FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Anwendung von kombinatorischen Optimierungsmethoden zur Rekonstruktion von in Streifen geschnittenen Papierdokumenten

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering & Internet Computing

eingereicht von

Thomas Winkler, BSc
Matrikelnummer 9450254

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung
Betreuer: ao. Univ.-Prof. Dipl.-Ing. Dr. techn. Günther Raidl
Mitwirkung: Univ.-Ass. Dipl.-Ing. Christian Schauer
Mag. Dipl.-Ing. Dr. Matthias Prandtstetter, AIT Mobility, DTS

Wien, 27.07.2011

(Unterschrift Verfasser) (Unterschrift Betreuer)

Erklärung zur Verfassung der Arbeit

Thomas Winkler

„Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe."

Wien, 27.07.2011 _____

(Unterschrift Verfasser)

Abstract

This thesis addresses the issue of reconstructing strip shredded text documents. This reconstruction process of such pages is interpreted as a combinatorial optimization problem, whereat the individual strips must be placed in their original order.

First, some approaches identifying the orientation of each strip will be presented. Based on the area between lines of text but also with a simple recognition technique for some distinctive characters it is tried to assign the correct orientation. Furthermore, a method for detecting the margin—i.e., those two strips on the left and right side of the original document—is applied for better readability and to increase the quality of the solution.

In this work the reconstruction is performed by using a variable neighborhood search. Additionally different methods will be discussed to further improve the reconstruction process, like grouping strips into blocks that are already in their correct order. Three different approaches are described and tested.

As benchmark instances ten documents each cut with four different strip widths were used to test the three approaches. It turns out that with more than a quarter of the test instances a complete and with more than half of the instances a readable reconstruction was possible.

Zusammenfassung

Diese Diplomarbeit behandelt das Thema der Rekonstruktion von in Streifen geschnittenen Textdokumenten. Das Wiederherstellen solcher Seiten wird als ein kombinatorisches Optimierungsproblem interpretiert, bei dem die einzelnen Streifen in die ursprüngliche Reihenfolge gebracht werden müssen.

Zuerst werden einige Ansätze zur Behandlung des Problems der unbekannten Orientierung der einzelnen Streifen vorgestellt. Anhand des Bereichs zwischen den Textzeilen aber auch mit einer einfachen Erkennung von markanten Buchstaben wird versucht eine Zuordnung der Orientierung zu bewerkstelligen. Weiters wird eine Methode zur Erkennung der Randstücke – also jene beiden Streifen, die sich im Original links und rechts auf der Seite befunden haben – angewandt, um die Lesbarkeit und die Qualität der Lösung zu erhöhen.

Im Zuge dieser Arbeit wird die Rekonstruktion mittels einer variablen Nachbarschaftssuche durchgeführt. Es werden zusätzlich unterschiedliche Methoden gezeigt, wie man bei der Rekonstruktion eine Verbesserung erhält, indem man die bereits richtig angeordneten Streifen zu Blöcken zusammenfasst. Drei unterschiedliche Ansätze werden beschrieben und getestet.

Als Testinstanzen werden zehn Dokumente mit je vier verschiedenen Streifenbreiten zerschnitten und die drei Ansätze darauf getestet. Es zeigt sich, dass bei einem Viertel aller Testinstanzen eine vollständige und bei mehr als der Hälfte der Testinstanzen eine lesbare Rekonstruktion möglich ist.

Danksagung

Ich danke allen, die so lange Geduld mit mir hatten, jenen, die mich angetrieben haben, jenen, die gut fanden, was ich tat, jenen, die mich motiviert haben und vor allem jenen, die mich inspiriert haben weiterzumachen.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Bei der Aufbewahrung sensibler Daten und Informationen über Personen und Unternehmen ist nach einer längeren Archivierung der Informationen in Papierform am Ende die Vernichtung des Papiers vorgesehen. Dabei ist es von größtem Interesse, dass die entsorgten Papierseiten nicht einfach von Unbefugten gefunden und gelesen werden können. Deswegen werden diese Dokumente üblicherweise vor der Entsorgung auf spezielle Art unkenntlich gemacht beziehungsweise zerstört.

Eine der häufigsten Möglichkeiten, die genützt wird, um zu entsorgende Papierseiten vor einem unerwünschten Zugriff zu schützen, ist ein *Schredder*. Ein Schredder ist eine Maschine, die eine oder mehrere Seiten Papier durch mehrere Schnitte in einzelne Fragmente aus Papier zerkleinert. Man findet unterschiedliche Formen von Schreddern, wobei sich je nach Typ die Größe der Schnipsel unterscheidet. In der DIN 32757 sind unterschiedliche Sicherheitsstufen definiert: Sicherheitsstufe 1: 12 mm Streifen, Sicherheitsstufe 2: 6 mm Streifen, Sicherheitsstufe 3: 2 mm Streifen, Sicherheitsstufe 4: 2 mm breite und maximal 15 mm lange Partikel (30 mm^2), Sicherheitsstufe 5: 0,8 mm breite und maximal 15 mm lange Partikel (12 mm^2). Grundsätzlich gliedern sich alle Arten von Schredder in zwei Grundklassen, nämlich:

- Der Streifenschnitt-Schredder schneidet das Papier vertikal in lange dünne Streifen.

- Der Kreuzschnitt-Schredder schneidet das Papier zuerst vertikal in Streifen und danach werden die entstandenen länglichen Streifen noch mit horizontalen Schnitten zusätzlich in kleinere Schnipsel zerschnitten.

Der Streifenschnitt-Schredder kann eine größere Menge an Papier mit weniger Aufwand verarbeiten und ist zudem meist der billigere Typ. Der Kreuzschnitt-Schredder ist teurer, aber dafür sind die zerstörten Dokumente schwerer wieder herzustellen und bietet somit eine höhere Sicherheit. In der Abbildung 1.1 sind die Schneidewerkzeuge eines Streifenschnittschredders dargestellt und die Abbildung 1.2 zeigt, wie eine Seite Papier aussieht, nachdem sie aus diesem Schredder kommt.

Abbildung 1.1: Die Schneidevorrichtung eines Streifenschnitt Schredders

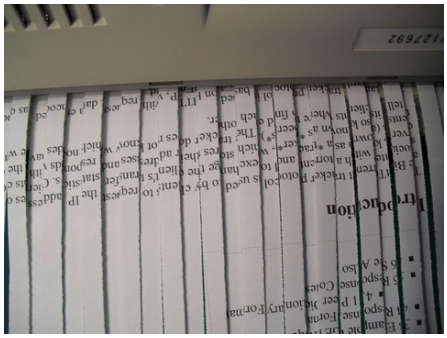


Abbildung 1.2: Ein Blatt Papier aus einem Schredder kommend

Zusätzlich zur maschinellen Zerstörung von Dokumenten durch einen Schredder sind auch andere Arten der Aktenvernichtung möglich, etwa durch händisches Zerreißen. Der Vorteil dabei ist, dass man keine Maschinen benötigt. Allerdings ist die Qualität der Unkenntlichmachung nicht so hoch, da im Vergleich zum Schredder oft eine geringere Anzahl an Schnipseln entsteht. Außerdem sind händisch zerrissene Schnipsel einfacher wiederherzustellen, da sie ungleichmäßig sind und man diese Eigenschaft bei der Rekonstruktion ausnützen könnte.

Eine andere Möglichkeit bietet das Verbrennen des Papiers, welche vermutlich die sicherste Methode der Vernichtung ist. Allerdings ist diese Variante zum einen mit einem hohen organisatorischen Aufwand verbunden (Ofen, Kamin, etc.) und zum anderen sollte man zusätzlich nach dem Verbrennen die

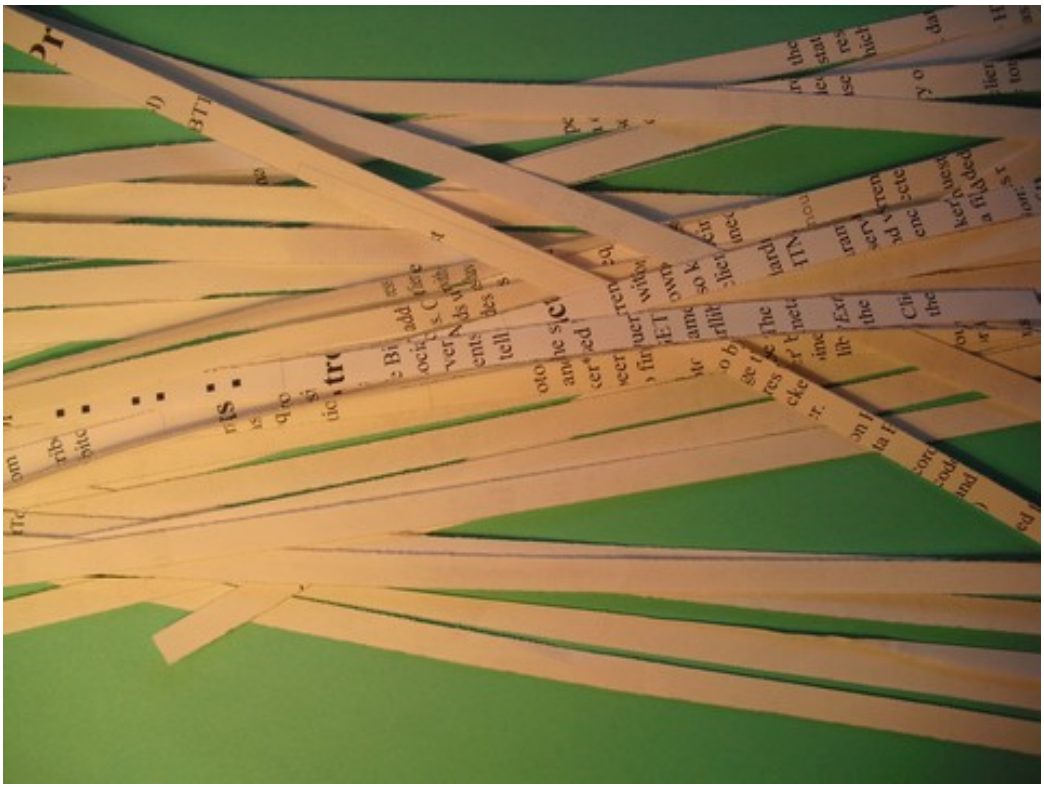


Abbildung 1.3: Eine in ungeordnete Streifen geschnittene Seite Papier

zurückbleibenden Aschereste noch Kontrollieren und gegebenenfalls „Durchmischen“, denn solange das ehemalige Blatt Papier nicht zerfallen ist, kann man den Text oft noch einwandfrei lesen.

Hat man die Seiten eines Dokuments mittels eines Schredders in Streifen geschnitten (und dieses eventuell unbeabsichtigt oder vorschnell getan beziehungsweise jemand versucht Informationen zu verheimlichen indem er Akten mit einem Schredder zerschneidet), so kann es auch vorkommen, dass man die auf diesen Papierseiten enthaltenen Informationen doch wieder benötigt und man möchte diese doch wieder zusammensetzen.

Leider werden die Streifen allerdings nicht geordnet sein, wie in der Abbildung 1.2 ersichtlich, sondern zufällig durchmischt, wie in der Abbildung 1.3 zu sehen. Dreht man diese Streifen, sodass sie alle gleich orientiert sind, und legt sie horizontal nebeneinander, so erhält man ein Bild wie in der Abbil-

dung 1.4 (hierbei handelt es sich um ein Bild, das von der im Rahmen der Diplomarbeit entwickelten Anwendungssoftware erstellt wurde).

Historisch betrachtet lässt sich feststellen, dass die Wiederherstellung von zerschnittenen Papierseiten schon seit geraumer Zeit von großem Interesse ist. Der Einsatz von Werkzeugen zur Rekonstruktion von zerstörten Dokumenten muss dabei nicht zwangsläufig in einem ethisch fragwürdigen Bereich stattfinden – wie zum Beispiel Spionagetätigkeiten – sondern wird durchaus auch in legalen Bereichen eingesetzt. Vor allem im Bereich der Forensik hat die Wiederherstellung von Dokumenten eine besonders wichtige Bedeutung. Zur Forensik zählen jene Arbeitsgebiete, die sich mit der Rekonstruktion und Analyse von kriminellen Handlungen befassen. Oft kann es in diesem Bereich zur Überprüfung von getätigten Aussagen von verdächtigen Personen notwendig sein, unbrauchbar gemachte Dokumente wiederherstellen zu können. Auch in der Archäologie findet man immer wieder Anwendungen, bei denen zerstörte Dokumente wieder hergestellt werden müssen, wie z.B. bei den Stasi-Akten [19]. Es kommt aber auch vor, dass Personen Zettel oder Geldscheine, die zwischen andere Akten geraten sind, unabsichtlich zerstört haben. Im einfachsten Fall hat jemand irrtümlich oder vorschnell eine Seite Papier in einen Schredder gesteckt. In jedem dieser Fälle möchte man das ursprüngliche Dokument wieder zusammensetzen.

Diese Arbeit befasst sich ausschließlich mit der Rekonstruktion von durch Streifenschnitt-Schreddern zerstörten Dokumentseiten. Um die Lesbarkeit zu erhöhen werden im Folgenden „Streifenschnitt-Schredder“ nur als Schredder bezeichnet.

Die Problematik beim automatischen Zusammensetzen mittels Computerprogrammen besteht darin, den Algorithmus ähnlich der menschlichen Vorgehensweise zu implementieren. Deshalb werden stattdessen auf eine vorher eindeutig bestimmte Weise gewisse Parameter überprüft, um daraus eine hinreichend genaue Lösung zu errechnen, die die Lesbarkeit des Textes gewährleistet. Während ein Mensch sich beim Betrachten von zwei Streifen häufig direkt darüber klar wird, ob sie gut zusammenpassen, muss man konkrete Algorithmen finden, um diese Aufgabe zu lösen. Dafür kann der Compu-

Abbildung 1.4: Eine in nebeneinander liegend angeordnete Streifen geschnittene Seite Papier.

ter in der gleichen Zeit, die ein Mensch benötigt nur zwei Streifen miteinander zu vergleichen, bereits einige hundert oder sogar tausende Streifen miteinander verglichen haben. Bei dem in dieser Arbeit vorgestellten Ansatz geht man bei der Analyse von zwei Streifen grundsätzlich so ähnlich vor, wie es ein Mensch auch tun würde. Indem man die Kanten der Streifen vergleicht, findet man heraus, ob die Streifen an allen Stellen zusammen passen. So die Person des Lesens und der Schrift mächtig ist, die sich auf dem Papier befindet, wird diese eindeutig Buchstaben, Wörter oder sogar Sätze identifizieren können. Eventuell wird der menschliche Zusammensetzer auch konkret nach gewissen Buchstaben suchen, wenn er eventuell eindeutige Anfänge von Wörtern auf dem Streifen identifizieren kann. Des Weiteren kann man anhand der Buchstaben in Wörtern, sofern man diese kennt, immer eindeutig entscheiden, ob der Streifen in der korrekten Orientierung vorliegt. Bei dem beschriebenen Ansatz untersucht man die Kanten der einzelnen Streifen und vergleicht die entlang der Kante liegenden Bildpunkte (des einen Streifens) mit jenen Bildpunkten, die an der Kante des anderen Streifens liegen. Das Ergebnis des Vergleichs wird errechnet, indem der Wert erhöht wird, sobald der Farbwert eines Punktes entlang der Kante nicht mit dem Farbwert des Punktes seines Nachbarstreifens in derselben Zeile übereinstimmt. Die Funktion, die dieses Ergebnis für zwei Streifen errechnet, wird *Bewertungsfunktion* genannt. Mit ihrer Hilfe wird die Qualität definiert, mit der zwei Streifen an den jeweiligen Kanten zueinander passen. Zwei Streifen, bei denen die jeweils verglichenen Kanten genau ident mit den zum Vergleich herangezogenen Farbwerten sind, liefern bei dieser Art der Bewertungsfunktionen einen Wert von 0. Je mehr Punkte nicht mit jenen ihrer Nachbarstreifen übereinstimmen, umso höher fällt das Ergebnis der Bewertungsfunktion aus.

Von den vier Seiten eines rechteckigen Streifens, sind bei einem Schredder die beiden längeren Kanten immer absolut eindeutig erkennbar, da sie um ein Vielfaches größer sind, als die beiden kürzeren (sich gegenüberliegenden) Seiten. In jedem Fall kann man annehmen, dass sich die Schnitte orthogonal zu den Textzeilen auf dem Papier befinden, egal ob das Blatt im Hoch- oder Querformat geschreddert wurde (wodurch sich im letzteren Fall die Anzahl

der Streifen erhöhen und die Länge der Streifen verringern würde). Daher können von den vier Möglichkeiten, die Streifen zu orientieren immer zwei direkt ausgeschlossen werden (nämlich die beiden Möglichkeiten, bei denen sich die beiden kurzen Seiten des Streifens links und rechts befinden würden). Somit kann die Orientierung jedes Streifens nur einer der zwei restlichen möglichen entsprechen: auf dem Kopf stehend oder richtig orientiert, also der Originalorientierung des Dokuments folgend.

Diese Tatsache ist bei einem Schredder mit anderer Schnittform, wie zum Beispiel bei Kreuzschnitt, nicht mehr so eindeutig vorhanden, da dort die geometrischen Abmessungen in Richtung quadratisches Fragment gehen könnten.

Daher kann man die Orientierung anfangs an den geometrischen Abmessungen des Streifens erkennen. Weiters kann der Algorithmus aber auch versuchen, die Orientierung der Streifen zu erkennen, indem er versucht, einzelne Buchstaben zu identifizieren oder Zeilenabstände zu erkennen. Diese beiden Möglichkeiten werden auch in meiner Arbeit behandelt.

Das *Ziel der Rekonstruktion* ist es, das ursprüngliche Dokument am Ende wieder herzustellen. Dazu müssen alle Streifen richtig nebeneinander und jeweils in der richtigen Orientierung liegen, damit es möglich wird, den Inhalt wieder lesen zu können beziehungsweise zu erkennen. Um dieses Ziel zu erreichen muss alle Streifen jeweils paarweise verglichen werden. Es ist nicht ausreichend nur den jeweils besten Streifen mit seinem vermeintlichen Nachbarn zusammenzulegen. Der Grund, warum dieses Vorgehen alleine nicht ausreichend ist, wird in den Abbildungen 1.5, 1.6 und 1.7 weiter unten beschrieben. Um die optimale und richtige Lösung zu finden, müssen alle Streifen immer in ihrer Gesamtheit untersucht werden. Dabei ist eine mögliche Lösung eine einer bestimmten Reihenfolge folgende Anordnung aller Streifen nebeneinander. Um nun zwei Lösungen miteinander vergleichen zu können, wird eine Bewertungsfunktion eingeführt, die sich primär aus der Summe aller Bewertungsfunktionswerte der entsprechenden Nachbarschaftsbeziehungen der Streifen ergibt. Die in Abbildung 1.5 dargestellten Pfeile und Zahlen sind das Ergebnis einer Bewertungsfunktion von drei Streifen A, B und C jeweils zueinander, die als gerichteter Graph dargestellt werden. Der Graph, der die

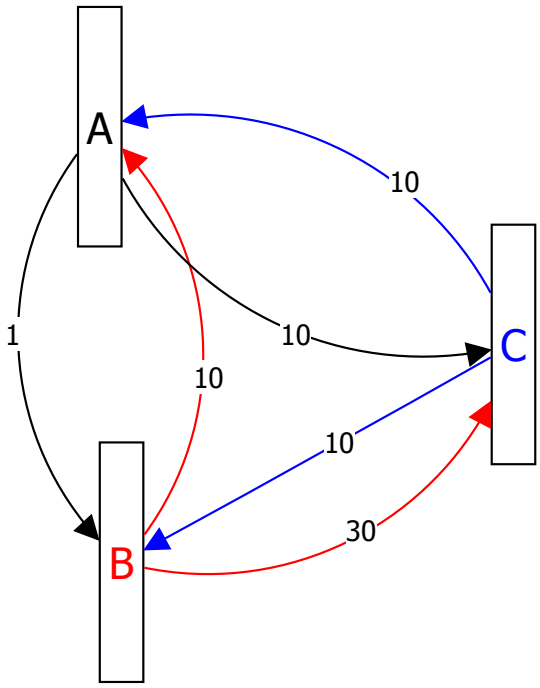


Abbildung 1.5: Graphische Darstellung der Ergebniswerte aus der Bewertungsfunktion von drei einzelnen Streifen A, B und C zueinander, dargestellt in Form eines gerichteten Graphen (wobei der Pfeil x nach y für den Wert steht, den die Bewertungsfunktion liefert, wenn x links von y platziert wird).

Verbindungen der Streifen zu den anderen in dieser Abbildung darstellt, zeigt das zu lösende Problem in einer alternativen Form; es wird der Weg, der alle Streifen enthält und die geringste Summe der Kanten ergibt, gesucht. Der Pfeil, der von A nach B zeigt (enthält den Wert 10), repräsentiert die Tatsache, dass A links neben B liegt, während der Pfeil von B nach A (der den Wert 1 enthält) die Situation B links neben A (oder alternativ A rechts neben B) darstellt.

Bildet man aus diesem Beispiel eine Lösung und verwendet dazu jeweils die besten Verbindungen untereinander, erhält man nicht automatisch die beste Lösung wie im Folgenden gezeigt wird: In Abbildung 1.6 sieht man, dass die Lösung $A \rightarrow B \rightarrow C$ nicht optimal ist, obwohl der beste passende Streifen neben A verwendet wurde – nämlich Streifen B mit dem Wert 1 –, in weiterer Folge neben Streifen B nur Streifen C (mit dem Wert 30) liegen kann und

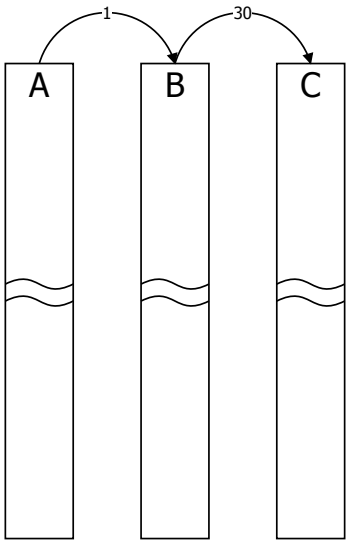


Abbildung 1.6: Die Summe aller Bewertungsfunktionen in dieser Lösung beträgt 31, obwohl sie die am besten zusammenpassenden Streifen A und B nebeneinander enthält.

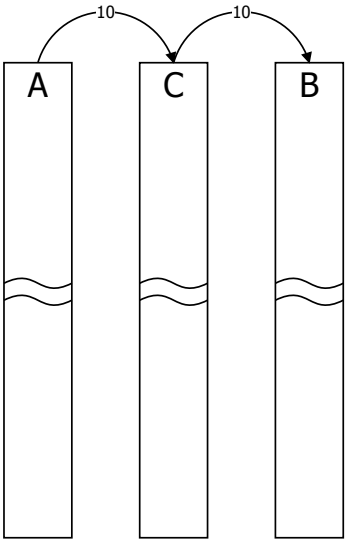


Abbildung 1.7: Diese Lösung wird mit der Summe 20 bewertet und ist somit besser (da der Wert geringer als 31 ist).

somit die gesamte Lösung mit $1+30 = 31$ bewertet wird, was nicht der besten Lösung entspricht, wie man im Weiteren sieht; denn in Abbildung 1.7 wird eine bessere Gesamtbewertung für die Lösung $A \to C \to B$ erzeugt, da man neben Streifen A den Streifen C (mit dem Wert 10) und neben Streifen C den Streifen B (der Wert ist ebenfalls 10) legt und dadurch einen Wert von $10 + 10 = 20$ erhält. Geringere Werte repräsentieren bessere Lösungen. Die zweite Lösung ist daher besser, obwohl dort neben Streifen A nicht der am besten passende Streifen platziert wurde. Dieses Beispiel soll verdeutlichen, dass der triviale Weg, einfach jene Streifen zusammenzulegen, die paarweise den kleinsten Wert bezogen auf die Bewertungsfunktion erzielen, in Summe nicht implizit den kleinsten Wert erzielen.

Die Anzahl der dadurch entstehenden möglichen Lösungen ist sehr groß (siehe ähnliches Problem unter [4]) und in den meisten Fällen können nicht alle

möglichen Lösungen untersucht werden. Die Aufgabe, die es (für den Computer) zu lösen gilt, wird als *kombinatorisches Optimierungsproblem* formuliert. Ziel dieser Arbeit ist es, Ansätze zu entwickeln, die am menschlichen Vorgehen bei der Rekonstruktion angelehnt sind, um so die Durchsuchung des Lösungsraumes auf die relevanten Teile zu beschränken und somit den benötigten Zeitaufwand möglichst gering zu halten.

Die Arbeit ist wie folgt aufgebaut:

Im folgenden Kapitel wird die Aufgabenstellung formal definiert und die Rahmenbedingungen der Problemstellung werden festgesetzt.

In Kapitel 3 werden andere Arbeiten zu diesem Thema vorgestellt sowie auf Relevanz in Bezug auf diese Arbeit eingegangen.

Kapitel 4 gibt einen Überblick über eine Auswahl an klassischen Methoden zum Lösen von kombinatorischen Optimierungsproblemen.

Die in dieser Arbeit verfolgten Lösungsansätze werden in Kapitel 5 diskutiert, wobei auch auf Details zur Implementierung der entwickelten Algorithmen eingegangen wird.

In Kapitel 6 wird konkret auf die entwickelte Applikation und die Implementierung eingegangen und diese erklärt.

Die Ergebnisse der Tests werden in Kapitel 7 präsentiert und analysiert.

Kapitel 2

Problembeschreibung

Ziel dieser Arbeit ist die Untersuchung möglicher Methoden zur Rekonstruktion von in Streifen geschnittenem Papier. Um dieses Rekonstruktion zu bewerkstelligen, wird unter anderem zur Orientierungserkennung eine stark vereinfachte Buchstabenerkennung verwendet, um die tatsächlich richtige Orientierung des Originaldokuments wieder herstellen zu können, sowie einen Algorithmus, um die gleichmäßige Orientierung aller Streifen anhand der Textzeilenzwischenräume zu erkennen und zu optimieren. Neben einer herkömmlichen variablen Nachbarschaftssuche mit Verschiebe- und Rotationsoperationen wird zusätzlich eine Bildung von Blöcken von Streifen in der Nachbarschaftssuche verwendet, um diese Operationen effektiver zu machen.

Die Rekonstruktion von zerstörten Dokumenten ist ein vielseitiges Problem und umfasst auch den interdisziplinären Bereich der Mustererkennung. Darum ist es notwendig einige Einschränkungen zu definieren um sich auf eine bestimmte Art von Problemstellung konzentrieren zu können. Weiters wird die eigentliche Aufgabe, die es zu erledigen gilt, definiert und formal notiert.

2.1 Limitierungen

Wie schon eingangs erwähnt, gibt es mehrere Arten der Zerstörung von Seiten gibt. Diese Diplomarbeit beschäftigt sich ausschließlich mit der Rekonstruk-

tion von Papierseiten, auf denen sich maschinengeschriebener Text befindet, die von einem Schredder zerschnitten wurden.

Der Aufwand, der notwendig ist, die Streifen einzuscannen und deren optische Parameter zu optimieren ist sehr groß. Außerdem fallen diese Verfahren vor allem in die Bereiche der Mustererkennung und des Bildverstehens. Daher werden aus Gründen der Vereinfachung, die Daten der Streifen künstlich erzeugt. Damit ist gemeint, dass eine vollständig vorhandene Seite eines Dokuments, die zum Beispiel als PDF oder Bitmap Datei vorliegt, nicht wirklich in einem Schredder in Streifen geschnitten wird, sondern der Vorgang, das Bild in mehrere Segmente aufzuteilen, mittels Computerprogramms ausgeführt wird. Im Weiteren werden diese künstlichen Streifen auch als *virtuell* erzeugte Streifen bezeichnet.

Die Tatsache, dass es sich bei den zu rekonstruierenden Seiten nur um virtuell zerschnittene Streifen handelt, ändert aber an den Funktionen und Ergebnissen, der hier untersuchten Algorithmen nichts wesentliches. Dies wurde von Ukovich und Ramponi in einem Versuch getestet [23]. Dort wurden sowohl physikalisch real existierende geschredderte Seiten Papier wie auch virtuell in Streifen geteilte Seiten, die zuvor eingescannt wurden, in einem Test verwendet. Die Autoren kommen zu dem Ergebnis, dass beide Arten von Streifen gleichermaßen verwendet werden können und keinen relevanten Einfluss auf die Ergebnisse der verwendeten Algorithmen zur Rekonstruktion haben. Somit macht es für den Prozess des Zusammensetzens keinen merkbaren Unterschied, ob man künstlich generierte oder real geschnittene Streifen als Basis verwendet.

Des Weiteren treffe ich einige Annahmen, die im Normalfall nicht immer so (einfach) anzutreffen wären, um die Aufgabenstellung in einen bewältigbaren Rahmen zu bringen.

- Es wird angenommen, dass der durch den Schnitt entstehende Spalt vernachlässigbar klein ist, was bedeutet, dass es keinen Verlust an Information an den Schnittkanten gibt. Da es sich um simulierte Schnitte handelt, werden die Streifen verlustfrei durch einen „Schnitt“ zwischen

zwei Bildpunkten erzeugt. In einer realen Situation müsste man sich eventuell damit auseinandersetzen, dass an den Schnittkanten ein Spalt in dem zusammengesetzten Ergebnis entstanden ist. Dies wird bei Balme [1] betrachtet, die in ihrer Arbeit die Umstände behandelt, dass bei ausgefransten Schnittkanten nicht die jeweils außen liegenden Spalten zum Vergleich herangezogen werden können oder sollten. Stattdessen wird die zweite oder dritte Bildpunktspalte des eingescannten Streifens verwendet.

- Alle Streifen sind gleich hoch. Das bedeutet, es existiert ein Blatt Papier mit einer beliebigen Höhe und alle Streifen, die aus diesem Blatt entstehen, sind von gleicher Höhe. Die Streifenbreite ist nicht notwendigerweise für alle Streifen gleich. Jedoch bietet diese keine Information, die in dieser Arbeit weiterverwertet wird, da im Normalfall in einer realen Situation alle Streifen immer gleich breit sein werden. Sonst würde aufgrund der Information, dass zum Beispiel jeder zweite Streifen breiter als die anderen ist, ein bedeutender Vorteil bei der Rekonstruktion entstehen, der begünstigend genützt werden könnte. Somit wäre ein Schredder, der unterschiedlich breite Streifen erzeugt, wesentlich unsicherer und man kann annehmen, dass eine solche Art Schredder kaum bis gar keine Anwendung finden würde. Dasselbe gilt für die Schnittwinkel der Streifen: es wird angenommen, dass alle Streifen „echte“ Rechtecke sind und alle vier Ecken aus 90° Winkel bestehen. Würden die Streifen einen von einem rechten Winkel erkennbar unterschiedlichen Schnittwinkel haben, dann kann diese Information dazu genützt werden die Vorder- und Rückseiten zu erkennen, wie es zum Beispiel in der Abbildung 2.2 gezeigt wird.

- Es sind alle Streifen eines Blattes, die Informationen enthalten, vorhanden. Die leeren Streifen, die üblicherweise an den beiden Außenrändern einer Seite entstehen, werden vor der Rekonstruktion explizit entfernt, da leere Streifen keine Information enthalten. Diese können, auch dann weggelassen werden, wenn sie nicht am Rand sondern in der Mitte der Seite entstehen (beispielsweise zwischen zwei Spalten eines mehrspalti-

gen Artikels).

- Auch wird vorausgesetzt, dass sich der geschriebene Text immer normal auf die Schnittkante befindet. Ein Text, dessen Ausrichtung parallel zur Schnittkante läge – wie das bei einem Blatt Papier, das im Querformat bedruckt ist, aber im Längsformat geschreddert worden ist, der Fall wäre – könnte ohne besondere Hilfsmittel gelesen werden. Selbst in den seltenen Fällen, wenn ein Schnitt genau durch eine Textzeile durchgeht, kann man im Selbsttest[1] sehr einfach feststellen, dass ohne zusätzliche Hilfsmittel ein problemloses Lesen der Zeile möglich ist. Die Rekonstruktion würde in so einem Fall eher anhand des Textinhaltes erfolgen, und man würde dann also versuchen die Satzteile wieder in der richtigen Anordnung zusammenzusetzen.

  Weiters wird auch angenommen, dass sich der Text nicht schräg auf dem Blatt Papier befindet.

- Die Orientierung der Streifen ist nicht bekannt[2]; das heißt, jeder Streifen ist entweder richtig orientiert oder er steht „auf dem Kopf“, was einer Drehung um 180° entspricht.

- Die Rückseite der Streifen wird nicht beachtet, beziehungsweise wird sie als leer angenommen, da man einen doppelseitig bedruckten Streifen Papier immer so darstellen kann, als ob der Streifen mit beiden Seiten in der doppelten Länge vorhanden wäre. Dazu würde man die Rückseite virtuell nach vorne klappen und die beiden Seiten würden sich an der gemeinsamen Oberkante (alternativ dazu an der Unterkante) „treffen“. Die Zusammengehörigkeit der Vorder- und Rückseite des Streifens bleibt durch ihre physikalische Verbundenheit eindeutig. Das gilt sowohl für virtuelle Streifen als auch für physikalisch existierende eingescannte Streifen. Die Abbildung 2.1 zeigt den Vorgang des virtuellen Auseinanderklappens. Nach dieser Umwandlung kann der Streifen

[1] Dazu muss man nur mit einem Blatt Papier einige Textzeilen zur Hälfte abdecken und testen, ob die Zeile trotzdem lesbar bleibt.

[2] Beim Zerschneiden werden die Streifen zufällig gedreht.

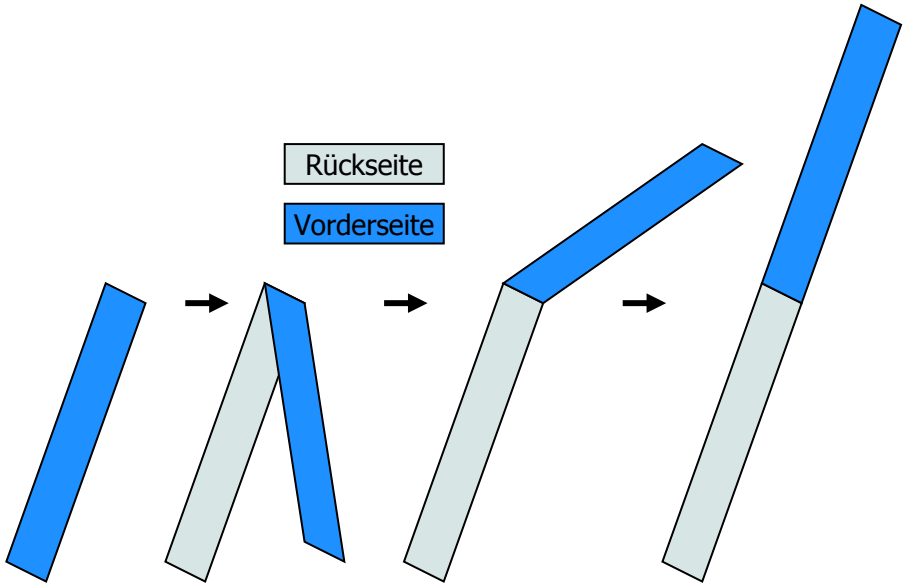


Abbildung 2.1: Umwandlung eines doppelseitig bedruckten Streifens in einen Streifen doppelter Länger, der die ehemalige Vorder- und Rückseite auf einer Seite hat.

als einseitige Version behandelt werden. Dadurch entsteht nach dem Zusammensetzen der Streifen ein Dokument, bei dem die Rückseite am Kopf über der Vorderseite steht (oder umgekehrt).

Zudem sind viele Dokumente einseitig gedruckt, da die Druckgeschwindigkeit dann höher ist und der langsamere doppelseitige Druck seltener verwendet wird.

Außerdem entstehen, da das Blatt Papier meistens händisch in den Schredder geschoben wird, oft keine absolut korrekten 90° Winkel zwischen Papierkante und Schnittkante. Anhand dieses Winkels kann man eindeutig auf die Vorder- beziehungsweise Rückseite schließen, wie dies in der Abbildung 2.2 zu ersehen ist. Zusätzlich erhält man durch schräge Schnitte auch eine Information über die Position der Streifen, die sich am Rand der Dokumentseite befinden müssen. Diese haben eine spe-

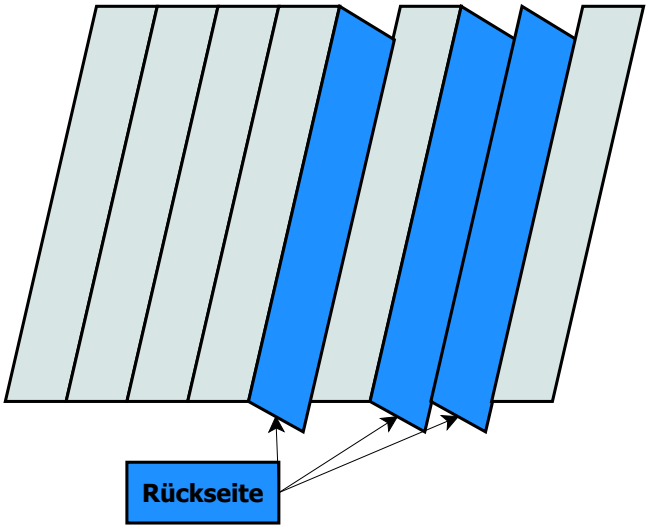


Abbildung 2.2: Darstellung der möglichen Zusammensetzung von schräg geschnittenen Streifen, wobei die Vorder- und Rückseite erkannt werden kann.

zielle Form, die sie von allen anderen Streifen eindeutig unterscheidet: Die Längskanten solcher Randstreifen sind nicht parallel. Die Abbildung 2.3 zeigt diesen Effekt, wobei in diesem Beispiel jeweils zwei Randstücke eine unterschiedliche Form aufweisen. Dies ist im Allgemeinen bei schrägen Schnittkanten für mindestens einen Streifen pro Rand der Fall; je schräger – also je mehr der Schnittwinkel von 90° abweicht – desto höher die Anzahl der Streifen, die eine eindeutig unterscheidbare Form erhalten. Dadurch lässt sich mehr Information aus der Form der Streifen erhalten.

2.2 Aufgabenstellung

Nachdem die Streifen aus einem digital vorliegenden Dokument virtuell erzeugt wurden, werden sie durch Zufallsgenerator vermischt und in einer willkürlichen Ausgangsposition angeordnet. Anschließend wird auch die Ori-

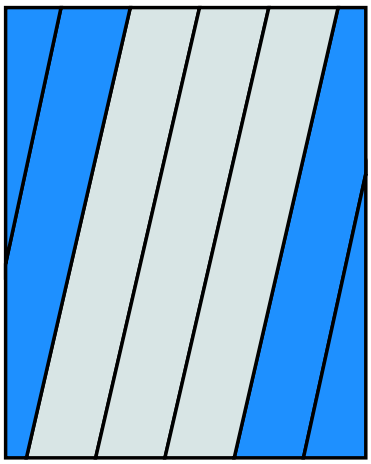


Abbildung 2.3: Randstücke haben bei schrägen Schnittlinien eine unterschiedliche Form im Vergleich zu allen anderen Streifen.

entierung zufällig gewählt. Die Aufgabe ist nun, diese zufällig angeordneten und zufällig orientierten Streifen wieder in die richtige Reihenfolge und Orientierung zu bringen, dass es dadurch möglich wird, den Inhalt der Seite wiederherzustellen und zu lesen. Selbstverständlich werden die Informationen, wie der Streifen ursprünglich orientiert war und an welcher Stelle sich der Streifen im Originaldokument befunden hat, bei der Rekonstruktion nicht verwendet. Allerdings steht diese Information zur abschließenden Bewertung der Korrektheit der Lösung im Testfall natürlich zur Verfügung.

2.3 Problemdefinition

Bei der Aufgabe der Rekonstruktion einer durch einen Schredder zerstörten Seite Papier hat man (gemäß der Limitierung aus 2.1) zu Beginn gleich große und rechtwinkelige Schnipsel Papier, die man als Streifen bezeichnet. Jeder Streifen hat die gleiche Höhe und eine – abhängig vom Schredder – gleiche Breite. Für die Höhe der Streifen l – später auch als Länge bezeichnet – gilt immer, dass $l \geq 1$ Bildpunkt ist. Auch für die Breite b jedes Streifens gilt $b \geq 1$ Bildpunkt. Abgesehen davon ist die Breite der Streifen in dieser Arbeit ohne weitere Bedeutung, da aus der Breite des Streifens keine Information

gewonnen wird.

Zur formalen Beschreibung der Aufgabenstellung ergeben sich die folgenden Definitionen, die im Wesentlichen aus der Doktorarbeit von Prandtstetter [21] übernommen wurden, wie folgt: Die zu lösende Aufgabe besteht darin, aus einer Menge $S = \{1, \ldots, n\}$ von n gleich großen Streifen, die nicht notwendigerweise von nur einem einzelnen Blatt Papier stammen müssen, sofern alle betrachteten Blätter die gleiche Höhe besitzen, die richtige Reihenfolge zu finden, in der die Streifen aneinander gereiht gehören, sowie die richtige Orientierung zu finden, in welche jeder einzelne Streifen gedreht gehört, um die ursprünglichen Seiten wiederherzustellen. Zudem muss jeder Streifen dabei genau einmal vorkommen.

Eine Lösung $x =< \pi, o >$ des Rekonstruktionsproblems besteht aus einer Permutation $\pi : S \to \{1, \ldots, n\}$ der Elemente der Menge aller Streifen S sowie aus einem Vektor $o =< o_1, \ldots, o_n > \in \{0, 1\}^n = O^n$ der die Orientierung der einzelnen Streifen $i \in S$ bestimmt:

$$o_i = \begin{cases} 0 & \text{der Streifen } i \text{ wird nicht gedreht,} \\ 1 & \text{der Streifen } i \text{ wird um } 180° \text{ gedreht.} \end{cases}$$

Weiters bezeichnet π_i, mit $i \in S$, die Position des Streifens i in Bezug auf die Permutation π. Im Gegenzug bezeichnen wir mit s_k den Streifen, der sich an der Position k, mit $1 \leq k \leq n$, befindet. Das heißt, $\pi_i = k \Leftrightarrow s_k = i$ mit $i \in S$ und $1 \leq k \leq n$.

Zusätzlich definieren wir eine Kostenfunktion, $c(i, j, \omega) \geq 0$, im Weiteren auch als Bewertungsfunktion bezeichnet, die darüber Aufschluss geben soll, welcher Fehler gemacht wird – also wie viele gegenüberliegende Bildpunkte nicht zusammenpassen –, wenn sich zwei Streifen i und j mit den jeweiligen Orientierungen aus $\omega = (o_i, o_j) \in O^2$ Seite an Seite befinden; wobei der Streifen i links und der Streifen j rechts im Dokument liegen. Diese Bewertungsfunktion liefert einen höheren Wert für $c(i, j, \omega)$, je mehr Unterschiede die beiden Streifen an den Vergleichspunkten aufweisen, und einen geringen Wert, je mehr gleichfärbige Punkte beim Vergleich (der Kanten) der beiden

Streifen gefunden werden. Die detailiertere Definition der Bewertungsfunktion befindet sich in Abschnitt 5.1.

Das Ziel der Rekonstruktion ist es, eine Lösung x zu finden – das heißt, eine Permutation und Orientierung aller Streifen – so dass die Summe $Z(x)$ der jeweils paarweisen Bewertungsfunktionsergebnissen der einzelnen Nachbarstreifen möglichst gering ausfällt:

$$Z(x) = \sum_{k=1}^{n-1} c(\pi_k, \pi_{k+1}, \omega)$$

2.4 Komplexitätsanalyse

An dieser Stelle muss man sich fragen, mit welchem Aufwand das Auffinden der einen Permutation und Orientierung, die dem ursprünglichen Dokument entspricht, verbunden ist. In [21] wurde bewiesen, dass das Problem der Rekonstruktion von in Streifen geschnittenen Dokumenten $\mathcal{NP}$-vollständig ist, indem das Problem des Handlungsreisenden [17] auf dieses Problem abgebildet wurde.

Die Klasse der $\mathcal{NP}$-vollständigen Probleme bildet einen Kernpunkt der Forschung im Bereich der kombinatorischen Optimierung. Es ist bisher kein Algorithmus mit polynomieller Laufzeit bekannt, der eines der Probleme aus dieser Klasse beweisbar optimal lösen kann. Das Interessante an diesem Problem ist, dass es eine große Menge an $\mathcal{NP}$-vollständigen Problemen gibt (zum Beispiel das Handlungsreisendenproblem). Würde aber ein polynomieller Algorithmus gefunden werden, der eines dieser Probleme exakt löst, dann kann dieser Algorithmus mit polynomiellem Aufwand auf alle anderen Probleme erweitert werden.

Um ein $\mathcal{NP}$-vollständiges Problem in akzeptabler Zeit zu lösen, greift man häufig auf heuristische Ansätze zurück, die zwar keine optimale Lösung mehr garantieren können, aber dafür mit entsprechend großen Probleminstanzen umgehen können.

Bei der Problemstellung dieser Diplomarbeit handelt es sich um ein kombinatorisches Problem; im Konkreten um ein Anordnungsproblem mit zusätzlichen Nebenbedingungen. Für die Anordnung von unterscheidbaren Objekten unter Beachtung der Reihenfolge, ist die Formel zur Berechnung von allen möglichen Permutation $P(n)$ einer Menge von n unterscheidbaren Objekten $P(n) = n!$, wobei die Definition von $j! = j \cdot (j-1)!$ mit $1! = 1$ ist.

Im schlechtesten Fall (*worst case*) müssen alle Möglichkeiten untersucht werden, sofern man nicht durch Ausschlussverfahren gewisse Gruppen von Lösungen weglassen kann, wobei man sich sicher sein muss, dass diese nicht die optimale (oder eine bessere als die bisher beste) Lösung enthalten. Daher müssten im schlechtesten Fall alle n Streifen in allen $n!$ Möglichkeiten positioniert und jeweils die Summe der Bewertungen der einzelnen Streifen gebildet werden. Dazu kommt, dass man – unabhängig von der Position eines Streifens – noch zwei Möglichkeiten für die Orientierung jedes Streifens hat. Daraus ergeben sich für alle n Streifen 2^n Möglichkeiten, diese zu orientieren. Insgesamt sind es somit $n! \cdot 2^n$ mögliche Lösungen, die untersucht werden müssen, sofern man keine der Lösungen ausschliessen kann.

Kapitel 3

Verwandte Arbeiten

Oft ist es von Vorteil einen Blick auf andere Arbeiten zu werfen, die sich ebenfalls mit einer ähnlichen Problematik beschäftigen. Da es bereits einige Literatur dazu gibt, die sich mit der Rekonstruktion von Papierstreifen beschäftigt, lohnt es sich, anzusehen welche Erkenntnisse sie gewonnen haben um eventuell ähnliche Methoden zu adaptieren oder zu verbessern.

De Smet, De Bock und Philips [5] befassen sich in ihrer Arbeit zur halbautomatischen Rekonstruktion mit in Streifen geschnittenen Dokumenten. Da die Autoren real zerschnittene Dokumentseiten behandeln, weisen sie unter anderem darauf hin, dass sich von den geometrischen Attributen der einzelnen Streifen gewisse Informationen ableiten lassen. Vor allem zu Beginn kann unter anderem aus der Biegung der Streifen, die beim Schneiden entsteht, ein Rückschluss auf die originale Orientierung gezogen werden. Auch die Erkennung einzelner Buchstaben mittels OCR (*Optical Character Recognition*) oder andere Muster erkennende Software nach dem initialen Einscannen der Streifen wird erwähnt.

Im Weiteren wird eine Kette von Schritten beschrieben, die die einzelnen Streifen bei der Rekonstruktion durchlaufen. Unter anderem werden Streifen ohne Information, also jene Streifen, die eine bestimmte Grenze von (nicht weißen) Bildpunkten nicht überschreitet, entfernt. Auch die Erkennung von Textzeilen zu binären Blöcken, wobei eine ganze Zeile jeweils zu 0 oder 1

konvergiert, und anschließendem Vergleich mit anderen Streifen anhand der entstehenden Blöcke um eine Bewertungsfunktion zu erhalten und die in weiterer Folge entstehende Erkennung von oberen und unteren Rändern wird behandelt.

Die Autoren [5] erklären außerdem, dass es sinnvoll ist, die Streifen ähnlich einem Puzzle in Gruppen einzuteilen, um dann nur jeweils die aus einer Gruppe untereinander zu vergleichen, damit so die Anzahl der Vergleiche der Streifen reduziert werden kann.

In einem Artikel über die Rekonstruktion von Dokumenten, bei der die äußere Form der einzelnen Teile des Dokuments gar nicht betrachtet wird, beschreibt Balme [1], dass die Informationen der einzelnen Streifen nur aus dem Inhalt kommen können. Sie sieht ihre Arbeit als Erweiterung zu Veröffentlichungen von Kosiba et al. [16] und Goldberg et al. [8], die einen automatischen Puzzle-Löser behandeln, bei dem es nur um die äußere Form der Teile geht. Sie beschreibt den Ansatz von De Smet [5] für Farbfotos als ungeeignet, da der *binary text line* Algorithmus, vorwiegend nur bei Textdokumenten funktioniert. Weiters meint sie auch, dass bei Kreuzschnittschreddern die Methoden von De Smet nicht zufrieden stellend funktionieren würden. Sie sieht auch die Notwendigkeit, die Streifen vor einer vollständigen Bewertung zueinander in Gruppen einzuteilen.

Mögliche Gruppen sind:

- Länge: Länge der Streifen, wobei hier immer davon ausgegangen wird, dass sich die Streifen aus mehreren verschiedenen Dokumenten stammen und nicht nur aus einem einzelnen Blatt erzeugt worden sind.
- Farbe: Da es bei ihrer Arbeit hauptsächlich um Farbfotos geht, setzt sie die Streifen so in Gruppen zusammen, dass sie annähernd den gleichen Grauwert der Farben entlang der Kantenabschnitte haben.
- Information: Schließlich werden die wertlosen Streifen, also jene die keine Information enthalten, entfernt.

Des Weiteren beschreibt Balme [1] die Tests der Streifen untereinander in der

jeweiligen Gruppe als schwierig, da sie von real eingescannten Dokumenten ausgegangen war. Sie löst das Problem der Schatten an den Rändern, die durch Scanner und Schredder entstehen können, indem sie die äußeren zwei Pixelreihen ignoriert und die dritte Reihe heranzieht. Die dadurch entstehende Verschiebung der zusammenpassenden Werte wird dann damit ausgeglichen, dass die jeweils angrenzenden Pixel auch in die Bewertungsfunktion einbezogen werden. Diese Idee wurde auch von Prandtstetter [21] zur Definition der Bewertungsfunktion aufgegriffen.

Skeoch [22] beschreibt in ihrer Arbeit vor allem die Rekonstruktion von reinen Farbfotos. Die Streifen werden anhand von Farbwerten entlang der Kanten verglichen und zusammengefügt. Sie geht davon aus, dass bei einem Farbfoto die Farbverläufe „langsam“ sind und somit der Unterschied der Farben an den Rändern der einzelnen Streifen eher gering ist. Sie verwendet dazu Distanzfunktionen, unter anderem die *euklidische Distanz* und die *Cosinus Distanz*.

Skeoch verwendet in ihrer Arbeit zu großen Teilen Methoden der Mustererkennung, da sie sich mit einem Projekt befasst, bei dem tatsächlich die physikalisch existierenden Streifen eingescannt, erkannt und rekonstruiert werden sollen. Da die Ausrichtung der Streifen beim Einscannen aber so gut wie immer unkorrekt ist, geht es bei Skeoch in erster Linie darum, die „Verdrehung“ der Streifen zu erkennen und richtig zu stellen: Dabei geht es um die Richtigstellung der Rotierung der Streifen um wenige Winkelgrade, während die Orientierungserkennung sich mit der Richtigstellung jener Streifen beschäftigt, die um 180° gedreht wurde, also auf dem Kopf stehen. Diese Orientierung wird dann in der Phase der Dokumentenrekonstruktion geprüft und korrigiert.

Auch sehr bemerkenswert ist der Ansatz einer Bewertungsfunktion, bei dem sie eine Markierung einführt, die die Qualität ihrer Bewertungsfunktion (*Fitness Evaluation*) anzeigt. Sie errechnet die angrenzenden Streifen, indem sie die Farben der einzelnen Streifenränder Pixel für Pixel miteinander vergleicht und bewertet. Aus dem Mittelwert aller Pixel des gesamten Streifenrandes erhält sie einen Wert zwischen im Bereich $[0 \ldots 1]$; diesen Wert präsentiert

sie optisch über der Lösung. Bemerkenswert ist das deswegen, weil man auf den ersten Blick, sofern man die richtige Zusammensetzung bereits kennt, die Qualität der Bewertungsfunktion ersehen kann. Für Details siehe [22, Seite 40ff] wo man die oben beschriebene grafische Präsentation der Qualität der Bewertungsfunktion sieht.

Skeoch verwendet evolutionäre Algorithmen als Optimierungsmethode des Problems der Rekonstruktion der Streifen zu einem endgültigen Bild. Sie behandelt noch die Themen, dass ein Streifen fehlt sowie in weiterer Folge doppelseitige Streifen und ebenfalls Streifen, die aus mehrere Dokumente entstanden sind auf einmal rekonstruiert werden. Sie stellt fest, dass die von ihr verwendeten Methoden, für synthetische Daten (also künstlich generierte Streifen) gut funktionieren und reale Streifen nicht in der selben Güte. Außerdem funktioniert ihre Rekonstruktion bei Text nicht optimal während Farbbilder mit besonders feinen Übergängen (wie in ihrem Beispiel ein Regenbogenbild) gut funktionieren.

Morandell [18] behandelt in seiner Arbeit die Zusammensetzung von in Streifen geschnittenen Dokumente, ohne dabei die Orientierung zu beachten. Er geht von künstlich erzeugten Daten aus, also jenen die durch ein simuliertes Schreddern erzeugt werden. Ebenfalls bildet er das Problem der Rekonstruktion dabei auf das Problem des Handlungsreisenden (*traveling salesman*) ab, indem er die „Wege“ zwischen den Streifen gewichtet, die die Bewertungsfunktion der einzelnen Streifen als jeweilige Nachbarn definiert.

Seine Lösung wird durch variable Nachbarschaftssuche erzeugt, bei der er diverse Züge (*moves*) verwendet: (a) Einfügeoperationen, (b) Verschiebeoperationen und (c) Einfügen als Block. Zusätzlich implementiert er Optimierungsalgorithmen von iterativer lokaler Suche bis zum simulierten Abkühlen (*simulated annealing*).

Da man für heuristische Verfahren eine Startlösung benötigt, untersucht Morandell in seiner Arbeit auch, wie man gute, geeignete Startlösungen findet. Als mögliche Verfahren nennt er FPC (*forward page construction*) und DPC (*duplex page construction*).

Bei der *forward page construction* wird ein Streifen zufällig ausgewählt und anschließend der beste zu diesem Streifen passende Streifen daneben platziert. Das wird solange für den jeweils zuletzt gesetzten Streifen wiederholt, bis sämtliche Streifen platziert wurden. Bei der *duplex page construction* wird ebenfalls zu Beginn ein Streifen zufällig ausgewählt. Danach werden für den Streifen der beste linke und der beste rechte Nachbar bestimmt. Unter „bester Nachbar“ ist hier der „Nachbar“-Streifen gemeint, der das optimalste Bewertungsfunktionsergebnis aufweist. Der Nachbarstreifen aus den beiden ermittelten, der den besseren Wert erzielt wird schließlich akzeptiert und platziert. Dieses Verfahren wird für den jeweils linken und rechten Rand der entstehenden Seite so lange wiederholt – wobei jeweils immer der beste Streifen links oder rechts platziert wird – bis kein Streifen mehr übrig ist. In seiner Bewertung befindet er DPC als die bessere der beiden Varianten.

Prandtstetter [21] behandelt in seiner Dissertation unter anderem auch die Rekonstruktion von zerstörten Papierdokumenten. Es geht dabei neben der Rekonstruktion von in Streifen geschnittenen Seiten auch um die Rekonstruktion von händisch zerrissenen Papierseiten sowie (jene von) in kleine Rechtecke zerschnittene Dokumente. Prandtstetter löst die kombinatorischen Optimierungsprobleme jeweils dadurch, indem er die Problemstellungen in ein Handlungsreisenden Problem (*traveling salesman problem*) umwandelt um es dann mittels bekannter Algorithmen, wie die variable Nachbarschaftssuche, zu lösen. Zusätzlich verwendet er Lagrangerelaxierung um dafür Schranken zu berechnen.

Des Weiteren geht er bei der Rekonstruktion auf die Thematik ein, welche Möglichkeiten durch menschliche Interaktion entstehen. Es wird eine Lösung vorgestellt, die von Klau [15, 14] schon demonstriert wurde, die darauf basiert, dass für eine Tabu Suche (siehe Abschnitt 4.4) von einem Menschen die Tabuliste für die nicht zusammenpassenden Teilen aufgestellt wird, um dadurch zu verhindern, dass falsche Lösungen erzeugt werden beziehungsweise schlechte Lösungen überhaupt weiter untersucht werden.

Kapitel 4

Methoden

Das Lösen des Problems der Rekonstruktion der Papierstreifen fällt grundsätzlich in den Bereich der kombinatorischen Optimierung – siehe Abschnitt 4.1. Man sucht dabei aus einer endlichen, meist großen, Menge von möglichen Lösungen die optimale, die besser[1] als alle anderen Lösungen ist. Für dieses Problem (und selbstverständlich auch für andere) kann man entweder ein exaktes oder ein heuristisches Lösungsverfahren verwenden. Bei einem exakten Verfahren untersucht man systematisch – allerdings meist nur implizit, durch logisch bedingtes Weglassen von Teillösungen – alle Möglichkeiten, die vorhandenen Streifen zu kombinieren und findet daher immer die optimale Lösung, also jene Lösung, die unter allen Möglichkeiten die beste Qualität – bezogen auf eine vorher definierte Bewertung der Lösung – darstellt. Ein exakter Algorithmus findet zwar immer die beste Lösung, jedoch benötigt dies für größere Probleminstanzen zu viel Zeit, um in einem realen Umfeld noch von Wert zu sein. Aber auch exakte Verfahren sind in Bezug auf die optimale Lösung an die Qualität der Bewertungsfunktion gebunden. Denn wie es zum Beispiel in der Abbildung 4.1 gezeigt wird, ist bei bestimmen Ergebnissen die tatsächlich richtige Lösung nur im Kontext durch den menschlichen Betrachter zu entscheiden. Die Bewertungsfunktion – je nach Definition – wird die eine oder andere Lösung besser bewerten, jedoch ist im Vorfeld nicht

[1] bezogen auf das Resultat der Bewertungsfunktion

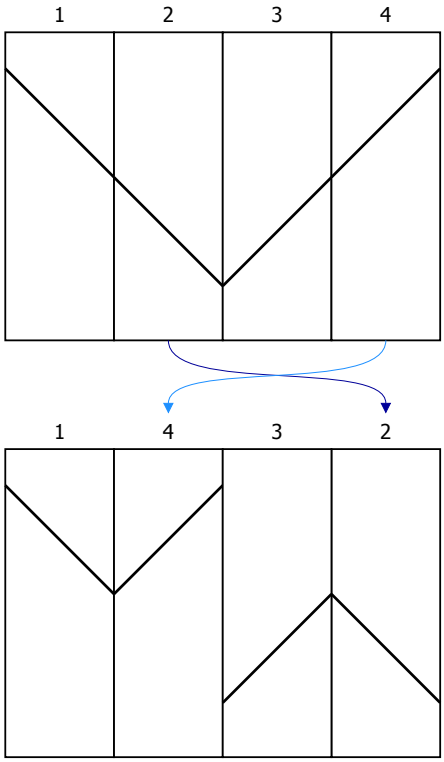


Abbildung 4.1: Demonstration von zwei gleichwertigen möglichen Lösungen, die nur *im Kontext des Dokuments* von einem menschlichen Leser unterscheidbar sind, weil anders nicht eindeutig festgestellt werden kann, welche der beiden Lösungen dem original Dokument entspricht.

eindeutig festlegbar, welche Möglichkeit dann tatsächlich die bessere wäre.

Im Gegensatz zum exakten Ansatz wird bei einem heuristischen Algorithmus nicht jede Möglichkeit ausprobiert beziehungsweise bewertet, sondern nur die Lösungen, die aus Verfeinerung von bereits existierenden, meist guten Lösungen entstanden sind. Dadurch findet der Algorithmus eine Lösung in einer vertretbaren Zeit und in den meisten Fällen ist die Lösung dennoch von einer ausreichenden Qualität – auch wenn die optimale Lösung unter Umständen niemals gefunden wird. Eine Heuristik kann auch dazu verwendet werden, nur eine einzelne Lösung zu erzeugen, wie dies zum Beispiel bei der initialen Anordnung der Streifen notwendig ist, was auch als Startlösung bezeichnet wird. In diesem Kapitel werden diese eben erwähnten Methoden und Begriffe noch kurz vorgestellt und formal definiert.

4.1 Kombinatorische Optimierung

Bei kombinatorischer Optimierung [20] geht es darum, aus einer großen Menge von Elementen eine Teilmenge zu konstruieren, die den gewünschten Bedingungen entspricht und die bezogen auf eine Kostenfunktion optimal ist. Eine Instanz eines kombinatorischen Optimierungsproblems ist ein Tupel (S, f), wobei S die Menge aller möglichen Lösungen bezeichnet und f eine Funktion ist, die $f \rightarrow \mathbb{R}$ abbildet und damit jedem $s \in S$ die Kosten $f(s)$ zuweist. Das Ziel einer kombinatorischen Optimierung in Bezug auf ein Minimierungsproblem ist es, (die) eine Lösung $s \in S$ zu finden, so dass es kein $u \in S$ gibt, für das $f(u) < f(s)$ gilt. Im weiteren Verlauf wird stets von einem Minimierungsproblem ausgegangen.

4.2 Lokale Suche

Bei der lokalen Suche [13, 21] wird versucht, eine bestehende (initiale) Lösung iterativ durch kleine Veränderungen zu verbessern, um dadurch zu einer (akzeptablen) Lösung zu gelangen. Die kleinen Veränderungen werden als *Züge* oder vom englischen übernommen auch als *moves* bezeichnet. Dabei wird der Begriff der Nachbarschaft verwendet, der die möglichen veränderten Lösungen beschreibt, die durch Änderungen aus der anfänglichen Lösung entstehen können. Formal beschrieben spricht man von einer Nachbarschaftsfunktion N, die eine Abbildung $N : S \rightarrow 2^S$ darstellt, welche einer beliebigen Lösung $x \in S$ aus der Menge der möglichen Lösungen S eine Menge von Nachbarn $N(x)$ zuweist, die so genannte Nachbarschaft von x. Die Lokale Suche nimmt eine Lösung x, bestimmt die Nachbarschaften dieser Lösung $N(x)$ und wählt dann eine Lösung x', für die die Suche weitergeführt wird, bis eine bestimmte Qualität beziehungsweise ein Abbruchkriterium erreicht wird. Sei f eine Funktion, die die Bewertung der Qualität der Lösung x durchführt: $f : S \rightarrow \mathbb{R}$. In weiterer Folge ist festzulegen, wie bei der Suche die neue Lösung x' aus den Nachbarschaften ausgewählt wird. Diese Auswahl wird *Schritt* oder vom englischen übernommen auch *step* genannt. Hier sind

die häufigsten Schrittfunktionen [21]:

- Beste Verbesserung (*best improvement*): Es wird jene Lösung x' aller $N(x)$ gewählt, die die größte Verbesserung im Vergleich zur Lösung x bringt.
- Erste Verbesserung (*first improvement*): Es wird bei der Untersuchung aller $N(x)$ das erste x' gewählt, das untersucht wurde und eine Verbesserung bringt.
- Zufällige Veränderung: Unter allen Nachbarn in $N(x)$ wird ein x' zufällig ausgewählt. Diese Methode ist besonders schnell, wobei eine Verbesserung nicht garantiert ist.

Verwendet man als Schrittfunktion die beste oder erste Verbesserung, wird irgendwann der Punkt erreicht, bei dem keine Verbesserung der Lösung mehr erzielt werden kann. Diesen Punkt bezeichnet man als ein *lokales Minimum*. Eine Lösung $\bar{x}$ kann als lokales Minimum der Nachbarschaftsfunktion N bezeichnet werden, wenn:

$$\forall x' \in N(x) \text{ gilt } f(\bar{x}) \leq f(x')$$

Zuletzt sollte noch ein globales Minimum (=globales Optimum) definiert werden: Eine Lösung x^* gilt als optimal, wenn gilt, dass

$$f(x^*) \leq f(x), \forall x \in S$$

wahr ist. Anders ausgedrückt ist eine Lösung global minimal, wenn sie für alle (möglichen) Nachbarschaftsfunktionen das lokale Minimum darstellt.

4.3 Variable Nachbarschaftssuche

Bei der *variablen Nachbarschaftssuche* [9, 10, 11, 12, 21] (*VNS*) versucht man den Nachteil zu Umgehen, dass die Lokale Suche oft bei einem lokalen

Minimum stecken bleibt. Im Gegensatz zu Multi-Start Heuristiken werden bei der VNS die bisherigen Lösungen nicht verworfen und an (zufälliger) neuer Stelle gestartet, sondern durch so genanntes *Rütteln* verändert, um so einem lokalen Minimum zu entkommen und im besten Fall das globale Minimum zu erreichen. Rütteln wird auch als *shaking* bezeichnet: es handelt sich dabei um Veränderungen, die aus einer anderen Nachbarschaftsstruktur gewählt werden können. Bei der VNS wird zunächst eine neue Lösung durch das Rütteln ausgewählt um anschließend eine lokale Suche durchzuführen. Dazu ist es nötig eine Reihe von Nachbarschaftsstrukturen N_i, mit $1 \leq i \leq k_{\max}$ zu definieren. Diese Methode beruht auf der Feststellung, dass viele lokale Minima nahe beieinander liegen und zufälliges Wechseln der Teile einer Lösung, die ein lokales Minimum darstellt, die Wahrscheinlichkeit erhöht, dass die Suche in einer nahen Region weitergeführt werden kann und dort ebenso relativ schnell zu einem lokalen Minimum führt.

Im Gegensatz zur VNS werden beim variablen Nachbarschaftsabstieg oder englisch *Variable Neigborhood Descent* (VND) systematisch die Nachbarschaftsstrukturen durchsucht. Wird in einer Nachbarschaftsstruktur eine Verbesserung durch die Schrittfunktion erzielt, so kehrt die Suche anschließend wieder zur ersten Nachbarschaftsstruktur zurück und beginnt von vorne. Wurde die Lösung in N_{i+1} in Vergleich zur Lösung aus N_i – also der vorher untersuchten – verbessert, so wird wieder mit $i = 1$, also der ersten Nachbarschaftsfunktion, begonnen. Daher ist es auch notwendig, dass die Nachbarschaftsstrukturen einer Ordnung unterliegen, so dass $N_i \not\subseteq N_{i+1}$ für alle $2 \leq i \leq k_{\max}$ gilt. Der Algorithmus endet, wenn alle Nachbarschaftsstrukturen $N_1, \ldots, N_k$ untersucht wurden und keine lokale Suche eine Verbesserung gebracht hat.

Bei einer *generellen VNS* wird so vorgegangen, dass die oben beschriebene VNS und VND miteinander kombiniert eingesetzt werden. Aus der Menge der k verfügbaren Nachbarschaftsfunktionen wird jeweils so lange eine lokale Suche in einer Nachbarschaftsstruktur durchgeführt, bis keine weitere Verbesserung möglich ist. Findet der Algorithmus eine Verbesserung, beginnt er wieder in der ersten Nachbarschaftsstruktur; findet man keine so wird ei-

ne neue Lösung mittels Rütteln ausgewählt. Dieses Vorgehen wird solange wiederholt, bis die Anzahl an vorgesehenen Iterationen erreicht wird. Im folgenden Dokument ist immer diese Form der VNS gemeint, wenn die variable Nachbarschaftssuche erwähnt wird.

4.4 Tabu Suche

Es kann vorkommen, dass sich bei der Untersuchung aller Nachbarschaftsstrukturen bei VNS eine bereits untersuchte Lösung erneut durch einen ausgeführten Zug ergibt; daraus könnte ein endloser Zyklus von Zügen entstehen, die zwar irgendwann, aufgrund der hohen Anzahl von Iterationen abgebrochen werden. Dadurch kann das globale Minimum allerdings nicht gefunden werden.

Zur Lösung dieses Problems kann die Tabu Suche [7, 6, 14] verwendet werden. Dabei wird eine bereits besuchte Lösung x für eine gewisse Anzahl an Iteration d tabu gesetzt – also verboten. Befindet sich eine Lösung in der Tabuliste, wird diese während der eigentlichen Suche nicht weiter untersucht, sondern solange nicht beachtet bis die Lösung wieder aus der Tabuliste entfernt wird. Eine Lösung x wird in eine Tabuliste L eingefügt, die aus d Elementen besteht; bei jeder neuen Lösung die eingefügt wird, fällt eine alte aus der Liste heraus und kann dann wieder besucht werden. Da das Untersuchen der Tabuliste ebenfalls Zeit benötigt, ist die Länge der Liste d der Dauer der Erzeugung und Bewertung der Lösungen anzupassen.

Kapitel 5

Lösungsansatz und Durchführung

Bei der Implementierung des Programms im Zuge der Diplomarbeit wurde die Programmiersprache *perl* verwendet, da diese sehr schnell arbeitet und auch sehr schnell programmiert werden kann, da man Datenstrukturen sehr dynamisch aufbauen kann. Dadurch ist diese Sprache ausgezeichnet geeignet um Prototypen zu entwerfen und Konzepte zu testen sowie deren Funktion zu beweisen beziehungsweise zu demonstrieren.

In diesem Kapitel werden einige Algorithmen beziehungsweise Datenstrukturen vorgestellt, die bei der Lösungsfindung angewandt wurden.

Zur Optimierung des Problems der Rekonstruktion des Dokuments, also der korrekten Zusammenstellung des geschreddeten Dokuments, wird eine Variable Nachbarschaftssuche verwendet.

5.1 Bewertungsfunktion

Die Bewertungsfunktion dient dazu, eine quantitative Aussage zu ermöglichen, wie gut ein Streifen an einen anderen Streifen passt. Die Bewertungsfunktion errechnet einen Wert, der auch als Fehler bezeichnet wird, der angibt, wie

gut oder schlecht eine Lösung ist. Das bedeutet, je mehr Unterschiede beim Vergleich zweier Streifen gefunden werden, desto höher ist der Fehlerwert der errechnet wird. Hierfür benötigt man zuerst eine Testfunktion, die eine Abbildung $t : p \to \mathbb{R}$ ist, die den Inhalt eines Bildpunktes p_s eines Streifens S darstellt. Die Bewertung der Streifen A und B erfolgt in dieser Form

$$f(A, B) = \sum_{p \in P} |t(p_a) - t(p'_b)| \,,$$

und wird als Fehler der beiden Streifen definiert, wobei P die Menge der zu überprüfenden Punkte ist und p_a der zu überprüfende Punkt auf Streifen A sowie p'_b der dazu passende zu überprüfende Punkt auf Streifen B ist. Die Zuordnung von p und p' erfolgt je nach angewandter Methode. Die entstandene Summe $f(A, B)$ ist das Ergebnis der Bewertungsfunktion, das auch als Fehler zwischen den beiden Streifen bezeichnet wird.

Passt der Streifen A gut an den Nachbarstreifen B, so ergibt die Bewertungsfunktion $f(A, B)$ einen geringen Wert, der gegen 0 geht, wobei 0 die perfekte Übereinstimmung der beiden Streifen repräsentiert. In diesem Fall wären alle Punkte entlang der Kanten von A und B exakt gleich.

Vergleich der Kanten

Natürlich sind komplexere Methoden und Algorithmen zur Bewertung der Randstreifen möglich und auch in der Literatur [1, 21] zu finden. Eine einfache Vorgehensweise ist der geradlinige Vergleich aller Punkte der rechten Seite eines Streifens mit der linken Seite seines rechten Nachbarstreifens. In der oben beschriebenen Formel wären hier alle Punkte p aus der rechten Spalte eines Streifens, und die Punkte p' aus der linken Spalte des anderen Streifens.

Hierbei zählt man, wie viele Bildpunkte in den jeweiligen Randspalten der zwei untersuchten Streifen gleich sind. Wenn also in derselben Zeile an den Rändern der Streifen beide Werte gleich sind, liefert die Bewertungsfunktion ein geringeres Ergebnis, als wenn sich dort unterschiedliche Bildpunkte befinden. Es wird die rechte Spalte des linken Streifens und die linke Spalte

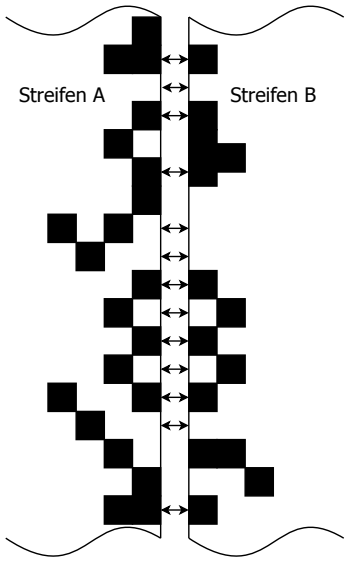


Abbildung 5.1: Stark vergrößerte Darstellung der Bildpunkte beim Vergleichen entlang der rechten Kanten von Streifen A und der linken Kante von Streifen B.

des rechten Streifens untersucht. Dieser Vergleich wird für alle n Streifen mit jeweils allen anderen $n-1$ Streifen durchgeführt. Diese Werte werden, da sie sich nicht verändern, anfangs einmalig errechnet und in einem Feld $v[x, y]$ einer $n \times n$ Matrix V gespeichert, wobei x der Streifen links und y der Streifen rechts ist. Natürlich sind diese Werte generell nicht symmetrisch, darum gilt normalerweise $v[x, y] \neq v[y, x]$. Allerdings gilt immer $v[x, y] = v[\bar{y}, \bar{x}]$, wobei $\bar{x}$ den um 180° gedrehten Streifen x darstellt. Das heißt, dass die rotierten und positionsvertauschten Streifen den gleichen Wert haben. Der Kantenvergleich wird in der Abbildung 5.1 für einige Bildpunkte dargestellt. Die Pfeile in der Abbildung zeigen jene Punkte an, die sowohl links als auch rechts gleich sind; das können sowohl schwarze als auch weiße Punkte sein.

5.2 Bewertung der Randstücke

In den folgenden Abschnitten werden Begriffe wie Bildpunkte, Spalten, Zeilen und Ränder verwendet, die kurz erklärt werden sollten. Unter einem Bildpunkt wird die digitale Speicherung der Helligkeitsinformation einer kleinsten Einheit eines Streifens verstanden. Der Streifen setzt sich aus mehreren Zeilen sowie Spalten von Bildpunkten zusammen. Unter Spalten (und Zeilen) versteht man, dass sich die Punkte entlang der y-Achse (und x-Achse) direkt nebeneinander befinden. Alle Punkte einer Spalte befinden sich somit in vertikaler Richtung, also von oben nach unten (oder umgekehrt). Die Ränder sind jene Spalten eines Streifens, die sich am äußersten linken beziehungsweise rechten Ende des aufrechten länglichen Streifens befinden. Weiters ist, bei nicht näher spezifizierter Erwähnungen von einem Bildpunkt immer ein nicht-weißer – somit fast immer schwarzer – Farbwert gemeint. Ist von *leerer* oder *keiner* Spalte oder Zeile die Rede ist damit gemeint, dass sich in dieser nur weiße Bildpunkte befinden.

Sind alle Streifen richtig orientiert, so ist es sehr häufig möglich, eindeutig zu erkennen, welche beiden Streifen die Randstücke sind. Der rechte Randstreifen hat an seinem rechten Rand keine schwarzen Bildpunkte und ebenso hat der linke Streifen keine schwarzen Bildpunkte am linken Rand. Diese beiden Streifen lassen sich daher relativ einfach identifizieren, indem man bei allen Streifen untersucht, welcher Streifen die geringste Anzahl an nicht rein weißen Spalten besitzt und an welcher Stelle der jeweilige Streifen seine erste und letzte Spalte hat. Der Streifen, der das geringste Maximum an Spalten hat – also jener Streifen, der, von seiner linken Seite beginnend, die wenigsten Spalten mit Bildpunkten belegt hat – ist der rechte Streifen. Umgekehrt ist der Streifen mit der höchsten Startspalte – also jene die am weitesten links beginnt – der linke Streifen.

Dieses Verfahren kann dann scheitern, wenn durch Zufall das Ende des Textes genau auf das Ende eines Streifens fällt. In diesem Fall ist es mit dieser Methode nicht mehr möglich, die Ränder mit Sicherheit zu bestimmen. Allerdings lässt sich dieser Fall dadurch erkennen, wenn bei einem Blatt Papier

mehr als zwei (anstatt genau zwei) Randstücke erkannt werden. Allerdings auch im Fall, dass genau zwei solcher Streifen entdeckt werden gilt: Eine richtige Lösung kann trotzdem niemals garantiert werden, da es durch Zufall möglich ist, dass die Randstreifen genau am Beginn beziehungsweise am Ende des Textes abgeschnitten wurden und dadurch mit dieser Methode nicht als Randstreifen entdeckt werden können, während zwei andere Streifen zufällig einen Abstand zwischen Text und Streifenrand aufweisen. Es ist allerdings im Allgemeinen eher unwahrscheinlich, dass auf einem normalen Blatt Papier mit Text ein solcher Effekt auftritt.

5.3 Orientierungserkennung

Eines der Grundprobleme bei jedem weiteren Versuch die Streifen richtig wieder zusammenzustellen ist, dass am Beginn der Suche eine ungleiche Orientierung der Streifen zu keiner effizienten Lösung führt.

Wenn man ein gutes Verfahren findet, mit dem man es schafft zuerst alle Streifen in die gleiche Richtung zu drehen, verringert man nicht nur den Lösungsraum um den Faktor 2^n, wobei n die Anzahl der Streifen ist, sondern man erspart sich vor allem in jeder Suchfunktion die Rotation der Streifen zu untersuchen. Diese Zahl ergibt sich aus der Tatsache, dass für jeden richtig rotierten Streifen in der Ausgangsanordnung eine von zwei möglichen Veränderungen wegfällt, somit reduziert sich die Anzahl der möglichen Lösungen pro Streifen um den Faktor 2, was bei n Streifen eine Reduzierung aller möglichen Lösungen um 2^n ergibt.

In der Literatur findet man einige aus dem Bereich der Mustererkennung stammende Ansätze, wie etwa die Zerlegung von Text zur Feststellung der in Ober- und Unterlängen der Zeichen. Diese Verfahren werden in [24, 3] vorgestellt, wobei die Ansätze immer von einem vollständig zusammenhängenden Dokument ausgehen, das um 180° gedreht wurde.

In dieser Arbeit werden zwei Verfahren vorgestellt, die mit der Tatsache umgehen können, dass es sich bei der Rekonstruktion um einzelne Streifen

handelt, die erst in eine gemeinsame Richtung orientiert werden müssen.

5.3.1 Orientierungsdetektion anhand der Zwischenräume der Zeilen

Die Methode zur Erkennung der Orientierung der Streifen beruht auf der Annahme, dass sich auf einer geschredderten Seite Papier zumindest einige Zeilen mit Text befinden.

Die Idee ist, dass man anhand der Abstände zwischen den Textzeilen erkennen kann, ob zwei oder mehr Streifen im gegenseitigen Bezug zueinander in der gleichen Richtung orientiert sind. Betrachtet man eine Seite Papier mit Schrift, so gibt es zeilenweise gesehen mehrere leere Linien, die vom linken Rand durchgehend und ohne Unterbrechung bis zum rechten Rand führen. Meistens befinden sich mehrere solcher leerer Bildpunkt-Zeilen übereinander auf einem Blatt. Man versucht die Ausrichtung der Streifen so zu ändern, dass möglichst auf allen Streifen die Ausrichtung gleich ist. Stellt man sich das Blatt Papier wie einen (dreidimensionalen) Quader vor und überlegt sich weiters, dass die leeren Zeilen auf einem Streifen im inneren des Quader Hohlräumen entsprechen, während die Stellen an denen sich Text befindet innen nicht durchgängig sind; so kann man sich den Lösungsansatz so vorstellen, als ob man von der Seite versucht in diese Hohlräume mit einer Nadel (oder ähnlichem Werkzeug) vorzudringen bis man den Quader vollständig durchquert hat. Stößt man dabei auf einen Widerstand versucht man jeweils den Streifen, bei dem die Nadel anstößt, zu drehen und überprüft anschließend ob sie danach weiterkommt.

Bei dieser Methode ist es selbstverständlich ohne Bedeutung, in welcher Reihenfolge die Streifen zu diesem Zeitpunkt angeordnet sind, da es nur darum geht, die Anzahl der nicht weißen Bildpunkte in einer Zeile zu zählen. Somit kann man bereits anfangs die Orientierung testen, bevor man die Streifen an die richtigen Positionen setzt.

Im Unterschied zu den Textzeilen, die fast nie vollständig auf einer Seite von ganz links nach ganz rechts durchgehend vorhanden sind, sind die Textzwi-

schenräume – auch Leerzeilen genannt – durchaus immer vorhanden, auch wenn sie bei den vorliegenden Streifen immer unterschiedlich breit sind. Die Information wird daher bei diesem Verfahren aus den weißen Zeilen gewonnen, also jenem Bereich wo keine Bildpunkte des schwarzen Textes vorhanden sind. Findet man das Minimum einer Leerzeilenhöhe auf allen Streifen, die in eine Richtung orientiert sind, vor, so hat man einen sehr guten Indikator, die Orientierung der Streifen in eine gemeinsame Richtung gefunden zu haben.

In Abbildung 5.2 wird der rote Text auf die vier Streifen A, B, C und D zerteilt. Zuerst wird (in der Abbildung links) pro Streifen der gesamte vertikale Bereich schwarz eingefärbt, in dem sich Text befindet. Danach untersucht man jene nicht-schwarzen Bereiche, die sich auf allen Streifen befinden. In Abbildung 5.2 auf der rechten Seite wird das Resultat grün dargestellt und beschreibt den Zeilenzwischenraum, der sich auf allen vier Streifen befindet. Die Abbildung 5.3 zeigt anschließend einen Streifen A und der um 180° rotierte Streifen A welcher als A' bezeichnet wird. Jene grünen Bereiche, die sich auf A und A' befinden haben für den Test keine Aussagekraft und werden daher am Weg zu M_1 entfernt. Im Zwischenergebnis M_1 sind alle Bereiche blau markiert, an denen entweder auf A oder auf A' aber nicht auf beiden ein grüner Zwischenraum vorhanden ist. Die Schnittmenge $M_2 = M_1 \cap A$ ergibt jene Maske, mit den grünen Leerzeilen, die zwar auf allen Streifen, aber nur in einer Richtung vorhanden sind. Mit der nun erhaltenen Maske sind einmalig alle zu untersuchenden Streifen mit M_2 zu testen um eine eindeutige Orientierung des Streifens zu erhalten. Hat der Streifen leere Zeilen an jenen Stellen die in Abbildung 5.3 in M_2 grün markiert sind, so ist der Streifen in die gewünschte Richtung orientiert, sonst muss er gedreht werden.

Seiten, die nur im oberen Teil beschrieben sind, wie beispielsweise das Ende von Kapiteln, können mittels dieser Methode so gut wie immer in wenigen Testschritten eindeutig einer gemeinsamen Orientierung zugeordnet werden. In weiterer Folge ist es für die Rekonstruktion ohne Bedeutung, ob alle Streifen aufrecht oder um 180° gedreht angeordnet wurden, da sich dadurch nur die Orientierung der Lösung verändert, nicht aber deren Zusammensetzung.

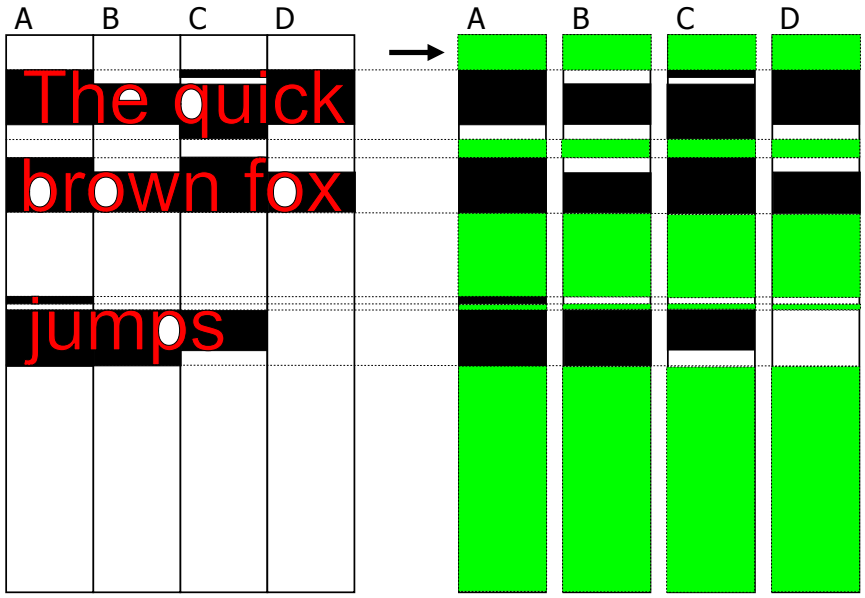


Abbildung 5.2: Erster Schritt zur Orientierungserkennung anhand der Textzwischenräume.

Algorithmus

Im ersten Schritt zur Erkennung der Orientierung wird jeder Streifen auf leere Zeilen untersucht. Als leer gilt eine Zeile dann, wenn sich keine Information in ihr befindet, das heißt die Anzahl der nicht weißen Bildpunkte in der Zeile Z gleich null ist.

$$\sum_{i \in Z} \chi_i = 0$$

wobei

$$\chi_i = \begin{cases} 0 & \text{Bildpunktwert geringer als 50\% Grauwert} \\ 1 & \text{Bildpunktwert höher als 50\% Grauwert} \end{cases}$$

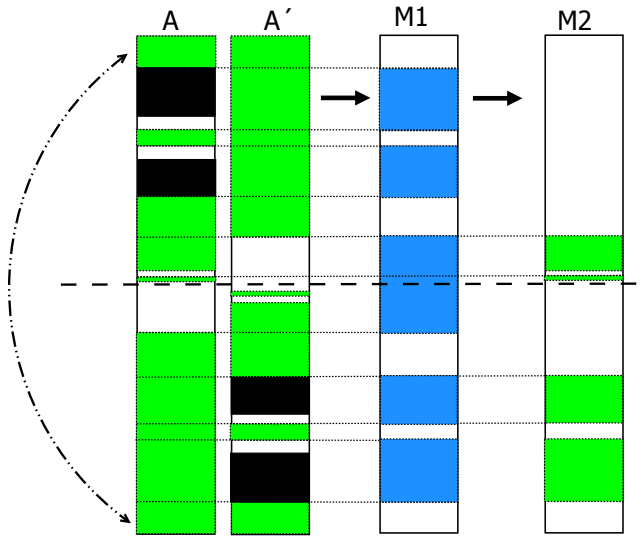


Abbildung 5.3: Erzeugung der Maske zur Erkennung der Orientierung.

Im nächsten Schritt fasst man alle unmittelbar angrenzenden leeren Zeilen zu Blöcken zusammen. Diese Blöcke haben den Vorteil, dass sie effizienter bei der Berechnung und Speicherung sind. Als Information wird für jeden Block der Beginn, also die erste Zeile, mit der der Block beginnt, und die Anzahl der weiteren, nachfolgenden leeren Zeilen gespeichert.

Man hat nun für jeden Streifen eine Blockliste, die alle zusammenliegenden leeren Zeilen eines Streifens enthält.

Als nächsten Schritt invertiert man die Blockliste. Diese Information ist entspricht der Blockliste, die man erhalten würde, wenn man denselben Streifen nur mit umgedrehter Orientierung – also um 180 Grad gedreht – untersuchen würde. Es ist effizienter die Blockliste des umgedrehten Streifens aus der Blockliste des Originals herzuleiten, indem man aus jedem Tupel (s, l) aus der „Originalliste“ ein neues Tupel (r, u) mittels folgender Gleichung

errechnen kann:

$$r = t - (s + l) \tag{5.1}$$

und

$$u = l \tag{5.2}$$

wobei t die maximale Länge eines Streifen ist und s beziehungsweise r der Start des Blocks und l beziehungsweise u die Längen des Blocks sind.

Zur Prüfung der Orientierung sind natürlich nur jene Leerblöcke von Bedeutung, die nicht auch im rotierten Streifen enthalten sind. Es geht also nur um jene Blöcke die im gespiegelten beziehungsweise rotierten Streifen nicht an derselben Stelle vorkommen. Beispielweise hätte eine leere Zeile in einem Streifen der Länge $l \geq 1$ an der Position ρ nur dann eine Bedeutung für die Untersuchung, wenn nicht ebenso an der Stelle $l-\rho$ eine leere Zeile vorhanden wäre.

Deswegen untersucht man nun im nächsten Schritt, welche Blöcke der Streifen zur weiteren Untersuchung von Relevanz sind und welche nicht. Dazu trägt man alle Blöcke in eine Liste ein und erhöht beim Eintragen jedes Mal einen Wert, der aussagt, wie häufig ein Block dieser Größe vorkommt. Anschließend sortiert man die Blöcke nach ihrer Größe absteigend und beginnt mit der Untersuchung der Einträge aus dieser Liste.

Für jeden Block (x, y) sucht man nun in allen verfügbaren Streifen, ob an der Position x eine y Bildpunkte lange Stelle von leere Zeile steht. Diese Überprüfung führt man in beiden Orientierungen durch. Hierbei kommt wieder die kompakte Speicherung zum Einsatz, denn um zu prüfen, ob sich der ganze Block (x, y) im Block (s, l) eines Streifens befindet muss man nur prüfen, ob

$$x \geq s \qquad \text{sowie} \qquad x + y \leq s + l \tag{5.3}$$

gilt. Ist dies der Fall, so ist (x, y) ein Teil von (s, l) und somit sind in beiden Streifen an der gleichen Stelle, nämlich x bis y, leere Zeilen.

Es gibt vier mögliche Fälle als Resultat:

1. der Block befindet sich in den beiden Blocklisten der unterschiedlichen Streifenorientierungen. Der gesuchte Block existiert demnach sowohl in der normalen Liste als auch in der invertierten Blockliste.
2. der Block befindet sich nur in der normalen Blockliste.
3. der Block befindet sich nur in der invertierten Blockliste.
4. der Block befindet sich in keiner der beiden Listen.

Findet man einen Block, den man durch eine Mindestgröße limitieren sollte, der auf allen verfügbaren Streifen nur die Resultate 2 und 3 liefert, so hat man das gesuchte Ergebnis bereits erhalten. Um alle Streifen in die gleiche Richtung auszurichten, muss man jetzt alle Streifen, die das Resultat 2 geliefert haben umdrehen oder man dreht alle Streifen um, die das Resultat 3 als Ergebnis hatten. Im zweiten Fall erhält man natürlich ebenfalls eine Ausrichtung aller Streifen in die gleiche Richtung, nur sind dann alle Streifen umgedreht. Die aus Lesersicht richtige Orientierung kann man mit dieser Methode nicht feststellen. Jedoch lässt sich mit der in 5.3.2 gezeigten Methode eine Kombination finden, die als Resultat auch immer die „wirklich richtige“ Orientierung hat (nämlich so, dass man es lesen kann).

Findet man keinen einzigen Block von Leerzeilen, der auf genau allen Streifen einmal vorkommt, so müsste man in weiterer Konsequenz alle $2^{(n-1)}$ Permutationen der Streifen drehen – was zu lange dauern würde – und kann daher stattdessen die im Folgenden beschriebene Heuristik anwenden. Dabei werden auch wieder nur die Blöcke herangezogen, die Resultat 2 oder 3 geliefert haben, also jene, in denen der gesuchte Block genau einmal vorkommt: entweder in der normalen oder in der invertierten Form. Diese Blöcke bekommen, abhängig davon, ob sie nun in der normalen oder der invertieren Liste eines Streifens gefunden wurden, einen Wert zu einem Array $a_1, a_2, \ldots, a_n$ hinzugefügt, wobei $a_i = (w_1, w_2) \in \mathbb{Z}^2$ mit $1 \leq i \leq n$. Anfangs sind alle $w_1 = 0$ und $w_2 = 0$.

Der hinzugefügte Wert v_b ist in dem erstellten Programm $v_b = d \cdot l_b$, wobei d die Anzahl der Streifen ist, auf denen der Block eindeutig vorgekommen ist und l_b die Länge des Blocks ist. Dadurch erreicht man einen sehr viel höheren Wert für Blöcke mit mehr leeren Zeilen, die auf vielen Streifen gefunden wurden, und Blöcke mit wenigen leeren Zeilen, die nur auf wenigen Streifen existieren, bekommen einen viel geringen Wert. Man fügt diesen Wert v_b nun dem Tupel $a_i = (w_1, w_2)$ bei einmaligem Auftreten des Blocks auf dem Streifen i hinzu:

- Der Block wurde in der normalen Blockliste des Streifens i gefunden: $w_1 = w_1 + v_b$
- oder der Block wurde in der inversen Blockliste des Streifens i gefunden: $w_2 = w_2 + v_b$

Nachdem man diese Berechnung für alle Blöcke durchgeführt hat, beinhalten die Werte $a_j = (w_1, w_2)$ für den Streifen j die Entscheidung, ob er gedreht werden sollte oder nicht. Ist $w_1 \geq w_2$ so wird die Orientierung des Streifens nicht verändert. Ist $w_2 > w_1$ dann wird der Streifen rotiert, das Ergebnis kann so interpretiert werden, dass mehr „dafür“ spricht, den Streifen zu drehen als es nicht zu tun.

Diese Methode ist davon abhängig in welcher Orientierung sich die Streifen zur Zeit der Untersuchung befinden, da der Vergleich der Blöcke auf den anderen Streifen diese nicht dreht. Daher kann und muss diese Berechnung nach jeder Veränderung der Lösung durch andere Nachbarschaftsstrukturen neu durchgeführt werden, um damit weitere Verbesserungen der Lösung zu erzielen.

Ergebnis

Im Allgemeinen kann man nicht feststellen, ob die Streifen für einen Menschen gesehen auf dem Kopf stehen oder nicht.

In der Praxis hat sich gezeigt, dass mit dieser Methode ein sehr gutes Resultat erzielt wird, wenn der obere beziehungsweise untere (weiße) Rand einer

Seite nicht gleich lange sind. Dieser, daraus entstehende, relativ große Block existiert dann immer auf allen Streifen auf nur einem Ende (entweder oben oder unten). Das lässt sich auch in der Abbildung 5.3 erkennen.

Die Methode ist unabhängig davon, ob sich eine Überschrift (die nicht über eine ganze Zeile geht) auf einer Seite befindet, da sie sich nur nach den Zwischenräumen zwischen den Textzeilen richtet (und ein solcher Zwischenraum befindet sich trotzdem auch an den Stellen auf den anderen Streifen, wo die Überschrift bereits zu Ende ist).

Limitierung

Will man diese Methode auf eine Probleminstanz ansetzen, bei der mehrere Seiten zugleich wiederhergestellt werden sollen und man im Vorfeld die Zuordnung der Streifen noch nicht kennt, funktioniert die Methode genauso wenig wie bei einer Seite eines Dokuments, auf der sich mehrere Spalten befinden. Da sich die verschiedenen Textspalten eines beispielsweise zweispaltigen Textes (siehe Abbildung 5.4) so auf dem Blatt Papier befinden könnten, dass sich auf der Höhe der Zwischenräume der ersten Spalten der Text der zweiten Spalte befindet, kann dadurch keine einzige durchgehende leere Zeile gefunden werden.

Allerdings könnte die Information der Leerzeilen – in einem etwas anders aufgebauten Verfahren – auf den Seiten (oder den Spalten) dazu verwendet werden, sich ähnelnde Streifen in Gruppen einzuteilen, indem man Streifen mit gleichen Leerzeilen - beziehungsweise daraus resultierenden Textzeilen - findet. Im Idealfall könnte diese Methode sogar unabhängig von der anfänglichen Orientierung der Streifen sein, wenn sich für jedes Blatt Papier eine nicht symmetrische leere Zeile finden ließe. Diese Idee könnte daher als Vorbereitung für die Textrekonstruktion mehrerer Seiten verwendet werden, wie dieses bei Ukovich [23] im Sinne von *preprocessing* vorgeschlagen wird.

Probleme machen Grafiken, die von der obersten Zeile bis zur untersten Zeile eines Blattes gehend dazu führen, dass sich im gesamten Bereich der Seite keine einzige leere Zeile finden lässt. Dies könnte man beheben, indem man

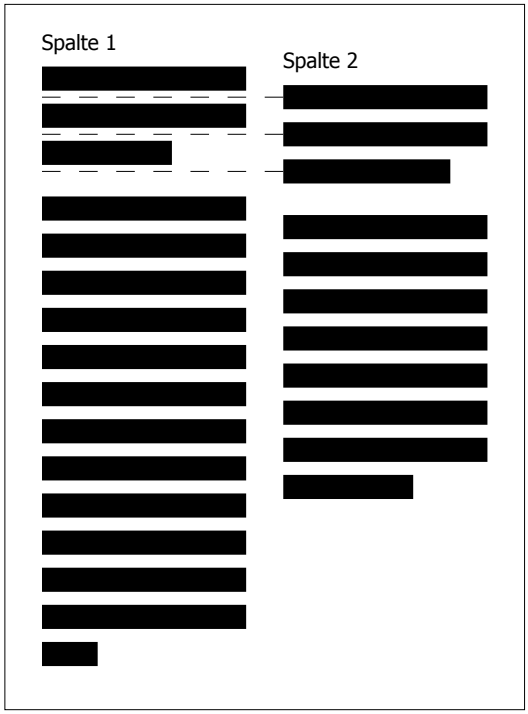


Abbildung 5.4: Darstellung eines zweispaltigen Blatts Papier, bei dem sich der Zwischenraum von Spalte 1 mit dem Text von Spalte 2 höhenmäßig überdeckt.

vorher bildverarbeitende Filter einsetzt, um dünne Linien zu eliminieren.

5.3.2 Buchstabenerkennung

Das Erkennen von Buchstaben ist für den Menschen, der des Lesens mächtig ist, mühelos möglich. Der Computer hat dabei allerdings große Schwierigkeiten, da er im Vergleich zum Menschen nicht den Buchstaben als ganzes und auf einmal erfassen kann, sondern immer nur einzelne Punkte und deren Nachbarpunkte prüfen kann, und so ein Zeichen als Buchstaben erken-

nen kann. Vergleichbar ist dieser Vorgang mit einem Menschen, der in einem Kornfeld steht und etwas erkennen sollte, das aus der Vogelperspektive leicht lesbar wäre. Der Mensch würde sich genauso wie der Computer sehr schwer tun, und könnte keineswegs mehr eindeutig und sofort erkennen, was in dem Kornfeld geschrieben steht.

Vor allem aber werden beim Lesen von Buchstaben durch den Menschen die einzelnen Punkte zu Linien zusammengefasst und diese wiederum zu einem Zeichen beziehungsweise Buchstaben zusammengesetzt. Daher erkennen wir auch verzerrte Buchstaben mühelos, da wir über *Metawissen* verfügen. Dieses Metawissen besteht in erster Linie daraus, dass wir sämtliche Buchstaben des Alphabets und die jeweiligen Unterschiede kennen. Versucht man dasselbe mit Schriftzeichen einer fremden Schrift, beispielsweise Chinesisch, erkennt man sehr schnell, vor welchen Problemen ein Computer bei der Schrifterkennung steht.

Algorithmus

Bei der Erkennung von Buchstaben wird versucht, einzelne eindeutige Zeichen als Buchstaben zu identifizieren und dadurch die Orientierung der Streifen eindeutig herauszufinden. Da diese Arbeit nicht aus dem Bereich der Mustererkennung entstammt, erschien es vernünftig einen sehr einfachen Ansatz zu wählen um nur eine äußerst beschränkte Menge von Buchstaben erkennen zu können, die aber ausreichend ist um eine gute Lösung zu erhalten. Wichtig ist es ein Muster und einen Algorithmus zu finden, die es schaffen einen Buchstaben häufiger als richtig zu erkennen, als sie es nicht tun – womit gemeint ist, dass ein richtig orientierter Buchstabe fälschlicherweise als etwas anderes identifiziert und umgedreht wird. In der Abbildung 5.5 wird das von mir definierte und eingesetzte Muster gezeigt, das beim Vergleich der Buchstaben verwendet wird um ein „T“ zu erkennen. Eine genauere Erklärung erfolgt unten in diesem Abschnitt.

Um möglichst unabhängig von Schriftsätzen zu sein, wird ein Verfahren angewandt, bei dem jeder Buchstabe B aus einer Menge von Linien definiert wird,

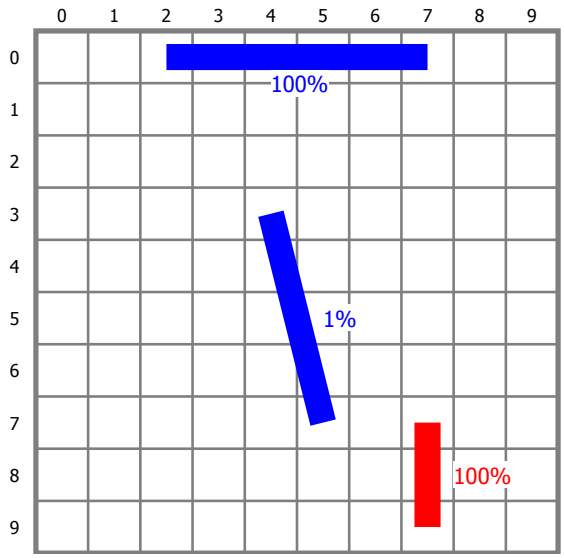


Abbildung 5.5: Definition des Musters zur Erkennung von „T“: Blaue Linien müssen schwarze Bildpunkte enthalten, während rote Linien weiße Bildpunkte enthalten müssen. In diesem Beispiel wird die kleine rote Linie notwendig um „T“ von der Ziffer „1“ zu unterscheiden.

die mit den auf den Streifen gefundenen Bildpunkten verglichen werden.

$$B = \{l_1, l_2, \ldots, l_n\} \text{ wobei } n \geq 1 \text{ gilt}$$

Jede Linie l_x besteht aus einem Tupel $(x_s, y_s, x_e, y_e, m, p)$ für das gilt, dass (x_s, y_s) den Startpunkt und (x_e, y_e) den Endpunkt der Linie beschreiben. Der Wert $m \in (0, 1)$ beschreibt, ob entlang der Linie Punkte vorhanden sein sollen $(m = 1)$ oder nicht vorhanden sein dürfen $(m = 0)$. Schließlich gibt $p \in [0; 100]$ an, auf wieviel Prozent der möglichen Punkte der Linie auch tatsächlich Punkte getroffen werden müssen. In den Abbildungen 5.5 bis 5.8 sind die Werte für $m = 0$ – also dem Nichtvorhanden sein von schwarzen Bildpunkten entlang der Linie – in roter Farbe dargestellt und die Werte für $m = 1$ – also jene Punkte, die einen nicht weißen Farbwert haben müssen – in blauer Farbe. Die Werte für p befinden sich in den Abbildungen jeweils

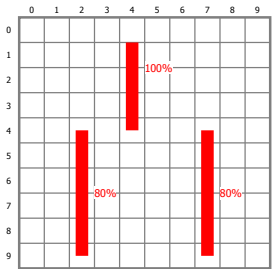


Abbildung 5.6: Erkennung von „v“ und „M“

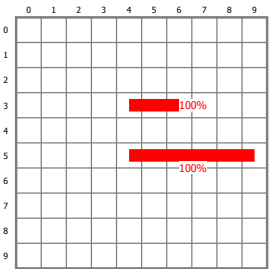


Abbildung 5.7: Muster für „c“ und „e“

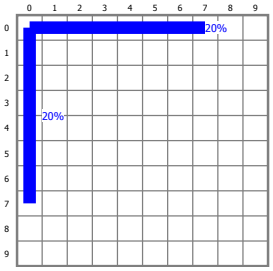


Abbildung 5.8: Identifikation von „B“, „E“, „F“ oder „P“

neben den Linien.

Um unabhängig von unterschiedlichen Größen von einzelnen Buchstaben der verschiedenen Schriftarten zu sein, wird jede Linie l_x mit normierten Werten angegeben, das heißt Fließkommawerte zwischen 0 und 1, die die Koordinaten der Anfangs und Endpunkte beschreiben. Diese Werte werden dann beim Vergleich mit der Punktmatrix des Buchstaben in die realen Integerwerte einmalig umgerechnet und verglichen. Dadurch macht es keinen Unterschied mit welcher Auflösung das Schriftstück, beziehungsweise die Streifen desselben, eingescannt wurden; auch die Buchstabengröße kann daher vernachlässigt werden.

Erfahrungen

Beim Abarbeiten der Testinstanzen zeigte sich, dass zum Beispiel der Buchstabe „T“ einigermaßen gut erkennbar ist. Auch wenn dieser Buchstabe normalerweise nicht so häufig auftritt, um davon ausgehen zu können, dass er auf jedem Streifen einmal vorkommt, kann er neben der Erkennung der Orientierung einiger weniger Streifen auf jeden Fall immer dafür verwendet werden, die endgültige Lösung automatisch in die richtigen Orientierung zu drehen.

Wesentlich häufiger als ein „T“ kommen die Buchstaben „c“ und „e“ vor. Auch diese Buchstaben sind anhand eines einfachen Musters zu identifi-

zieren, da sie nicht symmetrisch sind. Laut einem Artikel über Buchstabenhäufigkeiten [2] sind von 100 aufeinander folgenden Buchstaben durchschnittlich drei ein „c“ und 17 (in englischen Texten 13) ein „e“. Somit kann man grob abschätzen, dass im Durchschnitt jedes sechste Zeichen eines Textes ein „c“ oder „e“ ist, und man dadurch eine gute Wahrscheinlichkeit erhält auf jedem Streifen einen solchen Buchstaben zu finden, um die Orientierung eindeutig zu bestimmen. Das in Abbildung 5.7 gezeigte Muster würde auf den ersten Blick nicht unbedingt mit den genannten Buchstaben in Verbindung gebracht werden. Es erlaubt aber dennoch eine sehr schnelle und vor allem recht zuverlässige Erkennung von falsch gedrehten Buchstaben und ermöglicht in weiterer Folge falsch gedrehte Streifen zu korrigieren. Alle anderen Buchstaben (außer „c“ und „e“) haben nämlich entlang des Bereichs der rot markierten Linien immer einige schwarze Bildpunkte und sind somit leicht zu unterscheiden. Ein weiteres Muster, das einige der Großbuchstaben – wie B, E, F, P und so weiter – richtig detektiert, ist in der Abbildung 5.8 zu sehen. In den Tests hat sich die Suche nach diesen Buchstaben sehr bewährt und es wurden bei der Anwendung auf Textseiten fast immer nur auf den Randstreifen keine solchen erkennbaren Buchstaben vorgefunden. Dadurch erhält man in vielen Fällen bereits eine fast vollständig richtig orientierte Lösung.

5.4 Blockbildung

Bei der Lösung des Problems, kann man – zweifellos als Mensch, aber auch der Computer anhand der Bewertungsfunktionsergebnisse – erkennen, dass einige (kleine) Gruppen von Streifen zusammen gehören und nicht getrennt voneinander betrachtet werden sollten. Solche Streifengruppen können zu einem oder mehreren Blöcken zusammenfügt werden.

Ein solcher Block ist dann ein Vektor $B =< S_1, S_2, \ldots, S_b >$ von b zusammengefügten verschiedenen Streifen, wobei $1 \leq b \leq n$ ist und n die Anzahl aller Streifen ist. Eine Lösung L besteht aus mehreren verschiedenen Blöcken $L =< B_1, B_2, \ldots, B_l >$, wobei $1 \leq l \leq n$ ist. In weiterer Folge betrachtet

man bei der Lösung dann die jeweiligen Außengrenzen jedes Blocks: für die linke Seite des Blocks B wird der linke Rand von S_1 verwendet und statt der rechten Seite des Blocks B verwendet man den rechten Rand von S_b. Dadurch müssen statt n Streifen nur mehr l Blöcke kombiniert werden. Das entspricht einer Reduktion von $n - l$ Einheiten, die kombiniert werden müssen.

Anschließend werden die Berechnungen mit den einzelnen Streifen beziehungsweise den (Streifen-)Blöcken durchgeführt. Dazu betrachtet man den oder die gebildeten Blöcke als jeweils einen Streifen und iteriert mit diesen über sämtliche Nachbarschaftsstrukturen, wobei die Blöcke nicht über die gesamte Suche hinweg bestehen müssen, sondern währenddessen auch wieder aufgelöst werden können. Im Folgenden werden unterschiedliche Ideen zur Vorgehensweisen und Ansätze zur Bildung und Auflösung von einem Block oder mehreren Blöcken beschrieben.

5.4.1 Vorgehensweise

Vor der Überlegung welche Strategien man bei der Blockbildung verfolgen kann, sollte man sich zuerst veranschaulichen, welcher Zusammenhang zwischen der Anzahl der Streifen, der Anzahl der Schnitte (zwischen den Streifen), der Anzahl der Blöcke und der Größe der Blöcke besteht. In den folgenden Überlegungen wird ein Block, der aus nur einem Streifen besteht, als *trivialer Block* bezeichnet. Folgende Abhängigkeiten lassen sich festhalten:

- In einer Lösung aus n Streifen befinden sich $n - 1 = s$ Schnitte.
- In einer Lösung mit b Blöcken gibt es $b - 1$ Schnitte – damit sind die Verbindungen zwischen den Streifen gemeint –, da es in diesem Kontext keinen Unterschied zwischen Streifen und Blöcken gibt.
- Vereint man von n Streifen insgesamt x Schnitte, so erhält man $s - x$ oder anders gesagt $n - 1 - x$ Schnitte.
- Entfernt man von s Schnitten insgesamt x Schnitte durch Vereinigung,

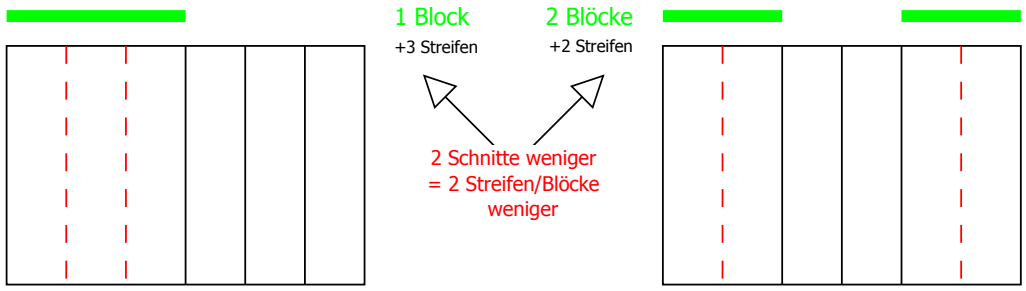


Abbildung 5.9: Vergleich von zwei Möglichkeiten zur Bildung von Blöcken bei gleichbleibender Anzahl an Schnitten, die entfernt werden.

so entstehen minimal 1 und bis zu x nicht triviale Blöcke unterschiedlicher Größe.

- Ein Block mit y Streifen enthält $y-1$ Schnitte und verringert daher die gesamte Lösung um eben diese $y-1$ Schnitte.
- Aus n Streifen kann man $1 \ldots n$ Blöcke erstellen, wobei die Blöcke auch trivial sein können.
- Aus n Streifen kann man maximal $\lfloor \frac{n}{a} \rfloor$ Blöcke bilden, die jeweils mindestens a Streifen pro Block enthalten.
- Erzeugt man aus n Streifen b Blöcke – inklusive trivialer Blöcke –, so ist die Anzahl der Streifen im größten Block durch $n-b+1$ beschränkt, da die restlichen $b-1$ Blöcke nur aus einem Streifen bestehen.
- Im Allgemeinen gilt $s-k=b+r$, wobei k die Anzahl der Schnitte ist, die durch b nicht triviale Blöcke wegfallen und r die Anzahl an trivialen Blöcken (oder auch die Anzahl der Einzelstreifen) beschreibt. Als Beispiel betrachte man die Abbildung 5.9, wobei in beiden Beispielen $n=6$ und $k=2$ ist; im linken Teil der Abbildung ist $b=1$ und $r=3$, während im rechten Teil $b=2$ und $r=2$ ist.

Aus einigen dieser Bedingungen lassen sich nun unterschiedliche Strategien ableiten.

Fixe Anzahl der Blöcke: Ein erster möglicher Ansatz zur Blockbildung der Streifen ist, eine Anzahl von x nicht trivialen Blöcken zu bilden. Bei der Bildung von x Blöcken wird eine beliebige Anzahl an Streifen pro Block gewählt, aber zumindest 1 und maximal $n - b + 1$ (siehe oben). Es ist dabei wichtig zu beachten, dass die Anzahl der trivialen Blöcke in der Lösung nach der Blockbildung nicht konstant sein darf, da sonst die Größe der gebildeten Blöcke auch nicht unterschiedlich sein könnte. Der Algorithmus zur Bildung der Blöcke entfernt solange Schnitte – und fügt damit einen Streifen zu einem Block hinzu – bis die Anzahl der Blöcke die maximale Anzahl übersteigen würde und terminiert anschließend. Die Reihenfolge, in der Schnitte entfernt werden, entspricht einer sortierten Liste, wobei mit dem Schnitt begonnen wird, der die schlechteste Bewertung erzeugt.

Diese Methode hat sich in ersten Tests allerdings als problematisch erwiesen, da sich bereits bei einer geringen Anzahl von gewünschten Blöcken relativ schnell sehr große Blöcke bilden, die dann sofort alle verfügbaren Streifen enthalten. Meistens erhält man statt der gewünschten Anzahl von Blöcken allerdings nur einen großen Block, der alle Streifen enthält. Der Grund dafür ist, dass die Schnitte direkte neben einem Block mit diesem zusammengefügt werden, wodurch es jedes Mal nur zu einer Vergrößerung eines vorhandenen Blocks kommt, und die Gesamtanzahl an Blöcken immer klein bleibt, bis kein Streifen mehr außerhalb des Blocks verbleibt. Daher wurde diese Strategie allein in dieser Form nicht weiter untersucht, sondern durch die im Folgenden beschriebene ersetzt.

Fixe Anzahl und maximale Größe der Blöcke: Als Lösung dieses Problems bietet sich an, die Größe der entstehenden Blöcke mittels einer Obergrenze zu beschränken. Zu einem Block, der die maximale Größe erreicht, wird kein weiterer Streifen mehr hinzugefügt, sondern ein neuer Block muss entstehen; dies kann mittels einer Tabuliste (siehe 4.4) realisiert werden.

Das heißt diese Methode erzeugt bei einer fixen Anzahl an Blöcken eine

variablen Anzahl an Schnitten, die zusammengefasst werden, und führt daher zu einer variablen Größe der Blöcke und Anzahl der einzelnen Streifen.

Fixe Anzahl an Vereinigungen: Ein weiterer Ansatz zur Blockbildung ist es, jene y Schnitte weiterhin als unzusammenhängend zu betrachten, die die schlechteste Bewertung in der vorhandenen Gesamtlösung haben, während alle anderen Schnitte zu Blöcken zusammengefasst werden. Die Anzahl an nicht trivialen Blöcken, die sich daraus ergibt, ist je nach Lage der besten und schlechtesten Schnitte unterschiedlich, wie dies in der Abbildung 5.9 an den Schnitten (rote Linie), die zu einem Block zusammengefasst werden, erkennbar ist. Hat man in einer Lösung beispielsweise 20 Streifen, so erzeugt man eine Lösung, die von den 19 Schnitten zwischen diesen 20 Streifen nur mehr die 5 schlechtesten Schnitte überlässt. Das hat zur Folge, dass die nachfolgenden Berechnungen mit nur 6 vorhanden „Streifen“ erfolgen. Am Ende löst man alle Blöcke wieder auf und beginnt mit der Blockbildung von vorne, da sich durch die Veränderung der Lösung inzwischen andere schlechtere Schnitte ergeben haben könnten.

Zusammengefasst ist diese Methode jene, bei der eine fixe Anzahl von Schnitten, die vereinigt werden, zu einer variablen Anzahl von Blöcken führt.

Qualitative Bedingung für Anzahl an Blöcken: Als weiterer Schritt zur Verbesserung besteht die Möglichkeit, dass man nicht immer eine fixe Anzahl von Blöcken bildet. Stattdessen werden nur die Streifen zu einen Block zusammenfasst, welche eine gewisse Bedingung bezüglich der Qualität der Bewertung erfüllen. Eine solche Bedingung ist zum Beispiel: Jene Kanten, die besser als der arithmetische Mittelwert alle Kantenbewertungen sind, werden zusammenfasst. In weiterer Folge kann man diese Bedingung noch zusätzlich verschärfen und nimmt jene Kanten, die besser als $p\%$ (zum Beispiel $p = 50$) des Mittelwerts aller Kanten sind.

Selbstverständlich lassen sich die beschriebenen Methoden zusätzlich sinnvoll miteinander kombinieren.

5.4.2 Erfahrungen

In den ersten Tests hat sich gezeigt, dass es nicht sehr sinnvoll ist, zu (wenige) große Blöcke zu bilden, weil dann die Wahrscheinlichkeit viel höher ist, dass falsche Papierstreifen miteinander zu einem Block zusammengefasst werden. Wenn man zu (viele) kleine Blöcke erzeugen lässt, erhält man seltener die richtige Lösung, weil die Streifen und ihre richtigen Nachbarn immer noch als einzelne Streifen getrennt betrachtet werden.

5.5 Variable Nachbarschaftssuche

Die in der implementierten Lösung verwendete Heuristik ist eine *Variable Nachbarschaftssuche* (VNS) (siehe 4.3). Bei der Implementierung der VNS musste vor allem berücksichtigt werden, dass sämtliche Operation innerhalb der Nachbarschaftsstrukturen auch auf die (Streifen-)Blöcke als Ganzes angewendet werden und deshalb auch die Streifen pro Block in ihrer Reihenfolge wie ursprünglich festgelegt erhalten bleiben müssen. Daher müssen sämtliche Nachbarschaftsstrukturen auf Lösungen aufgebaut werden, die aus Blöcken bestehen, auch wenn nur ein Streifen pro Block vorhanden sein sollte, wie dies auf jeden Fall in der Startlösung der Fall ist. Zum besseren Verständnis wird im weiteren Verlauf der Begriff Streifen auch als Synonym für einen Block verwendet. Die verwendeten Nachbarschaftsstrukturen werden im Folgenden kurz beschrieben:

- $\mathcal{N}_1$: Die Suche nach den Zwischenräumen auf den Streifen, um alle Streifen in die gleiche Richtung orientieren zu können, wie in Abschnitt 5.3.1 beschrieben sowie das Drehen der Streifen und das anschließende Verschieben gemäß der Nachbarschaftsstruktur $\mathcal{N}_3$.

- $\mathcal{N}_2$: Das Drehen eines Streifens um dessen Orientierung zu ändern, wobei die Position des Streifens in der Gesamtlösung nicht verändert wird. Es wird jeder Streifen an der Stelle, an der er sich in der Lösung befindet gedreht und getestet, ob sich die Bewertung verbessert. Gegebenenfalls wird dieser Zug beibehalten.

- $\mathcal{N}_3$: Beim Umsetzen wird der Streifen mit der schlechtesten Bewertung in die Position verschoben, die die größte Verbesserung der Bewertung der Lösung erzielt. Wird ein Streifen herausgenommen und an anderer Stelle eingesetzt, dann verschieben sich alle Streifen zwischen der alten und der neuen Position in die Richtung der alten Position um jeweils einen Platz. Diese Operation ist aufwändiger, entspricht aber eher dem menschlichen Verhalten beim Lösen eines Puzzles, und führt im Normalfall schnell zu einem besseren Ergebnis.

- $\mathcal{N}_4$: Das Verschieben der Streifen von ihren Positionen nach links, wobei Streifen, die sich jenseits des linken Randes befinden, am rechten Rand wieder angefügt werden. Diese Verschiebeoperation kann die Position der Streifen korrigieren, damit sich die echten Randstücke in der endgültigen Lösung tatsächlich am Rand befinden. Bei gegebener Verbesserung wird der Streifen an seine entsprechende Randposition verschoben.

In der angewendeten Heuristik werden die Nachbarschaftsstrukturen $\mathcal{N}_1$ bis $\mathcal{N}_4$ ausgeführt, wobei dies zweimal geschieht, da erst beim zweiten Durchlauf Blöcke gebildet werden. Zur zufälligen Veränderung der Lösungen (*dem Rütteln*) wird eine eigene Nachbarschaftsstruktur $\mathcal{N}_5$ verwendet:

- $\mathcal{N}_5$: Zwei Streifen tauschen ihren Platz, ohne dass es einen Einfluss auf die restlichen Streifen in der Lösung hat. Diese Nachbarschaftsstruktur wird bei der implementierten VNS als *Rütteloperation* verwendet, wobei zufällig zwei oder mehrere Streifen ihre Position tauschen. Beim i-ten mal Rütteln werden $k^{i \bmod n}$ Streifen vertauscht, wenn n die Anzahl der Streifen und k eine konstante Zahl, die den fortlaufenden Anstieg von vertauschten Streifen steuert, in der Lösung ist.

Die Reihenfolge der Durchsuchung der Nachbarschaftsstrukturen ist schematisch in Algorithmus 1 beschrieben.

Algorithmus 1: Algorithmus zur Rekonstruktion

Eingabe : *Lösung* (=eine Anordnung von Streifen)
Ausgabe : beste gefundene Lösung

```
1  Beginn
2      Vorverarbeitung der Streifen;
3      vnslauf = 0; besteLösung = Lösung;
4      n = Anzahl der Streifen;
5      solange vnslauf < 500 tue
6          für i = 1...4 tue
7              für alle j Streifen in Lösung tue
8                  führe N_i auf Streifen j aus;
9          bilde Blöcke;
10         für i = 1...4 tue
11             für alle j Blöcke in Lösung tue
12                 führe N_i auf Block j aus;
13         wenn Lösung < besteLösung dann
14             besteLösung = Lösung;
15         Blöcke auflösen;
16         führe N_5 auf Lösung k^(vnslauf mod n) mal aus;
17         vnslauf = vnslauf + 1;
18     retourniere besteLösung;
```

Das in Zeile 2 erwähnte Vorbearbeiten besteht aus folgenden drei Schritten:

1. Dem in Abschnitt 5.3.2 beschriebenen Verfahren zum Erkennen von Buchstaben-Mustern um die Orientierung der einzelnen Streifen zu erkennen. Da sich diese Suche nicht auf eine bestimmte Lösung bezieht, wird sie nur dazu verwendet, die Startlösung möglichst gut zu wählen und daher initial am Anfang ausgeführt.

2. Danach wird versucht einen Streifen zu finden der an den linken Rand passt. Dieser wird aus der Menge der Streifen entfernt und an die erste

Stelle der Lösung platziert. Findet man keinen – was vorkommen kann – wählt man einen zufälligen Streifen.

3. An diesen und alle folgenden Streifen wird jeweils der am besten daran passende Streifen aus allen verbleibenden Streifen daneben gesetzt, bis alle Streifen in der Startlösung enthalten sind.

Oft erhält man auf diese Weise bereits eine sehr hochwertige Lösung, die manchmal gar nicht mehr weiter verbessert werden konnte.

Kapitel 6

Implementierung

Im Zuge dieser Diplomarbeit ist eine Applikation entstanden, die den Zweck erfüllt, die Ideen und Überlegungen sowie die daraus entstandenen Algorithmen zu testen und weiter zu optimieren. Einige der erwähnenswerten Details zu dem vollständig selbst erstellten Programm werden in diesem Kapitel kurz beschrieben.

6.1 Allgemeine Informationen zur Anwendung

Das Programme wurde in der Programmiersprache *perl* (Version 5.10.1) entwickelt und programmiert. Damit die Eingangsdaten – also die Instanzen als Bilddaten – angezeigt werden können wurde für die graphische Oberfläche der Applikation das Modul *Tk* (Version 804)[1] verwendet. Das Design der Oberfläche, kann in Abbildung 6.1 betrachtet werden.

Kommandozeilenmodus

Neben der Möglichkeit die Rekonstruktion graphisch zu betrachten wurde auch eine Alternative geschaffen, die Rekonstruktion vollkommen automatisiert durchzuführen. Diese läuft ohne jede Benutzerinteraktion ab und öffnet

[1] Das dafür zu installierende Paket heißt in den meisten Distributionen „perl-tk“.

dabei auch keine graphische Oberfläche. Neben einigen anderen Optionen, die in einer Kommandozeile übergeben werden können, lässt sich der Verlauf der Rekonstruktion in einem Logfile aufzeichnen. Ebenso lässt sich damit steuern, welche Instanz verwendet werden soll, in wie viele Streifen diese zerschnitten wird und welcher Blockbildungsalgorithmus beim Wiederzusammensetzen angewendet werden soll. Um die Ressourcen optimal zu nützen kann auch die Anzahl an automatischen Durchläufen angegeben werden. In einem solchen Fall wird nach jedem Durchlauf zwar immer eine neue zufällige Ausgangsformation der Streifen erzeugt, allerdings müssen die Strukturdaten – wie die Werte in der Matrix der Ergebnisse der Bewertungsfunktion oder die Bildpunkte der Streifen – nicht jedes Mal neu geladen und initialisiert werden, was die Laufzeit bei x hintereinander durchgeführten Tests im Vergleich zu einem x-maligen Neustart der Anwendung signifikant verringert.

6.2 Visualisierte Buchstabenerkennung

Die in 5.3.2 beschriebene Erkennung von falsch orientierten Buchstaben wird in der graphischen Oberfläche ebenfalls angezeigt, damit die Möglichkeit besteht die Wirksamkeit der Methode zu überprüfen. Die Buchstaben, die erkannt werden, sind mit einer farblichen Markierung gekennzeichnet. Rote Markierungen beschreiben ein auf dem Kopf stehendes Zeichen, während grüne die korrekte Orientierung des Buchstabens hervorheben. Die in Abbildung 6.2 gezeigten markierten Zeichen entsprechen dem in Abbildung 5.7 definiertem Muster, das vorwiegend die Zeichen „e“ und „c“ erkennen soll. Man ersieht daraus sofort, dass die tatsächliche Größe der Buchstaben keine Rolle spielt, da sich die Buchstabenmuster der Größe anpassen. In der Abbildung 6.3 sieht man die Buchstaben noch einmal in einer stark vergrößerten Form.

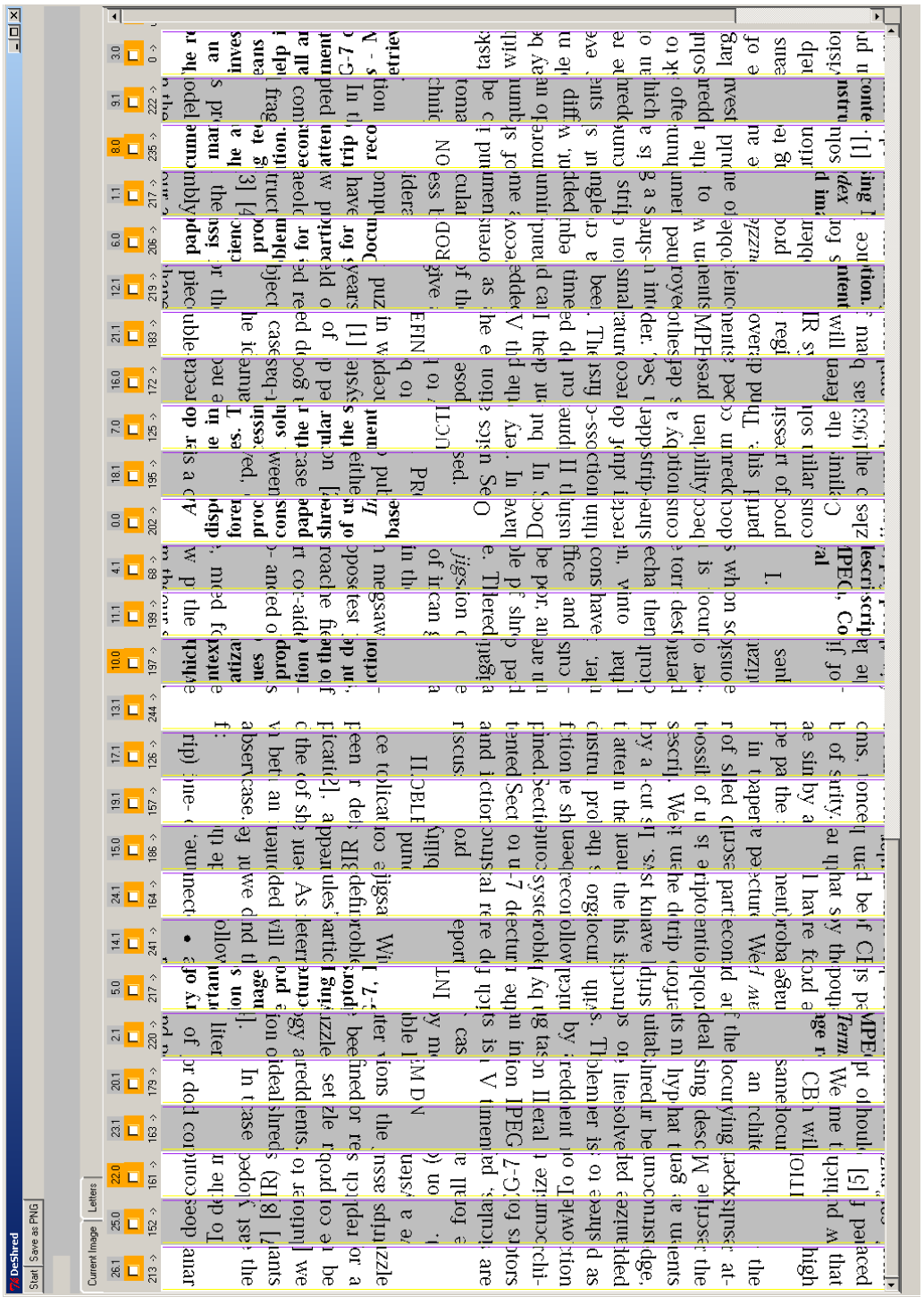


Abbildung 6.1: Screenshot einer geschredderten Seite in der Anwendung

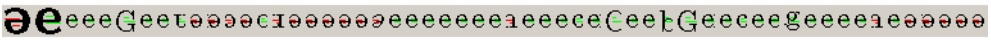


Abbildung 6.2: Optische Markierung der erkannten Zeichen und ihrer vermuteten Orientierung.

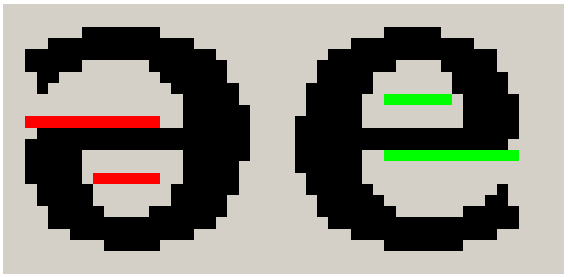


Abbildung 6.3: Vergrößerte Darstellung der in Abbildung 6.2 gezeigten Markierungen. Die roten und grünen Striche sind die in Abbildung 5.7 definierten Muster, wobei die rote Version als am Kopf stehend erkannt wird.

6.3 Datenstruktur

Grundsätzlich gibt es in der Sprache *perl* zwar auch die Möglichkeit einen objektorientierten Ansatz zu verwenden, jedoch entspricht das einer Verwendung von dynamisch benannten Strukturen,wie es auch im gewählten Ansatz verwendet wurde. Somit können Strukturen wie Objekte gesehen werden, auf die bestimmte Methoden angewandt werden können. Zur Speicherung, dem algorithmischen Lösen des kombinatorischen Problems und zur Bildung einer Lösung wurde die Datenstruktur verwendet, die in Abbildung 6.4 gezeigt wird und die im Folgenden kurz erklärt wird.

Jede Lösung ist eine Liste (oft auch als *Array* bezeichnet) von einem oder mehreren Blöcken; die Blöcke wiederum bestehen aus einer Liste von einem oder mehreren Streifen. Die Streifen sind eine Datenstruktur, in der neben den einzelnen Bildpunkten auch die Höhe und Breite des Streifens in *Bildpunkten* (auch als *pixel* bezeichnet) gespeichert sind.

Bei der Bildung einer Lösung werden die Streifen zu mehreren Blöcken zusammengefasst, wobei die Reihenfolge der Streifen in den Blöcken für die Lösung relevant ist, da die gut zusammenpassenden Streifen nebeneinander

gestellt werden. In der gleichen Art ist die Reihenfolge der Blöcke innerhalb der Lösung ebenso relevant, denn zur Erzeugung des rekonstruierten Bilddokumentes werden alle Blöcke und deren Streifen in der Reihenfolge und Orientierung wie sie in der Liste stehen nebeneinander stehend zusammengefügt.

6.4 Visualisierte Lösungsdarstellung

Zur besseren Darstellung wurde zusätzlich die Möglichkeit geschaffen, eine Lösung wieder in eine Bilddatei zurückzuspeichern. Dazu werden die einzelnen Streifen der Reihenfolge der Lösung entsprechend nebeneinander in eine PNG Datei übertragen. Ein solches Bild ist beispielsweise in der Abbildung 7.7 zu sehen.

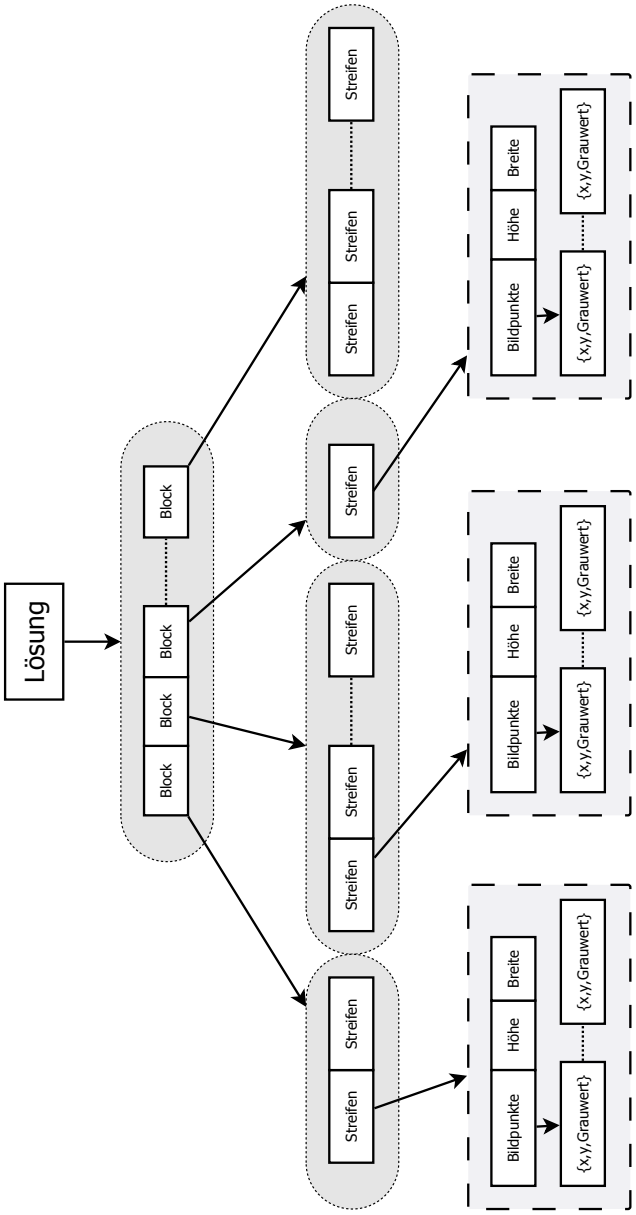


Abbildung 6.4: Die im Programm verwendetet Datenstruktur bei der Erzeugung der Lösungen.

Kapitel 7

Tests

Die bisher beschriebenen Algorithmen wurden in der im vorigen Kapitel kurz vorgestellten Software implementiert und anschließend getestet. Grundsätzlich werden Tests bei heuristischen Verfahren – falls es sich um Methoden handelt, die mit einem Zufallsgenerator arbeiten – mehrfach wiederholt um statistische Ausreißer abzufangen und so bessere Aussagen über die Qualität der Verfahren abzuleiten.

In erster Linie wurde getestet, wie sich drei der vier in 5.4.1 definierten Strategien im Vergleich zueinander verhalten haben um feststellen zu können, welche unter ihnen die vielversprechendste ist. Die vierte Methode – die Bildung von einer konstanten Menge an Blöcken – hat sich schon in den Vorbereitungen als ungeeignet erwiesen und wurde daher nicht getestet. Die im Folgenden erwähnten Blockbildungsstrategien werden durchnummeriert und sind wie folgt definiert:

- Strategie 1 ist die Blockbildung mit einer fixierten Anzahl an Blöcken und einer Blockgrößenbeschränkung.
- Strategie 2 ist die Blockbildung mit einer konstanten Anzahl an Schnitten, die vereinigt werden, was zu einer variablen Anzahl an Blöcken führt.
- Strategie 3 ist die Blockbildung mittels qualitätsbezogenem Kriterium

für die Vereinigung der Schnitte.

7.1 Konfiguration

Zunächst soll die Konfiguration beschrieben werden, die gewählt wurde um die Tests durchzuführen. Damit sind die Variablen in den Formeln und teilweise die Formeln selbst gemeint, die für die Testergebnisse relevant waren. Sämtliche Werte wurden durch Vorabtests empirisch festgelegt.

Getestet wurde auf einem *Athlon 64 X2 Dual Core Prozessor 5200+* mit 4GB RAM und einem Windows XP Betriebssystem. Die benutzte Programmiersprache war – wie auch schon bei der Implementierung erwähnt – *perl V5.10.1* mit *Tk V804* für die graphische Oberfläche.

Die Anzahl an durchgeführten Nachbarschaftssuchen war mit $v = 500$ Durchläufen für alle Tests gleich festgelegt. Die Parameter für die drei getesteten Strategien waren weiters immer von der Anzahl i der bereits durchlaufenen variablen Nachbarschaftssuchen und der Anzahl an Streifen n, die insgesamt in der Lösung vorhanden sind, abhängig:

Strategie 1 Die Anzahl an b Blöcken, die gebildet werden soll, wurde durch

$$b = \left\lfloor \frac{n \cdot (25 + \frac{60}{v-1}) \cdot i}{100} \right\rfloor$$

bestimmt und die maximale Größe der Blöcke wurde mit $m = 7$ festgelegt.

Strategie 2 Die Anzahl von s Schnitten, die zu Blöcken vereinigt werden sollen, wurde durch

$$s = \left\lfloor \frac{n \cdot (25 + \frac{60}{v-1}) \cdot i}{100} \right\rfloor$$

bestimmt.

Strategie 3 Der Prozentsatz p (in %), der den Grenzwert für die Schnitte festlegt unter dem sie zusammengefügt werden, wurde durch

$$p = 120 - 40 \cdot \frac{v - i}{v}$$

festgelegt.

Rütteln Die Konstante k der Nachbarschaftsstruktur $\mathcal{N}_5$ wurde nach einigen empirischen Versuchen als $k = 1.3$ gewählt, um ein anfänglich langsameres Ansteigen der zu tauschenden Streifen zu ermöglichen.

Insgesamt wurden 10 verschiedenen Seiten aus [21] getestet (siehe Anhang A), die jeweils in Streifen von vier unterschiedlichen Breiten (virtuell) zerschnitten wurden. Die Streifenbreite wurde so gewählt, dass jeweils (circa) 20, 30, 40 und 50 Streifen aus einer Seite entstanden; durch Rundungsfehler[1] ergab sich bei einigen Instanzen eine Abweichung von ± 1 Streifen. Für jede der so entstandenen 40 Instanzen wurde die Rekonstruktion 30-mal durchgeführt. Dies führt pro Blockbildungsstrategie zu 1200 Testläufen beziehungsweise insgesamt zu 3600 Testläufen über alle drei Strategien. Zusätzlich wurden weitere 1200 Testläufe ohne Blockbildungsstrategie ausgeführt.

7.2 Wirkung der Blockbildung

Als erstes sollte man untersuchen, wie stark sich die Blockbildung auswirkt. Dazu vergleicht man am besten die Ergebnisse der Rekonstruktionen mit Blockbildungsstrategien mit jenen ohne Blockbildung. Jeder Punkt in der Abbildung 7.1 repräsentiert das durchschnittliche Ergebnis von 30 Testläufen einer bestimmten Instanz mit einer bestimmten Anzahl an Streifen. Die Nulllinie in der Grafik entspricht den Rekonstruktionsergebnissen ohne Blockbildung. Je nachdem ob ein Ergebnis oberhalb oder unterhalb der Linie dargestellt wird zeigt dies an, um wie viel besser oder schlechter die Rekonstruktion

[1] Die Anzahl der Streifen, wird intern in die Breite der Streifen umgerechnet und dort gerundet. Es gibt keine Streifen mit 40.25 Bildpunkten Breite sondern entweder 40 oder 41 Punkte.

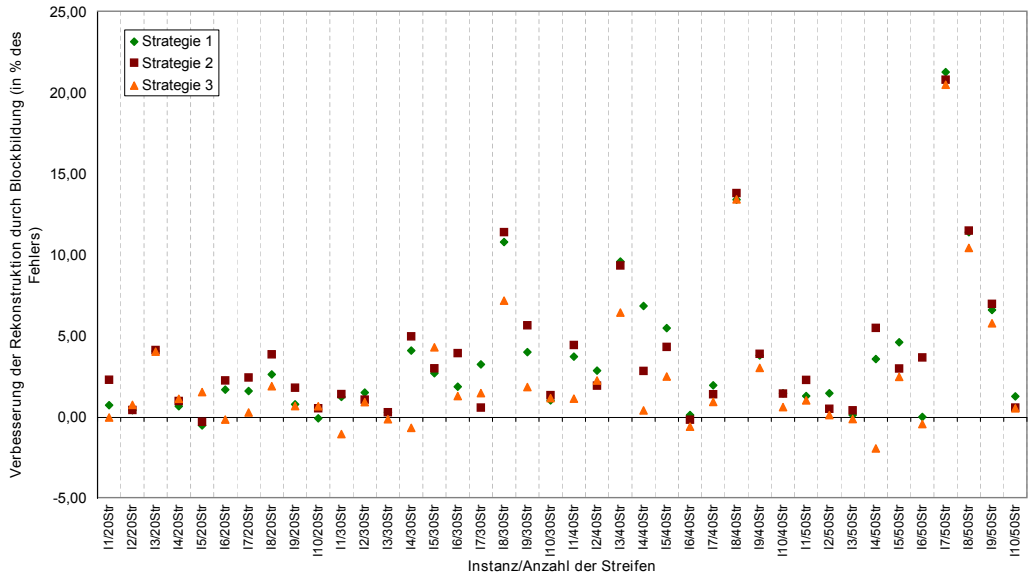


Abbildung 7.1: Durchschnittliche Verbesserung der Rekonstruktion durch den Einsatz der drei Blockbildungsstrategien. Die Bezeichnung „Ix/yStr“ entspricht der Instanz x, die in y Streifen geschnitten wurde.

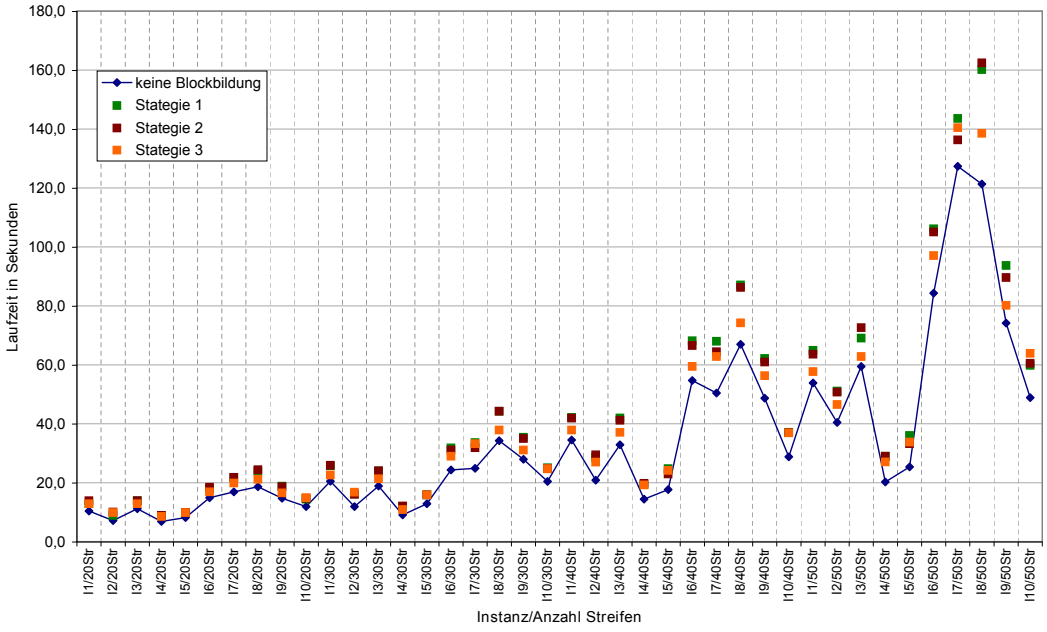


Abbildung 7.2: Durchschnittliche Laufzeiten der variablen Nachbarschaftssuchen.

durch die jeweilige Strategie im Vergleich zur Rekonstruktion ohne Blockbildung bewertet wurde. Man kann sich davon überzeugen, dass fast alle Punkte im positiven Bereich liegen und somit grundsätzlich in allen getesteten Instanzen eine Verbesserung durch Blockbildung erzielt werden konnte. Daher erscheint ein weiterer Vergleich von Ergebnissen ohne Blockbildung als nicht sinnvoll. Zusätzlich erkennt man, dass bei steigender Anzahl der Streifen die Blockbildung zu einer stärken Verbesserung im Vergleich zu keiner Blockbildung führt.

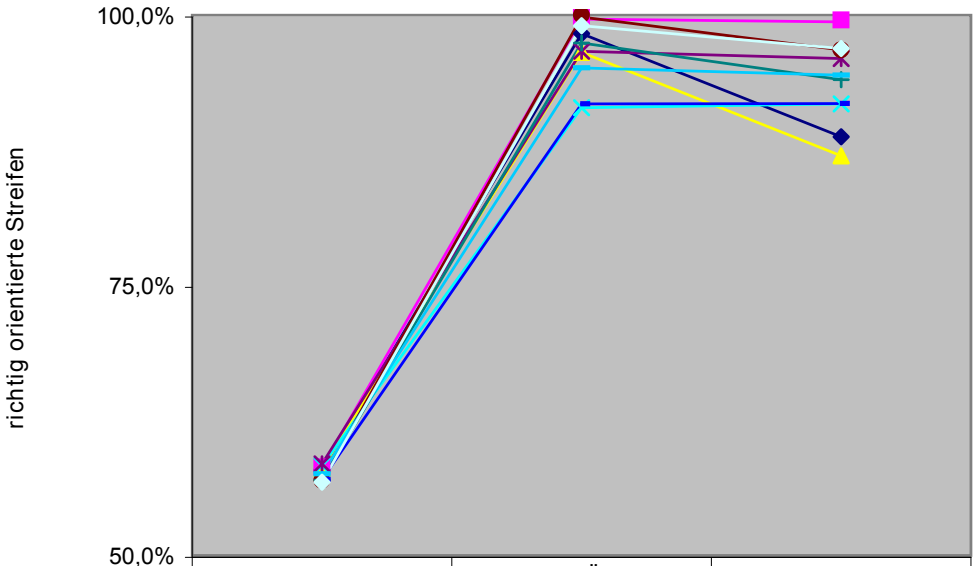
Betrachtet man zusätzlich die Zeit, die es dauert alle 500 Iterationen der VNS durchzuführen so ermöglicht dies eine Untersuchung des Nutzens der Blockbildung. So wäre es bei geringer Verbesserung durch die Blockbildung und gleichzeitiger Verdoppelung der Laufzeit überlegenswert auf die Blockbildung gänzlich zu verzichten. Aus Abbildung 7.2 ersieht man jedoch schnell, dass die Blockbildung nur einen sehr geringen Einfluss auf die Rechenzeit der Rekonstruktion hat. Man erkennt weiters den (erwarteten) Zusammenhang zwischen steigender Anzahl an Streifen und der steigenden Dauer der Rekonstruktionszeit.

7.3 Wirkung der Buchstabenerkennung

Nachdem der in 5.3.2 vorgestellte Algorithmus zur Erkennen der Orientierung der einzelnen Buchstaben unabhängig von der Position der Streifen aber auch unabhängig von der jeweiligen Orientierung und einer untersuchten Lösung funktioniert, wird dieser Algorithmus nur am Anfang einmalig ausgeführt. Daher sollte man überprüfen, wie gut dieses Verfahren wirkt. Das Ergebnis, das in der Abbildung 7.3 dargestellt wird, zeigt an, wie viele der Streifen zu welchem Zeitpunkt richtig orientiert waren. Die drei untersuchten Zeitpunkte sind:

START beschreibt den Zustand der Streifen nachdem ihre Positionen und Orientierungen zufällig verteilt worden sind.

INITIALLÖSUNG ist der für die Untersuchung relevante Zustand. Zu



	START	INITIALLÖSUNG	ENDE
Instanz 1	58,1%	98,5%	88,9%
Instanz 2	58,4%	99,8%	99,6%
Instanz 3	58,2%	96,7%	87,2%
Instanz 4	58,5%	91,6%	92,0%
Instanz 5	58,7%	96,8%	96,2%
Instanz 6	57,1%	100,0%	96,9%
Instanz 7	57,1%	97,6%	94,2%
Instanz 8	57,2%	92,0%	92,0%
Instanz 9	57,7%	95,3%	94,6%
Instanz 10	57,0%	99,2%	97,1%

Abbildung 7.3: Die durchschnittliche Anzahl der richtig orientierten Streifen im Verlauf der drei untersuchten Phasen der VNS

diesem Zeitpunkt sind die drei in Abschnitt 5.5 beschriebenen Ansätze – darunter die Orientierungserkennung der Buchstaben – zur Bildung einer Initiallösung durchgeführt worden. Mit dieser Startlösung startet die VNS.

ENDE beschreibt den Zustand der Streifen in der am Schluss gefundenen besten Lösung.

Dabei ist es nicht relevant, ob die Streifen tatsächlich richtig oder um 180° gedreht stehen, sondern es wird die Mehrzahl der Richtung der Streifen als die richtige betrachtet. Aus diesem Grund gibt es klarerweise keine Werte geringer als 50%. Betrachtet man die Ergebnisse, so sieht man, dass der Unterschied zwischen den 10 untersuchten Testinstanzen gering ist, und dass durchschnittlich mehr als 96% aller Streifen in der Startlösung die gleiche Orientierung besitzen. Ebenfalls zu merken ist, dass sich die Orientierung der Streifen im Verlauf der VNS Iterationen bei einigen Instanzen wieder verschlechtert.

Aus der Untersuchung ergibt sich folgender Grund für die unterschiedlich guten Initiallösungen: Es gibt immer wieder Streifen – hauptsächlich sind das die Randstücke – auf denen sich keine oder nur wenige vollständig vorhandene Buchstaben befinden. Befinden sich darunter keine Buchstaben, die von der Methode zur Orientierung verwendet werden können, so bleibt der Streifen in der ursprünglichen Ausrichtung. Darum ist nur vom Zufall abhängig, ob diese Randstücke in der Initiallösung bereits richtig orientiert sind. Erst durch die Verwendung der in Abschnitt 5.3.1 gezeigten Methode zur Erkennung der Orientierung anhand der Zeilenzwischenräume (im Zuge der VNS Nachbarschaftsstruktur) werden diese Randstreifen gegebenenfalls auch richtig orientiert.

Konsequenterweise muss man noch überprüfen, wie sich die Verwendung der Orientierungserkennung auf das Endergebnis der Rekonstruktion nach der VNS auswirkt. Dazu wird die Rekonstruktion jeweils mit und ohne der Orientierungserkennung beim Erzeugen der Initiallösung durchgeführt. Den Vergleich der durchschnittlichen Abweichung von der richtigen Lösung je-

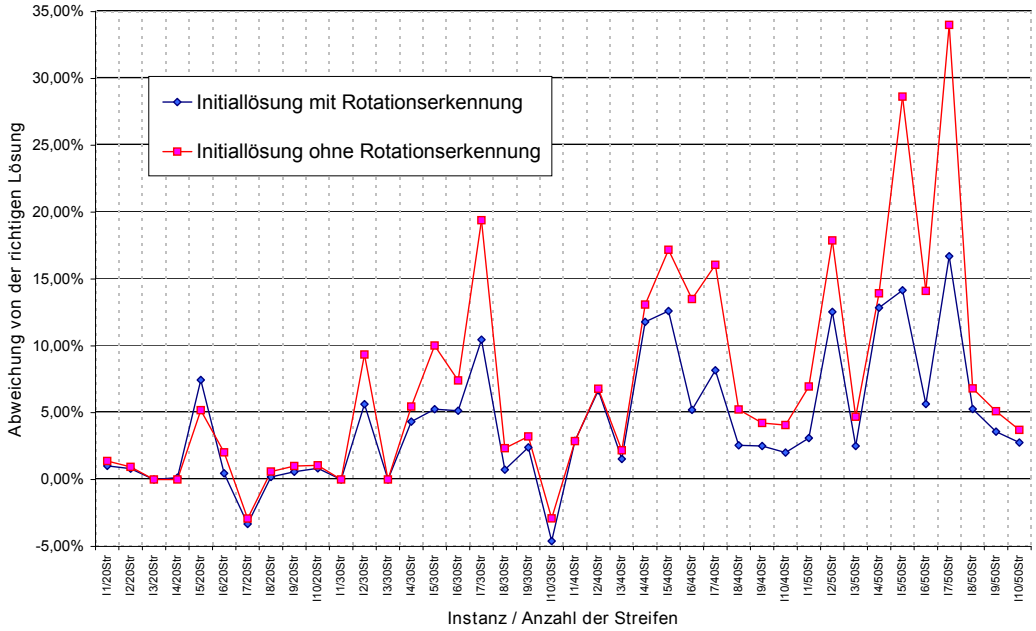


Abbildung 7.4: Der Vergleich des Fehlers der Rekonstruktion jeweils mit und ohne Verwendung der Rotationserkennung bei der Bildung der Initiallösung.

der Instanz ersieht man aus Abbildung 7.4. Wie man erkennt, ist der Fehler des Ergebnisses der Rekonstruktion ohne Rotationserkennung bei der Erzeugung der Initiallösung fast immer größer als bei der Rekonstruktion mit einer solchen. Somit erzeugt die Verwendung des Verfahrens eine signifikate Verbesserung. Man sieht auch, dass sich die Verbesserung mit steigender Anzahl von Streifen erhöht.

7.4 Statistische Auswertung des Ergebnisses

In der Tabelle 7.1 sieht man, wie sich die einzelnen Blockbildungsalgorithmen im Vergleich verhalten. Die Spalten enthalten die verwendete Testinstanz (*Inz*), die Anzahl der Streifen in die die Seite geschnitten wurde (*Str*) sowie jenen *Wert*, den die Bewertungsfunktion für die korrekte Lösung errechnet. Die Spalten für alle drei Strategien zeigen den Mittelwert (MittW) und die quadratische Abweichung (Abw) der Differenz zwischen der Bewer-

tung (Wert) der richtigen Lösung und der erhaltenen Lösung in den jeweils 30 durchgeführten Testrekonstruktionen pro angegebener Instanz, sowie den Vergleichswert $p_{k,l}$, der das Resultat aus einem *t-Test* mit 5% Fehler zwischen den Ergebniswerten der Strategie k und jenen der Strategie l zeigt.

			Strategie 1			Strategie 2			Strategie 3		
Inz	Str	Wert	MittW	Abw	$p_{1,2}$	MittW	Abw	$p_{2,3}$	MittW	Abw	$p_{3,1}$
1	20	1506	2.6%	(2.6)	$>$	1.0%	(1.7)	$<$	3.3%	(3.6)	$\approx$
1	30	2192	0.0%	(0.0)	$=$	0.0%	(0.0)	$<$	2.5%	(2.6)	$>$
1	40	3092	3.6%	(1.3)	$>$	2.9%	(1.5)	$<$	6.2%	(1.8)	$>$
1	49	3751	4.1%	(0.7)	$>$	3.1%	(0.9)	$<$	4.4%	(0.6)	$\approx$
2	20	1059	0.8%	(1.0)	$\approx$	0.8%	(0.8)	$\approx$	0.5%	(1.1)	$\approx$
2	30	1176	5.2%	(2.8)	$\approx$	5.6%	(4.0)	$\approx$	5.8%	(3.9)	$\approx$
2	40	1601	5.8%	(2.0)	$\approx$	6.7%	(2.7)	$\approx$	6.4%	(2.3)	$\approx$
2	49	1787	11.6%	(4.4)	$\approx$	12.5%	(3.6)	$\approx$	12.9%	(6.7)	$\approx$
3	20	1269	0.1%	(0.1)	$>$	0.0%	(0.0)	$<$	0.1%	(0.1)	$\approx$
3	30	1997	0.0%	(0.0)	$=$	0.0%	(0.0)	$<$	0.4%	(1.3)	$>$
3	40	2776	1.3%	(1.8)	$\approx$	1.5%	(2.0)	$<$	4.4%	(2.8)	$>$
3	49	3519	2.8%	(0.3)	$>$	2.5%	(0.7)	$<$	3.0%	(0.5)	$>$
4	20	640	0.5%	(1.4)	$\approx$	0.1%	(0.8)	$\approx$	0.0%	(0.0)	$<$
4	30	1371	5.2%	(2.6)	$\approx$	4.3%	(3.3)	$<$	10.0%	(4.2)	$>$
4	40	1431	7.8%	(2.5)	$<$	11.8%	(3.7)	$<$	14.2%	(3.3)	$>$
4	49	1695	14.8%	(3.4)	$>$	12.9%	(4.1)	$<$	20.3%	(6.1)	$>$
5	20	478	7.7%	(4.5)	$\approx$	7.4%	(4.9)	$\approx$	5.6%	(4.8)	$<$
5	30	1138	5.6%	(2.0)	$\approx$	5.3%	(2.2)	$>$	4.0%	(2.9)	$<$
5	40	1231	11.5%	(3.0)	$\approx$	12.6%	(3.8)	$<$	14.4%	(3.6)	$>$
5	49	1726	12.5%	(1.7)	$<$	14.2%	(1.7)	$\approx$	14.7%	(2.5)	$>$
6	19	905	1.0%	(2.4)	$\approx$	0.5%	(1.4)	$<$	2.9%	(3.6)	$>$
6	30	1355	7.2%	(2.8)	$>$	5.2%	(1.9)	$<$	7.8%	(2.6)	$\approx$
6	39	1875	4.9%	(1.0)	$\approx$	5.2%	(0.9)	$<$	5.6%	(0.7)	$>$
6	50	2274	9.3%	(1.8)	$>$	5.6%	(1.8)	$<$	9.8%	(1.8)	$\approx$
7	20	1027	-2.5%	(1.6)	$>$	-3.3%	(0.9)	$<$	-1.2%	(2.0)	$>$
7	31	1369	7.8%	(2.9)	$<$	10.5%	(2.8)	$\approx$	9.6%	(3.7)	$>$
7	40	1569	7.6%	(3.1)	$\approx$	8.2%	(2.9)	$\approx$	8.6%	(2.6)	$\approx$
7	50	2196	16.2%	(3.6)	$\approx$	16.7%	(3.6)	$\approx$	17.0%	(3.6)	$\approx$
8	20	1672	1.4%	(1.9)	$>$	0.2%	(0.8)	$<$	2.2%	(2.8)	$\approx$
8	29	2413	1.3%	(2.1)	$\approx$	0.7%	(1.7)	$<$	5.0%	(1.2)	$>$
8	39	3334	3.0%	(0.7)	$>$	2.6%	(1.0)	$<$	2.9%	(0.5)	$\approx$
8	49	3996	5.4%	(1.4)	$\approx$	5.3%	(1.3)	$<$	6.3%	(0.9)	$>$
9	20	2419	1.6%	(1.6)	$>$	0.6%	(1.0)	$<$	1.7%	(2.1)	$\approx$
9	29	3664	4.1%	(1.8)	$>$	2.4%	(1.3)	$<$	6.2%	(2.3)	$>$
9	40	4713	2.6%	(0.6)	$\approx$	2.5%	(0.8)	$<$	3.4%	(0.5)	$>$
9	49	5949	4.0%	(1.0)	$\approx$	3.6%	(1.2)	$<$	4.8%	(0.9)	$>$
10	20	2595	1.5%	(1.0)	$>$	0.8%	(1.1)	$\approx$	0.7%	(1.1)	$<$
10	29	4000	-4.3%	(1.0)	$\approx$	-4.6%	(1.2)	$\approx$	-4.4%	(1.4)	$\approx$
10	40	4901	2.0%	(1.2)	$\approx$	2.0%	(1.3)	$<$	2.9%	(1.3)	$>$
10	49	6104	2.1%	(1.2)	$<$	2.8%	(1.4)	$\approx$	2.8%	(1.3)	$>$

Tabelle 7.1: Das Resultat aus den Testläufen.

Man kann erkennen, dass sich die Strategie 2 im Vergleich zu den anderen beiden als effektiver herausgestellt hat. Interessanterweise ist die Strategie 1 auch in fast allen Fällen besser als die Strategie 3, was in der Form eigentlich nicht zu erwarten war, da auf Grund des Vorgehens von Strategie 3 –

nämlich die qualitätsbezogene Strategie – eigentlich ein besseres Abschneiden erwartet wurde. Dennoch war die Strategie 3 diejenige, die am schlechtesten abgeschnitten hat.

Eine mögliche Erklärung wäre, dass die Zahlen aus der Bewertung der Schnittkanten zwischen den Streifen durch fortlaufende Verbesserung der Lösung immer näher zusammenrücken und somit auch den Durchschnitt (mit steigender Qualität der Gesamtlösung) weiter reduzieren. Daher werden zu viele Schnittkanten zu Blöcken zusammengeführt. Obwohl die kontinuierliche Veränderung des Parameters diesem Effekt ohnehin entgegenwirkt, da sich der Schwellwert kontinuierlich erhöht, hat sich die Verbesserung durch das Bilden der Blöcke auf diese Art als zu wenig effektiv herausgestellt.

7.5 Analyse der Iterationen

Nach den $v = 500$ durchgeführten Nachbarschaftssuchen wurde jeweils analysiert, in welchem Durchlauf die letzte Verbesserung gefunden wurde, um ein Maß zu erhalten, ob eventuell zu viele oder zu wenige Suchen durchgeführt wurden. Die Anzahl der Iterationen wurde in Gruppen zu je 50 zusammengefasst, das heißt, in der Kategorie bis 50 wird die Anzahl aller Suchen, die ihr Ergebnis in der Iteration 1 bis 50 gefunden haben, aufgezeigt.

Als Ergebnis sieht man in der Abbildung 7.5, wie sich die Anzahl der VNS Durchläufe bis zur richtigen Lösungsfindung, im Vergleich der drei unterschiedlichen Blockbildungsstrategien verhält.

In der Abbildung 7.6 sieht man im Unterschied dazu die Anzahl an Iteration, bis keine weitere Verbesserung mehr gefunden wurde. Dies unterscheidet sich insofern von der vorherigen Statistik, dass hier auch die Ergebnisse, die kein vollständig richtiges Resultat dargestellt haben, miteinbezogen wurden.

Man sieht daraus, dass sich die Anzahl an gefundenen Verbesserungen mit der Anzahl der Iterationen sogar steigert. So könnte sich eine Erhöhung der Anzahl an VNS Iterationen sogar positiv auswirken. Auf der anderen Seite sollte man sich überlegen, wie sinnvoll es ist, eine solche Lösung, die nach einer ho-

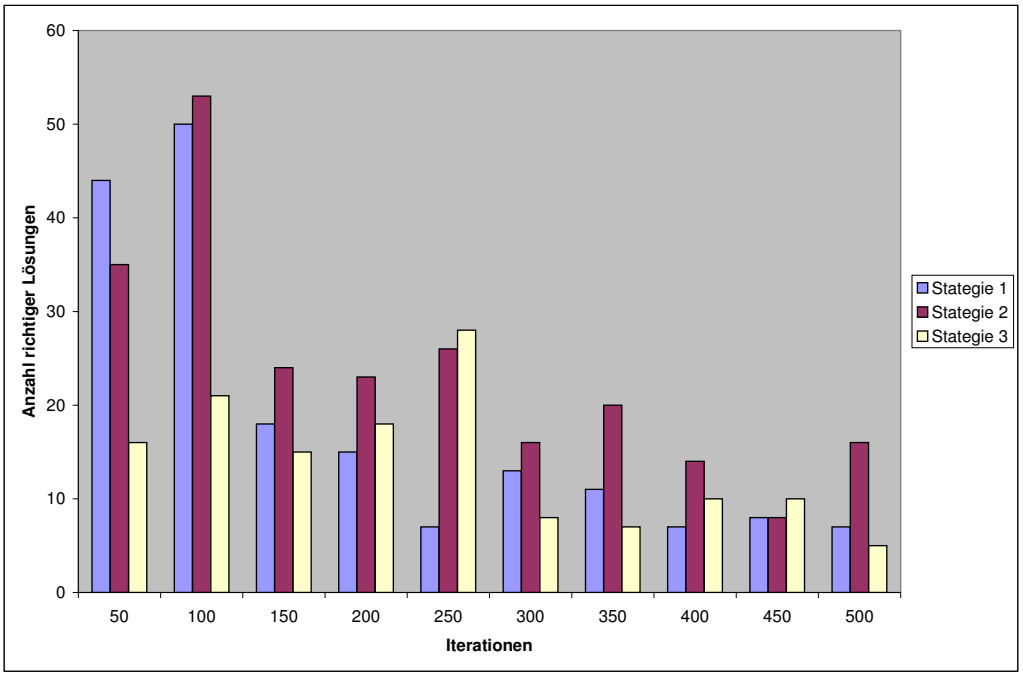


Abbildung 7.5: Anzahl der VNS Läufe bis die richtige Lösung gefunden wurde.

hen Anzahl von Durchläufen gefunden wurde, weiter zu verbessern. Die Zeit, die es dauern würde die richtige Lösung zu finden steht unter Umständen in keinem Verhältnis zu der Zeit, die man benötigt, die verbleibende Lösung zusammenzufügen. Man betrachte dazu eine fast fertige Seite und überlege sich selbst wie lange es dauert, eine solche Lösung zu einer korrekten zusammenzufügen. Durch einfaches Lesen einer Zeile findet man innerhalb kürzester Zeit die richtige Lösung.

Dazu soll das Beispiel in der Abbildung 7.7 dienen, in dem die grauen und roten Linien die Schnitte der Seite darstellen. Man sieht an dieser teilweise rekonstruierten Seite, dass von einem Mensch ohne viel Aufwand die richtige Lösung erfasst werden kann. Auch wenn die roten Striche noch eine zusätzliche Hilfe darstellen, die nur wegen der Tatsache zur Verfügung stehen, dass dem Programm im Testfall die richtige Lösung bekannt ist, würde auch ohne diese Markierung eine vollständige Richtigstellung sehr einfach möglich sein. Dazu bräuchte man nur eine der Zeilen (verstehend) zu lesen

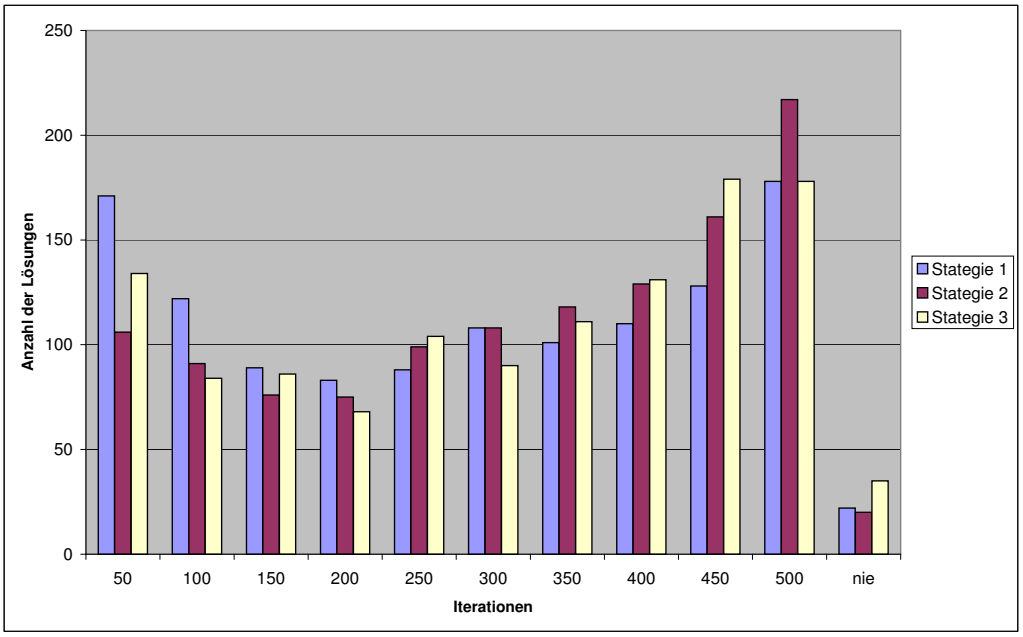
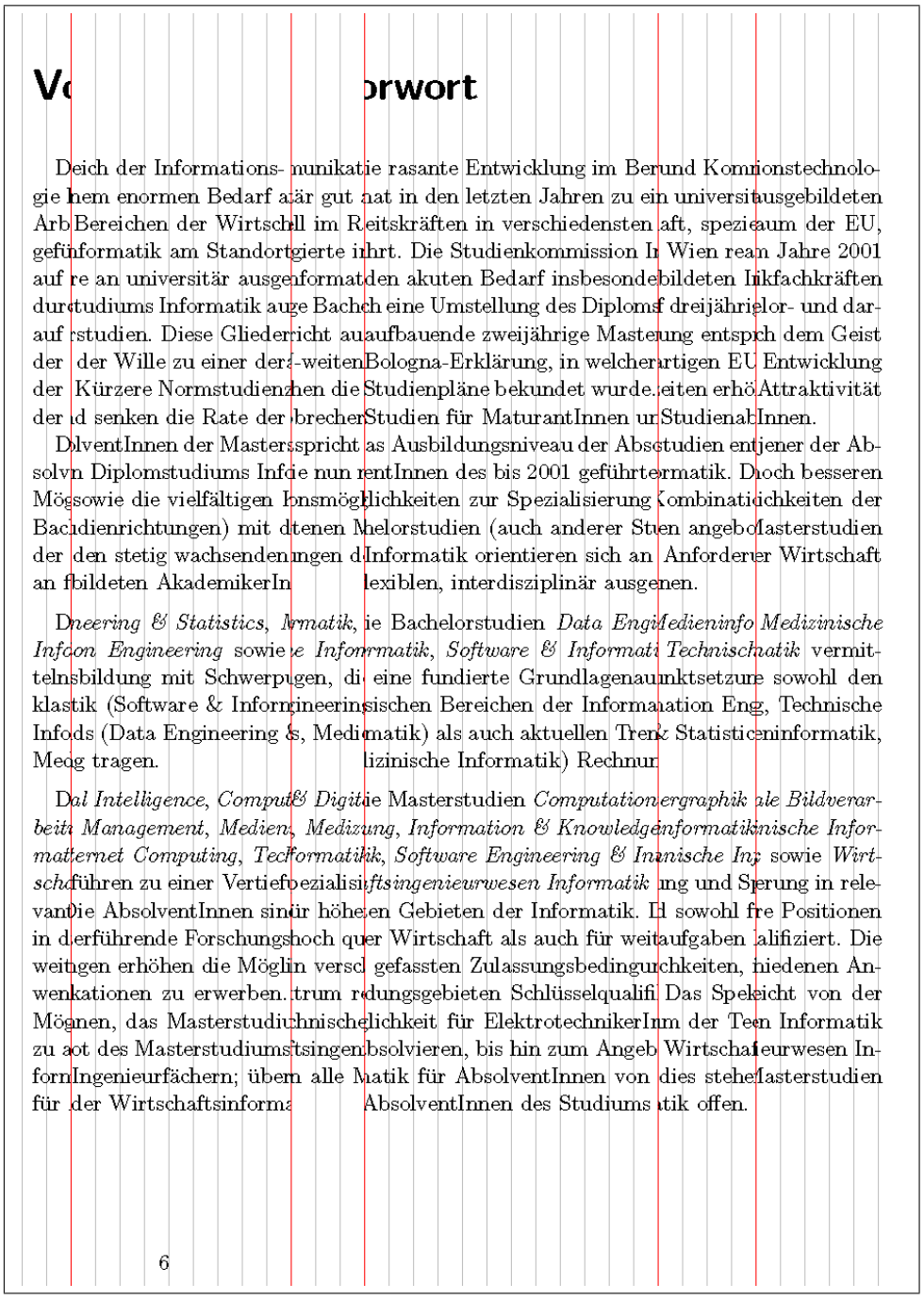


Abbildung 7.6: Anzahl der VNS Läufe bis die letzte Verbesserung und somit beste, aber nicht unbedingt richtige Lösung gefunden wurde.

und würde aus dem Kontext der Worte sofort ersehen, welcher Buchstabe als nächstes folgen müsste. Dadurch könnte man in einem einzigen Durchgang (durch eine einzige Zeile) die richtige Lösung erzeugen.

7.6 Wirkung von Nachbarschaftsstrukturen

Um Herauszufinden, wie wirksam die einzelnen Nachbarschaftsstrukturen in den Testläufen waren, wurden die Testläufe statistisch ausgewertet und die Ergebnisse graphisch dargestellt. Die Namen in den Graphiken entsprechen in folgender Weise den Nachbarschaftsstrukturen, die in Abschnitt 5.5 definiert wurden:



Vo orwort

Deich der Informations- munikatie rasante Entwicklung im Berund Komnionstechnologie hem enormen Bedarf a:är gut nat in den letzten Jahren zu ein universitausgebildeten Arb Bereichen der Wirtschll im Reitskräften in verschiedensten aft, spezieaum der EU, gefünformatik am Standortgierte inhrt. Die Studienkommission In Wien rean Jahre 2001 auf re an universitär ausgenformatden akuten Bedarf insbesondebildeten Inikfachkräften durtudiums Informatik auge Bachch eine Umstellung des Diplomsf dreijährielor- und daraufstudien. Diese Gliederricht auaufbauende zweijährige Masteung entspch dem Geist der der Wille zu einer der:-weiten Bologna-Erklärung, in welcherartigen EU Entwicklung der Kürzere Normstudienzhen die Studienpläne bekundet wurde.eiten erhöAttraktivität der d senken die Rate der brecher Studien für MaturantInnen ur StudienabInnen.

DolventInnen der Mastersspricht as Ausbildungsniveau der Absstudien entjener der Absolvn Diplomstudiums Infie nun rentInnen des bis 2001 geführtermatik. Doch besseren Mögsowie die vielfältigen Insmöglichkeiten zur Spezialisierung Kombinatichkeiten der Bacdienrichtungen) mit dtenen Melorstudien (auch anderer Stuen angebolasterstudien der den stetig wachsenden ngen d Informatik orientieren sich an Anforderer Wirtschaft an fbildeten AkademikerIn lexiblen, interdisziplinär ausgenen.

D*neering & Statistics*, *Irmatik*, ie Bachelorstudien *Data Engifedieninfo Medizinische Infcon Engineering* sowie *e Infonrmatik*, *Software & Informati Technischatik* vermittelnsbildung mit Schwerpugen, di eine fundierte Grundlagenaunktsetzure sowohl den klastik (Software & Informgineerin sischen Bereichen der Informaation Eng, Technische Infods (Data Engineering &, Medimatik) als auch aktuellen Trenk Statisticeninformatik, Medg tragen. lizinische Informatik) Rechnur

D*al Intelligence*, *Comput& Digit* ie Masterstudien *Computationergraphik ale Bildverarbeit Management*, *Medien*, *Medizung*, *Information & Knowledgenformatiknische Informatternet Computing*, *Tecformatik*, *Software Engineering & Innische In* sowie *Wirtschf*führen zu einer Vertiefbezialisi*ftsingenieurwesen Informatik* ng und Sperung in relevanDie AbsolventInnen sindr höhen Gebieten der Informatik. El sowohl fre Positionen in derführende Forschungshoch quer Wirtschaft als auch für weitaufgaben alifiziert. Die weitgen erhöhen die Möglin versc gefassten Zulassungsbedingurchkeiten, hiedenen Anwenkationen zu erwerben. trum rdungsgebieten Schlüsselqualifi Das Spekeicht von der Mögnen, das Masterstudiuchnischelichkeit für ElektrotechnikerInm der Teen Informatik zu ot des Masterstudiumsitsingenbsolvieren, bis hin zum Angeb Wirtschaeurwesen InformIngenieurfächern; übern alle Matik für AbsolventInnen von dies stehelasterstudien für der Wirtschaftsinforma AbsolventInnen des Studiums atik offen.

6

Abbildung 7.7: Eine Lösung der Instanz 01, die in 50 Streifen zerschnitten und nicht vollständig korrekt zusammengesetzt wurde. Die roten Linien zeigen die Schnitte zwischen falsch zusammengesetzten Streifen an.

	Bezeichnung	Beschreibung
$\mathcal{N}_1$	ROT-DETECT&POS	Rotationserkennung mit Positionsverschiebung
$\mathcal{N}_2$	ROT	Rotation eines Streifen
	ROT+B	Rotation eines Blocks
$\mathcal{N}_3$	POS	Umsetzen eines Streifen an eine neue Position
	POS+B	Umsetzen eines Blocks an eine neue Position
$\mathcal{N}_4$	SHI	Verschieben der Streifen
	SHI+B	Verschieben der Blöcke

Die erste in Abbildung 7.8 dargestellte Graphik zeigt wie häufig –bezogen auf die gesamte Anzahl an durchgeführten variablen Nachbarschaftssuchen – mit Hilfe einer Nachbarschaftsstruktur eine Verbesserung erzielt wurde. Man sieht, dass der Zug POS eindeutig am häufigsten durchgeführt wurde, wenn es zu einer Verbesserung der Lösung kam. Allerdings lässt sich das auch erklären, weil die Qualität der Startlösung normalerweise recht gut war, da die Vorverarbeitung mittels der Buchstabenerkennung und der Rotationserkennung anhand der Leerzeilen meistens eine richtige Orientierung für alle Streifen produziert hat. Die in der nächsten Abbildung 7.9 dargestellte Statistik zeigt, wie gut die Verbesserung war, sofern eine solche mittels der angezeigten Nachbarschaftsstruktur gefunden wurde. Es zeigt sich in dieser Graphik, dass die Strategie 2 im Vergleich zu den beiden anderen Strategien immer eine stärkere Verbesserung erzielen konnte. Vor allem Strategie 3 hat beim Umsetzen von Blöcken (POS+B) durchschnittlich schlecht abgeschnitten.

7.7 Vergleich der Instanzen

In diesem Abschnitt wird kurz das Verhalten der Algorithmen bezogen auf die Testinstanzen betrachtet. In allen folgenden Abbildungen ist der relative Fehler der gefundenen Lösungen bezogen auf die richtige Lösung dargestellt. In der Abbildung 7.10 erkennt man den „erwarteten“ Verlauf, dass bei steigender Anzahl an Streifen (in der Instanz 02) die Abweichung ansteigt. Alle drei Strategien verhalten sich jedes Mal sehr ähnlich und es ist kaum ein

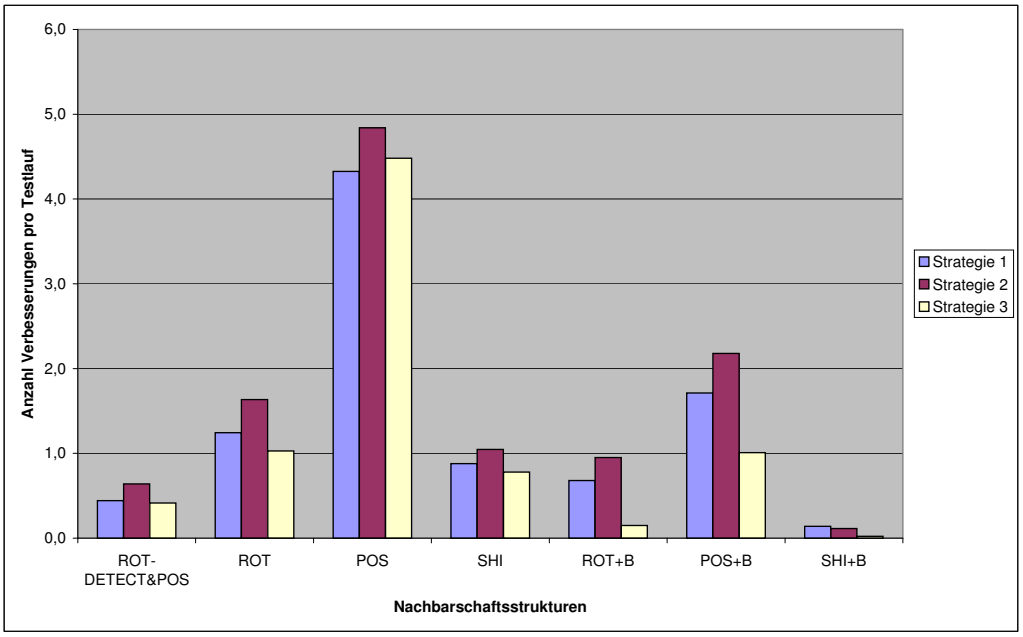


Abbildung 7.8: Die Anzahl an gefunden Verbesserungen für jede Nachbarschaftsstruktur über alle Testdurchläufe.

Trend erkennbar, welche Strategie besser ist.

Sehr viel anders sieht die Situation in der Abbildung 7.11 aus. Hier sieht man, dass sich die Strategie 2 mit steigender Streifenzahl relativ gut verhält. Die Strategie 1 ist etwas schlechter als die Strategie 2 und das Ergebnis wird auch mit steigender Anzahl von Streifen schlechter, was aber durchaus erwartet wurde. Die Strategie 3 hat grundsätzlich immer, vor allem aber bei den ersten beiden Instanzen mit 20 und 29 Streifen, schlechter abgeschnitten, was eigentlich nicht erwartet werden konnte.

Den Einfluss des Zufalls kann man in der Abbildung 7.12 sehen. Wie an etlichen Stellen beobachtet, ist durch eine ungünstige Wahl von Schnitten ein vollkommen anderes Ergebnis zu erhalten. In diesem Beispiel der Instanz 10 sieht man, dass sich die drei Strategien zwar relativ gleich verhalten, aber die Schnittkanten bei der Instanz mit 29 Streifen – und zwar nur dort – (zufällig) so angeordnet waren, dass die erzeugten Lösungen durch die verwendete Bewertungsfunktion nicht mehr der Lösung entsprechend korrekt

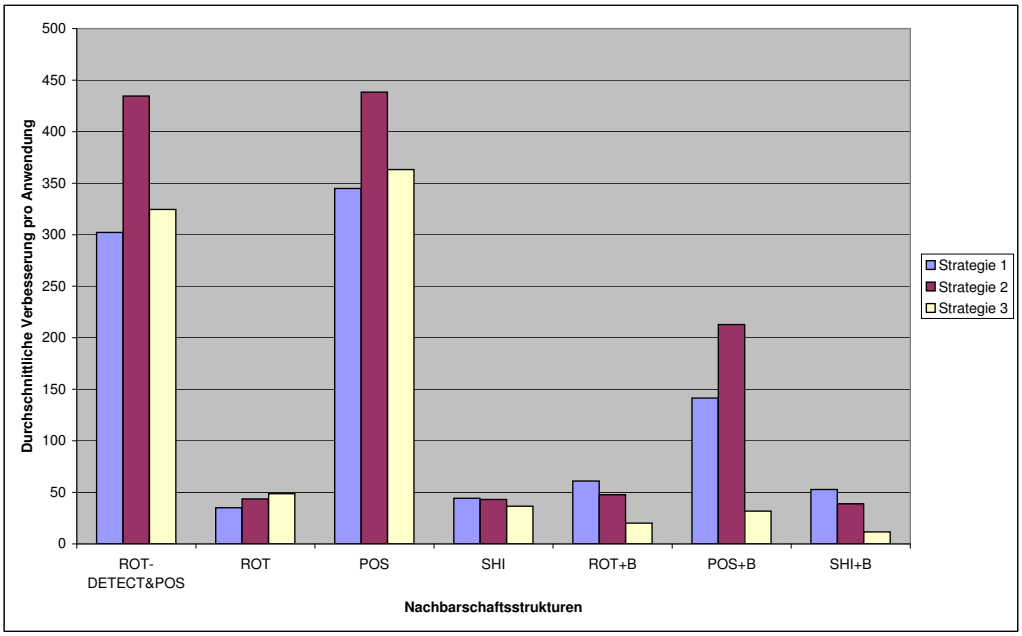
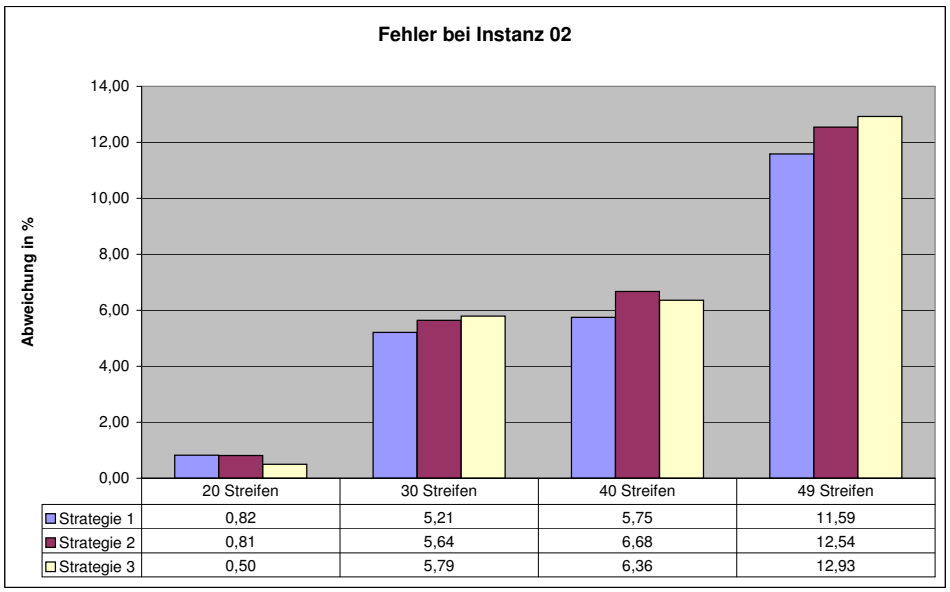


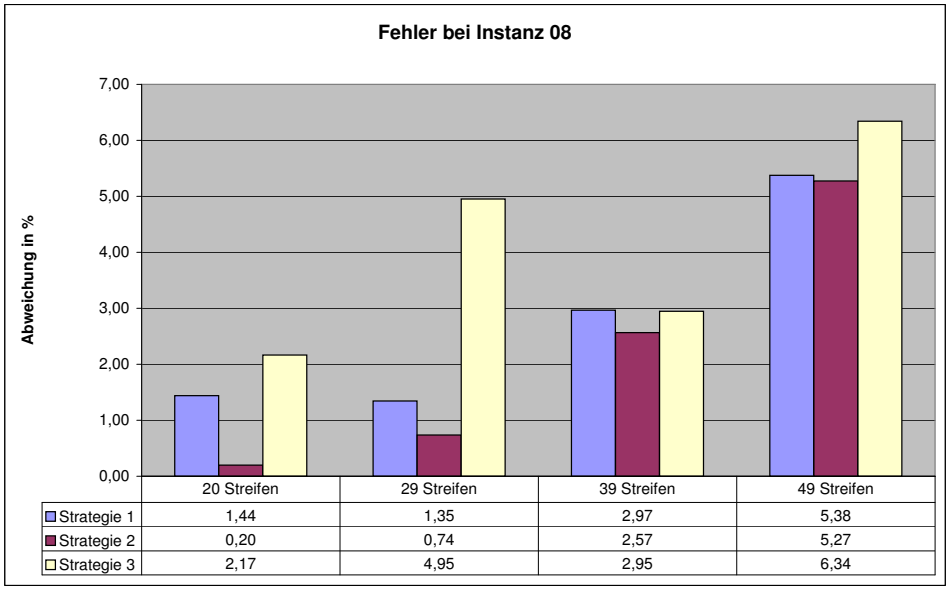
Abbildung 7.9: Die erzielte durchschnittliche Verbesserung in den Fällen, wo eine Verbesserung erreicht werden konnte.

berechnet werden konnten. Daher kam es hier (bei allen Strategien) zu einer negativen Abweichung, also einer Lösung, die besser bewertet wurde als die tatsächlich richtige Anordnung der Streifen.



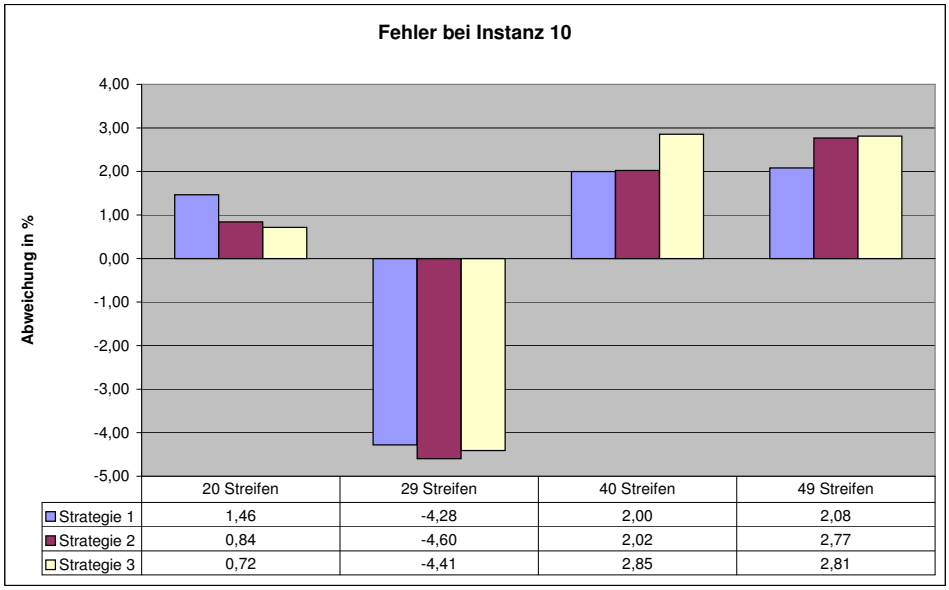
	20 Streifen	30 Streifen	40 Streifen	49 Streifen
Strategie 1	0,82	5,21	5,75	11,59
Strategie 2	0,81	5,64	6,68	12,54
Strategie 3	0,50	5,79	6,36	12,93

Abbildung 7.10: Darstellung der relativen Abweichungen von der richtigen Lösung bei Instanz 02.



	20 Streifen	29 Streifen	39 Streifen	49 Streifen
Strategie 1	1,44	1,35	2,97	5,38
Strategie 2	0,20	0,74	2,57	5,27
Strategie 3	2,17	4,95	2,95	6,34

Abbildung 7.11: Darstellung der relativen Abweichungen von der richtigen Lösung bei Instanz 08.



	20 Streifen	29 Streifen	40 Streifen	49 Streifen
Strategie 1	1,46	-4,28	2,00	2,08
Strategie 2	0,84	-4,60	2,02	2,77
Strategie 3	0,72	-4,41	2,85	2,81

Abbildung 7.12: Darstellung der relativen Abweichungen von der richtigen Lösung bei Instanz 10.

Kapitel 8

Schlussfolgerung und Zukünftiges

In dieser Diplomarbeit wurde das Problem behandelt, eine Textseite, die in Streifen geschnitten wurde, wieder zusammenzusetzen. Diese Aufgabe wurde als kombinatorisches Optimierungsproblem formuliert und eine variablen Nachbarschaftssuche zur Lösungssuche entwickelt und implementiert.

Zur Lösung des Problems wurde zunächst eine Bewertungsfunktion eingeführt, die den Fehler bewertet, der sich ergibt, wenn zwei Streifen nebeneinander liegen. Die Summe über alle diese Fehler soll bei dieser Problemstellung minimiert werden. Teilweise kann es aber passieren, dass eine Lösung besser bewertet wird als die ursprüngliche, in Streifen zerschnitte Textseite. Die Wahl einer ambitionierteren Bewertungsfunktion würde helfen dieses Problem zu vermeiden. Dies ist aber Teil des interdisziplinären Forschungsbereichs der Mustererkennung und hätte daher den Rahmen der Diplomarbeit überstiegen.

Die Problematik, dass die Orientierung der einzelnen Streifen nicht von vornherein bekannt ist, war bei der Implementierung der Lösungssuche eine der Herausforderungen. Zwei neue Methoden sind entstanden und wurden getestet um die Orientierung der Streifen richtig bestimmen zu können. Zum einen wurde eine simple Buchstabenerkennung entwickelt, die sich auf die

Erkennung von Mustern stützt, die im Anwendungsfall aus bis zu drei Linien sowie jeweils einem Geltungsbereich bestanden. Die Resultate waren viel versprechend und in vielen Fällen wurden daher die Streifen so behandelt, als ob die Orientierung von Anfang an bekannt und richtig gewesen wäre. Als zweite Methode wurde eine Erkennung der Zeilenzwischenräume entwickelt, die die Asymmetrie des Textes auf einer Seite nützt, um alle Streifen gemeinsam in die gleich Richtung orientieren zu können. Auf einigen der Textseiten konnte auf eindeutige Weise festgestellt werden, welche Streifen falsch und welche richtig gedreht waren und dadurch die Aufgabenstellung sofort auf eine einheitlich gedrehte Problemstellung reduziert werden. In den anderen Fällen wurde mit derselben Methode ein heuristischer Ansatz gefunden, mit dem einige der Streifen der Orientierung der anderen angepasst werden konnten.

Um die Wirkung der Anwendung der Nachbarschaftsstrukturen zu erhöhen wurde ein Blockbildungsverfahren implementiert, das eine Reihe von Streifen zu einem Block zusammenfasst, der in weiterer Folge wie ein einzelner Streifen behandelt wird. Später wird der Block wieder aufgelöst und die Blockbildung startet von vorne mit zumeist anderen Streifen. Dafür wurden drei unterschiedliche Strategien eingesetzt, deren Parameter mit steigender Anzahl von Iterationen der variablen Nachbarschaftssuche verändert wurden. Als beste hat sich jene Methode herausgestellt, die eine konstante Anzahl von Vereinigungen (von Schnitten) zu einem Block zusammenführt, wobei diese Anzahl ansteigt, je öfter die VNS durchgeführt wurde.

Bei der Rekonstruktion von mehreren Seiten kann die Methode der Erkennung der Zwischenräume auch zukünftig dazu dienen, die Streifen – auch von mehreren Seiten – zu Gruppen zusammenzufassen, die jeweils den ganzen Seiten entsprechen. In dieser Arbeit wurde jedoch kein Test in diese Richtung durchgeführt.

Als Verbesserung könnte man eine Texterkennung mittels Wörterbuch verwenden um viele der Wörter im Text zu finden; so könnte man zusätzlich die Blockbildung dahingehend abändern, dass diese Blöcke vermutlich über den gesamten Verlauf der Rekonstruktion nicht aufgelöst werden müssen. Eine Alternative dazu wäre es, das Ergebnis durch menschliche Interaktion

zu optimieren. Dazu zählt zum einen die Blöcke permanent festzulegen und zum anderen eine fast richtige Lösung endgültig zusammenzusetzen, da diese oft aus wenigen Blöcken mit richtigen Streifen (aber diese Blöcke in falscher Reihenfolge) besteht.

Literaturverzeichnis

[1] Joanna Balme. Reconstruction of shredded documents in the absence of shape information. Technical report, Yale University, USA, 2007. Tech Report.

[2] Karl-Heinz Best. Zur Häufigkeit von Buchstaben, Leerzeichen und anderen Schriftzeichen in deutschen Texten. *Glottometrics*, 11:9–31, 2005.

[3] Robert S. Caprari. Algorithm for text page up/down orientation determination. *Pattern Recognition Letters*, 21(4):311 – 317, 2000.

[4] Jill Cirasella, David Johnson, Lyle McGeoch, and Weixiong Zhang. The asymmetric traveling salesman problem: Algorithms, instance generators, and tests. In Adam Buchsbaum and Jack Snoeyink, editors, *Algorithm Engineering and Experimentation*, volume 2153 of *Lecture Notes in Computer Science*, pages 32–59. Springer Berlin / Heidelberg, 2001.

[5] Patrick De Smet, Johan De Bock, and Wilfried Philips. Semiautomatic reconstruction of strip-shredded documents. In Amir Said and John G. Apostolopoulos, editors, *Image and Video Communications and Processing 2005*, volume 5685 of *Proceedings of SPIE*, pages 239–248, San Jose, CA, USA, 2005. SPIE.

[6] F. Glover and G. Kochenberger, editors. *Handbook of Metaheuristics*, volume 57 of *International Series in Operations Research & Management Science*. Kluwer Academic Publishers, Norwell, MA, 2003.

[7] Fred Glover, James P. Kelly, and Manuel Laguna. Genetic algorithms and tabu search: Hybrids for optimization. *Computers & Operations Research*, 22(1):111 – 134, 1995. Genetic Algorithms.

[8] David Goldberg, Christopher Malon, and Marshall Bern. A global approach to automatic solution of jigsaw puzzles. *Computational Geometry Theory and Applications*, 28(2-3):165–174, 2004.

[9] Pierre Hansen and Nenad Mladenović. Variable neighborhood search: Principles and applications. *European Journal of Operational Research*, 130(3):449–467, 2001.

[10] Pierre Hansen and Nenad Mladenović. A tutorial on variable neighborhood search. Technical Report G-2003-46, Les Cahiers du GERAD, HEC Montréal and GERAD, Canada, 2003.

[11] Pierre Hansen and Nenad Mladenović. Variable neighborhood search methods. In Christodoulos A. Floudas and Panos M. Pardalos, editors, *Encyclopedia of Optimization*, pages 3975–3989. Springer, 2009.

[12] Pierre Hansen, Nenad Mladenović, and José A. Moreno-Pérez. Variable neighbourhood search: methods and applications. *4OR*, 6(4):319–360, 2008.

[13] Holger Hoos and Thomas Stützle. *Stochastic Local Search: Foundations & Applications*. Morgan Kaufmann Publishers Inc., 2004.

[14] Gunnar W. Klau, Neal Lesh, Joe Marks, and Michael Mitzenmacher. Human-guided tabu search. In *Eighteenth national conference on Artificial intelligence*, pages 41–47, Menlo Park, CA, USA, 2002. American Association for Artificial Intelligence.

[15] Gunnar W. Klau, Neal Lesh, Joe Marks, and Michael Mitzenmacher. Human-guided search. *Journal of Heuristics*, 16:289–310, June 2010.

[16] D.A. Kosiba, P.M. Devaux, S. Balasubramanian, T.L. Gandhi, and K. Kasturi. An automatic jigsaw puzzle solver. In *Pattern Recognition,*

1994. Vol. 1 - Conference A: Computer Vision & Image Processing., Proceedings of the 12th IAPR International Conference on, volume 1, pages 616 –618 vol.1, 1994.

[17] E.L. Lawler, A.H.G. Rinnooy-Kan, J.K. Lenstra, and D.B. Shmoys. *The Traveling salesman problem: a guided tour of combinatorial optimization.* Wiley Series in Discrete Mathematics and Optimization Series. Wiley, 1985.

[18] Wolfgang Morandell. Evaluation and reconstruction of strip-shredded text documents. Master's thesis, Vienna University of Technology, Austria, 2008.

[19] B. Nickolay and J. Schneider. The world´s biggest puzzle : The stasi snippet project - the ultimative challenge for digital image processing. *Inspect*, 9(3), 2008.

[20] Christos H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms And Complexity.* Prentice-Hall, 1982.

[21] Matthias Prandtstetter. *Hybrid Optimization Methods for Warehouse Logistics and the Reconstruction of Destroyed Paper Documents.* PhD thesis, Vienna University of Technology, Austria, 2009.

[22] Anna Skeoch. *An Investigation into Automated Shredded Document Reconstruction using Heuristic Search Algorithms.* PhD thesis, University of Bath, UK, 2006.

[23] Anna Ukovich and Giovanni Ramponi. Feature extraction and clustering for the computer-aided reconstruction of strip-cut shredded documents. *Journal of Electronic Imaging*, 17(1):013008–1–013008–13, 2008.

[24] Joost van Beusekom, Faisal Shafait, and Thomas Breuel. Combined orientation and skew detection using geometric text-line modeling. *International Journal on Document Analysis and Recognition*, 13:79–92, 2010.

Anhang A

Instanzen

Die in dieser Arbeit verwendeten Instanzen entsprechen denen aus [21], werden aber der Vollständigkeit halber hier angeführt.

Vorwort

Die rasante Entwicklung im Bereich der Informations- und Kommunikationstechnologie hat in den letzten Jahren zu einem enormen Bedarf an universitär gut ausgebildeten Arbeitskräften in verschiedensten Bereichen der Wirtschaft, speziell im Raum der EU, geführt. Die Studienkommission Informatik am Standort Wien reagierte im Jahre 2001 auf den akuten Bedarf insbesondere an universitär ausgebildeten Informatikfachkräften durch eine Umstellung des Diplomstudiums Informatik auf dreijährige Bachelor- und darauf aufbauende zweijährige Masterstudien. Diese Gliederung entspricht auch dem Geist der Bologna-Erklärung, in welcher der Wille zu einer derartigen EU-weiten Entwicklung der Studienpläne bekundet wurde. Kürzere Normstudienzeiten erhöhen die Attraktivität der Studien für MaturantInnen und senken die Rate der StudienabbrecherInnen.

Das Ausbildungsniveau der AbsolventInnen der Masterstudien entspricht jener der AbsolventInnen des bis 2001 geführten Diplomstudiums Informatik. Die nun noch besseren Möglichkeiten zur Spezialisierung sowie die vielfältigen Kombinationsmöglichkeiten der Bachelorstudien (auch anderer Studienrichtungen) mit den angebotenen Masterstudien der Informatik orientieren sich an den stetig wachsenden Anforderungen der Wirtschaft an flexiblen, interdisziplinär ausgebildeten AkademikerInnen.

Die Bachelorstudien *Data Engineering & Statistics*, *Medieninformatik*, *Medizinische Informatik*, *Software & Information Engineering* sowie *Technische Infomatik* vermitteln eine fundierte Grundlagenausbildung mit Schwerpunktsetzungen, die sowohl den klassischen Bereichen der Informatik (Software & Information Engineering, Technische Infomatik) als auch aktuellen Trends (Data Engineering & Statistics, Medieninformatik, Medizinische Informatik) Rechnung tragen.

Die Masterstudien *Computational Intelligence*, *Computergraphik & Digitale Bildverarbeitung*, *Information & Knowledge Management*, *Medieninformatik*, *Medizinische Informatik*, *Software Engineering & Internet Computing*, *Technische Informatik* sowie *Wirtschaftsingenieurwesen Informatik* führen zu einer Vertiefung und Spezialisierung in relevanten Gebieten der Informatik. Die AbsolventInnen sind sowohl für höhere Positionen in der Wirtschaft als auch für weiterführende Forschungsaufgaben hoch qualifiziert. Die weit gefassten Zulassungsbedingungen erhöhen die Möglichkeiten, in verschiedenen Anwendungsgebieten Schlüsselqualifikationen zu erwerben. Das Spektrum reicht von der Möglichkeit für ElektrotechnikerInnen, das Masterstudium der Technischen Informatik zu absolvieren, bis hin zum Angebot des Masterstudiums Wirtschaftsingenieurwesen Informatik für AbsolventInnen von Ingenieurfächern; überdies stehen alle Masterstudien für AbsolventInnen des Studiums der Wirtschaftsinformatik offen.

Abbildung A.1: Instanz 01

Abbildung A.2: Instanz 02

3. Medieninformatik

3.1. Präambel

Das Studium *Medieninformatik* versteht sich als spezielle anwendungsorientierte Informatik, die die Bereiche Design, Computergraphik, Bildverarbeitung und Multimedia – kurz: die zunehmende Auseinandersetzung mit dem Begriff des *Visuellen* – in den Mittelpunkt stellt. Diese Bereiche entwickelten in den letzten Jahren in und außerhalb der Informatik eine starke Dynamik, die die Lehrinhalte beeinflusst und neue Berufsfelder erschließt. Ihre kompetente Bearbeitung verlangt nicht nur eine andere Gewichtung und informatikinterne Ausweitung der traditionellen Studieninhalte, sondern auch die Ergänzung um Themen aus dem Bereich Design.

Im Mittelpunk der Medieninformatik steht der Umgang mit dem Visuellen, vornehmlich mit Bildern, bildhaften Darstellungen und graphischen Symbolen, der in allen Aspekten studiert wird, und zwar unter besonderer Berücksichtigung der Verwendung von Computern. Genau aus diesem Grund auch wird der Studiengang auf Initiative der Informatik vorangetrieben.

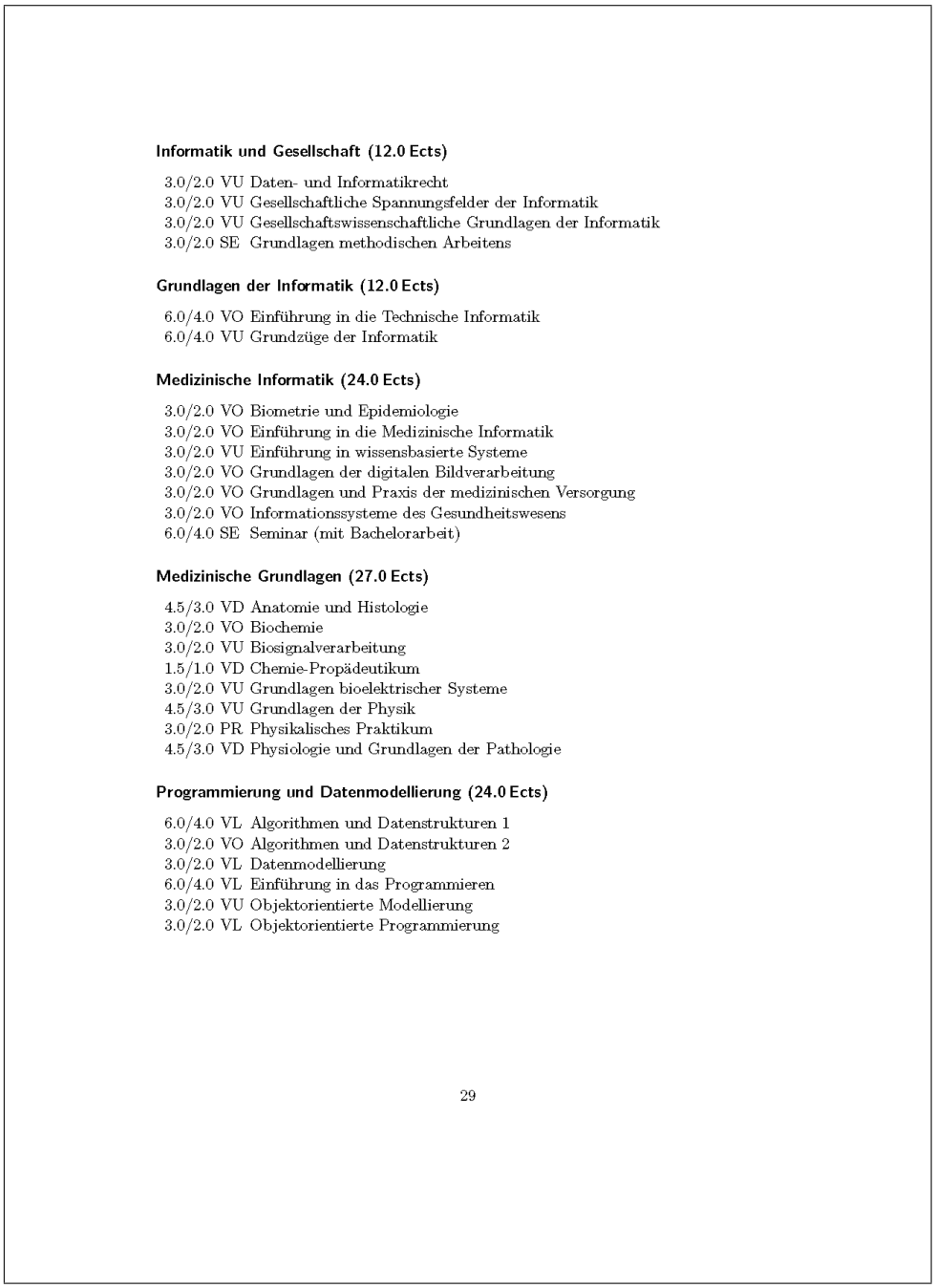
Multimedia und ihre Anwendungen gelten als ein wichtiger Zukunftsbereich in der Informatik. Aufgaben wie die Präsentation von Informationen mit unterschiedlichen Medien, die Gestaltung der interaktiven Schnittstellen und die Navigation durch virtuelle Welten stellen derart hohe Qualifikationsansprüche an zukünftige MedieninformatikerInnen, dass die Einrichtung eines eigenen Studiums dafür unbedingt notwendig ist.

Hierzu wird als Kern des Studienganges eine solide Grundausbildung in der Informatik angeboten, mit einer Spezialisierung auf visuelle Themen wie Design, Computergraphik, Bildverarbeitung und Mustererkennung. Hier sind Gebiete wie die technische Bildaufnahme, Bildvorverarbeitung, Bildauswertung und automatische Bildinterpretation vertreten, aber auch neben Bildwiedergabe und Bildkommunikation alle Aspekte der Bildsynthese, der virtuellen Realität und der wissenschaftlichen Visualisierung.

3.2. Qualifikationsprofil der Absolventinnen und Absolventen

Das Studium soll eine wissenschaftlich geprägte Ausbildung vermitteln, die Theorie, Fachwissen und praktische Kenntnisse von Medientechnik, Computergraphik, der digitalen Bildverarbeitung und Mustererkennung einschließt. Es soll die Studierenden in die Lage versetzen, Methoden und Werkzeuge aus den oben genannten Gebieten zu verstehen, anzuwenden sowie sich eigenständig an ihrer Erforschung und Weiterentwicklung zu beteiligen. Studienziel ist weiters die Vermittlung von Wissen um die kreative Gestaltung der Medien und deren Produktionsprozess. Dazu gehört die Befähigung der Auszu-

Abbildung A.3: Instanz 03



Informatik und Gesellschaft (12.0 Ects)

3.0/2.0 VU Daten- und Informatikrecht
3.0/2.0 VU Gesellschaftliche Spannungsfelder der Informatik
3.0/2.0 VU Gesellschaftswissenschaftliche Grundlagen der Informatik
3.0/2.0 SE Grundlagen methodischen Arbeitens

Grundlagen der Informatik (12.0 Ects)

6.0/4.0 VO Einführung in die Technische Informatik
6.0/4.0 VU Grundzüge der Informatik

Medizinische Informatik (24.0 Ects)

3.0/2.0 VO Biometrie und Epidemiologie
3.0/2.0 VO Einführung in die Medizinische Informatik
3.0/2.0 VU Einführung in wissensbasierte Systeme
3.0/2.0 VO Grundlagen der digitalen Bildverarbeitung
3.0/2.0 VO Grundlagen und Praxis der medizinischen Versorgung
3.0/2.0 VO Informationssysteme des Gesundheitswesens
6.0/4.0 SE Seminar (mit Bachelorarbeit)

Medizinische Grundlagen (27.0 Ects)

4.5/3.0 VD Anatomie und Histologie
3.0/2.0 VO Biochemie
3.0/2.0 VU Biosignalverarbeitung
1.5/1.0 VD Chemie-Propädeutikum
3.0/2.0 VU Grundlagen bioelektrischer Systeme
4.5/3.0 VU Grundlagen der Physik
3.0/2.0 PR Physikalisches Praktikum
4.5/3.0 VD Physiologie und Grundlagen der Pathologie

Programmierung und Datenmodellierung (24.0 Ects)

6.0/4.0 VL Algorithmen und Datenstrukturen 1
3.0/2.0 VO Algorithmen und Datenstrukturen 2
3.0/2.0 VL Datenmodellierung
6.0/4.0 VL Einführung in das Programmieren
3.0/2.0 VU Objektorientierte Modellierung
3.0/2.0 VL Objektorientierte Programmierung

29

Abbildung A.4: Instanz 04

Herkunfts-studium	Basismodule (à 18.0 Ects)			Vertiefungsmodule (à 6.0 Ects)			
	Ing.wiss.	Wirtschaft	Informatik	Ing.wiss.	Wirtschaft	Informatik	beliebig
Informatik	1	1	0	0 oder 1	1	2 oder 1	0
Wirtschafts-informatik	1	0	0	1	1	2	2
Ingenieur-wiss.	0	1	1	0 oder 1	1	2 oder 1	0
Wirtschafts-ing.wes. MB	0	0	1	1	1	2	2
Wirtschaft	1	0	1	0 oder 1	1	2 oder 1	0

Für Herkunftsstudien, die nicht von diesem Schema erfasst werden, kann das studienrechtliche Organ Basismodule festlegen.

Im Rahmen der Vertiefungsmodule sind Seminare im Umfang von 3.0 Ects bis 6.0 Ects zu absolvieren.

Freie Wahlfächer und Soft Skills (9.0 Ects)

Siehe Abschnitt 7.4.

Diplomarbeit (30.0 Ects)

Siehe Abschnitt 7.5.

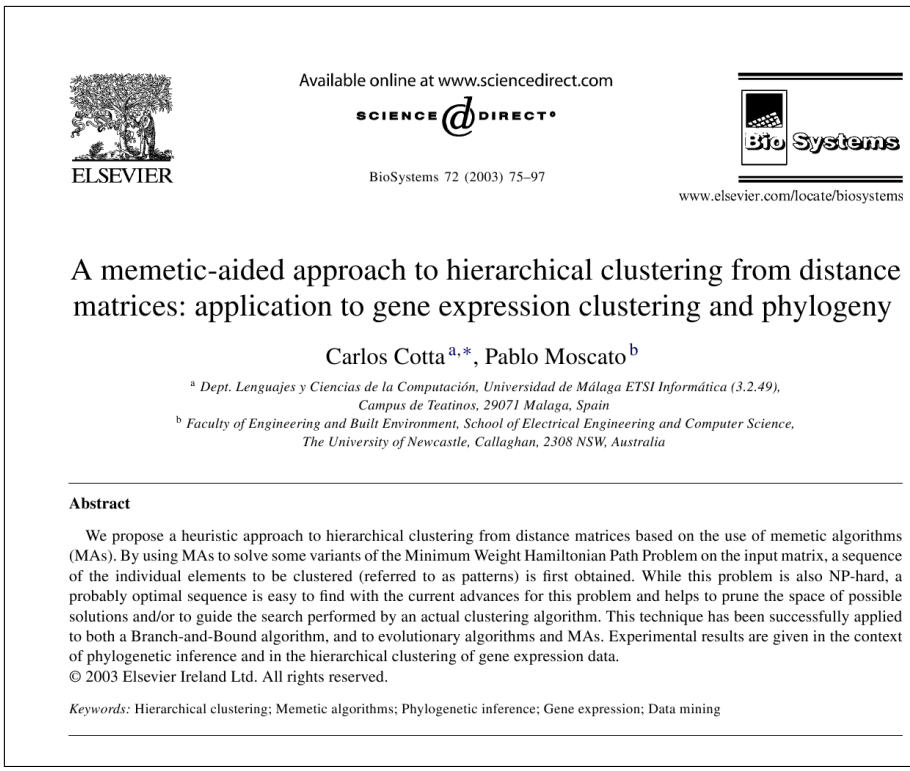
15.5. Basis- und Vertiefungsmodule

Basismodul Informatik

Es sind Lehrveranstaltungen im Umfang von 18.0 Ects aus den folgenden Lehrveranstaltungen zu wählen. Die Kenntnisse der Lehrveranstaltungen dieses Moduls werden als Voraussetzung für das Verständnis der Vertiefungsmodule aus Informatik erwartet.

6.0/4.0 VL Algorithmen und Datenstrukturen 1
3.0/2.0 VL Datenmodellierung
6.0/4.0 VO Einführung in die Technische Informatik
3.0/2.0 VU Objektorientierte Modellierung
3.0/2.0 VL Objektorientierte Programmierung
3.0/2.0 VO Software Engineering und Projektmanagement
6.0/4.0 LU Software Engineering und Projektmanagement
6.0/4.0 VU Theoretische Informatik und Logik

Abbildung A.5: Instanz 05



Available online at www.sciencedirect.com
SCIENCE DIRECT®
ELSEVIER
BioSystems 72 (2003) 75–97
BioSystems
www.elsevier.com/locate/biosystems

A memetic-aided approach to hierarchical clustering from distance matrices: application to gene expression clustering and phylogeny

Carlos Cotta [a,*], Pablo Moscato [b]

[a] *Dept. Lenguajes y Ciencias de la Computación, Universidad de Málaga ETSI Informática (3.2.49), Campus de Teatinos, 29071 Malaga, Spain*
[b] *Faculty of Engineering and Built Environment, School of Electrical Engineering and Computer Science, The University of Newcastle, Callaghan, 2308 NSW, Australia*

Abstract

We propose a heuristic approach to hierarchical clustering from distance matrices based on the use of memetic algorithms (MAs). By using MAs to solve some variants of the Minimum Weight Hamiltonian Path Problem on the input matrix, a sequence of the individual elements to be clustered (referred to as patterns) is first obtained. While this problem is also NP-hard, a probably optimal sequence is easy to find with the current advances for this problem and helps to prune the space of possible solutions and/or to guide the search performed by an actual clustering algorithm. This technique has been successfully applied to both a Branch-and-Bound algorithm, and to evolutionary algorithms and MAs. Experimental results are given in the context of phylogenetic inference and in the hierarchical clustering of gene expression data.

Keywords: Hierarchical clustering; Memetic algorithms; Phylogenetic inference; Gene expression; Data mining

Abbildung A.6: Instanz 06

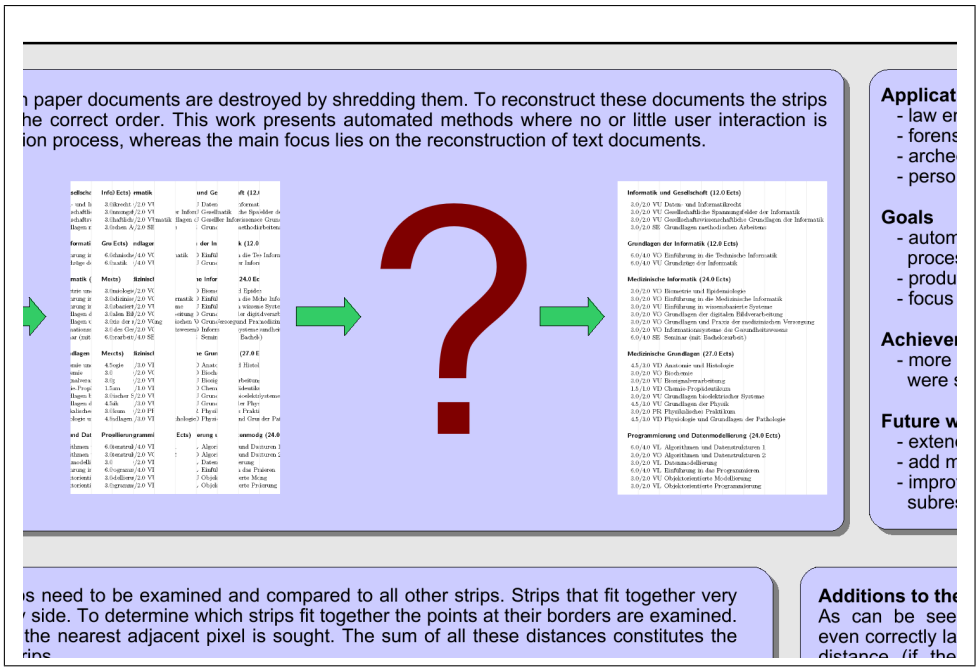


Abbildung A.7: Instanz 07

***Abstract*— The recovery of paper documents which have been disposed of is an important issue in many contexts, such as forensics and investigation sciences. The automatization of the process by means of image processing techniques can give a considerable help in the problem solution. We propose in this paper an overall architecture for the reconstruction of strip-cut shredded documents, paying particular attention to the possibility of using MPEG-7 descriptors for the strip content description.**

***Index Terms* - MPEG-7, Document reconstruction, Content-based image retrieval**

I. Introduction

One of the tasks which forensics and investigation science have to deal with is the recovery of shredded documents. Documents may be torn by hand, but more often are destroyed using a suitable mechanical equipment, which cuts them into thin strips or even, with a cross-cut shredder, into small rectangles. The reconstruction of documents that have been shredded by an office strip-shredder is a difficult and time-consuming task to be performed by a human operator, and can become an insoluble problem when the number of shredded documents is large. The problem could be considered as a particular case of *jigsaw puzzle*. The automatization of the process by means of image processing techniques can give a considerable help in the problem solution.

Computer vision methods for the solution of jigsaw puzzles have been proposed since 1963 [1]. In the latest years, jigsaw puzzle approaches have been adopted in the field of archaeology and art conservation, for the computer-aided reconstruction of two- and three-dimensional fragmented objects [2] [3] [4].

In the literature, most of the solutions proposed for the assembly of jigsaw puzzles define a model for the piece contour and perform the matching based on the contour shape, the same, rectangular, shape; thus the contour shape does not provide the necessary information for the matching. To define content features, the techniques developed in the last years for content-based image retrieval (CBIR) systems [7] [8] [9] represent a good starting point. In order to find a solution for the shredded document reconstruction problem in the context of CBIR systems, the concept of match should be replaced by the concept of similarity. We will assume that strips that are found to be similar by a CBIR system will have a high probability to be part of the same region (document).

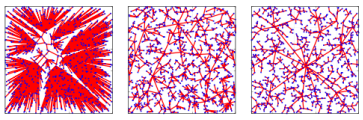
We propose in this paper an overall architecture for the reconstruction of shredded documents, paying particular attention to the possibility of using MPEG-7 descriptors for the strip content description. We hypothesize that the documents have been cut by a strip-cut shredder. To our best knowledge, this is the first attempt in the literature to solve the shredded document reconstruction problem. The paper is organized as follows. In Section II the shredded document reconstruction problem is defined. In Section III the general system architecture is presented. In Section IV the MPEG-7 descriptors are described and in Section V the experimental results are reported and discussed.

II. Problem definition

With reference to publications in which the jigsaw puzzle problem has been either defined [1] [5] or redefined for a particular application [2], a set of "puzzle rules" can be determined for the case of shredded documents. As in [3] we will distinguish between an ideal case of shredded remnants and the real, observed, case. In the ideal case we define the following rules:

- a piece (strip) is a one- or double-sided connected planar region

Abbildung A.8: Instanz 08



(a) CBTC (271.4). (b) RTC (41.18). (c) ACO (31.09).

Figure 1: A diameter constrained tree with $D = 10$ constructed using (a) the CBTC heuristic, compared to (b) RTC (best solution from 1000 runs) and (c) a solution obtained by an ant colony optimization approach (complete, Euclidean graph with 1000 nodes distributed randomly in the unit square, the corresponding objective values are given in parentheses).

of relatively short edges and the majority of the nodes have to be connected to this backbone via rather long edges, see the example in Fig. 1(a). On the contrary, a reasonable solution for this instance, shown in Fig. 1(c), demonstrates that the backbone should consist of a few longer edges to span the whole area to allow the large number of remaining nodes to be connected as leaves by much cheaper edges. In a pure greedy construction heuristic this observation is difficult to realize. In the *randomized tree construction approach* (RTC, Fig. 1(b)) from [13] not the cheapest of all nodes is always added to the partial spanning tree but the next node is chosen at random and then connected by the cheapest feasible edge. Thus at least the possibility to include longer edges into the backbone at the beginning of the algorithm is increased. For Euclidean instances RTC has been so far the best choice to quickly create a first solution as basis for exact or metaheuristic approaches.

In the following we will introduce a new construction heuristic for the BDMST problem which is especially suited for very large Euclidean instances. It is based on a hierarchical clustering that guides the algorithm to find a good backbone. This approach is then refined by a local improvement method and extended towards a greedy randomized adaptive search procedure (GRASP) [17]. A preliminary version of this work can be found in [10].

3. THE CLUSTERING HEURISTIC

The clustering-based construction heuristic can be divided into three steps: Creating a hierarchical clustering (*dendrogram*) of all instance nodes based on the edge costs, deriving a height-restricted clustering (HRC) from this dendrogram, and finding for each cluster in the HRC a good root (center) node.

3.1 Hierarchical Clustering

For the purpose of creating a good backbone especially for an Euclidean instance of the BDMST problem agglomerative hierarchical clustering seems to provide a good guidance. To get spatially confined areas, two clusters A and B are merged when $\max\{c_{a,b} : a \in A,\ b \in B\}$ is minimal over all pairs of clusters (complete linkage clustering [12]).

The agglomeration starts with each node being an individual cluster, and stops when all nodes are merged within one

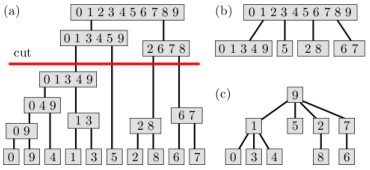


Figure 2: Hierarchical clustering (a), height-restricted clustering (b), and the resulting diameter constrained tree with $D = 4$ (c) after choosing a root for each cluster in (b).

single cluster. The resulting hierarchical clustering can be illustrated as a binary tree, also referred to as a *dendrogram*, with $|V|$ leaves and $|V| - 1$ inner nodes each representing one merging operation during clustering; see Fig. 2(a) for an example with $|V| = 10$. An inner node's distance from the leaves indicates when the two corresponding clusters – relative to each other – have been merged.

3.2 Height-Restricted Clustering

After performing the agglomerative hierarchical clustering, the resulting dendrogram is transformed into a height-restricted clustering (HRC) for the BDMST, i.e. into a representation of the clustering respecting the diameter and thus the height condition. The dendrogram itself cannot directly act as HRC since in general it will violate this constraint, see Fig. 2(a). Therefore, some of the nodes in the dendrogram have to be merged to finally get a tree of height $H - 1$, the HRC for the BDMST; see Fig. 2(b) for a diameter of $D = 4$.

For the quality of the resulting tree this merging of dendrogram nodes is a crucial step, worth significant effort. It can be described by $H - 1$ *cuts* through the dendrogram defining which nodes of it will also become part of the height-restricted clustering and which are merged with their parent clusters. As an example, starting at the root containing all instance nodes agglomerated within one single cluster the cut illustrated in Fig. 2(a) defines the dendrogram nodes $\{0, 1, 3, 4, 9\}$, $\{5\}$, $\{2, 8\}$, and $\{6, 7\}$ to become direct successors of the root cluster in the height-restricted clustering.

One fundamental question arising in this context is the way of defining the cutting positions through the dendrogram. After preliminary tests, the identification of the precise iteration at which two clusters have been merged in the agglomeration process turns out to be a good criterion. This *merge number* (or *merge#*), which allows a fine-grained control of the cutting positions, can be stored within each node of the dendrogram, with the leaves having a merge number of zero and the root $|V| - 1$.

Based on the merge numbers cutting positions ς are computed as

$$\varsigma_i = (|V|-1) - 2^{i \cdot \frac{\log_2 x}{H-1}} \qquad i = 1, \ldots, H-1, \qquad (1)$$

where x is a strategy parameter. This formula is motivated by a perfectly balanced tree, where parameter x can be interpreted as the number of nodes that shall form the backbone.

These cutting positions can now be used to build the height-restricted clustering for the BDMST, as depicted in

Abbildung A.9: Instanz 09

Table 1: Averaged objective values over all 15 Euclidean Steiner tree instances of Beasley's OR-Library with 1000 nodes for various diameter bounds and (meta-)heuristics, the standard deviations are given parentheses. In addition, the averaged maximum running times of the clustering heuristics that were used as time limit for CBTC and RTC are listed.

	without VND							with VND						
D	CBTC		RTC		Cd^A		Cd^B	t_{max}(C) [s]	RTC		Cd^B		ACO		t_{max}(C) [s]
4	329.0261	(6.02)	146.4919	(3.88)	68.3241	(0.72)	**68.3226**	(0.70)	2.54 (0.09)	65.2061	(0.55)	**65.1598**	(0.56)	65.8010	(0.48)	5.56 (1.01)
6	306.2655	(9.02)	80.8636	(2.40)	47.4045	(4.85)	**47.1702**	(4.61)	4.55 (0.49)	41.4577	(0.36)	**41.3127**	(0.50)	42.1167	(0.26)	9.94 (1.52)
8	288.3842	(7.52)	53.2535	(1.33)	37.0706	(1.35)	**36.9408**	(1.34)	5.92 (0.42)	35.0511	(0.35)	**34.2171**	(0.29)	34.7489	(0.23)	11.61 (1.61)
10	266.3665	(9.01)	41.1201	(0.68)	33.5460	(0.67)	**33.3408**	(0.66)	6.79 (0.42)	32.1181	(0.31)	**30.9704**	(0.24)	31.0388	(0.20)	13.43 (2.16)
12	250.0016	(8.01)	35.7590	(0.47)	32.2571	(0.48)	**31.9561**	(0.44)	7.11 (0.33)	30.2897	(0.29)	29.1796	(0.26)	**28.6356**	(0.23)	14.68 (2.49)
14	237.1403	(6.28)	33.3644	(0.30)	31.3790	(0.37)	**31.0176**	(0.33)	7.00 (0.64)	29.0940	(0.28)	28.0093	(0.23)	**26.6524**	(0.32)	15.05 (3.00)
16	224.3123	(5.72)	32.1965	(0.24)	30.7937	(0.33)	**30.4287**	(0.29)	7.20 (0.72)	28.2433	(0.28)	27.1363	(0.19)	**25.5760**	(0.19)	15.63 (2.89)
18	210.9872	(7.63)	31.5826	(0.24)	30.5182	(0.29)	**30.1348**	(0.27)	7.32 (0.81)	27.6008	(0.27)	26.5601	(0.20)	**24.8811**	(0.16)	16.78 (3.61)
20	197.1772	(7.99)	31.2682	(0.22)	30.3116	(0.31)	**30.0384**	(0.28)	7.57 (0.76)	27.1091	(0.26)	26.1079	(0.23)	**24.3698**	(0.15)	18.54 (3.89)
22	183.0157	(8.03)	31.0864	(0.22)	30.2344	(0.30)	**30.0739**	(0.28)	8.56 (0.98)	26.6984	(0.28)	25.8048	(0.21)	**24.0129**	(0.17)	21.39 (5.19)
24	172.8251	(10.59)	30.9921	(0.23)	**30.0202**	(0.23)	30.1603	(0.27)	8.28 (1.41)	26.3648	(0.27)	25.4523	(0.24)	**23.7723**	(0.20)	21.36 (6.42)
5	241.3032	(5.09)	117.3238	(2.22)	62.2867	(0.76)	**62.0646**	(0.67)	24.59 (2.02)	58.9883	(0.53)	**58.7930**	(0.56)	59.5964	(0.49)	30.82 (3.28)
7	222.1441	(4.50)	67.7577	(1.31)	46.7291	(3.92)	**46.4112**	(3.73)	27.94 (1.79)	39.4703	(0.34)	**39.3817**	(0.46)	39.9948	(0.25)	38.79 (4.03)
9	204.6141	(6.00)	47.3168	(0.85)	37.0224	(1.25)	**36.8904**	(1.27)	18.27 (1.68)	33.9677	(0.30)	**33.2142**	(0.25)	33.5907	(0.23)	32.51 (4.88)
11	189.7513	(4.62)	38.4754	(0.50)	33.4140	(0.70)	**33.1749**	(0.66)	13.97 (0.71)	31.3661	(0.29)	30.3683	(0.20)	**30.2701**	(0.19)	29.47 (4.70)
13	175.7382	(4.23)	34.5154	(0.32)	32.1094	(0.43)	**31.8041**	(0.41)	12.79 (1.17)	29.7644	(0.28)	28.7554	(0.21)	**28.1224**	(0.20)	29.94 (6.28)
15	163.1926	(4.31)	32.7069	(0.25)	31.2654	(0.35)	**30.8941**	(0.32)	11.03 (1.27)	28.6966	(0.26)	27.6899	(0.20)	**26.3893**	(0.25)	28.54 (6.29)
17	149.9852	(5.14)	31.8467	(0.23)	30.7699	(0.33)	**30.3664**	(0.30)	8.93 (0.94)	27.9309	(0.27)	26.9097	(0.19)	**25.3794**	(0.23)	28.47 (6.19)
19	139.9730	(4.32)	31.4048	(0.21)	30.5350	(0.29)	**30.0837**	(0.27)	7.91 (1.08)	27.3691	(0.26)	26.3784	(0.20)	**24.7705**	(0.18)	29.67 (7.37)
21	128.1830	(4.90)	31.1697	(0.23)	30.3017	(0.30)	**30.0384**	(0.27)	7.60 (0.71)	26.9015	(0.26)	25.9415	(0.20)	**24.3128**	(0.18)	30.05 (6.74)
23	119.5551	(4.46)	31.0421	(0.22)	**30.0627**	(0.24)	30.1166	(0.31)	6.96 (0.81)	26.5346	(0.27)	25.6021	(0.21)	**23.9719**	(0.21)	28.55 (7.05)
25	110.6725	(4.39)	30.9772	(0.23)	**29.9450**	(0.21)	30.1393	(0.24)	6.68 (0.89)	26.2126	(0.26)	25.2289	(0.21)	**23.7773**	(0.25)	25.59 (6.02)

which sub-cluster to choose can be based on the same criterion as in the agglomeration process the choice which clusters to merge, i.e. a root node is assigned to the sub-cluster where the maximum distance to a node of it is minimal.

7. COMPUTATIONAL RESULTS

The experiments have been performed on an AMD Opteron 2214 dual-core machine (2.2GHz) utilizing benchmark instances already used in the corresponding literature (e.g. [15, 11]) from Beasley's OR-Library [3, 2] originally proposed for the Euclidean Steiner tree problem. These complete instances contain point coordinates in the unit square, and the Euclidean distances between each pair of points are taken as edge costs. As performance differences are more significant on larger instances, we restrict our attention here to the 15 largest instances with 1000 nodes.

Table 1 summarizes the results obtained for various heuristics. Given are the objective values averaged over all 15 instances, where for each instance 30 independent runs have been performed, together with the standard deviations in parentheses. Considered are the two previous construction heuristics CBTC and RTC as well as the clustering heuristic C where each cluster a good root node is assigned using one of the two dynamic programming approaches d^A (restricted search space) and d^B (approximating optimal cluster roots using a correction value κ). Since d^A and d^B derive no optimal trees for a given clustering local improvement is used to further enhance their solutions.

Binary search to identify a good value for x was performed within $\frac{|V|}{2}$ and $|V|$, except when $D < 6$. In this latter case the interval bounds have been set to $\frac{|V|}{20}$ and $\frac{|V|}{8}$. In GRASP a mean μ of 0 and, after preliminary tests, a variance σ^2 of 0.25 was used, and the procedure was aborted after $l_{\max} =$ 100 iterations without improvement. The time (in seconds) listed is the over all instances averaged maximum running time of Cd^A and Cd^B, which was also used as time limit for the corresponding executions of CBTC and RTC. To verify statistical significance paired Wilcoxon signed rank tests have been performed.

Clearly, CBTC is not suitable for this type of instances, its strength lies in problems with random edge costs. The clustering heuristic outperforms RTC for every diameter bound, where the gap in solution quality is huge when D is small and becomes less with increasing diameter bound. In general, Cd^B outperforms all other heuristics significantly with an error probability of less than $2.2 \cdot 10^{-16}$. Only when the diameter bound gets noticeably loose the first dynamic programming approach Cd^A dominates Cd^B (error probability always less than $2.13 \cdot 10^{-9}$). It can also be seen that in the even-diameter case the runtime of the clustering heuristic only increases moderately with the number of levels in the height-restricted clustering. When D is odd the search for a good center edge dominates the runtime. Thus, with increasing D the clustering heuristics even get faster since the number of potential center edges to be considered, determined in the presented preprocessing step, decreases (less direct successors of the root cluster in the HRC).

We also performed test where a strong variable neighborhood descend (VND) as proposed in [11] has been applied to the best solutions of the various construction heuristics. As expected, it flattens the differences but still the BDMSTs derived from clustering heuristic solutions are in general of higher quality. On instances with diameter bounds less than approximately 10, these trees – computed in a few seconds – can also compete with results from the leading metaheuristic, the ACO from [11], which requires computation times of one hour and more.

Abbildung A.10: Instanz 10