{ij} = \left\{ \begin{array}{ccc} n & : & \text{i und j liegen in der gleichen Teilmenge} \\ \frac{(n-1)k - \alpha n(k-1)}{(1-\alpha k)k} & : & \text{sonst} \end{array} \right\}.$$

Literaturverzeichnis

[AW79] R. Ahlswede und I. Wegener. *Suchprobleme*. Teubner Verlag, Stuttgart, 1979.

[A98] S. Albers. Improved randomized on-line algorithms for the list update problem. *SIAM Journal on Computing*, 27:682–693, 1998.

[A98a] S. Albers. A competitive analysis of the list update problem with lookahead. *Theoretical Computer Science*, 197:95–109, 1998.

[AM97] S. Albers und M. Mitzenmacher. Revisiting the COUNTER algorithms for list update. *Information Processing Letters*, 64:155–160, 1997.

[AM98] S. Albers und M. Mitzenmacher. Average case analyses of list update algorithms, with applications to data compression. *Algorithmica*, 21:312–329, 1998.

[ASW95] S. Albers, B. v. Stengel, und R. Werchner. A combined BIT and Timestamp algorithm for the list update problem. *Information Processing Letters*, 56:135–139, 1995.

[AW98] S. Albers und J. Westbrook. Self-organizing data structures. In *Online Algorithms*, LNCS 1442, Seiten 13–51. Springer, 1998.

[A95] D. Aldous. On simulating a markov chain stationary distribution. In *Discrete Probability and Algorithms*, Band 72 von *IMA Series in Mathematics and its Applications*, Seiten 1–9. Springer, 1995.

[ALW96] D. Aldous, L. Lovász, und P. Winkler. Mixing times for uniformly ergodic markov chains. Technischer Bericht Nr. 1100, Yale University, 1996.

[AM78] B. Allen und I. Munro. Self-organizing binary search trees. *Journal of the ACM*, 25(4):526–535, 1978.

[ABCO96] V. Almeida, A. Bestavros, M. Crovella, und A. de Oliveira. Characterizing reference locality in the WWW. Technischer Bericht TR-96-11, Boston University, 1996.

[AMS96] N. Alon, Y. Matias, und M. Szegedy. The space complexity of approximating the frequency moments. In *28th ACM Symposium on the Theory of Computing (STOC)*, Seiten 20–29, 1996.

[AM80] D. Altenkamp und K. Mehlhorn. Codes: Unequal probabilities, unequal letter costs. *Journal of the ACM*, 27:412–427, 1980.

[AL94] F. d'Amore und V. Liberatore. The list update problem and the retrieval of sets. *Theoretical Computer Science*, 130:101–123, 1994.

[AMN93] F. d'Amore, A. Marchetti-Spaccamela, und U. Nanni. The weighted list update problem and the lazy adversary. *Theoretical Computer Science*, 108:371–384, 1993.

[ANW82] E. J. Anderson, P. Nash, und R. R. Weber. A counterexample to a conjecture on optimal list ordering. *Journal of Applied Probability*, 19:730–732, 1982.

[A77] J. P. Arnaud. *Sur quelques problèmes concernant les librairies*. Dissertation, Université Paul Sabatier, Toulouse, 1977.

[ACK87] O. I. Aven, E. G. Coffman, und Y. A. Kogan. *Stochastic Analysis of Computer Storage*. Reidel, Dordrecht, 1987.

[BE97] R. Bachrach und R. El-Yaniv. Online list accessing algorithms and their applications: Recent empirical evidence. In *8th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 1997.

[B75] P. J. Bayer. Improved bounds on the costs of optimal and balanced binary search trees. Technischer Bericht MIT/LCS/TM-69, MIT, 1975.

[BCW90] T. C. Bell, J. G. Cleary und I. H. Witten. *Text Compression*. Prentice Hall, 1990.

[B83] M. E. Bellow. *Performance of Self-Organizing Sequential Search Heuristics under Stochastic Reference Models*. Dissertation, Carnegie-Mellon University, Pittsburgh, 1983.

[BCL90] J. L. Bentley, K. L. Clarkson, und D. B. Levine. Fast linear expected-time algorithms for computing maxima and convex hulls. *Algorithmica*, 9:168–183, 1993.

[BM85] J. L. Bentley und C. C. McGeoch. Amortized analyses of self-organizing sequential search heuristics. *Communications of the ACM*, 28(4):404–411, 1985.

[BS78] J. L. Bentley und M. I. Shamos. Divide and conquer for linear expected time. *Information Processing Letters*, 7(2):87–91, 1978.

[BSTW86] J. L. Bentley, D. S. Sleator, R. E. Tarjan, und V. K. Wei. A locally adaptive data compression scheme. *Communications of the ACM*, 29:320–330, 1986.

[BBKTW94] S. Ben-David, A. Borodin, R. Karp, G. Tardos, und A. Wigderson. On the power of randomization in on-line algorithms. *Algorithmica*, 11:2–14, 1994.

[B79] J. R. Bitner. Heuristics that dynamically organize data structures. *SIAM Journal of Computing*, 8(1):82–110, 1979.

[B84] A. Björner. Ordering of coxeter groups. *Contemporary Mathematics*, 34:175–195, 1984.

[BH97] A. Boneh und M. Hofri. The coupon-collector problem revisited. *Stochastic Models*, 31:39–66, 1997.

[BP96] S. Boneh und V. G. Papanicolaou. General asymptotic estimates for the coupon collector problem. *Journal of Computational and Applied Mathematics*, 67:277–289, 1996.

[BoE97] A. Borodin und R. El-Yaniv. On randomization in online computation. In *Proceedings 12th IEEE Conference on Computational Complexity*, Seiten 226–238, 1997.

[BE98] A. Borodin und R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.

[BIRS95] A. Borodin, S. Irani, P. Raghavan, und B. Schieber. Competitive paging with locality of reference. *Journal of Computer and System Sciences*, 50:244–258, 1995.

[B63] R. K. Brayton. On the asymptotic behaviour of the number of trials necessary to complete a set with random selection. *Journal of Mathematical Analysis and Applications*, 7:31–61, 1963.

[BW94] M. Burrows und D. J. Wheeler. A block-sorting lossless data compression algorithm. Technischer Bericht 124, Digital Equipment Corporation, Systems Research Center, 1994.

[BK73] P. J. Burville und J. F. C. Kingman. On a model for storage and search. *Journal of Applied Probability*, 10:697–701, 1973.

[C65] L. L. Campbell. A coding theorem und Rényi's entropy. *Information and Control*, 8:423–429, 1965.

[Ch93] Ph. Chassaing. Optimality of move-to-front for self-organizing data structures with locality of references. *Annals of Applied Probability*, 3:1219–1240, 1993.

[CSO89] H. T. Ch'ng, B. Srinivasan, und B. C. Ooi. Study of selforganizing heuristics for skewed access patterns. *Information Processing Letters*, 30:237–244, 1989.

[CG96] V. S.-N. Choi und M. Golin. Lopsided trees: Analyses, algorithms, and applications. In *International Colloquium on Algorithms, Languages and Programming (ICALP)*, LNCS 1099, Seiten 538–549. Springer, 1996.

[CN98] M. Chrobak und J. Noga. Competitive Algorithms for Multilevel Caching and Relaxed List Update. In *9th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, Seiten 87–96, 1998.

[C67] K. L. Chung. *Markov Chains with Stationary Transition Probabilities*. Springer-Verlag, Berlin, 2. Auflage, 1967.

[CHS88] F. R. K. Chung, D. J. Hajela, und P. D. Seymour. Self-organizing sequential search and Hilbert's inequalities. *Journal of Computer and System Sciences*, Seiten 148–157, 1988.

[CFV98] J. Clément, Ph. Flajolet, und B. Vallée. The analysis of hybrid trie structures. In *9th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, Seiten 531–539, 1998.

[CW90] C. Courcoubetis und R. R. Weber. The move-to-front rule for multiple lists. *Probability in the Engineering and Informational Sciences*, 4:19–27, 1990.

[DS72] P. Denning und S. Schwartz. Properties of the working set model. *Communications of the ACM*, 15:191–198, 1972.

[D99] L. Devroye. A note on the expected time for finding maxima by list algorithms. *Algorithmica*, 23:97–103, 1999.

[D94] R. Dobrow. *Markov chain analysis for some self-organizing schemes for lists and trees*. Dissertation, John Hopkins University, Baltimore, 1994.

[DF95] R. Dobrow und J. Fill. The move-to-front rule for self-organizing lists with markov dependent requests. In *Discrete Probability and Algorithms*, Band 72 von *IMA Series in Mathematics and its Applications*, Seiten 57–80. Springer, 1995.

[E75] P. Elias. Universal codeword sets and representations of the integers. *IEEE Transactions on Information Theory*, 21:194–203, 1975.

[E87] P. Elias. Interval and recency rank source coding: Two on-line adaptive variable-length schemes. *IEEE Transactions on Information Theory*, 33:3–10, 1987.

[E96] R. El-Yaniv. There are infinitely many competitive-optimal online list accessing algorithms. Manuskript, 1996.

[EFT97] R. El-Yaniv, S. Fine, und N. Tishby. Agnostic classification of markovian sequences. In *Advances in Neural Information Processing Systems*, Band 10, Seiten 465–471. MIT Press, 1997.

[F68] W. Feller. *An Introduction to Probability Theory and its Applications*, Band 1. John Wiley & Sons, New York, 3. Auflage, 1968.

[F96] P. M. Fenwick. The Burrows-Wheeler transform for block sorting text compression. *Computer Journal*, 39:731–740, 1996.

[FW98] A. Fiat und G. Woeginger, Hrsg. *Online Algorithms*. LNCS 1442. Springer, 1998.

[F96] J. A. Fill. An exact formula for the move-to-front rule for self-organizing lists. *Journal of Theoretical Probability*, 9:113–160, 1996.

[F96a] J. A. Fill. Limits and rates of convergence for the distribution of search costs under the move-to-front rule. *Theoretical Computer Science*, 164:185–206, 1996.

[FH96] J. A. Fill und L. Holst. On the distribution of search cost for the move-to-front rule. *Random Structures and Algorithms*, 8:179–186, 1996.

[F85] Ph. Flajolet. Approximate counting: a detailed analysis. *BIT*, 25:113–134, 1985.

[FGT92] Ph. Flajolet, D. Gardy, und L. Thimonier. Birthday paradox, coupon collector, caching algorithms and self-organizing search. *Discrete Applied Mathematics*, 39:207–229, 1992.

[FS95] Ph. Flajolet und R. Sedgewick. Mellin transforms and asymptotics: Finite differences and Rice's integrals. *Theoretical Computer Science*, 144:101–124, 1995.

[F84] G. N. Frederickson. Self-organizing heuristics for implicit data stuctures. *SIAM Journal of Computing*, 13(2):277–291, 1984.

[F60] E. Fredkin. Trie memory. *Communications of the ACM*, 3:490–500, 1960.

[G97] T. Garefalakis. A new familiy of randomized algorithms for list accessing. In *European Symposium on Algorithms (ESA)*, LNCS 1284, Seiten 200–209. Springer, 1997.

[G94] M. J. Golin. A provably fast linear-expected-time maxima-finding algorithm. *Algorithmica*, 11:501–524, 1994.

[GR98] M. J. Golin und G. Rote. A dynamic programming algorithm for constructing optimal prefix-free codes with unequal letter costs. *IEEE Transactions on Information Theory*, 44, 1998.

[GY96] M. J. Golin und N. Young. Prefix codes: Equiprobable words, unequal letter costs. *SIAM Journal on Computing*, 25:1281–1292, 1996.

[GMS81] G. H. Gonnet, J. I. Munro, und H. Suwanda. Exegesis of self-organizing linear search. *SIAM Journal of Computing*, 10(3):613–637, 1981.

[GOR97] M.T. Goodrich, M. Orletsky, und K. Ramaiyer. Methods for achieving fast query times in point location data structures. In *8th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 1997.

[GKP94] R. L. Graham, D. E. Knuth, und O. Patashnik. *Concrete Mathematics*. Addison-Wesley, 2. Auflage, 1994.

[GRVW95] D. Grinberg, S. Rajagopalan, R. Venkatesan, und V. Wei. Splay trees for data compression. In *6th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, Seiten 522–530, 1995.

[HR89] T. Hagerup und C. Rüb. A guided tour of Chernoff bounds. *Information Processing Letters*, 33:305–308, 1989.

[HLP52] G. H. Hardy, J. E. Littlewood, und G. Polya. *Inequalities*. Cambridge University Press, 1952.

[H72] W. J. Hendricks. The stationary distribution of an interesting markov chain. *Journal of Applied Probability*, 9:231–233, 1972.

[H73] W. J. Hendricks. An extension of a theorem concerning an interesting markov chain. *Journal of Applied Probability*, 10:886–890, 1973.

[H76] W. J. Hendricks. An account of self-organizing systems. *SIAM Journal on Computing*, 5:715–723, 1976.

[HH85] J. H. Hester und D. S. Hirschberg. Self-organizing linear search. *ACM Computing Surveys*, 17:295–311, 1985.

[HH87] J. H. Hester und D. S. Hirschberg. Self-organizing search lists using probabilistic back-pointers. *Communications of the ACM*, Seiten 1074–1079, 1987.

[HK96] M. Hofri und N. Kechris. Probabilistic counting of a large number of events. Technischer Bericht TR-UH-CS-92-23, University of Houston, 1992.

[HS91] M. Hofri und H. Shachnai. Self-organizing lists and independent references: A statistical synergy. *Journal of Algorithms*, 12:533–555, 1991.

[HS91a] M. Hofri und H. Shachnai. On the optimality of the counter scheme for dynamic linear lists. *Information Processing Letters*, 37:175–179, 1991.

[HS98] M. Hofri und H. Shachnai. The list update problem: Improved bounds for the counter scheme. *Algorithmica*, 22:650–659, 1998.

[H90] G. Hotz. *Algorithmen, Sprachen und Komplexität, Festvortrag zur Eröffnung des Wintersemesters 1990/91*. Saarbrücker Universitätsreden, Band 32, 1990.

[H93] G. Hotz. Search trees and search graphs for markov sources. *Journal of Information Processing and Cybernetics*, 29:283–292, 1993.

[H97] G. Hotz. *Algorithmische Informationstheorie*. Teubner, 1997.

[HS97] G. Hotz und F. Schulz. Sorting and searching in view of the noiseless coding theorem, Manuskript, 1997.

[H60] R. A. Howard. *Dynamic Programming and Markov Processes*. MIT Press, 1960.

[H52] D. A. Huffman. A method for the construction of minimum redundancy codes. *Proceedings Inst. Radio Engineering*, 40:1098–1101, 1952.

[HM93] L. C. K. Hui und Ch. Martel. Unsuccessful search in self-adjusting data structures. *Journal of Algorithms*, 15:447–481, 1993.

[HM96a] L. C. K. Hui und Ch. Martel. Analyzing self-adjusting linear list algorithms with deletions and unsuccessful searches. *Information Processing Letters*, 58:231–236, 1996.

[HM96b] L. C. K. Hui und Ch. Martel. Rendomized competitive algorithms for successful and unsuccessful search. *The Computer Journal*, 39:427–438, 1996.

[JS91] Ph. Jacquet und W. Szpankowski. Analysis of digital tries with markovian dependence. *IEEE Transactions on Information Theory*, 37:1470–1475, 1991.

[JS99] Ph. Jacquet und W. Szpankowski. Entropy calculations via analytic depoissonization. *IEEE Transactions on Information Theory*, 1999.

[J98] P. D. Jelenković. Asymptotic approximation of the move-to-front search cost distribution and least-recently-used caching fault probabilities, Manuskript, Mai 1998.

[J88] D. W. Jones. Application of splay trees to data compression. *Communications of the ACM*, 31(8):996–1007, 1988.

[K94] R. Kannan. Markov chains and polynomial time algorithms. In *35th IEEE Symposium on Foundations of Computer Science (FOCS)*, Seiten 656–671, 1994.

[KR80] Y. C. Kan und S. M. Ross. Optimal list order under partial memory constraints. *Journal of Applied Probability*, 17:1004–1015, 1980.

[KMMO94] A. R. Karlin, M. S. Manasse, L. A. McGeoch, und S. Owicki. Competitive randomized algorithms for nonuniform problems. *Algorithmica*, 11:542–571, 1994.

[KPR92] A. Karlin, S. Phillips, und P. Raghavan. Markov paging. In *33rd IEEE Symposium on Foundations of Computer Science (FOCS)*, Seiten 208–217, 1992.

[KR90] R. Karp und P. Raghavan. Private Kommunikation zitiert in [RWS94], 1990.

[KK77] J. Keilson und A. Kester. Monotone matrices and monotone markov processes. *Stochastic Processes and their Applications*, 5:231–241, 1977.

[KS60] J. G. Kemeny und J. L. Snell. *Finite Markov Chains*. Van Nostrand, Princeton, 1960.

[K61] J. H. B. Kemperman. *The First Passage Problem for a Stationary Markov Chain*. University Press, Chicago, 1961.

[KPS89] P. Kirschenhofer, H. Prodinger, und W. Szpankowski. On the variance of the external path length in a symmetric digital trie. *Discrete Applied Mathematics*, 25:129–143, 1989.

[K98] D. E. Knuth. *The Art of Computer Programming, Vol. 2: Sorting and Searching*. Addison-Wesley, 3. Auflage, 1998.

[KV81] L. K. Konneker und Y. L. Varol. A note on heuristics for dynamic organization of data structures. *Information Processing Letters*, 12(5):213–216, 1981.

[KP94] E. Koutsoupias und C. Papadimitriou. Beyond competitive analysis. In *35th IEEE Symposium on Foundations of Computer Science (FOCS)*, Seiten 394–401, 1994.

[L84] K. Lam. Comparison of self-organizing linear search. *Journal of Applied Probability*, 21:763–776, 1984.

[LLS84] K. Lam, M. Y. Leung, und M. K. Siu. Self-organizing files with dependent accesses. *Journal of Applied Probability*, 21:343–359, 1984.

[LH90] L. L. Larmore und D. S. Hirschberg. A fast algorithm for optimal length-limited Huffman codes. *Journal of the ACM*, 37:464–473, 1990.

[L94] R. J. Lipton. A new approach to information theory. In *Symposium on the Theoretical Aspects of Computer Science (STACS)*, LNCS 775, Seiten 699–708. Springer, 1994.

[L95] R. J. Lipton. Coding for noisy feasible channels. In *WITS: Proceedings of the IEEE-IMS Workshop on Information Theory and Statistics*, 1995.

[LW95] L. Lovász und P. Winkler. Efficient stopping rules for markov chains. In *ACM Symposium on Theory of Computing (STOC)*, Seiten 76–82, 1995.

[LW96] L. Lovász und P. Winkler. Reversal of markov chains and the forget time. Technischer Bericht Nr. 1099, Yale University, 1996.

[LP94] F. Luccio und A. Pedrotti. A parallel list update problem. *Information Processing Letters*, 52:277–284, 1994.

[M92] H. Mahmoud. *Evolution of Random Search Trees*. Wiley, 1992.

[MR98] C. Martínez und S. Roura. Randomized binary search trees. *Journal of the ACM*, 45:288–323, 1998.

[MP90] R. Mathar und D. Pfeifer. *Stochastik für Informatiker*. Teubner, 1990.

[M96] R. Mathar. *Informationstheorie*. Teubner, 1996.

[MRB80] D. Matthews, D. Rotem, und E. Bretholz. Self-organizing doubly linked lists. *International Journal of Computer Mathematics, Sect. A*, 8:99–106, 1980.

[M65] J. McCabe. On serial files with relocatable records. *Operations Research*, 12:609–618, 1965.

[Mc77] R. J. McEliece. *The Theory of Information and Coding*. Addison-Wesley, 1977.

[M75] K. Mehlhorn. Nearly optimal binary search trees. *Acta Informatica*, 5:287–295, 1975.

[Me77] K. Mehlhorn. *Effiziente Algorithmen*. Teubner Verlag, 1977.

[M88] K. Mehlhorn. *Sortieren und Suchen*, Band 1 von *Datenstrukturen und effiziente Algorithmen*. Teubner Verlag, Stuttgart, 2. Auflage, 1988.

[M68] D. R. Morrison. PATRICIA–practical algorithm to retrieve information coded in alphanumeric. *Journal of the ACM*, 15:514–534, 1968.

[N75] P. Nath. On a coding theorem connected with Rényi's entropy. *Information and Control*, 29:234–242, 1975.

[N82] P. R. Nelson. The transposition replacement policy with a partial memory constraint. *Journal of Applied Probability*, 19:733–736, 1982.

[Ne96] M. Nelson. Data compression with the Burrows-Wheeler transform. *Dr. Dobb's Journal*, September, 1996.
http://dogma.net/markn/articles/bwt/bwt.htm.

[NS60] D. J. Newman und L. Shepp. The double dixie cup problem. *American Mathematical Monthly*, 67:58–61, 1960.

[NO89] D. T. H. Ng und B. J. Oommen. Generalizing singly-linked reorganization heuristics for doubly linked lists. In *Mathematical Foundations of Computer Science*, LNCS 379, Seiten 380–389. Springer, 1989.

[Ni96] N. Nisan. Extracting randomness: How and why (a survey). In *Proceedings 11th IEEE Conference on Computational Complexity*, Seiten 44–58, 1996.

[P94] A. Pedrotti. Analysis of a list-update strategy. *Information Processing Letters*, 52:115–121, 1994.

[RWS94] J. Westbrook, N. Reingold, und D. Sleator. Randomized competitive algorithms for the list update problem. *Algorithmica*, 11:15–31, 1994.

[OOW93] E. J. O'Neil, P. E. O'Neil, und G. Weikum. The LRU-K page replacement algorithm for database disk buffering. In *Proceedings of the 1993 ACM International Conference on Management of Data*, Seiten 297–306, 1993.

[OOW98] E. J. O'Neil, P. E. O'Neil, und G. Weikum. An optimality proof of the LRU-K page replacement algorithm. *Journal of the ACM*, 45, 1998.

[OHM90] B. J. Oommen, E. R. Hansen, und J. I. Munro. Deterministic optimal and expedient move-to-rear list organizing strategies. *Theoretical Computer Science*, 74:183–197, 1990.

[ON93] B. J. Oommen und D. T. H. Ng. An optimal absorbing list organization strategy with constant memory requirements. *Theoretical Computer Science*, 119:355–361, 1993.

[OD97] J. Oommen und J. Dong. Generalized swap-with-parent schemes for self-organizing sequential linear lists. In *International Symposium on Algorithms and Computation (ISAAC)*, LNCS 1350, Seiten 414–423, 1997.

[P91] R. M. Phatarfod. On the matrix occuring in a linear search problem. *Journal of Applied Probability*, 28:335–346, 1991.

[P94] R. M. Phatarfod. On the transition probabilities of the move-to-front scheme. *Journal of Applied Probability*, 31:570–574, 1994.

[PT80] R. I. Phelps und L. C. Thomas. On optimal performance in self-organizing paging algorithms. *Journal Inf. Optimization Science*, 1:80–93, 1980.

[P92] S. M. Pincus. Approximating markov chains. *Proceedings of the National Academy of Sciences of the USA*, 89:4432–4436, 1992.

[RS89] P. Raghavan und M. Snir. Memory versus randomization in on-line algorithms. In *International Colloquium on Algorithms, Languages and Programming (ICALP)*, LNCS 372, Seiten 687–703. Springer, 1989.

[RW90] N. Reingold und J. Westbrook. Optimum off-line algorithms for the list update problem. Technischer Bericht YALEU/DCS/TR-805, Yale University, 1990.

[R98] M. Reinstädtler. Verlustfreie Datenkompression mit selbstorganisierenden Listen. Diplomarbeit, Universität des Saarlandes, 1998.

[R61] A. Rényi. On measures of entropy and information. In *Proceedings 4th Berkeley Symp. Math. Statist. Probab.*, Seiten 547–561, 1961.

[R92] D. Richards. Studies of self-organizing placement algorithms for linear and finger lists. *Information Sciences*, 63:153–172, 1992.

[R79] J. Riordan. *Combinatorial Identities*. R. Krieger Publishing, 1979.

[R76] R. Rivest. On self-organizing sequential search heuristics. *Communications of the ACM*, 19(2):63–67, 1976.

[R95] J. S. Rosenthal. Convergence rates of Markov chains. *SIAM Review*, 37:387–405, 1995.

[Sa98] K. Sadakane. A Fast Algorithm for Making Suffix Arrays and for Burrows-Wheeler Transformation. In *Data Compression Conference (DCC)*, Seiten 129–138, 1998.

[SS96] F. Schulz und E. Schömer. Self-organizing data structures with dependent accesses. In *International Colloquium on Algorithms, Languages and Programming (ICALP)*, LNCS 1099, Seiten 526–537. Springer, 1996.

[S98] F. Schulz. Two new families of list update algorithms. In *International Symposium on Algorithms and Computation (ISAAC)*, LNCS 1533, Seiten 99–108. Springer, 1998.

[SF96] R. Sedgewick und Ph. Flajolet. *Analysis of Algorithms*. Addison-Wesley, 1996.

[S97] R. Seidel. Convex Hull Computations. In J. E. Goodman und J. O'Rourke, Hrsg., *Handbook of Discrete and Computational Geometry*. CRC Press, 1997.

[S48] C. E. Shannon. Mathematical theory of communication. *Bell System Technical Journal*, 27:379–423, 1948.

[ST72] G. S. Shedler und C. Tung. Locality in page reference strings. *SIAM Journal of Computing*, 1:218–241, 1972.

[ST85] D. D. Sleator und R. E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28:202–208, 1985.

[ST85a] D. D. Sleator und R. E. Tarjan. Self-adjusting binary search trees. *Journal of the ACM*, 32:652–686, 1985.

[S56] W. E. Smith. *Naval Research Logistics Quarterly*, 3:59–66, 1956.

[S99] J.-B. Son. Konvexe Hüllen in erwarteter Linearzeit. Diplomarbeit, Universität des Saarlandes, 1999.

[Sz88] W. Szpankowski. The evaluation of an alternating sum with applications to the analysis of some data structures. *Information Processing Letters*, 28:13–19, 1988.

[Sz98] W. Szpankowski. Techniques of average case analysis of algorithms. In M. Atallah, Hrsg., *Handbook of Algorithms and Theory of Computation*. CRC Press, 1998.

[T93] B. Teia. A lower bound for randomized list update algorithms. *Information Processing Letters*, 47:5–9, 1993.

[TN82] A. M. Tenenbaum und R. M. Nemes. Two spectra of self-organizing sequential search algorithms. *SIAM Journal of Computing*, 11:557–566, 1982.

[T63] M. L. Tsetlin. Finite automata and models of simple forms of behaviour. *Russian Mathematics Surveys*, 18:1–27, 1963.

[VO93] R. S. Valiveti und B. J. Oommen. Self-organizing doubly-linked lists. *Journal of Algorithms*, 14:88–114, 1993.

[WB] I. H. Witten und T. Bell. The calgary/canterbury text compression corpus. Anonymous FTP von ftp.cpsc.ucalgary.ca: /pub/projects/text.compression.corpus/text.compression.corpus.tar.Z .

[XZL98] Z.-B. Xu, J.S. Zhang und Y.-W. Leung. An approximate algorithm for computing multidimensional convex hulls. *Applied Mathematics and Computation*, 94:193–226, 1998.

[Y98] N. E. Young. Bounding the diffuse adversary. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 1998.

[Z96] E. Zeidler, Hrsg. *Teubner-Taschenbuch der Mathematik*. Teubner Verlag, 1996.

[Z49] G. K. Zipf. *Human Behaviour and the Principle of Least Effort*. Addison-Wesley, Cambridge, 1949.

Index

Absorbierende Verfahren 65
ABSTEIGENDE-HÄUFIGKEIT 11
Amortisierte Analyse 31
ARRAY 98
Austausch
  bezahlter 8
  freier 8
Average-Case Analyse
  siehe mittlere Analyse

Batched(k)-Heuristik 15
bidirektionale Suche 89
BIT 12, 76, 86
Burrows-Wheeler-Transformation 110

Calgary/Canterbury Compression Corpus 105
Chernoff-Schranke 40
COMBINATION, COMB 12
COUNTER-MOVE-TO-FRONT, CMTF 83
Coupon Collector Problem 69

Datenkompression 101
deterministische Verfahren 10
digitaler Suchbaum 140
DMTF 89
Double Dixie Cup Problem 69

Elias-Kodes 102
Entropie 124
  einer Markov-Kette 119
  empirische 104
Erst-Eintritts-Zeit 151

FREQUENCY-COUNT 11
Fundamentalmatrix 152

gedächtnislose Verfahren 10
gewichtete Zugriffskosten 83

harmonische Zahlen 47
Huffman-Algorithmus 126

$(i-1)$-Kostenmodell 8

k-Mal-Hintereinander-Strategien 15
k-PLÄTZE-VORWÄRTS-Regeln 11, 22
Kodes 123
  alphabetische (ordnungserhaltende) 131
kompetitive Analyse 9
Konvergenzgeschwindigkeit 57
konvexe Hülle 113

Listenfaktorisierung 30
Listenproblem 7
Lokalität 118
  blockweise 155
  elementweise 154
Lotka-Verteilung 150

MAPLE 42, 140
Markov-Kette 150
  ergodische (*ergodic*) 152
  rückwärtslaufende (*time-reversed*) 153
  absorbierende (*absorbing*) 152
  erweiterte (*extended*) 22
  expandierte (*expanded*) 17
  zeit-umkehrbare (*time-reversible*) 153
  partitionierbare (*lumpable*) 23
Mean first-passage-time 151
mehrfache Listen 95
MEHRFACHLISTEN-RAND, MRAND 97
mittlere Analyse 9
MOVE-FORWARD 10
MOVE-FRACTION(d), MF(d) 11
MOVE-TO-FRONT, MTF 7, 10
MOVE-TO-RECENT-ITEM(k), MRI(k) 12

MOVE-TO-ROOT, MTR 122
MOVE-UP(k), MU(k) 11, 22

NACH-VORNE-SCHIEBEN-Regel 7, 10

Offline-Algorithmus 9
Online-Algorithmus 9
Overwork 57

paarweise Unabhängigkeit 29
POS(k) 22
Projektion auf Paare 30

Randomisierte Verfahren 12, 73
RANDOM MOVE-TO-FRONT, RMTF) 83
RANDOM MOVE-TO-FRONT(p), RMTF(p) 12, 74
Rényi-Entropie 124

Simple(k)-Heuristik 15
Suchbaum 130
- adaptiver 122
- digitaler 140
- externe Pfadlänge 135
- optimaler 134
- Tiefe 131
- zufälliger 135

SWITCH(k) 22
SORT-BY-DELAY(k), SBD(k) 27
SORT-BY-RANK(α), SBR(α) 27
SORT-BY-TIME (q), SBT(q) 28
SORTIERE-NACH-RANG(α), SBR(α) 27
SORTIERE-NACH-ZEIT(q), SBT(q) 28
SORTIERE-NACH-ZUGRIFF(k), SBD(k) 27

Tabu-Menge 14, 151
TIMESTAMP, TS 11
Trägheit 118
TRANSPOSITION, TR 10
Trie 140

Ungleichung von Jensen 103
Ungleichung von Kraft 123
Ungleichung von Hilbert 41

verallgemeinerte Zipf-Verteilung 45, 150

VERTAUSCHEN-Regel 10
VORWÄRTS-Regeln 22

Zipf-Verteilung 45, 149
ZUR-WURZEL-ROTIEREN-Regel 122