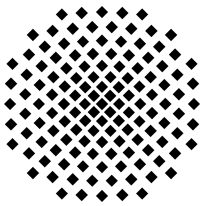
Zwischenbericht der
Projektgruppe
Fahrgemeinschaften
Bericht Nr. 1997/10



Universität Stuttgart

Zwischenbericht der Projektgruppe Fahrgemeinschaften

Herbert Heid
Daniela Nicklas
Alexander Porrmann
Thomas Schäffer
Volker Scholz

Betreuung
Prof. Dr. Volker Claus
Dipl.-Inf. Wolfgang Reissenberger
Dipl.-Inform. Friedhelm Buchholz
Dipl.-Math. Nicole Weicker

Abteilung Formale Konzepte
Fakultät Informatik
Universität Stuttgart

7. August 1997

Prof. Dr. Volker Claus
Abteilung Formale Konzepte
Institut für Informatik
Universität Stuttgart

Breitwiesenstr. 20-22
D-70565 Stuttgart

Telefon:

0711-7816-300 (Prof. Dr. V. Claus)
0711-7816-301 (Sekretariat)
0711-7816-330 (FAX)

E-Mail: claus@informatik.uni-stuttgart.de

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Die Projektgruppe im Informatikstudium

Das Studium der Informatik vermittelt dem Studierenden zwar einen großen Teil des nötigen Fachwissens, jedoch stellt das Berufsleben noch weitere Anforderungen an Informatikerinnen und Informatiker. Teamfähigkeit und Erfahrung spielen gerade bei der Mitarbeit an großen Software–Projekten eine wichtige Rolle. Hier verfolgt die Idee der Projektgruppe folgende Ausbildungsziele:

- Arbeiten im Team
- Analyse von Problemen, Strukturierung von Lösungen und gemeinsamer Entwurf geeigneter Systeme
- Selbständige Erarbeitung von Lösungsvorschlägen und deren Vorstellung und Verteidigung in einer Gruppe
- Übernahme von Verantwortung für die Lösung von Teilaufgaben und die Erstellung von Modulen
- Mitwirkung an einer umfassenden Dokumentation
- Erstellen eines Software–Produktes, das ein Einzelner innerhalb des vorgegebenen Zeitraumes unmöglich bewältigen kann
- Projekt–Planung und Kosten/Nutzen–Analyse
- Einsatz von Werkzeugen
- Persönlichkeitsbildung (Übernahme von Verantwortung, Selbstvertrauen, Verläßlichkeit, Rücksichtnahme, Durchsetzungsfähigkeit usw.)

An der Projektgruppe nehmen in der Regel acht bis zwölf Studierende des Hauptstudiums teil. Sie erarbeiten im Laufe eines Jahres ein Software–Produkt, welches einem Zeitaufwand von mehreren Personenjahren entspricht. Hierbei sollen sämtliche Phasen eines Software–Lifecycles — von der Planung bis

zur Wartung — durchlaufen werden, was in anderen Lehrveranstaltungen nicht üblich ist. Bei Software– und Fachpraktika wird zumeist eine gegebene, genau festgelegte Aufgabenstellung in ein Programm umgesetzt.

Eine Projektgruppe vereinigt die Lehrveranstaltungsformen „Hauptseminar“ (2 SWS), „Fachpraktikum“ (4 SWS) und „Studienarbeit“ (10 SWS) in sich. Demzufolge ist eine Projektgruppe mit 16 SWS einzustufen.

Der Ablauf einer Projektgruppe folgt meist folgendem Schema: Seminar–, Planungs–, Entwurfs–, Implementierungs–, Integrations–, Experimentier– und Schlußphase. Diese Phasen werden im folgenden genauer erläutert.

Seminarphase: Die Themenstellung wird gründlich analysiert. Dazu werden von den Mitgliedern Originalpublikationen durchgearbeitet und die Ergebnisse vorgetragen. Ergebnisse dieser Phase sind viel Wissen, je eine Vortragsausarbeitung und eine zusammenfassende Darstellung der Literaturauswertung.

Planungsphase: Die Projektgruppe analysiert den Problembereich, stellt Einsatzmöglichkeiten und Anwendungen zusammen, erarbeitet einen Anforderungskatalog und diskutiert Lösungsmöglichkeiten für diese Fragestellungen. Hierbei werden die in der Literatur bekannten Lösungsvorschläge und eigene Ideen gegeneinander abgewogen. Insbesondere wird frühzeitig diskutiert, welche Hard– und Software für die jeweiligen Lösungen erforderlich ist, welche sonstigen Kosten entstehen, wie hoch der Zeitaufwand sein wird, usw. Wichtig ist eine frühe Spezifizierung der Eigenschaften des Systems (Robustheit, Antwortverhalten, Flexibilität, Schutzmechanismen, Erweiterbarkeit, Verteiltheit, ...).
Inhaltliches Ergebnis ist eine möglichst eindeutige, ausschnittsweise sogar formale Spezifikation. Für jede ins Auge gefaßte Anwendung wird darüber hinaus ein Szenario bzgl. des Einsatzes, der Nutzung, der Tests und der Wartung skizziert.
Organisatorische Ergebnisse sind ein grober Zeitplan und die erste Aufteilung von Aufgabengebieten. Hier setzt auch eine Spezialisierung der Gruppenmitglieder ein.

Entwurfsphase: Voraussetzung für die Entwurfsphase ist, daß Begriffsbestimmungen, Anwendungen und Modelle weitgehend geklärt sind. Nach Festlegung des grundsätzlichen Lösungsverfahrens werden Teilprobleme und charakteristische Objekte herauskristallisiert, miteinander in Beziehung gesetzt, auf ihre Realisierbarkeit geprüft und grundlegende Datenstrukturen und Kommunikationswege festgelegt. Dabei werden die Schnittstellen der Einzelteile des Systems untereinander genau definiert. Ergebnis ist ein Plan des zu erstellenden (oder zu modifizierenden) Systems. Stehen die einzelnen Aufgaben fest, werden sie auf die Mitglieder verteilt. Die Implementierungssprache(n) sowie die erforderliche Hardware und die zu verwendenden Werkzeuge werden festgelegt. Eine Liste von Beispielen, die das System später positiv bewältigen muß, wird für die Testphase erstellt.

In der *Implementationsphase* und *Integrationsphase* wird der Programmcode erstellt, zusammengebunden (integriert) und getestet.

Die *Experimentierphase* schließt weitere Tests mit speziellen Anwendungen ein.

Zur *Schlußphase* zählt in erster Linie der Abschluß der *Dokumentation*, die ständig parallel zur Projektgruppenarbeit erstellt und auf den neuesten Stand

gebracht wird.

Das Konzept der Projektgruppe wird bereits seit Jahren an anderen Universitäten wie z.B. in Oldenburg und Dortmund erprobt und durchgeführt. Dort sind Projektgruppen z.T. schon Pflichtveranstaltungen im Rahmen des Informatikstudiums. An der Universität Stuttgart gibt es seit 1994 Projektgruppen im Fach Informatik; im neu eingerichteten Studiengang Software-Technik ist die Teilnahme verbindlich vorgeschrieben.

1.2 Aufgabenstellung beim Projekt Fahrgemeinschaften

Im Rahmen der Projektgruppe soll das Programm Mobidick (Mobil durch intelligentes computerunterstütztes Kombinieren) entstehen, das ausgehend von Personen- und Verkehrsdaten Aufteilungen in Fahrgemeinschaften berechnet. Es handelt sich hierbei um einen Prototyp für ein System, das beispielsweise in einer Mitfahr- oder Mobilitätszentrale eingesetzt werden kann, um für große Personenmengen Fahrgemeinschaften zu bestimmen und zu verwalten.

Die Personendaten enthalten Informationen über Start-, Zielorte, Arbeitszeiten und Eigenschaften der Personen (z.B. Geschlecht, Raucher/Nichtraucher usw.). Außerdem können die Personen angeben, wie ihre Wunschfahrgemeinschaft aussehen sollte, d.h. welche Kriterien ihnen besonders wichtig sind (Umweg, Arbeitszeit, Eigenschaften der Mitfahrer und persönliche Zu- bzw. Abneigung gegenüber bestimmten Personen).

Die Verkehrsdaten (Stadtplan) liefern die Grundlage für die Berechnung der besten Routen mit den kürzesten Umwegen. Davon ausgehend soll nun eine optimale oder heuristische Lösung gefunden werden. Die Güte von Fahrgemeinschaften und Einteilungen des Personenstamms kann anhand einer Bewertungsfunktion beurteilt werden. In diese Bewertungsfunktion gehen die o.g. Kriterien Umweg, Arbeitszeiten, Personeneigenschaften und Zu-/Abneigungen ein.

Die Personen-, Verkehrs- und Fahrgemeinschaftsdaten müssen verwaltet werden und leicht änderbar sein.

Besonderer Wert wird im Projekt auf die Austauschbarkeit der Algorithmen gelegt. Der Prototyp ermöglicht die Untersuchung verschiedener Algorithmen zur Wegsuche und zur Einteilung in Fahrgemeinschaften.

Kapitel 2

Vorgehensweise

Dieses Kapitel gibt einen Überblick der Aktivitäten der Projektgruppe „Fahrgemeinschaften“ von Anfang an bis zur Erstellung des Zwischenberichts. Dabei wird der Zeitplan sowie die einzelnen Phasen kurz vorgestellt und erläutert.

2.1 Arbeitsweise

Die Projektgruppe besteht aus den fünf studentischen Teilnehmern, dem Projektleiter Wolfgang Reissenberger und dem Kunden Friedhelm Buchholz, der insbesondere die Auswahl und Entwicklung der Algorithmen übernimmt. Für die Seminarvorträge und -ausarbeitungen ist Nicole Weicker zuständig.

Die zentrale Veranstaltung der Gruppe ist die Projektgruppensitzung. Bei jeder Sitzung gibt es einen Protokollführer und einen Sitzungsleiter, der in der Regel der Protokollant der letzten Sitzung war. Diese Ämter rotieren unter den Teilnehmern. Die Kommunikation erfolgt in erster Linie über die Sitzungen, aber auch elektronisch und über den Projektgruppenordner, in dem die Sitzungsprotokolle und andere relevante Dokumente abgelegt werden. In der Seminarphase traf sich die Projektgruppe etwa drei Mal die Woche. Davon waren an ein bis zwei Terminen Vorträge, am dritten wurde meist Organisatorisches geregelt. Später gab es in der Regel zwei Termine. Die Projektgruppensitzung dient hauptsächlich der Koordination. Zu verschiedenen Arbeiten werden Untergruppen gebildet, deren Ergebnisse dann anhand eines schriftlichen Berichts in der Sitzung diskutiert werden.

Für die Meilensteine (Anforderungsanalyse, Spezifikation) wurden technische Reviews durchgeführt. Dabei übernahm der Projektleiter die Rolle des Sitzungsleiters und die Teilnehmer die der Gutachter. Nach den internen Reviews wurden diese Dokumente noch mit dem Kunden besprochen und sind in ihrer jetzigen Form von diesem angenommen.

Die Arbeitsteilung in der Projektgruppe erfolgt dynamisch. Für jeden Meilenstein ist jemand anderes verantwortlich, der dann die Aufteilung in Arbeitspakete vornimmt und für die Integration und Gesamtgestaltung zuständig ist.

Für einzelne Gebiete wurden „Experten“ bestimmt, die sich dort besonders einarbeiten, wie z.B. für LaTeX, für Versionskontrolle oder für Dokumentation. Zu Themen, die der ganzen Projektgruppe neu sind, werden zum Teil externe Vorträge (z.B. von Mitarbeitern aus dem Haus) angeworben.

2.2 Zeitplan

Zu Beginn der Projektgruppe wurde folgender Zeitplan vereinbart, der sich an den Erfahrungen der vorangegangenen Projektgruppen orientiert.

14.10.96 – 29.11.96	**Seminarphase**	(7 Wo.)
3.12.96 – 3.1.97	**Anforderungsanalyse**	(4 Wo.)
6.1.97 – 14.2.97	**Spezifikation**	(6 Wo.)
17.2.97 – 18.4.97	**Entwurf**	(9 Wo.)
21.4.97 – 2.5.97	**Zwischenbericht**	(2 Wo.)
5.5.97 – 4.7.97	**Implementierung**	(9 Wo.)
7.7.97 – 29.8.97	**Test/Review**	(8 Wo.)
1.9.97 – 26.9.97	**Enddokumentation/Präsentation**	(4 Wo.)

Dabei ist zu beachten, daß sich die einzelnen Phasen nicht präzise voneinander trennen lassen und gewisse Überlappungen auftreten dürfen. Zu allen Phasen entsteht Dokumentation, die Bestandteil des jeweiligen Meilensteins ist. Die Dokumente der bisher erreichten Meilensteine sind Bestandteil des Zwischenberichts.

2.2.1 Seminarphase und Vorträge

Die Projektgruppe begann mit der Seminarphase. In dieser wurden von den Teilnehmern Vorträge zu folgenden Themen gehalten:

- „Nachbarschaftssuche im $\mathbb{R}^d$ und Geometrische Datenstrukturen“ von Volker Scholz
- „Matching-Verfahren“ von Herbert Heid
- „Kürzeste Wege in planaren Graphen“ von Alexander Porrmann
- „Softwareengineering und objektorientierte Entwicklung“ von Daniela Nicklas
- „Contraint Programmierung“ von Thomas Schäffer

Die Ausarbeitungen dieser Vorträge finden sich in Kapitel 3. Desweiteren fanden zu verschiedenen anderen Themen Vorträge statt, um das Wissen der Projektgruppe z.B. für die Wahl der Programmiersprache zu erweitern.

Diese Vorträge waren im einzelnen:

- „Fahrgemeinschaften“ von Prof. Claus

- „Die Programmiersprache ML“ von Wolfgang Reissenberger
- „Komplexität von Partionierungen“ von Friedhelm Buchholz
- „Kürzeste Wegsuche“ von Friedhelm Buchholz
- „Clustering“ von Friedhelm Buchholz
- „Anforderungen an das Fahrgemeinschaftensystem“ von Friedhelm Buchholz
- „C++ und die Methode von Booch“ von Bernd Kawetzki
- „Die Programmiersprache Ada“ von Andreas Bergen
- „Die Programmierspache Java“ von Fritz Hohl

2.2.2 Anforderungsanalyse

Die Anforderungsanalyse dient dazu, das Umfeld der Anwendung zu analysieren, bestehende Anforderungen zu erfassen und noch fehlende zu evaluieren. In der Projektgruppe begann diese Phase damit, daß Friedhelm Buchholz in seiner Rolle als Kunde einen Vortrag über die Anforderungen hielt. In einer zweiten Runde befragte ihn die Projektgruppe dazu. Dann entwickelte sie Szenarien oder Use Cases, aus denen sich dann weitere Anforderungen ergaben. Dabei wurden wichtige Entscheidungen getroffen: Das System wird ein Prototyp werden, der ohne graphische Benutzungsoberfläche auskommt, dafür aber wiederverwendbare Module enthält und um weitere Algorithmen erweitert werden kann. Das Ergebnis dieser Phase ist das Dokument „Anforderungsanalyse“, das sich in Kapitel 4 findet.

2.2.3 Spezifikation

In der Spezifikationsphase wird aus dem Anforderungskatalog das äußere Systemverhalten definiert. Es wird beschrieben, was das System tut, nicht wie es das tut. Da die Spezifikation die Grundlage für den Entwurf ist - hier sollen keine wichtigen Entscheidungen über das Verhalten mehr getroffen werden - dauerte diese Phase auch sechs Wochen. Für die Spezifikation wurden die Use Cases aus der Anforderungsanalyse verwendet und ausgebaut. Eines der Hauptprobleme war die Datenhaltung und die Konsistenzsicherung zwischen den einzelnen Datenstämmen (Personen, Fahrgemeinschaften, Verkehrsdaten). Das Dokument „Spezifikation“ steht im Kapitel 5 und ist das Ergebnis des gleichnamigen Meilensteins.

2.2.4 Entwurf

Im Entwurf wird entwickelt, wie das System sich intern verhält. Er geht von einem Grobentwurf, in dem die Subsysteme und ihre Schnittstellen identifiziert und definiert werden in einen Feinentwurf über, bei dem am Ende die genauen Datenstrukturen und Algorithmen beschrieben werden. Zum Zeitpunkt des

Zwischenberichts befindet sich die Projektgruppe noch in der Entwurfsphase. Kapitel 6 enthält deswegen den Grobentwurf.

2.2.5 Zwischenbericht

Der Zwischenbericht erscheint etwa nach der Hälfte der Zeit der Projektgruppe. Er enthält die zentralen Dokumente, die bis zu diesem Zeitpunkt erschienen sind. Zusammen mit dem Endbericht stellt er die Studienarbeit der Teilnehmer dar.

2.2.6 Weitere Phasen

Nach der Entwurfsphase folgt die Implementation, in der nun die vorher spezifizierten Klassen mit Leben gefüllt werden. Die verwendete Sprache ist C++, wie im folgenden Abschnitt erläutert.

2.3 Sprachentscheidung

Für die Implementierung des Fahrgemeinschaftensystems wurden vier Programmiersprachen diskutiert: SML, ADA, JAVA und C++. SML ist eine funktionale Sprache mit strengem Typsystem, Java und C++ sind Vertreter der objektorientierten Sprachen und ADA wird als eine Sprache bezeichnet, die verschiedene Paradigmen unterstützt.

Die Grundlagen für die Sprachentscheidung waren spezielle Fachvorträge, die Kundenanforderungen und die persönlichen Erfahrungen der Projektgruppenteilnehmer. Aufgrund des gesammelten Wissen wurde wie folgt eine Programmiersprache ausgewählt:

Die Sprache SML scheiterte an der Kundenanforderung, eine prozedurale Programmiersprache zu verwenden. Es sind Probleme bei der Anbindung von Algorithmen, die prozedural geschrieben sind, zu erwarten, da SML eine funktionale Sprache ist. Die geforderte zukünftige Erweiterung, eine direkte Anbindung einer graphischen Benutzungsoberfläche, gab den Anlaß dafür, ADA zu streichen. Ein weiterer Grund war die Darstellung der Objekte, die nur durch Records repräsentiert werden können. Da Java eine recht neue Programmiersprache ist und nicht gewiß war, wann die neue Entwicklungsumgebungsversion kommt, wählte man für die Implementierung nicht Java, sondern die Programmiersprache C++. C++ besitzt all die Eigenschaften, die an die Programmiersprache gestellt wurden: prozedural, objektorientierte Programmierung, Anbindung einer graphischen Benutzungsoberfläche und Effizienz in bezug auf schnelle Algorithmen und große Datenmengen. Für die Implementierung werden die Standard Template Library und die Leda Library verwendet, die einige Standardfunktionen bieten. Zur Beschreibung und zur Darstellung der Klassen wird ein OO-Browser verwendet, der in einer Emacs-Umgebung für C++ arbeitet.

Kapitel 3

Seminarvorträge

In diesem Kapitel befinden sich die Ausarbeitungen der Vorträge, die im Rahmen der Seminarphase von den Teilnehmern gehalten wurden.

3.1 Nachbarschaftssuche im $\mathbb{R}^d$ und Geometrische Datenstrukturen

3.1.1 Einführung

Bei der Bildung von Fahrgemeinschaften soll der für den Fahrer entstehende Umweg eine gewisse Zumutbarkeitsgrenze nicht überschreiten. Da das Problem der Zuordnung von Personen zu Fahrgemeinschaften NP-hart ist (siehe [3]), müssen heuristische Verfahren angewendet werden. Zieht man z.B. nur Personen mit ähnlichen Start-, Zielorten und Fahrtzeiten für eine Fahrgemeinschaft in Betracht und notiert die Daten jeder Person als 6-Tupel $(x_{start}, y_{start}, x_{ziel}, y_{ziel}, t_{abfahrt}, t_{ankunft})$, so entspricht dies einer Nachbarschaftssuche auf Punkten im $\mathbb{R}^6$. Im folgenden sollen hierfür geometrische Algorithmen und Datenstrukturen vorgestellt werden, die man bei einer Implementierung verwenden könnte.

Im ersten Teil wird ein Algorithmus zur Bestimmung aller Punktepaare mit Maximalabstand δ vorgestellt. Daran schließt sich eine Laufzeitabschätzung an.

Im zweiten Teil werden drei geometrische Datenstrukturen zur Verwaltung einer Punktmenge betrachtet. Für die einzelnen Operationen auf den Datenstrukturen werden Laufzeitabschätzungen angegeben. Für Nachbarschaftsprobleme ist dabei insbesondere die Bereichssuche wichtig. Im abschließenden Vergleich der Datenstrukturen wird der Unterschied zwischen statischen und dynamischen Datenstrukturen hervorgehoben.

Um mit diesen Hilfsmitteln benachbarte Punkte zu bestimmen, könnte man in zwei Schritten vorgehen: Zunächst führt man eine Bereichssuche auf der gewählten Datenstruktur in der Umgebung eines festen Punktes durch und bestimmt dann im fraglichen Bereich alle benachbarten Punkte mit Hilfe des Algorithmus für δ-Paare.

3.1.2 Algorithmus für δ-Paare

3.1.2.1 Definitionen

Eine untere Schranke für die Laufzeit eines Algorithmus für benachbarte Punkte ist sicherlich der Wiedergabeaufwand, d.h. wieviele Punktepaare tatsächlich gefunden und ausgegeben werden müssen. Diese Anzahl hängt von der Dichte (bzw. Spärlichkeit) der betrachteten Punktmenge ab, deshalb definieren wir zunächst ein Maß für die maximale Dichte einer Punktmenge (vgl. [17]):

Definition 1 (Spärlichkeit) *Eine Punktmenge $S \subseteq \mathbb{R}^d$ hat die Spärlichkeit $c \in \mathbb{N}$ für $\delta > 0$, falls alle d-dimensionalen Würfel der Kantenlänge 2δ höchstens c Punkte enthalten.*

Zur Veranschaulichung betrachte man Abb. 3.1. Verschiebt man ein Quadrat der Seitenlänge 2 über die Punktmenge S, so liegen maximal vier Punkte im Quadrat, also beträgt die Spärlichkeit $c = 4$ für $\delta = 1$.

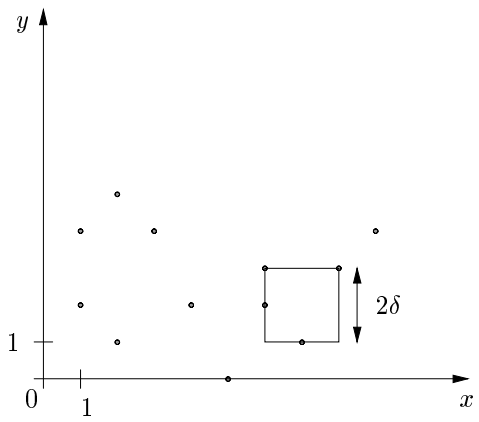


Abb. 3.1: Spärlichkeit $c = 4$ für $\delta = 1$

Bei einer hohen Spärlichkeit c für ein vorgegebenes δ muß man also bei N Punkten mit einer Laufzeit von $O(N^2)$ rechnen (alle Punktepaare werden zurückgegeben), nur bei dünnen Punktmengen ist eine Verbesserung zu erwarten. Als Abstandsmaß verwenden wir nicht den euklidischen Abstand, sondern betrachten jede Koordinate einzeln, wie in der folgenden Definition.

Definition 2 (δ-Paar) *Zwei Punkte $\mathbf{x} = (x_1, \ldots, x_d)^T$, $\mathbf{y} = (y_1, \ldots, y_d)^T \in \mathbb{R}^d$ sind ein δ-Paar*

$$:\Longleftrightarrow |x_i \Leftrightarrow y_i| \leq \delta \quad \forall i \in \{1, 2, \ldots, d\}$$

Mit diesen beiden Begriffen läßt sich nun unser Problem formulieren:

> Gegeben sei eine Punktmenge $S \subseteq \mathbb{R}^d$ mit Spärlichkeit c für δ und ein Maximalabstand δ. Gesucht sind alle δ-Paare in S.

3.1.2.2 Algorithmus NN (nearest neighbour)

Um dieses Problem mit einem divide-and-conquer-Ansatz zu lösen, teilt man die Punktmenge S durch eine Hyperebene l senkrecht zu einer Koordinatenachse x_k in zwei Mengen S_1 und S_2 (siehe Abb. 3.2). Die zur Koordinatenachse k senkrecht stehende Ebene l sei dabei durch die Gleichung $x_k = l_k$ gegeben. Löst man nun das Problem für diese Mengen rekursiv, so erhält man aber nur δ-Paare, die ganz in S_1 oder S_2 liegen. Für Paare, bei denen ein Punkt in S_1, der andere in S_2 liegt, genügt es, den Bereich in der δ-Scheibe S_δ um l zu untersuchen, wobei

$$S_\delta := \left\{ \mathbf{x} \in \mathbb{R}^d \,\middle|\, |x_k \Leftrightarrow l_k| \leq \delta \right\}.$$

Wenn ein Punkt gerade auf l liegt, kann der Partner höchstens δ von l entfernt sein.

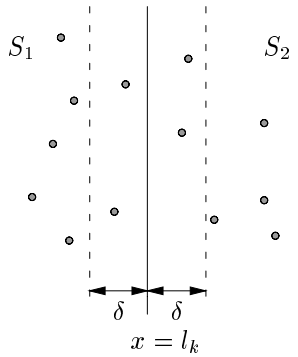


Abb. 3.2: im $\mathbb{R}^2$ ist l eine Gerade

Im ungünstigen Fall liegen allerdings alle Punkte der Ausgangsmenge S auch in S_δ (dichte Punktmenge), das ursprüngliche Problem wurde also nicht verkleinert. Deswegen projiziert man die Punkte in S_δ auf l und gewinnt somit eine Verkleinerung in der Dimension. Wenn man bei einer Dimension angelangt ist, können die δ-Paare durch direkten Vergleich der Nachbarn auf der Geraden bestimmt werden. Die Rekursion kann ebenfalls abgebrochen werden, wenn die Punktezahl unter eine bestimmte Schranke fällt, dann lohnt der Teilungsaufwand nicht mehr und man vergleicht alle Punktepaare direkt.

Algorithmus:

1. Eingabe: Punktmenge $S \subseteq \mathbb{R}^d$, Maximalabstand δ.
2. Ausgabe: alle δ-Paare in S.
3. Vorsortierung: Sortiere S bezüglich aller k Koordinaten in die Felder $F_1, \ldots, F_k$.

```
procedure deltapaare(S:Punktmenge, d:Dim.zahl)
begin
  if |S| ≤ K then
    vergleiche alle Punktepaare direkt
  else if d = 1 then
    for i := 1 to |S| ⇔1 do
      j:=i+1;
      while F'[j]_l ⇔F'[i]_l ≤ δ and j ≤ |S| do
        vergleiche Projektionslisten von F'[j] und F'[i];
        if δ-Paar gefunden then Paar ausgeben;
        j:=j+1;
      endwhile;
    endfor;
  else
    wähle Hyperebene l;
    spalte S in S_1, S_2 und S_δ auf;
    deltapaare(S_1, d);
    deltapaare(S_2, d);
    projiziere Punkte in S_δ auf l;
    deltapaare(S_δ, d ⇔1);
end
```

Nach der Durchführung der Vorsortierung wird *deltapaare* mit der zu untersuchenden Punktmenge S und der Anzahl der Raumdimensionen d aufgerufen. Die ersten beiden Fälle der **if**-Anweisung behandeln den Rekursionsabbruch, wir betrachten zunächst den **else**-Teil. Man wählt eine Hyperebene l und teilt S in S_1, S_2 und S_δ. Dabei werden die Felder $F_1, \ldots, F_k$ in die neuen Felder F_i^1, F_i^2 und F_i^δ (für $i = 1 \ldots k$) aufgespalten, wobei die Sortierung erhalten bleibt. Dann erfolgt Rekursion auf S_1 und S_2 in d Dimensionen. Bei der Projektion der Punkte in S_δ auf l wird die verlorengegangene Koordinate jedes Punktes in seiner Projektionsliste gespeichert. Danach erfolgt wiederum Rekursion auf der projizierten Menge in $d \Leftrightarrow 1$ Dimensionen.

Im Fall $|S| \leq K$ vergleicht man alle Punktepaare direkt. Bei nur einer Dimension ($d = 1$) liege die Punktmenge nach der Koordinate x_l sortiert im Feld F' vor. Man betrachtet dann für jeden Punkt nur Nachbarn in der δ-Umgebung (**while**-Schleife, $F'[j]_l \Leftrightarrow F'[i]_l$ ist der Abstand in der Koordinate x_l) und stellt anhand der Projektionslisten fest, ob ein δ-Paar vorliegt.

Als Beispiel betrachte man Abb. 3.3. Im ersten Bild sind die in der Ausgangsmenge S vorhandenen δ-Paare durch gestrichelte Linien verbunden. Im ersten Teilungsschritt liegen zwei Punkte in S_δ und werden auf l projiziert. Während in S_δ und S_2 die δ-Paare direkt bestimmt werden können (Rekursionsabbruch wegen $d = 1$ bzw. $|S| \leq 3$), wird S_1 nochmals in S_1', S_δ' und S_2' geteilt. Man findet schließlich alle drei δ-Paare, die in S_δ' und S_2' gefundenen Paare sind identisch.

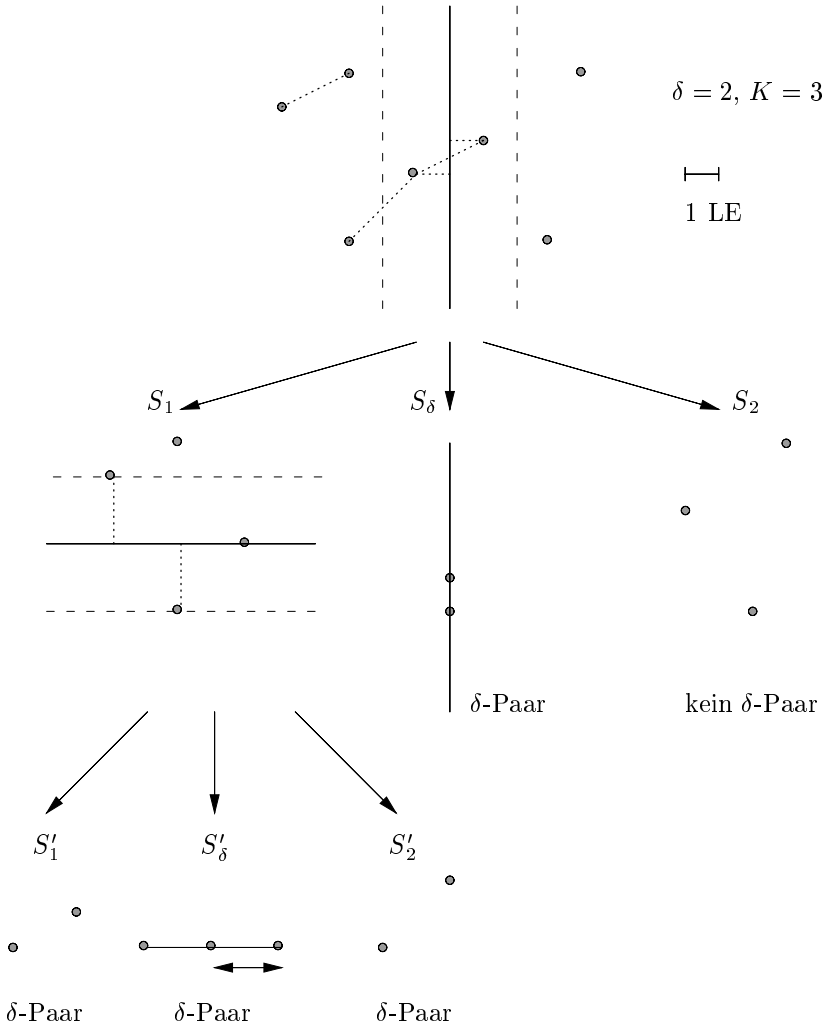


Abb. 3.3: Beispiel

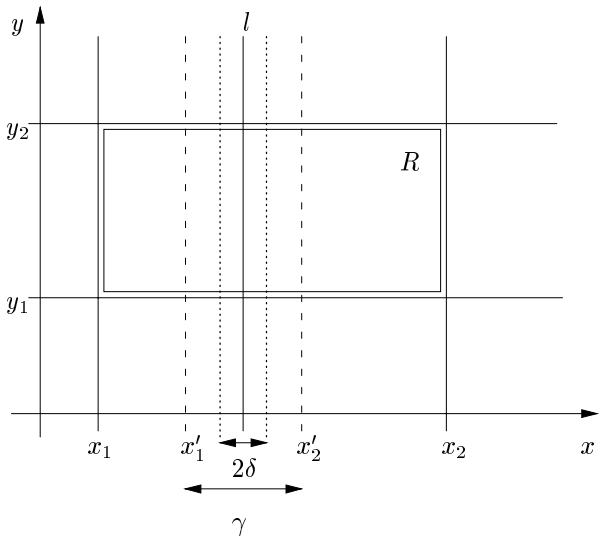
3.1.2.3 Wahl der Hyperebene

Der folgende Satz zeigt, wie man bei der Bestimmung der Hyperebene vorgehen kann, um eine „gerechte“ Teilung von S zu erreichen. Wie auch bei anderen divide-and-conquer-Verfahren (Bsp. quicksort) wäre die Laufzeit bei einem großen Ungleichgewicht zwischen $|S_1|$ und $|S_2|$ wiederum in $O(N^2)$.

Satz 1 (Existenz einer günstigen Hyperebene) *Für eine Punktmenge $S \subseteq \mathbb{R}^d$, $|S| = N$ mit Spärlichkeit c für δ gibt es eine Hyperebene l und eine dazu senkrechte Koordinatenachse mit folgenden Eigenschaften:*

1. *Die Mengen S_1 und S_2 zu beiden Seiten von l enthalten jeweils mindestens $\frac{N}{4d}$ Punkte.*
2. *In der δ-Scheibe S_δ um l liegen höchstens $d \cdot c \cdot N^{1-1/d}$ Punkte.*

Beweis: für $d = 2$



O.B.d.A. sei $N = 8N'$. Bestimme ein Intervall $[x_1, x_2]$, so daß rechts und links des vertikalen Streifens $T := \{(x, y) \in R^2 | x_1 \leq x \leq x_2\}$ jeweils $\frac{N}{8}$ Punkte liegen. Bestimme $[y_1, y_2]$ analog. Die beiden Intervalle spannen das Rechteck $R := [x_1, x_2] \times [y_1, y_2]$ auf. Falls beide Intervalle die Länge null haben, gibt es kein l mit den gewünschten Eigenschaften. Alle Punkte sind dann identisch und es gilt $N \leq c$ aufgrund der Spärlichkeitseigenschaft. Falls nur ein Intervall verschwindet, wählen wir l so, daß das andere Intervall halbiert wird. In S_δ liegen dann maximal c Punkte, weil in einem Quadrat der Seitenlänge 2δ höchstens c Punkte enthalten sind (Definition der Spärlichkeit). Im folgenden nehmen wir also an, daß $x_2 \Leftrightarrow x_1 > 0$ und $y_2 \Leftrightarrow y_1 > 0$. Bestimme nun auf der x-Achse das größte Intervall $[x'_1, x'_2]$ in $[x_1, x_2]$, das die Projektionen von höchstens $2c \cdot N^{1/2}$

Punkten enthält. Ebenso $[y_1', y_2']$. Sei γ das Maximum der beiden Intervallängen, o.B.d.A. $\gamma = x_2' \Leftrightarrow x_1'$.

Die Schnittlinie l mit der Gleichung $x = \frac{1}{2} \cdot (x_1' + x_2')$ hat die gewünschten Eigenschaften. Dazu zeigen wir $\gamma \geq 2\delta$. Dann gilt 1., weil rechts und links von $[x_1, x_2]$ $\frac{N}{8}$ Punkte liegen. Außerdem gilt 2., weil in $[x_1', x_2']$ maximal $2c \cdot N^{1/2}$ Punkte liegen.

Annahme: $\gamma < 2\delta$
Dann enthält jeder vertikale Streifen der Breite 2δ im Intervall $[x_1, x_2]$ mehr als $2c \cdot N^{1/2}$ Punkte (wegen der Maximalität von γ).

$$\lfloor \frac{x_2 \Leftrightarrow x_1}{2\delta} \rfloor \cdot 2cN^{1/2} \leq \frac{3}{4}N \Longleftrightarrow \lfloor \frac{x_2 \Leftrightarrow x_1}{2\delta} \rfloor \leq \frac{3N^{1/2}}{8c}$$

$$\text{analog}: \qquad \lfloor \frac{y_2 \Leftrightarrow y_1}{2\delta} \rfloor \leq \frac{3N^{1/2}}{8c}$$

R enthält höchstens $\lceil \frac{x_2 - x_1}{2\delta} \rceil \cdot \lceil \frac{y_2 - y_1}{2\delta} \rceil$ Quadrate der Seitenlänge 2δ, die nach Voraussetzung höchstens c Punkte enthalten (Definition der Spärlichkeit). Für die Punktezahl in R ergibt sich:

$$\begin{aligned} \lceil \frac{x_2 \Leftrightarrow x_1}{2\delta} \rceil \cdot \lceil \frac{y_2 \Leftrightarrow y_1}{2\delta} \rceil \cdot c &\leq (\lfloor \frac{x_2 \Leftrightarrow x_1}{2\delta} \rfloor + 1) \cdot (\lfloor \frac{y_2 \Leftrightarrow y_1}{2\delta} \rfloor + 1) \cdot c \\ &\leq (\frac{3N^{1/2}}{8c} + 1)^2 \cdot c \\ &\leq (\sqrt{2} \cdot \frac{3N^{1/2}}{8c})^2 \cdot c \end{aligned}$$

Die letzte Ungleichung gilt wegen $N \gg c$. Der letzte Term wird wegen $N \gg c$ für $c = 1$ maximal, also enthält R höchstens $2 \cdot (\frac{3}{8})^2 N \approx 0.28N$ Punkte. Außerhalb von R liegen höchstens $2 \cdot \frac{1}{4} \cdot N = \frac{1}{2}N$ Punkte ($|A| = \frac{1}{4}N$, $|B| = \frac{1}{4}N$ und $|A| \cup |B| \leq |A| + |B|$), also kann R nicht weniger als $0.5N$ Punkte enthalten. Widerspruch!

3.1.2.4 Aufwandsabschätzung für NN

$T(N, d)$ bezeichne die Laufzeit des Algorithmus für N Punkte im $\mathbb{R}^d$. Die Spärlichkeit der Punktemenge sei wiederum c. Wir beschränken uns auf den Fall $d = 2$. Für die einzelnen Phasen von NN ergeben sich dann folgende Abschätzungen:

1. **Vorsortierung**:
   Sortiere S nach jeder Dimension (Felder F_1, F_2).
   Aufwand: $O(2N \cdot \log N) = O(N \cdot \log N)$.

2. **Teilungsschritt**:

   - Bestimmung von $[x_1, x_2]$ und $[y_1, y_2]$ in $O(1)$.
   - Bestimmung von $[x_1', x_2']$ und $[y_1', y_2']$ in $O(2N)$, indem die Intervalle $[x_1, x_2]$ und $[y_1, y_2]$ einmal durchlaufen werden.

- S in S_1, S_2 und S_δ teilen, indem die sortierten Felder F_1, F_2 in F_i^1, F_i^2 und F_i^δ geteilt werden. $(1 \leq i \leq 2)$
  Wenn man z.B. bezüglich der x-Dimension teilt, wird F_1 in drei Teile gespalten, F_2 muß ebenfalls einmal durchlaufen werden, um die Punkte einzeln in die F_2^α einzufügen. Die Sortierung wird dabei beibehalten.
  Aufwand: $O(2N)$

Gesamtaufwand: $O(N)$.

3. **Rekursionsabbruch**:
   (a) $T(K,d) \in O(1)$. Für das direkte Vergleichen von K Punkten benötigt man $\frac{1}{2} \cdot K \cdot (K \Leftrightarrow 1)$ Operationen.
   (b) $T(N,1) = 2c \cdot N$:
   Man geht die auf einer Geraden liegenden, sortierten Punkte nacheinander durch. Dies sind maximal N Punkte, im Umkreis δ jedes Punktes liegen maximal c Punkte, bei denen man zwei Koordinaten prüfen muß.

4. **Rekursion**:
   - für S_1, S_2: $T(\alpha \cdot N, 2)$ und $T((1 \Leftrightarrow \alpha) \cdot N, 2)$
   - für S_δ: $T(2c \cdot \sqrt{N}, 1) = 4c^2 \cdot \sqrt{N}$

   wobei $\frac{1}{8} \leq \alpha \leq \frac{7}{8}$ (vgl. Satz 1).

Damit ergibt sich folgende Rekursionsgleichung:

$$T(N,2) = \overbrace{T(\alpha \cdot N, 2)}^{S_1} + \overbrace{T((1 \Leftrightarrow \alpha) \cdot N, 2)}^{S_2} + \underbrace{\overbrace{O(N)}^{Teilen} + \overbrace{\underbrace{T(2c \cdot \sqrt{N}, 1)}_{=4c^2 \cdot \sqrt{N}}}^{S_\delta}}_{O(N)}$$

mit der bekannten Lösung (vgl. z.B. mergesort):

$$T(N,2) \in O(N \cdot \log N)$$

Die Vorsortierung ist ebenfalls von dieser Ordnung, also ergibt sich eine *Gesamtlaufzeit* von $O(N \cdot \log N)$.

3.1.3 Datenstrukturen zur Verwaltung einer Punktmenge

3.1.3.1 Einführung

Im folgenden werden drei geometrische Datenstrukturen vorgestellt, die man zur Verwaltung der Personendaten verwenden könnte. Dabei sind folgende Operationen möglich ($p \in \mathbb{R}^d$ sei ein beliebiger Punkt, $S \subseteq \mathbb{R}^d$ die gespeicherte Punktmenge):

- Einfügen eines Punktes
- Löschen eines Punktes
- Punktsuche:
  Hier soll die Frage $p \in S$? beantwortet werden.
- Bereichssuche:
  Gesucht sind die Punkte aus S, die im Hyperquader $[l_0, h_0] \times \ldots \times [l_{d-1}, h_{d-1}]$ liegen.

Zur Lösung des Nachbarschaftsproblems interessiert uns insbesondere die Bereichssuche.

3.1.3.2 k-D-Bäume

Einführung
Analog zu binären Suchbäumen für den eindimensionalen Fall möchte man eine hierarchische Strukturierung der Daten erreichen, um eine schnelle Suche zu ermöglichen. Zum Aufbau eines k-D-Baums teilt man die vorliegende Punktemenge durch eine Hyperebene (Gerade), die mindestens einen dieser Punkte enthält. Dieser wird zu Wurzel mit den Unterbäumen $T_<$, $T_>$ und $T_=$, die die Punktemengen rechts, links und innerhalb der Ebene enthalten. Auf diesen Punktemengen setzt man rekursiv fort.
Als Beispiel betrachte man Abb. 3.4. Wählt man die erste Trennungsgerade

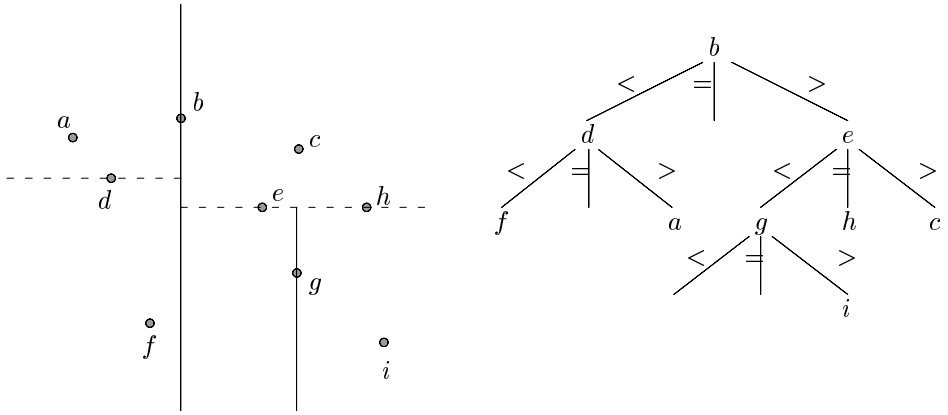


Abb. 3.4: 2-D-Baum

senkrecht durch den Punkt b, somit wird b zur Wurzel. Die Punktmenge zur linken Seite trennt man waagrecht durch d, dieser Punkt wird also der $<$-Sohn von b usw. Dabei unterscheidet man von Baumlevel zu Baumlevel abwechselnd nach x- und y-Koordinaten. Damit ergibt sich eine eineindeutige Beziehung zwischen Baumknoten und Punkten. Zu jedem Knoten v gehört außerdem eine Region $R(v) \subseteq \mathbb{R}^k$. Darunter versteht man man das Raumgebiet, das vom Teilbaum mit der Wurzel v abgedeckt wird. $R(c)$ entspricht also z.B. dem Quadranten

rechts oben. Man erhält also eine Partitionierung des Raums in Abhängigkeit von der gespeicherten Punktmenge.

Bei allgemeiner Dimension k geht man die Koordinaten von Baumlevel zu Baumlevel zyklisch durch (vgl. [15]):

Definition 3 (k-D-Baum) *Sei $S \subseteq \mathbb{R}^k$, $|S| = n$ und $\mathbf{x} = (x_0, x_1, \ldots, x_{k-1}) \in \mathbb{R}^k$. Ein bei Koordinate i beginnender k-D-Baum wird folgendermaßen definiert:*
1. für $k = n = 1$ besteht er aus einem einzigen Blatt $\mathbf{x} \in S$.
2. für $k > 1$ oder $n > 1$ besteht er aus

- *einer Wurzel $\mathbf{w} \in S$, $w_i \in \mathbb{R}$ bezeichnet die trennende Hyperebene senkrecht zur Koordinate i. (Ebenengleichung $x_i = w_i$)*
- *den Unterbäumen $T_<$, $T_=$, $T_>$*
  - *$T_<$ ist ein bei Koordinate $(i+1) \bmod k$ beginnender k-D-Baum für $S_< = \{\mathbf{x} \in S | x_i < w_i\}$.*
  - *$T_>$ ist ein bei Koordinate $(i+1) \bmod k$ beginnender k-D-Baum für $S_> = \{\mathbf{x} \in S | x_i > w_i\}$.*
  - *$T_=$ ist ein bei Koordinate $i \bmod (k \Leftrightarrow 1)$ beginnender $(k \Leftrightarrow 1)$-D-Baum für*
    $$S_= = \{(x_0, \ldots, x_{i-1}, x_{i+1}, \ldots, x_{k-1}) \in \mathbb{R}^{k-1} \mid \exists \mathbf{x} \in S : \mathbf{x} = (x_0, \ldots, x_{i-1}, w_i, x_{i+1}, \ldots, x_{k-1})\} \setminus \{w\}$$

Wie bei den binären Suchbäumen führt man balancierte Bäume ein, um die Suchzeiten gering zu halten (vgl. [15]):

Definition 4 (idealer k-D-Baum) *Ein k-D-Baum ist ideal, wenn für jeden Knoten v gilt: Die Unterbäume $T_<$ und $T_>$ enthalten jeweils höchstens die Hälfte aller Knoten im Teilbaum mit Wurzel v.*

Aufbau eines idealen k-D-Baums
Für eine gegebene Punktmenge S läßt sich ein idealer k-D-Baum durch eine einfache Median-Strategie konstruieren, um so die gewünschte Balancierung zu erreichen. Man bestimmt den Median bezüglich der Koordinate, die zum entsprechenden Baumlevel gehört und teilt in $S_<$, $S_>$ und $S_=$.

Algorithmus:

1. Eingabe: Punktmenge $S \subseteq \mathbb{R}^k$, $|S| = n$.
2. Ausgabe: idealer k-D-Baum für S.
3. Vorsortierung (einmal):
   Sortiere S nach jeder Koordinate. (Felder $F^0, \ldots, F^{k-1}$)
   Aufwand: $O(k \cdot n \cdot \log n)$
4. Teilungsschritt:
   Sei i die Koordinate, die zum aktuellen Baumlevel gehört. Bestimme den Median von F^i (in $O(1)$) und teile S in $S_<$, $S_=$ und $S_>$, also F^j in $F^j_<$, $F^j_=$ und $F^j_>$ (für $j = 0 \ldots k \Leftrightarrow 1$) auf.
   Aufwand: $O(k \cdot n)$, weil die Sortierung beibehalten wird.

5. Rekursion:
   Teilbäume für $S_<$, $S_=$ und $S_>$ aufbauen.

Pro Baumlevel werden höchstens $k \cdot n$ Operationen ausgeführt, wegen der Balancierung gibt es $\log n$ Level. Inklusive Vorsortierung ergibt sich also ein *Gesamtaufwand* von $O(k \cdot n \cdot \log n)$.

Operationen

1. Einfügen
Ein nachträgliches Einfügen von Punkten mit der naiven Methode führt zu unausgeglichenen Bäumen mit schlechten Suchzeiten. Eine Rebalancierung wie z.B. bei AVL-Bäumen ist schwierig, da man nicht auf Rotationen zurückgreifen kann (zyklischer Koordinatenwechsel). Der in 3.1.3.2 vorgestellte Algorithmus setzt eine statische Punktmenge voraus, in diesem Fall sind ideale Bäume möglich, die Baumtiefe beträgt dann $O(\log n)$.

2. Löschen
Wird ein innerer Knoten gelöscht, muß der abgetrennte Teilbaum neu aufgebaut werden. Im worst case wird die Wurzel gelöscht und der gesamte Baum muß neu aufgebaut werden, *Aufwand* $O(k \cdot n \cdot \log n)$.

3. Punktsuche
Wie bei binären Suchbäumen sucht man je nach Suchkoordinate in $T_<$, $T_=$ oder $T_>$ rekursiv weiter, bis man auf ein Blatt trifft oder den vorgegebenen Punkt findet. Bei idealen k-D-Bäumen läßt sich die Suchzeit folgendermaßen abschätzen: Wegen der Balancierung liegen höchstens $\log n$ <- oder >-Zeiger auf dem Suchpfad (vgl. Definition 4). Da es k Koordinaten gibt, sind maximal k =-Zeiger möglich. Der *Suchaufwand* beträgt also $O(k + \log n)$.

4. Bereichssuche
Dazu geben wir die entsprechende Prozedur in Pseudocode an:

Bezeichnungen:

- Suchbereich $R = [x_1, x_2] \times [y_1, y_2]$ (im $\mathbb{R}^2$)
- $P(v)$: Punkt, der zum Knoten v gehört.
- $S(v)$: Punktmenge im Baum mit Wurzel v.
- $v.>$, $v.<$, $v.=$: Söhne von v.
- $R(v)$: Region von v.

```
procedure region(v:Knoten, R:Rechteck)
begin
  if P(v) ∈ R then P(v) ausgeben;
  if |S(v)| > 1 then
  begin
    if R(v.>) ∩ R ≠ ∅ then region(v.>, R);
    if R(v.<) ∩ R ≠ ∅ then region(v.<, R);
```

```
    if R(v. =) ∩ R ≠ ∅ then region(v. =, R);
  end
end.
```

Beim Prozeduraufruf steht in v die Wurzel des zu durchsuchenden k-D-Baums, in R der gewünschte Suchbereich. Liegt der dem Knoten v zugeordnete Punkt im Suchbereich, wird dieser ausgegeben. Falls v kein Blatt ist, werden die Unterbäume rekursiv durchsucht, deren Regionen sich mit dem Suchbereich überlappen. Eine Laufzeitabschätzung hierzu findet man in [15] und [4]:

Satz 2 *Sei T ein idealer k-D-Baum für $S \subseteq \mathbb{R}^k$, $|S| = n$. Dann ist eine Bereichsanfrage in der Zeit*

$$O(k \cdot 4^k \cdot n^{1-1/k} + k \cdot |A|)$$

möglich, wobei A die Menge der Antworten ist.

Beweis: für $k = 2$: $O(32 \cdot \sqrt{n} + 2 \cdot |A|)$
Sei $R(v)$ die zum Knoten v zugehörige Region, d.h. alle Punkte im Teilbaum mit der Wurzel v liegen in dieser Region. Es gibt 7 Regionentypen (Abb. 3.5). Die Wurzel von T habe den Baumlevel 0, dann treten auf Level 1 die Typen 1 und 5 (Halbebene und Gerade) auf. Ab Level 2 gibt es Quadranten (Typ 2) und Halbgeraden (Typ 6). Ab Level 3 hat man drei Begrenzungslinien zu Verfügung, deswegen ist Typ 3 möglich (Streifen). Ab Level 4 tauchen Rechtecke (Typ 4) und Strecken (Typ 7) auf. Zur Aufwandsabsschätzung zählen wir die Anzahl der

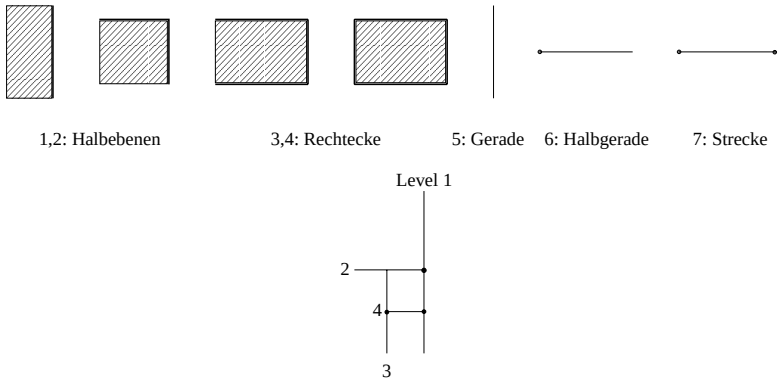


Abb. 3.5: Regionentypen

Knoten im Teilbaum T', der bei der Bearbeitung der Bereichsanfrage für das Rechteck $R = [x_1, x_2] \times [y_1, y_2]$ besucht wird. Ein Knoten v wird genau dann besucht, wenn sich Region und Suchbereich überlappen, also:

$$v \in T' \Longleftrightarrow R(v) \cap R \neq \emptyset$$

T' zerlegen wir in drei disjunkte Knotenmengen X, Y und Z:

$$\begin{aligned} X &= \{v \in T' | R(v) \subseteq R\} \\ Y &= \{v \in T' | R(v) \cap R \neq \emptyset, R(v) \Leftrightarrow R \neq \emptyset \text{ und } R \Leftrightarrow R(v) \neq \emptyset\} \\ Z &= \{v \in T' | R \subset R(v)\} \end{aligned}$$

$|X|$ kann leicht durch $|A|$ nach oben abgeschätzt werden, weil $X \subseteq A$ (die Region liegt ganz im Suchbereich, also auch der Knoten selber).Beim Test $P(v) \in R$ fallen dann $2 \cdot |A|$ Operationen an, da beide Koordinaten geprüft werden müssen.

$$\text{Anzahl der Tests } P(v) \in R\text{:} \quad 2 \cdot |A| \tag{3.1}$$

Für Z gilt $|Z| \leq \log n$, weil alle Knoten $v_i \in Z$ auf einem Pfad liegen $(R(v_1) \supseteq R(v_2) \supseteq \dots)$ und $\log n$ die Baumtiefe ist.

$$|Z| \leq \log n \tag{3.2}$$

Gilt $v \in Y$, so muß $R(v)$ mindestens einmal den Rand von R schneiden, weil ein Teil von $R(v)$ innerhalb von R liegt, ein Teil außerhalb. Der Rand von R besteht aus vier Randstücken (Rechteckseiten). Sei L das Randstück, das am häufigsten von solchen Knoten geschnitten wird und t_{max} die Anzahl der Knoten $v \in Y$, deren Region L schneidet. Für Y erhalten wir dann

$$|Y| \leq 4 \cdot t_{max} \tag{3.3}$$

Um t_{max} zu bestimmen, nehmen wir im folgenden o.B.d.A. an, daß L der rechte Rand von R ist. Die Menge der Knoten, die L schneiden, zerlegen wir in V und W, dabei wird nach der Gestalt der Knotenregionen unterschieden:

$$\begin{aligned} V &= \{v \in Y | R(v) \text{ ist eindimensional } \} \qquad \text{(Typ 5-7)} \\ W &= \{v \in Y | R(v) \text{ ist zweidimensional}\} \qquad \text{(Typ 1-4)} \\ t_{max} &= |V| + |W| \end{aligned}$$

Auf dem Pfad zu einem Knoten $w \in W$ kommt also kein =-Zeiger vor, bei einem Knoten $z \in V$ genau einer (eine Koordinate ist schon fest).

$W_l \subseteq W$ sei die Menge der Knoten aus W auf dem Level l des Baums, $W = \bigcup_{l=0}^{\log n} W_l$. Jeder Knoten $v \in W_l$ hat zwei Level tiefer höchstens zwei Enkel aus W, also $|W_{l+2}| \leq 2 \cdot |W_l|$. Dazu betrachte man Abb. 3.6. Nach der Wahl der

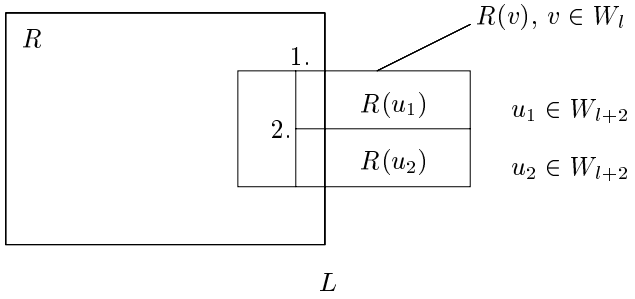


Abb. 3.6: zwei Enkel

ersten Splitlinie kommt der Bereich, der ganz in R liegt (oder derjenige, der ganz außerhalb von R liegt), für W nicht mehr in Frage. Nach dem zweiten Splitten entstehen also höchstens zwei Enkel in W_{l+2}, wie in der Abbildung dargestellt.

Mit der Anfangsbedingung $|W_0| \leq |W_1| \leq 2$ ergibt sich

$$|W_{2m}| \leq 2|W_{2m-2}| \leq \ldots \leq 2^m|W_0| \leq 2 \cdot 2^m \tag{3.4}$$
$$|W_{2m+1}| \leq 2|W_{2m-1}| \leq \ldots \leq 2^m|W_1| \leq 2 \cdot 2^m \tag{3.5}$$

Nun zählen wir die Knoten in V, sie sind Söhne von Knoten aus W, weil eine eindimensionale Region immer in einer zweidimensionalen liegt und der Vater ebenfalls L schneiden muß.

Bezeichne $S(u)$ für $u \in S$ die im Teilbaum mit Wurzel u gespeicherte Punktmenge. Sei $z \in V$ der $=$-Sohn von $w \in W_l$, dann gilt $R(z) \subseteq R(w)$ und somit $|S(z)| \leq |S(w)| \leq \frac{n}{2^l}$ (ein Teilbaum mit Wurzel w auf Level l hat höchstens $\frac{n}{2^l}$ Knoten wegen der Balancierungseigenschaft) (siehe Abb. 3.7).

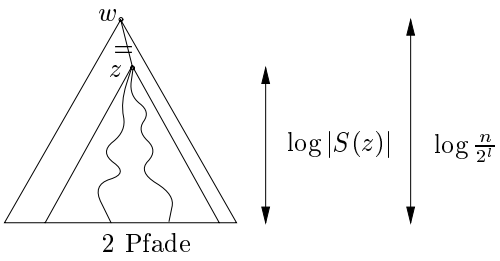


Abb. 3.7: $w \in W_l$ und $z \in V$

Weiterhin hat z höchstens $2 \cdot \log|S(z)|$ Nachfolger, die ebenfalls in V liegen. $R(z)$ ist nämlich eine horizontale oder vertikale Linie (siehe Abb. 3.8). Im ersten Fall

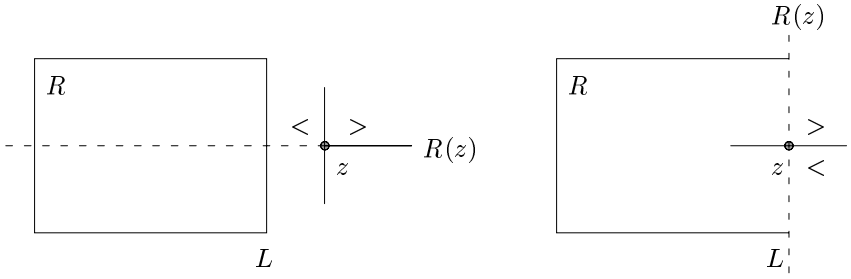


Abb. 3.8: $R(z)$ horizontal bzw. vertikal

liegen alle Nachfolger von z, die L schneiden, auf *einem* Pfad im Baum T, weil man entweder den $<$- oder den $>$-Zeiger verfolgen muß. Für die Region des Nachfolgers von z (gestrichelt) gilt wiederum dasselbe. Im zweiten Fall sind zwei Pfade möglich, man verfolgt beide Zeiger und hat dann bei den beiden Nachfolgern wiederum den ersten Fall vorliegen. Die Anzahl der Nachfolger ist also höchstens

$$2 \cdot \log|S(z)| \leq 2 \cdot \log \frac{n}{2^l} \tag{3.6}$$

Um $|V| + |W|$ nach oben abzuschätzen, zählen wir für jeden Baumlevel l die Knoten $w \in W_l$ und deren Nachfolger in V, wobei hierbei natürlich Knoten aus V doppelt gezählt werden (siehe Abb. 3.9).

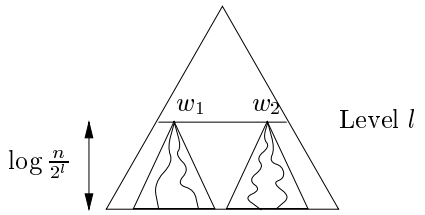


Abb. 3.9: pro $w \in W_l$ sind zwei Pfade möglich

$$
\begin{aligned}
t_{max} = |V| + |W| &\leq \sum_{l=0}^{\log n} |W_l| \cdot 2 \log \frac{n}{2^l} && \text{(mit 3.6)} \\
&\leq \sum_{l=0}^{\log n} 2 \cdot 2^{l/2} \cdot 2 \cdot \log \frac{n}{2^l} && \text{(mit 3.5)} \\
&= 4 \cdot \sum_{l=0}^{\log n} 2^{l/2} \cdot (\log n \Leftrightarrow l) \\
&= 4\sqrt{n} \cdot \sum_{l=0}^{\log n} 2^{\frac{l-\log n}{2}} \cdot (\log n \Leftrightarrow l) \\
&= 4\sqrt{n} \cdot \sum_{l=0}^{\log n} (\frac{1}{\sqrt{2}})^l \cdot l \in O(4\sqrt{n})
\end{aligned}
$$

Für Y ergibt sich also $|Y| \in O(16 \cdot \sqrt{n})$ (wegen 3.3) und $|Y| + |Z| \in O(16 \cdot \sqrt{n} + \log n) = O(16 \cdot \sqrt{n})$ (mit 3.2). Für jeden besuchten Knoten fallen beim Test $P(v) \in R$ zwei Operationen an, also erhält man mit (3.1) den *Gesamtaufwand:*

$$O(32 \cdot \sqrt{n} + 2 \cdot |A|)$$

■

3.1.3.3 Quad-Trees

Einführung
Die wohl bekannteste geometrische Datenstruktur ist der Quad-Tree (siehe [1]). Wir betrachten wiederum nur den zweidimensionalen Fall, die Verallgemeinerung für höhere Dimensionen (Okt-Trees usw.) macht keine besonderen Schwierigkeiten. Während bei den k-D-Bäumen eine Unterteilung in Halbebenen erfolgte, unterscheidet man nun vier Quadranten, es ergeben sich also Bäume mit dem Verzweigungsgrad vier (im $\mathbb{R}^k$ Verzweigungsgrad 2^k). Die Zuordnung der

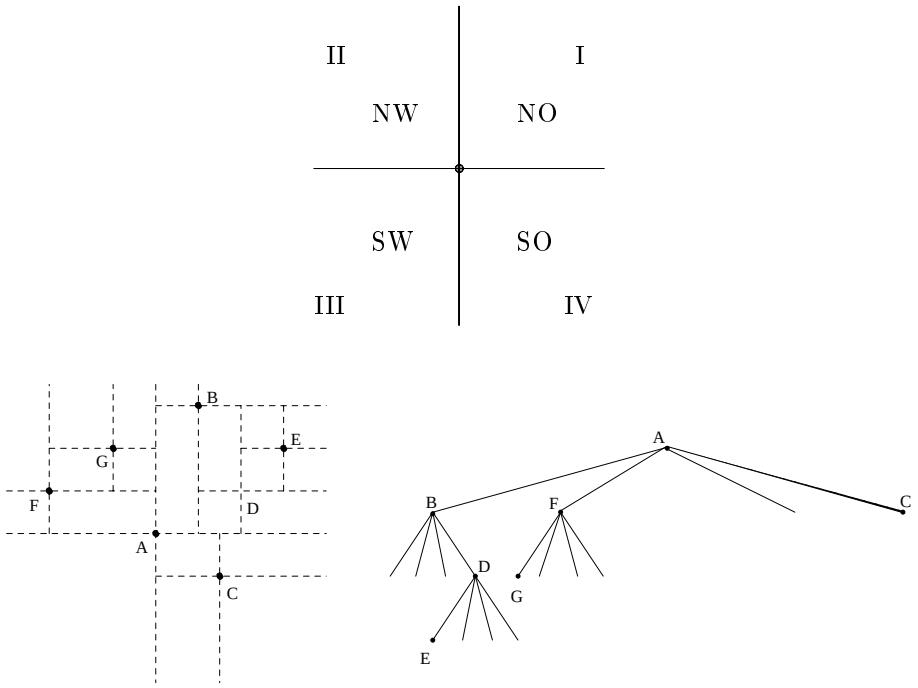


Abb. 3.10: Punktmenge und zugehöriger Quad-Tree

Punkte auf den Trennungslinien kann man willkürlich wählen, z.B. seien im folgenden Quadrant I und Quadrant III abgeschlossen.

Als Beispiel betrachte man Abb. 3.10. Wählt man den Punkt A als Wurzel, so liegt B im ersten Quadranten von A, wird also der erste Sohn von A. C liegt im vierten Quadranten und wird der vierte Sohn. D liegt im ersten Quadranten von A, da aber dort schon der Sohn B existiert, bestimmt man den Quadranten von D bzgl. von B usw.

Operationen

1. Einfügen, Punktsuche
Verzichtet man auf eine Balancierung der Bäume, kann man die übliche Einfügeprozedur verwenden:

```
procedure insert(v:Knoten, p:Punkt)
begin
  direction:=compare(v,p);        (* Quadrant von p bzgl. v*)
  if v.direction≠nil
    then insert(v.direction, p)
    else ^v.direction:=p;
end.
```

Beim Prozeduraufruf enthält v die Wurzel des zu durchsuchenden Quad-Trees und p ist der gesuchte Punkt. Zunächst bestimmt man den Quadranten, in dem

p in Bezug auf v liegt. Wenn kein entsprechender Sohn von v existiert, kann p eingefügt werden, ansonsten Rekursion auf dem Unterbaum. Die Punktsuche erfolgt durch den üblichen rekursiven Abstieg.

Empirische Untersuchungen für den average case in [1] ergaben eine Baumtiefe von $O(\log_4 n)$, so daß Einfügen und Punktsuche ebenfalls in $O(\log_4 n)$ möglich sind. Der worst case liegt z.B. dann vor, wenn die Punkte auf einer Geraden liegen, in diesem Fall entartet der Quad-Tree zu einer linearen Liste. Damit steigt der Aufwand für das Einfügen und die Punktsuche auf $O(n)$.

Um dies zu vermeiden, kann man wiederum ideale Quad-Trees einführen. Wählt man als Wurzel den Median bezüglich einer Koordinate, dann enthält jeder der vier Teilbäume höchstens die Hälfte der Knoten. Damit ist die Baumtiefe wieder in $O(\log_4 n)$. Wie bei den idealen k-D-Bäumen setzt dies aber eine statische Punktmenge voraus, ein nachträgliches Einfügen zerstört die Balancierung.

2. Löschen
Wird ein innerer Knoten aus dem Baum entfernt, so muß der abgetrennte Teilbaum wieder angefügt werden. Dabei bleibt einem nichts anderes übrig, als die Knoten dieses Teilbaums wieder einzeln einzufügen, falls die Wurzel gelöscht wurde, beträgt der Aufwand $O(n \cdot \log_4 n)$.

3. Bereichssuche
Der Algorithmus entspricht dem für die Bereichssuche in k-D-Bäumen mit dem Unterschied, daß nun eine vierfache Rekursion aufgrund des Verzweigungsgrads vier auftritt. Eine worst-case-Analyse für balancierte Bäume in [12] ergab einen Aufwand von $O(\sqrt{n})$ ($O(n^{1-1/k})$ für den k-dimensionalen Fall). Damit ist die Bereichssuche in Quad-Trees ebenso effizient wie in 2-D-Bäumen.

3.1.3.4 Grid-File

Einführung

Bei den oben vorgestellten Baumstrukturen erhält man eine Unterteilung des Raums in Abhängigkeit von der gespeicherten Punktmenge. Im Grid-File wird ein anderer Ansatz verfolgt; man gibt die Unterteilung des Suchraums explizit vor (siehe [20]). Legt man ein Gitter über die Punktmenge S und speichert die Punkte einer Gitterzelle in einer Punktliste, muß man bei einer Bereichsanfrage nur die überdeckten Gitterzellen durchsuchen. Um die Punktzahl pro Gitterzelle konstant zu halten, paßt man die Feinheit des Gitters der Punktdichte lokal an. Beim Grid-File gilt für die Verfeinerung das Matrixprinzip, d.h. die Gitterlinien durchziehen den ganzen Suchraum. In Abb. 3.11 sieht man links eine Veranschaulichung des Gitters, rechts ein Beispiel für ein Gridfile: In der rechten Abbildung findet man die Einteilung in neun Gitterzellen wieder. Da bei geringer Besetzung einer Gitterzelle eine eigene Punktliste (im folgenden Datenblock) nicht lohnt, faßt man benachbarte Gitterzellen zu Blockregionen zusammen (abgerundete Boxen). Eine Blockregion hat immer die Form eines Rechtecks und umfaßt die Gitterzellen, die zum selben Datenblock gehören.

Zur Speicherung der Intervallteilungen legt man für jede Dimension ein eindimensionales Feld an, ein sogenannter *Scale* (im Bsp.: 0, 25, 50, 100 für x und

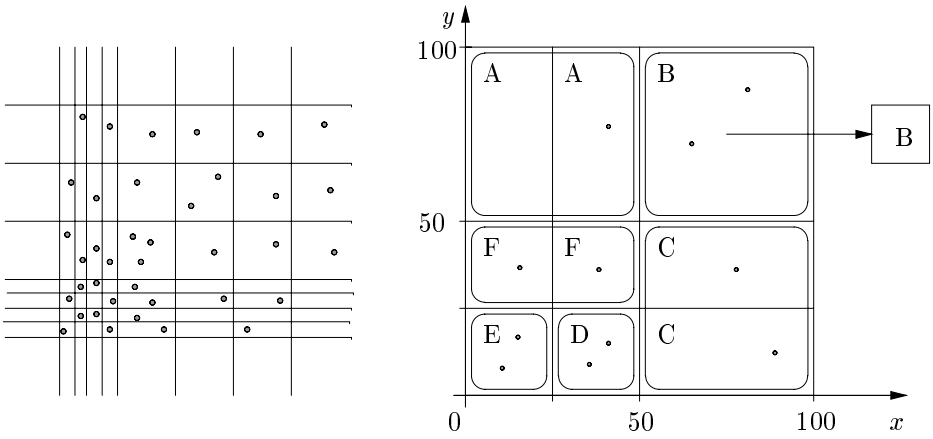


Abb. 3.11: Gridfile

y). Die Elemente der *Directory-Matrix* (=Gitterzellen) enthalten einen Zeiger auf den zugehörigen Datenblock. In den *Datenblöcken* werden die eigentlichen Punktkoordinaten gespeichert. Da diese auf Festplatte gespeichert werden, ergibt sich aus der Plattenblockgröße eine Blockkapazität von b Punkten (im Bsp. $b = 2$).

Als das Gridfile Mitte der 80er Jahre entwickelt wurde, wurden lediglich die *Scales* im (damals kleinen) Hauptspeicher gehalten, während *Directory-Matrix* und *Datenblöcke* auf Festplatte gespeichert wurden. Bei den heutigen Hauptspeichergrößen ist diese strikte Trennung nicht mehr nötig.

Operationen

Zur Aufwandsabschätzung gehen wir von dieser getrennten Datenhaltung aus und zählen die Anzahl der Zugriffe auf Directory-Matrix und Datenblöcke.

1. Punktsuche
Bei der Punktsuche gilt das Zwei-Zugriffs-Prinzip, d.h. man kommt mit zwei Plattenzugriffen aus:

1. Eingabe: Punkt (x, y)
2. Mittels x- und y-Scale Spalte und Zeile der Directory-Matrix bestimmen. *(Hauptspeicher)*
3. Directory-Element holen. *(1. Plattenzugriff)*
4. Mittels Zeiger den Datenblock holen. *(2. Plattenzugriff)*
5. Datenblock nach (x, y) durchsuchen.

2. Einfügen
Zunächst wird der entsprechende Datenblock wie bei der Punktsuche von der Platte geholt. Befinden sich darin weniger als b Punkte (Blockkapazität), so

kann der neue Punkt einfach eingefügt werden. Im anderen Fall müssen der Datenblock und die zugehörige Blockregion geteilt werden. In Abb. 3.12 soll ein neuer Punkt in den Datenblock D eingefügt werden, letzterer wird in die neuen Blöcke D und E geteilt:

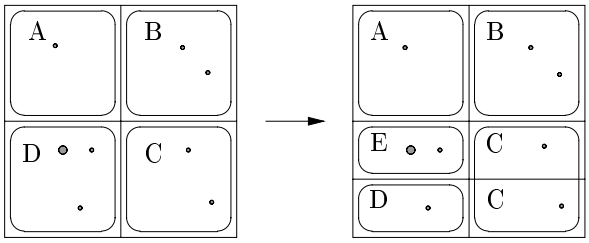


Abb. 3.12: Einfügen

Da hierbei auch unbeteiligte Blockregionen geteilt werden (im Bsp.: Region C), wächst das Directory stärker als eigentlich nötig. Der worst case tritt bei einer starken Häufung der Punkte auf einem Fleck auf, dadurch werden viele Teilungslinien nötig, die den ganzen Suchraum durchziehen.

Bei der Wahl der Splitdimension kann man unterschiedliche Strategien verfolgen, als Beispiele seien genannt:

- teile die längste Seite der Blockregion.
- teile gemäß schon vorhandenen Gitterzellen (im Bsp.: Region C).
- teile die Dimension mit der kleinsten Anzahl von bisher erfolgten Teilungen.

3. Löschen
Zunächst wird wieder der Datenblock aufgesucht und der zu löschende Punkt entfernt. Um keinen Speicherplatz zu verschwenden, sollte die Blockauslastung nicht zu klein werden. Deswegen wird noch überprüft, ob eine Blockverschmelzung durchgeführt werden kann. Dafür müssen zwei Bedingungen erfüllt sein:

1. Blockauslastung $<$ untere Schranke (z.B. 30%)
2. Blockauslastung im resultierenden Block $<$ obere Schranke (z.B. 70%)

Die zweite Bedingung verhindert dabei, daß auf eine Verschmelzung sofort wieder eine Teilung folgt.

Bei der Auswahl der Nachbarregion, mit der die Verschmelzung durchgeführt wird, muß natürlich beachtet werden, daß die resultierende Blockregion wieder rechteckig sein muß. Für die Auswahl gibt es unterschiedliche Strategien. Bei der *Bruderstrategie* macht eine Verschmelzung gerade eine Teilung rückgängig, d.h. nur „Brüder“, die aus einer Teilung hervorgegangen sind, können wieder verschmolzen werden (Bsp.: s.o., Rückrichtung). Die weniger restriktive *Nachbarstrategie* läßt alle Nachbarregionen zu, sofern sie die Rechteckbedingung erfüllen:

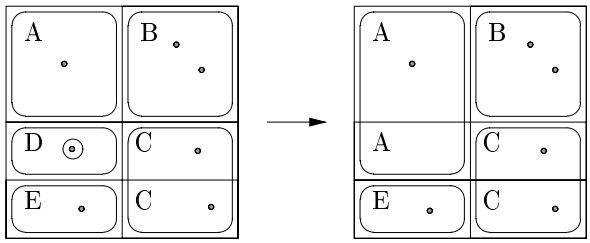


Abb. 3.13: Löschen

Nach dem Löschen des markierten Punktes in Datenblock D werden die Blöcke A und D verschmolzen (Abb. 3.13).

4. Bereichssuche
Der Suchbereich sei wieder durch das Rechteck $R = [x_1, x_2] \times [y_1, y_2]$ gegeben. Zunächst sucht man nach dem Punkt (x_1, y_1) (Ecke links unten) im Grid-File und überprüft die Punkte im zugehörigen Datenblock. Danach muß im Directory nach oben und nach rechts weitergesucht werden, was einer Bereichsanfrage auf der Directory-Matrix entspricht. Wir erläutern das Vorgehen an einem Beispiel. In Abb. 3.14 ist ein Grid-File mit entsprechendem Suchbereich dargestellt. Für

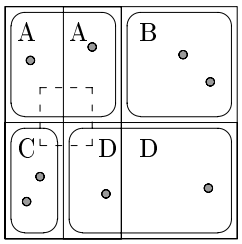


Abb. 3.14: einstufiges Grid-File

die Bereichssuche sind in diesem Beispiel 7 Plattenzugriffe nötig, und zwar für 4 Directoryelemente und 3 Datenblöcke (A, C und D).

Dieses einstufige Grid-File läßt sich zu einem zweistufigen erweitern. Hierbei werden Gitterzellen zu Blockdirectories zusammengefaßt und darüber ein Wurzeldirectory im Hauptspeicher gehängt, das Zeiger auf die Blockdirectories enthält (Abb. 3.15). Mit dieser Maßnahme braucht man nur noch 5 Plattenzugriffe, 2 für die beiden Blockdirectories und 3 für die Datenblöcke.

3.1.3.5 Vergleich

k-D-Bäume und Quad-Trees sind *statische* Datenstrukturen, ein nachträgliches Einfügen und Löschen von Punkten führt zu unausgeglichenen Bäumen und kommt somit nicht in Betracht. Bei beiden ist eine effiziente Bereichsanfrage mit dem Aufwand $O(\sqrt{n} + |A|)$ möglich (für den zweidimensionalen Fall). Eine

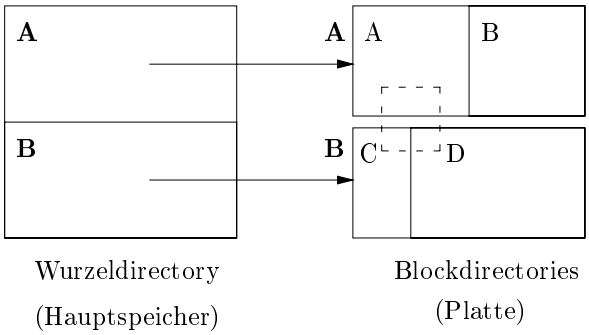


Abb. 3.15: zweistufiges Grid-File

Implementierung von Pointerstrukturen ist nur im Hauptspeicher sinnvoll, man würde sonst zuviele Plattenzugriffe benötigen.

Demgegenüber ist das Grid-File eine *dynamische* Datenstruktur, Einfügen und Löschen ist mit geringem Overhead (split und merge) möglich. Die Bereichsanfrage ist ebenfalls sehr effizient und die dafür benötigte Zeit hängt nur von der Größe des Suchbereichs, nicht der gesamten Punktezahl ab. Durch mehrstufige Grid-Files kann die Effizienz noch verbessert werden. Das Grid-File wurde ursprünglich für Sekundärspeicher konzipiert, könnte aber durchaus auch im Hauptspeicher gehalten werden.

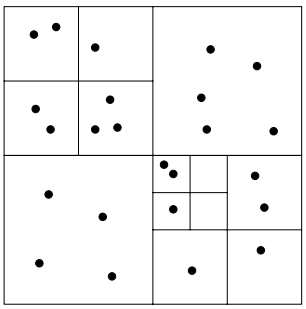


Abb. 3.16: Grid-Quad-Baum

Quad-Tree und Gridfile lassen sich zum Grid-Quad-Baum kombinieren. Dabei handelt es sich um ein Gridfile, bei dem die Aufteilung des Raums nicht nach dem Matrixprinzip, sondern wie bei den Quad-Trees erfolgt (s. Abb. 3.16). Man baut als einen Quad-Tree auf, bei dem nur die Blätter Datenblöcke der Kapazität b enthalten. Wie beim Gridfile werden die abzuspeichernden Punkte in die Datenblöcke eingetragen. Beim Teilen von Datenblöcken werden vier Unterquadranten erzeugt, beim Verschmelzen geht man nach der Bruderstrategie vor. Durch diese neue Aufteilung des Raums vermeidet man den hohen Speicheraufwand, der beim Matrixprinzip entsteht.

3.2 Matching Verfahren

3.2.1 Einleitung

Diese Ausarbeitung zeigt wie das 2'er FGM-Problem effizient gelöst werden kann. Dazu wird zunächst der Übergang von diesem Problem zu dem in der Graphentheorie bekannten Problem des Maximalen Matchings erläutert. Dann wird ein Algorithmus vorgestellt, der ein Maximales Matching mit einem Aufwand von höchstens $O(|V|^{2,5})$ Operationen findet. Dieser Algorithmus besteht aus zwei Teilen. Der erste Teil, die äußere Schleife, wurde von Hopcroft und Karp [8] entwickelt und der zweite Teil, die innere Schleife, von Micali und Vazirani [16].

3.2.2 Vom 2'er FGM-Problem zum Maximalen Matching

3.2.2.1 2'er FGM-Problem

Der Ausgangspunkt ist eine Menge von Personen, von denen Eigenschaften wie z.B. Wohnort, Arbeitszeit, Arbeitsort und Raucher/Nichtraucher, bekannt sind. Das Problem besteht darin, aus dieser Personenmenge eine maximale Anzahl von 2'er-Fahrgemeinschaften zu bilden. Dabei sollen der für den Fahrer entstehende Umweg und die Eigenschaften der einzelnen Personen berücksichtigt werden. Zur effizienten Lösung diese Problems werden Erkenntnisse aus der Graphentheorie benutzt.

3.2.2.2 Personengraph

Aus der in Abschnitt 3.2.2.1 vorgestellten Personenmenge und den Eigenschaften der einzelnen Personen wird ein Personengraph $G = (V, E)$ gebildet. Die Knoten $v \in V$ repräsentieren dabei die Personen und eine Kante $e \in E$ bedeutet, daß die verbundenen Personen zusammen in einer Fahrgemeinschaft fahren könnten. Zur Erstellung des Personengraphen wird für jedes Personenpaar geprüft, ob eine Fahrgemeinschaft möglich ist, und wenn ja, dann wird eine Kante in den Graph eingefügt. Ab jetzt wird das Problem nur noch aus Sicht der Graphentheorie gesehen. Es gibt nur noch einen Graphen mit Knoten und Kanten, dessen Entstehung für die Lösung des Problems zunächst nicht mehr von Bedeutung ist.

3.2.2.3 Matching

Definition

Sei $G = (V, E)$ ein Graph mit der Knotenmenge V und der Kantenmenge E. Dann heißt die Menge $M \subseteq E$ **Matching**, wenn kein Knoten $v \in V$ mit mehr als einer Kante aus M verbunden ist; (Abbildung 3.17).

Eine Kante $e \in E$ heißt **'matched'**, wenn $e \in M$ ist.
Eine Kante $e \in E$ heißt **'unmatched'**, wenn $e \in (E \Leftrightarrow M)$ ist.

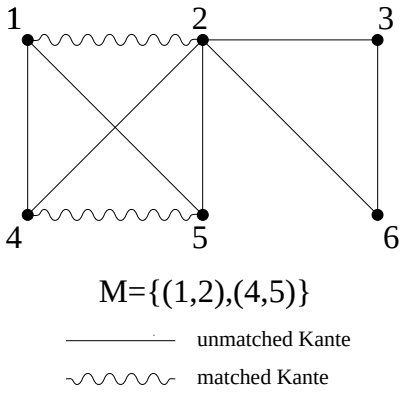


Abb. 3.17: Beispiel für ein Matching

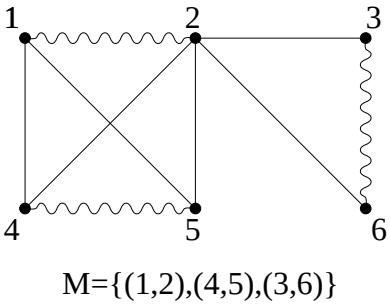


Abb. 3.18: Beispiel für ein Maximales Matching

Bei allen Abbildungen werden 'matched' Kanten als Wellenlinie und 'unmatched' Kanten als gerade Linien dargestellt.

3.2.2.4 Maximales Matching

Definition

Ein Matching M auf einem Graphen G heißt **Maximales Matching**, wenn kein Matching M' existiert, das mehr Kanten als M enthält; (Abbildung 3.18).

Ein Maximales Matching ist nicht eindeutig, es kann zu einem Graphen mehrere Maximale Matchings geben, nur die Größe dieser maximalen Matchings ist eindeutig.

3.2.3 MM-Algorithmus

Zur Berechnung eines Maximalen Matchings auf einem ungerichteten Graphen kann der von Hopcroft und Karp [8] und Micali und Vazirani [16] entwickelte

Algorithmus verwendet werden. In diesem und den folgenden Abschnitten wird die Funktionsweise vorgestellt und die Laufzeit des MM-Algorithmus betrachtet. Um den Algorithmus verstehen zu können, müssen zunächst einige neue Begriffe eingeführt werden.

Definitionen

Sei $G = (V, E)$ ein Graph und M ein Matching.

- Ein Knoten $v \in V$ heißt **freier Knoten**, falls v mit keiner Kante aus M verbunden ist.
- Ein Pfad $(v_1, v_2, v_3, \ldots, v_k)$ mit $(v_i, v_{i+1}) \in E$, $1 \leq i \leq k$ und $k \in \mathbb{N}$, heißt **alternierender Pfad**, falls dessen Kanten (v_i, v_{i+1}) abwechselnd 'matched' und 'unmatched' sind.
- Ein Pfad $(v_1, v_2, v_3, \ldots, v_k)$ mit $(v_i, v_{i+1}) \in E$, $1 \leq i \leq k$ und $k \in \mathbb{N}$, heißt **augmentierender Pfad**, falls er alternierend ist und die Knoten v_1 und v_k freie Knoten sind.

Um ein gefundenes nicht maximales Matching zu vergrößern, wird während der Berechnung immer wieder das Matching mit einem augmentierenden Pfad durch **'disjunktives oder'** verknüpft. Diese Verknüpfung wird an dieser Stelle kurz erläutert, bevor der Algorithmus näher beschrieben wird.

In Abbildung 3.19 ist am Beispiel zu sehen, wie Matchings (M_i) mit augmentierenden Pfaden (P_j) durch 'disjunktives oder' $(\oplus)$ verknüpft werden.

Das Ergebnis der Verknüpfung $M_{i-1} \oplus P_i$ ist M_i. M_i ist wieder ein Matching und enthält genau eine Kante mehr als M_{i-1}. Wie auch im Beispiel zu sehen, besteht M_i aus allen Kanten, die entweder zu M_{i-1} **oder** zu Pi aber **nicht** zu M_{i-1} **und** zu P_i gehören.

Der MM-Algorithmus berechnet zu einem ungerichteten Graphen $G = (V, E)$ ein Maximales Matching. Dazu wird, beginnend beim leeren Matching $M = \emptyset$, die maximale Menge kürzester augmentierender Pfade bezüglich M gesucht. Die Pfade dieser Menge werden dann nacheinander durch 'disjunktives oder' mit M verknüpft, wodurch das Matching M erweitert wird. Diese Suche der kürzesten augmentierenden Pfade und die anschließende Erweiterung von M wird solange wiederholt, bis keine augmentierenden Pfade mehr gefunden werden. Dann wird der Algorithmus abgebrochen, da das Maximale Matching gefunden wurde. In der äußeren Schleife des Algorithmus wird die Verknüpfung mit 'disjunktivem oder' durchgeführt und in der inneren Schleife wird die maximale Menge kürzester augmentierender Pfade bezüglich M gesucht. Der Aufwand zur Berechnung des Maximalen Matchings liegt für den hier vorgestellten MM-Algorithmus in $O(\sqrt{|V|}|E|)$.

Eingabe Ein ungerichteter Graph $G = (V, E)$.

Schritt 0 $M \leftarrow \emptyset$

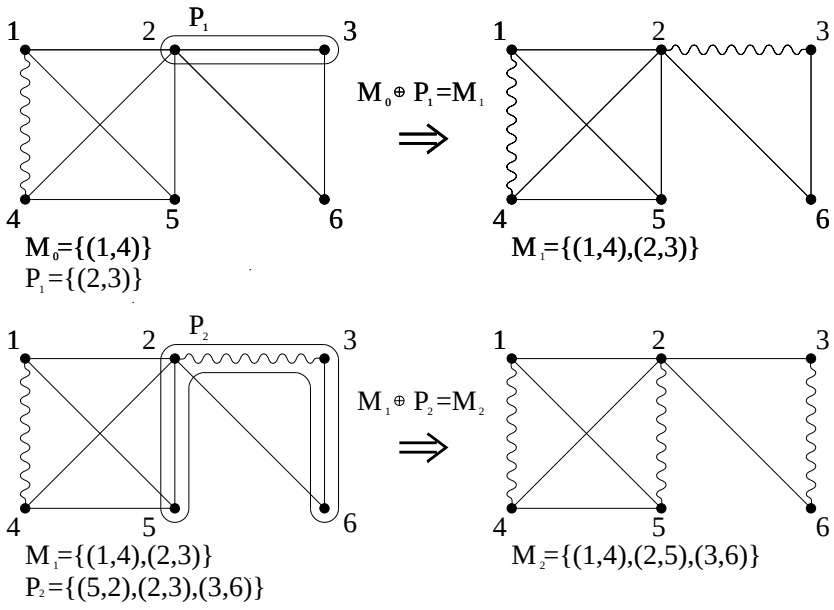


Abb. 3.19: Beispiel für Verknüpfungen mit 'disjunktivem oder'

Schritt 1 Sei $l(M)$ die Länge eines kürzesten augmentierenden Pfades bezüglich M. Finde eine maximale Menge von Pfaden $\{P_1, P_2, \ldots, P_t\}$ mit folgenden Eigenschaften:

a) für alle i ist P_i ein augementierender Pfad bezüglich M und $|P_i| = l(M)$;

b) alle P_i sind knotendisjunkt

stoppe, wenn kein solcher Pfad mehr existiert.

Schritt 2 $M \leftarrow M \oplus P_1 \oplus P_2 \oplus \cdots \oplus P_t$; **gehe zu Schritt 1**

Ausgabe Ein maximales Matching M mit $M \subset E$.

3.2.4 Äußere Schleife

In der äußeren Schleife des MM-Algorithmus wird nach der Initialisierung in Schritt 0 die innere Schleife in Schritt 1 ausgeführt. Nach der Abarbeitung des Schritt 1 wird dann in Schritt 2 das Matching M um die, bei Ausführung der inneren Schleife gefundenen augmentierenden Pfade $\{P_1, P_2, \ldots, P_t\}$ ergänzt. Dann wird wieder Schritt 1 aufgerufen. Werden bei einem Aufruf der inneren Schleife keine augmentierenden Pfade mehr gefunden, so ist M das Maximale Matching und der Algorithmus wird abgebrochen.

Es kann gezeigt werden, daß der Schritt 1 des MM-Algorithmus zur Berechnung eines Maximalen Matchings M mit $|M| = s$ höchstens $2\lfloor\sqrt{s}\rfloor + 2$ mal ausgeführt werden muß. Deshalb liegt der Aufwand für die äußere Schleife in $O(\sqrt{|V|})$. Die Beweisführung ist in [8] und [3] enthalten.

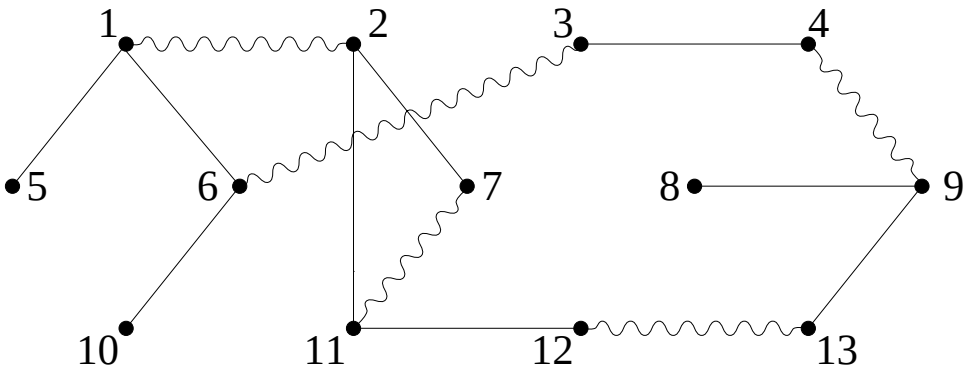


Abb. 3.20: Graph G und Matching M_5

3.2.5 Innere Schleife

In der Inneren Schleife des MM-Algorithmus wird eine maximale Menge kürzester augmentierender Pfade bezüglich eines Matchings M gesucht. Die Vorgehensweise wird zunächst an einem einfachen Beispiel gezeigt und dann im Detail erläutert.

3.2.5.1 Einfaches Beispiel zur Inneren Schleife

Die Berechnung eines Maximalen Matchings auf dem Graphen $G = (V, E)$ ist bei dem Matching M_5 angelangt (Abbildung 3.20). Die 'matched' Kanten sind in Abbildung 3.20 wieder als Wellenlinie und die 'unmatched' Kanten als gerade Linien gezeichnet.

In der nun folgenden Abarbeitung der inneren Schleife wird eine maximale Menge kürzester augmentierender Pfade gefunden. Dazu wird der Graph mit Breitensuche ausgehend von allen freien Knoten durchlaufen.

Im Beispiel sind die Knoten 5, 8 und 10 die Startpunkte der Suche nach den augmentierenden Pfaden. Da ein augmentierender Pfad an den Enden 'unmatched' Kanten besitzt, werden im ersten Schritt der Suche nur solche Kanten berücksichtigt. Der nach dem ersten Schritt der Suche entstandene Breitensuchbaum ist in Abbildung 3.21 dargestellt. Beim Knoten 5 wurde die Kante $(5, 1)$, beim Knoten 8 die Kante $(8, 9)$ und beim Knoten 10 die Kante $(10, 6)$ gefunden.

Im nächsten Schritt wird nur nach 'matched' Kanten gesucht, da augmentierende Pfade abwechselnd aus 'matched' und 'unmatched' Kanten bestehen. Das Ergebnis des zweiten Schritts ist die Erweiterung des Suchbaums um die Kanten $(1, 2)$, $(9, 4)$ und $(6, 3)$. Im dritten Schritt werden dann wieder 'unmatched' Kanten gesucht, wobei die Kanten $(2, 7)$, $(2, 11)$ und $(4, 3)$ gefunden werden. Der nach den beiden Schritten erstellte Breitensuchbaum ist in Abbildung 3.22 zu sehen.

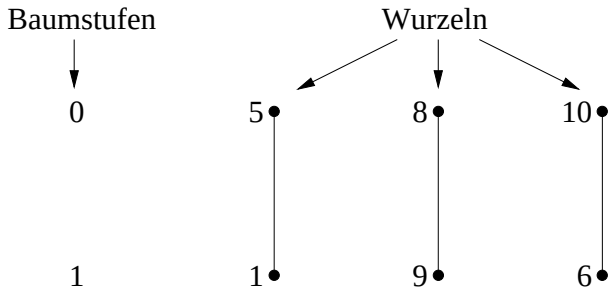


Abb. 3.21: Breitensuchbaum nach dem ersten Schritt der Suche

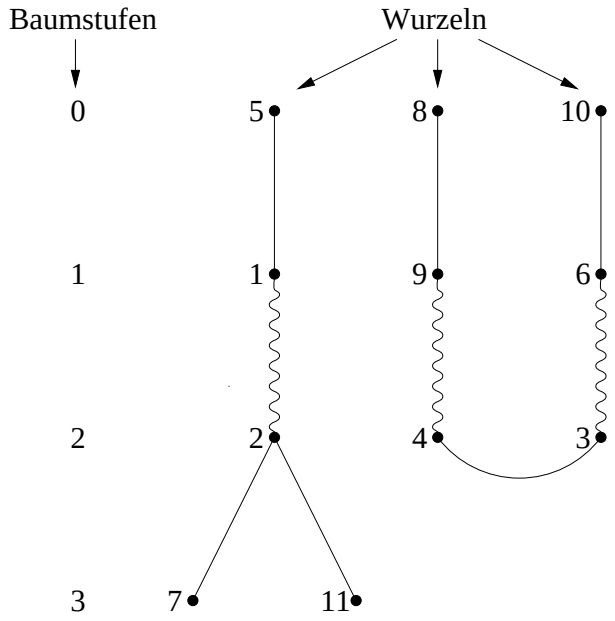


Abb. 3.22: Breitensuchbaum nach dem dritten Schritt der Suche

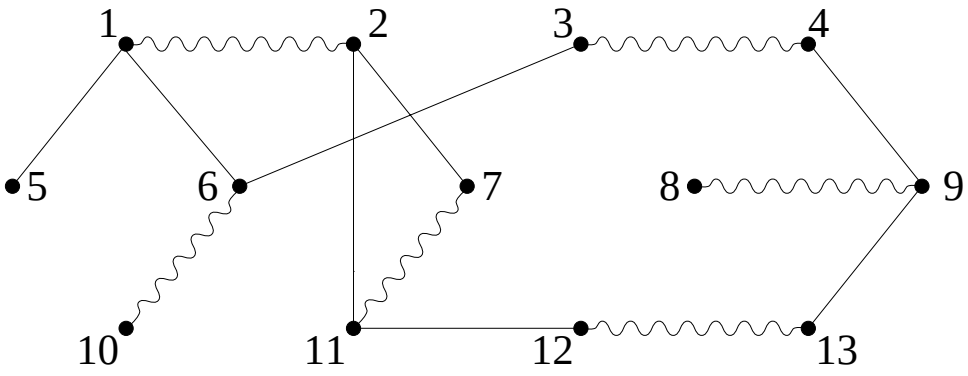


Abb. 3.23: Graph G und Matching M_6

Beim dritten Schritt haben sich die Äste mit den Wurzelknoten 8 und 10 getroffen, womit ein kürzester augmentierender Pfad P gefunden wurde. Da sich im dritten Schritt nur zwei Äste getroffen haben, besteht die maximale Menge kürzester Augmentierender Pfade aus genau einem Pfad. Würde im nächsten Schritt ein augmentierender Pfad gefunden, so wäre dieser länger als der Pfad P von 8 nach 9 und deshalb kein kürzester Pfad. Da die maximale Menge kürzester Augmentierender Pfade im dritten Schritt gefunden wurde, wird die Ausführung der inneren Schleife beendet.

Nach dem Ende der inneren Schleife würde nun die äußere Schleife weiter fortgesetzt, wobei sich durch die Verknüpfung $M_5 \oplus P$ das in Abbildung 3.23 dargestellte Matching M_6 ergibt.

Der in dem Beispiel angedeutete Algorithmus zum Berechnen einer maximalen Menge kürzester augmentierender Pfade bezüglich eines Matchings M, wurde von Micali und Vazirani [16] entwickelt. Für diese Berechnung benötigt der Algorithmus höchstens $O(|E|)$ Operationen. Die wesentlichen Bestandteile des Algorithmus sind die vier Routinen SEARCH, BLOSS-AUG, FINDPATH und TOPOLOGICAL ERASE, die im Weiteren genau beschrieben werden.

3.2.5.2 SEARCH

Die Routine SEARCH durchläuft den Graphen, in dem augmentierende Pfade gesucht werden, mit Breitensuche und baut dabei einen Breitensuchbaum auf. Für jeden Knoten und einige Kanten des Suchbaums werden Eigenschaften bestimmt und abgespeichert. Zur Beschreibung dieser Eigenschaften ist die Einführung einiger Begriffe nötig.

Definitionen

evenlevel(v) Ist die Länge des kürzesten alternierenden Pfades vom Knoten v zu einem freien Knoten, wobei die Anzahl der Kanten des Pfades **gerade**

ist. Wenn kein solcher Pfad existiert, dann ist evenlevel unendlich.

oddlevel(v) Ist die Länge des kürzesten alternierenden Pfades vom Knoten v zu einem freien Knoten, wobei die Anzahl der Kanten des Pfades **ungerade** ist. Wenn kein solcher Pfad existiert, dann ist oddlevel unendlich.

level(v) Ist das Minimum aus evenlevel(v) und oddlevel(v).

outer Ein Knoten v heißt outer, wenn level(v) gerade ist.

inner Ein Knoten v heißt inner, wenn level(v) ungerade ist.

other level(v) Wenn v outer ist, dann ist other level(v) =oddlevel(v), wenn v inner ist, dann ist other level(v) =evenlevel(v).

bridge Eine Kante(u, v) heißt bridge, wenn etweder evenlevel(u) $< \infty$ und evenlevel(v) $< \infty$, oder oddlevel(u) $< \infty$ und oddlevel(v) $< \infty$ ist.

tenacity(u, v) Ist das Minimum aus ((evenlevel(u)+evenlevel(v)), (oddlevel(u)+oddlevel(v))) + 1.

Das Ziel der Suche ist erreicht, wenn eine maximale Menge kürzester augmentierender Pfade bezüglich eines Matchings M gefunden wurde. Die Operationenfolge zur Berechnung einer maximalen Menge wird als eine **Phase** bezeichnet. In einer solchen Phase wird jeder Knoten des Graphen von SEARCH höchstens einmal durchlaufen.

Während der Breitensuche wird der Breitensuchbaum aufgebaut und zu jedem besuchten Knoten werden Eigenschaften abgespeichert. Die Suche beginnt bei allen freien Knoten, d.h. zuerst wird von jedem freien Knoten mindestens ein Nachbarknoten besucht und erst dann wird die Suche, bei den in den Baum aufgenommenen Nachbarknoten, fortgesetzt.

Die Baumstufen des Breitensuchbaums werden als **level** bezeichnet, wobei die Zählung bei der Wurzel mit 0 beginnt. Das bedeutet, daß sich alle freien Knoten im level 0 befinden und deren Nachbarknoten bilden dann das level 1. Das level i gibt an wieviele Kanten ein Knoten v von dem nächsten freien Knoten entfernt ist. Dieser freie Knoten ist der Wurzelknoten, der Vorgänger von v ist.

Die Entfernung, zwischen v und dem freien Knoten, kann gerade und ungerade Länge haben und da dies für die Erkennung von augmentierenden Pfaden wichtig ist, bekommt jeder Knoten zwei Parameter. Die Paremeter sind **oddlevel** und **evenlevel** und werden mit unendlich initialisiert. Bei der Suche wird für einen Knoten, der in einem ungeraden level liegt, oddlevel auf den Wert der Entfernung zum nächsten freien Knoten gesetz und bei Knoten in geraden leveln entsprechend evenlevel. Der im Beispiel in Abschnitt 3.2.5.1 entstandene Breitensuchbaum ist in Abbildung 3.24 mit den Angaben zu level, evenlevel und oddlevel dargestellt.

Die Suche im level i wird in Abhängigkeit von i in folgender Weise fortgesetzt:

- i ist gerade
  - Bei der Suche wird von allen Knoten v mit evenlevel(v) $= i$ ausgegangen.

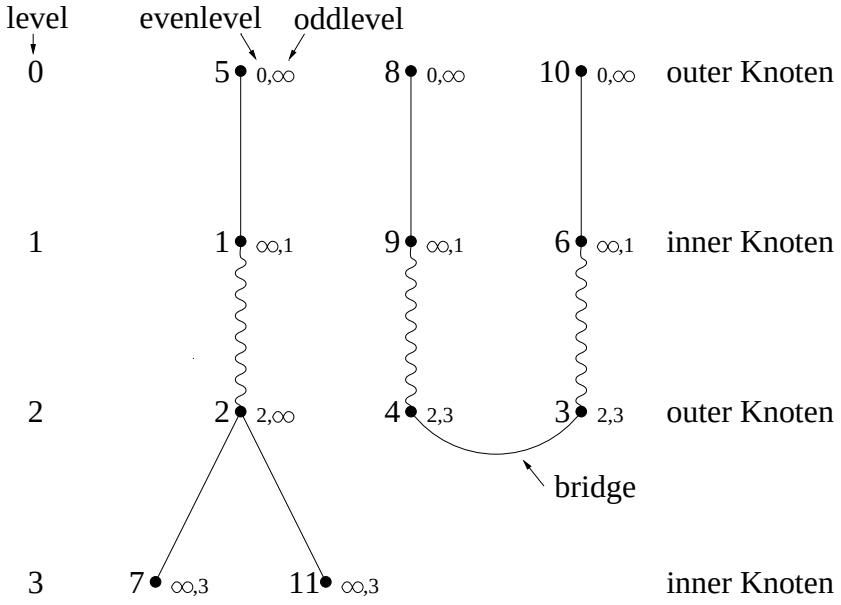


Abb. 3.24: Breitensuchbaum mit Eigenschaften der Knoten

  - Es werden alle Nachbarknoten u besucht, die mit v durch 'unmatched' und bloss-unbenutzte (siehe 3.2.5.3) Kanten verbunden sind.
  - Wenn oddlevel$(u) = \infty$, dann wird oddlevel(u) auf $i + 1$ gesetzt.

- i ist ungerade
  - Bei der Suche wird von allen Knoten v mit oddlevel$(v) = i$ ausgegangen.
  - Es wird ein Nachbarknoten u gesucht, der mit v duch eine 'matched' Kante verbunden ist.
  - Evenlevel(u) wird auf $i + 1$ gesetzt.

Beim Aufbau des Breitensuchbaums wird für jede neu hinzugekommene Kante geprüft, ob sie eine **bridge** ist. Immer wenn eine Kante in den Breitensuchbaum aufgenommen wird, die die bridge-Eigenschaft erfüllt, werden durch diese Kante zwei Äste des Baumes miteinander verbunden. Deshalb ist es möglich, daß diese bridge Teil eines augmentierenden Pfades ist. Da nicht jede bridge zu einem augmentierenden Pfad gehört, wird während der Suche im level i eine Liste bridges(i) angelget, in die alle gefundenen Kanten mit bridge-Eigenschaft eingetragen werden.

Nachdem die Suche an allen Knoten dieses Levels beendet ist, wird für jede gefundene bridge die Routine BLOSS-AUG (3.2.5.3) aufgerufen. BLOSS-AUG überprüft dann für jede einzelne Kante $e \in bridges(i)$, ob sie Teil eines augmentierenden Pfades ist und leitet gegebenenfalls eine Sonderbehandlung dieses Pfades ein.

Wenn bei der Abarbeitung der Menge bridges(i) durch BLOSS-AUG kein augmentierender Pfad gefunden wurde, so wird die Suche von SEARCH ausgehend vom level i fortgesetzt.

Wird jedoch mindestens ein augmentierender Pfad gefunden, dann bildet dieser zusammen mit allen weiteren in diesem level gefundenen Pfaden die maximale Menge kürzester augmentierender Pfade bezüglich des Ausgangsmatchings und die Phase wird durch Abbruch der inneren Schleife beendet. Jeder augmentierende Pfad, der bei Fortsetzung der Breitensuche durch SEARCH in einem höheren level $j >$ level i gefunden würde, ist kein kürzester Pfad mehr und gehört deshalb nicht zu der gesuchten Menge. Das liegt daran, daß ein im level i gefundener Pfad die Länge $2i + 1$ hat und wegen $j > i$ sind Pfade der Länge $2j + 1$ keine kürzesten augmentierenden Pfade.

Es gibt zwei Fälle, in denen eine Phase beendet wird. Im ersten Fall, der oben beschrieben ist, wird die Ausführung abgebrochen, wenn in dem aktuellen level augmentierende Pfade gefunden wurden. Der zweite Fall tritt ein, wenn die Suche einen level erreicht hat, in dem keine Kanten mehr erreichbar sind, die in den Suchbaum aufgenommen werden können. Der Suchbaum kann ausgehend von inner Knoten nur mit 'matched' Kanten und ausgehend von outer Knoten nur mit 'unmatched' Kanten erweitert werden.

Wird in einer Phase kein augmentierender Pfad mehr gefunden, so war das Matching schon zu Beginn der Phase maximal und der Algorithmus wird abgebrochen. Zum Abbruch des Algorithmus wird die innere und die äußere Schleife beendet.

3.2.5.3 BLOSS-AUG

Die Routine BLOSS-AUG wird von der Routine SEARCH am Ende jedes Levels für jede gefundene bridge aufgerufen. Um die in BLOSS-AUG enthaltene Erkennung der augmentierenden Pfade durchführen zu können, werden von allen Knoten außer oddlevel und evenlevel noch andere Eigenschaften benötigt. Deshalb wird für jeden Knoten die Menge predecessors(u) verwaltet.

Definitionen

predecessors(u) Ist die Menge aller direkten Vorgänger v des Knotens u.

ancestors(v) Ist die Menge aller Vorgänger des Knotens v.

Beim Aufruf von BLOSS-AUG wird eine bridge (u, v) als Parameter übergeben. BLOSS-AUG prüft dann, ob diese bridge Teil eines augmentierenden Pfades ist und ruft für einen gefundenen Pfad FINDPATH (siehe Abschnitt 3.2.5.4) auf. Ist (u, v) nicht Teil eines augmentierenden Pfades, so wird ein neues blossom konstruiert.

Was ist ein blossom?

Während des Aufbaus eines Breitensuchbaums von SEARCH kann es vorkommen, daß sich im level i ein Ast in mehrere Zweige aufteilt. Wenn sich dann in

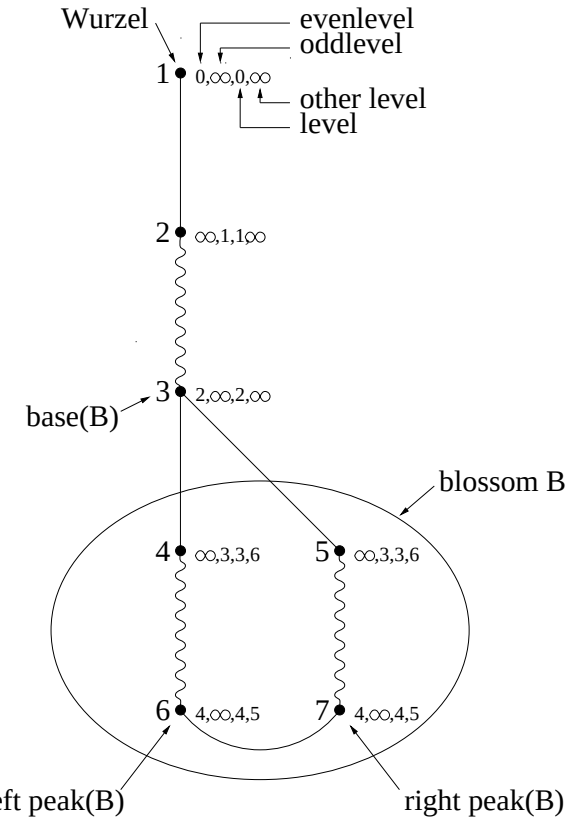


Abb. 3.25: Ausschnitt aus einem Breitensuchbaum

einem höheren level zwei der Zweige wieder treffen wird die Kante, die die Zweige verbindet, als bridge erkannt. Der Aufruf von BLOSS-AUG zu dieser bridge liefert aber keinen augmentierenden Pfad, da beide Zweige den gleichen freien Knoten als Ursprung haben. Damit die Laufzeit des MM-Algorithmus linear in der Anzahl der Kanten bleibt, wird aus den Knoten zwischen der Gabelung und der bridge ein blossom gebildet. Ein blossom ist eine Menge von Knoten, von denen einer als left peak(B) und einer als right peak(B) gekennzeichnet ist. Der Knoten an der Gabelung wird als base(B) bezeichnet und ist dem blossom B zugeordnet.

In Abbildung 3.25 ist ein Ausschnitt aus einem Breitensuchbaum dargestellt. Die Knoten 4,5,6 und 7, die sich innerhalb des eingezeichneten Kreises befinden bilden das neu entstandene blossom B. Der Knoten 3 wird als **base**(B) bezeichnet und die Knoten 6 und 7 sind die **peaks** des blossoms B.

Konstruktionsbedingungen eines blossoms

1. $\exists z : z \in \text{ancestors}(u) \wedge z \in \text{ancestors}(v)$
2. u und v haben keinen anderen Vorgänger mit dem gleichen Level wie z

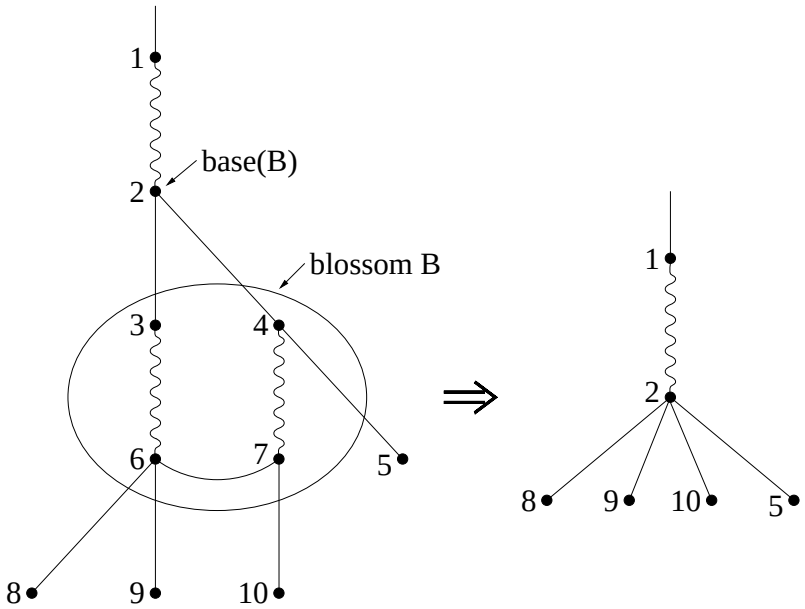


Abb. 3.26: Schrumpfung eines blossoms

Die Knoten u und v sind die Knoten der bridge (u, v), für die die Konstruktionsbedingungen zu prüfen ist.

Die Konstruktionsbedingungen sind erfüllt, wenn die beiden Knoten einer bridge im level i in einem level $j <$ level i genau einen gemeinsamen Vorgängerknoten haben.

Wurde festgestellt, daß für eine bridge (u, v) die Konstruktionsbedingungen gelten, so wird ein neues blossom B erstellt. Das blossom B besteht aus allen Knoten w, deren other level ∞ ist und mit dem Pfad, von w über die bridge (u, v) zu einem freien Knoten, auf einen endlichen Wert gesetzt werden kann (siehe Abbildung 3.25).

Blossoms werden gebildet, um die Laufzeit des MM-Algorithmus zu verbessern. Die Laufzeitverbesserung wird erreicht, da die blossoms nach ihrer Bildung für die weitere Suche in der aktuellen Phase als zusammengeschrumpft betrachtet werden. Das bedeutet, daß die Suche, wenn sie an einem Knoten des blossoms B angekommen ist, direkt an dem Knoten base(B) fortgesetzt wird. Dadurch wird verhindert, daß die Kanten innerhalb eines blossoms bei der Suche nach augmentierenden Pfaden mehrmals durchlaufen werden, so daß die Laufzeit nicht mehr linear in der Anzahl der Kanten wäre. Der Schrumpfungseffekt ist in Abbildung 3.26 zu sehen.

Algorithmus zur Erkennung eines blossoms

In diesem und dem nächsten Abschnitt wird von **Tiefensuche** gesprochen, deshalb wird diese zunächst kurz eingeführt. Bei der Tiefensuche geht, im Gegensatz

zu der in SEARCH benutzten Breitensuche, der Suchlauf zuerst in die Tiefe und erst dann in die Breite des Graphen. Ist die Suche an einem Knoten x angekommen, so wird zunächst zu einem der noch nicht besuchten Nachbarknoten gegangen und an diesem die Suche rekursiv fortgesetzt. Erst wenn die Suche an einem Knoten z angekommen ist, dessen Nachbarknoten alle schon besucht wurden, wird mit Backtracking zurück gegangen bis zu einem Knoten, dessen Nachbarknoten noch nicht alle besucht wurden.

Der Nachbarknoten von x, an dem die Suche fortgesetzt wird, kann zufällig gewählt werden. Es muß nur gewährleistet sein, daß keiner der Nachbarknoten mehrmals ausgewählt wird und daß das Backtracking erst zum Vaterknoten von x zurück geht, wenn alle Nachbarknoten von x besucht wurden.

Um entscheiden zu können, ob eine bridge (u, v) Teil eines augmentierenden Pfades oder eines blossoms ist, wird eine **doppelte Tiefensuche** durchgeführt. Bei der doppelten Tiefensuche werden parallel zwei Tiefensuchbäume mit den Knoten u und v als Wurzeln aufgebaut. Der Baum mit Wurzel u heißt T_l und der Baum mit Wurzel v heißt T_r. Ist die Suche bei den Knoten w_l und w_r angekommen, dann wird sie bei w_l fortgesetzt, wenn $\text{level}(w_l) \geq \text{level}(w_r)$ und sonst bei w_r.

Bei der Suche werden nur die predecessors eines Knoten berücksichtigt und jeder besuchte Knoten wird als bloss-benutzt markiert, damit er nicht mehrmals aufgesucht wird. Zusätzlich erhält noch jeder in T_l aufgenommene Knoten eine left-Markierung und jeder in T_r aufgenommene eine right-Markierung.

Wenn in T_l und T_r verschiedene freie Knoten gefunden wurden, dann gehört die bridge (u, v) zu einem augmentierenden Pfad und die Routine FINDPATH wird mit den gefundenen freien Knoten als Parameter aufgerufen. Treffen sich die beiden Suchbäume jedoch bei einem Knoten w, so muß eine Sonderbehandlung eingeleitet werden, da w wegen der Suche nach einem augmentierenden Pfad nur zu einem der Suchbäume gehören darf.

In der Sonderbehandlung wird entschieden, ob der Knoten w zu T_l oder zu T_r gehören soll. Zunächst erhält w eine left-Markierung und T_r versucht mit Backtracking einen Knoten $z \neq w$ mit gleichem level wie w zu finden. Bei erfolgreicher Suche mit Backtracking wird die doppelte Tiefensuche an den Knoten z in T_r und w in T_l fortgesetzt.

War es T_r nicht möglich einen Knoten z zu finden, so wird w right-markiert und T_l startet mit Backtracking die Suche nach einem Knoten z mit $z \neq w$. Wurde von T_l ein Knoten z gefunden, dann wird die doppelte Tiefensuche an den Knoten w in T_r und z in T_l fortgesetzt. Wenn in keinem der beiden Teilbäume ein Knoten mit gleichem level wie $\text{level}(w)$ aufzufinden ist, wird ein neues blossom B gebildet, da für die bridge (u, v) die Konstruktionsbedingungen gelten. Zur $\text{base}(B)$ wird dabei der Knoten w und die Knoten u und v der bridge sind dann left peak und right peak des blossoms B.

Das von T_l und T_r durchgeführte Backtracking, kann einen negativen Einfluß auf die Laufzeit haben. Deshalb werden noch die zwei Variablen **DCV** (**d**eepest **c**ommon **v**ertex) und **barrier** eingeführt, um das Backtracking zu beschränken.

Die Variable DCV zeigt immer auf den tiefsten von beiden Teilbäumen gefundenen Knoten. Solange noch kein Knoten sowohl von T_l als auch von T_r gefunden

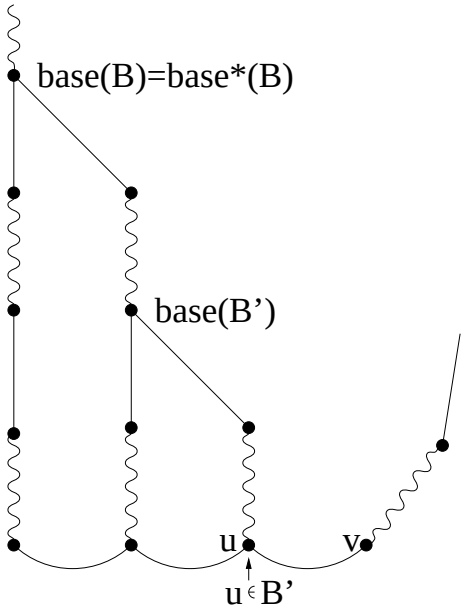


Abb. 3.27: geschachtelte blossoms

wurde ist DCV undefiniert. Mit der Variablen barrier wird das Backtracking in T_r begrenzt.

Angenommen T_l und T_r treffen sich an dem Knoten w und das Backtracking von T_r bleibt erfolglos, aber T_l findet einen Knoten mit gleichem level wie w. Bei einem erneuten Treffen der beiden Suchbäume in einem höheren level sollte T_r beim Backtracking nicht weiter als bis zum Knoten w zurückgehen. Deshalb wird bei jedem erfolglosen Backtracking von T_r barrier auf DCV gesetzt und dann nicht mehr weiter als zu barrier zurückgegangen.

Wie oben beschrieben wird ein blossom auf die base(B) zusammengeschrumpft, damit die Kanten des blossoms bei der Suche nicht mehrmals durchlaufen werden. Bei der bisherigen Betrachtung wurde immer nur ein einzelnes blossom betrachtet und deshalb stellt sich die Frage, wie geschachtelte blossoms behandelt werden.

Wenn blossoms wie in Abbildung 3.27 geschachtelt vorkommen, dann wird die Suche an der base mit dem geringsten level fortgesetzt. Dieser Knoten heißt dann base*(B).

3.2.5.4 FINDPATH

Findpath wird mit zwei Knoten als Parameter aufgerufen und berechnet den Weg zwischen diesen Knoten. Als weiteren Parameter kann noch ein Blossom B übergeben werden, wenn ein Weg innerhalb des Blossoms B gesucht wird. Die beiden übergebenen Knoten heißen high und low und dabei muß gelten, level(high) $\geq$ level(low). Das Ergebnis der Routine FINDPATH ist die genaue

Pfadbeschreibung des alternierenden Pfades zwischen den als Parameter übergebenen Knoten.

Findpath durchsucht den durch SEARCH und BLOSS-AUG erstellten Breitensuchbaum mit Tiefensuche, ausgehend von high. Diese Tiefensuche hat folgende Eigenschaften:

- Wurden als Parameter zwei Knoten und kein Blossom übergeben, so wird ein alternierender Pfad zwischen den zwei Knoten gesucht.
- Wurden als Parameter zwei Knoten und ein Blossom B übergeben, dann wird innerhalb des Blossoms ein alternierender Pfad zwischen den angegebenen Knoten gesucht. Alle Knoten u, die zusammen mit einem Blossom B als Parameter übergeben wurden, sind entweder base(B) oder $u \in B$. Wechselt der Ausgangspunkt der Suche von v zu einem Knoten $u \in$ predecessors(v), so wird v zum Vaterknoten von u. Ein Aufruf von FINDPATH mit einem Blossom als Parameter erfolgt immer von OPEN (siehe Abschnitt 3.2.5.4).
- Bei der Suche werden alle Blossoms außer dem als Parameter übergebenen als geschrumpft angesehen. Wenn die Suche an einem Knoten w ankommt mit $w \in B'$, so wird sie direkt bei base(B') $= b$ fortgesetzt und w wird zum Vaterknoten von b.
- In dem Blossom B wird die Suche nur mit Knoten fortgesetzt, deren 'left/right-Markierung' gleich der von high und deren level kleiner als level(low) ist.

Wenn die Suche beim Knoten low angekommen ist, dann konstruiert FINDPATH den allgemeinen Pfad von high zu low, durch Zurückgehen entlang der Vaterknotenkette von low zu high. Der entstandene Pfad ist ein allgemeiner Pfad, da er Knoten enthalten kann, die zu einem Blossom gehören. Die Suche wurde bei diesen Knoten direkt an der base des blossoms fortgesetzt und deshalb werden diese blossoms geöffnet, um zu entschieden, auf welchem Pfad sie durchlaufen werden. Erst wenn von allen blossoms der richtige Pfad bekannt ist, kann gewährleistet werden, daß der Pfad von low zu high ein alternierender Pfad ist. Das Öffnen der blossoms wird mit der Routine OPEN durchgeführt.

OPEN

OPEN öffnet ein als Parameter übergebenes blossom und sucht mit Hilfe eines Aufrufs von FINDPATH den Pfad durch das blossom. Als Parameter wird außer dem blossom B' noch der im allgemeinen Pfad enthaltene Knoten $w \in B'$ übergeben. Wenn w ein outer Knoten ist, dann wird FINDPATH mit den Parametern w, base(B') und B' aufgerufen. Ist w ein inner Knoten, dann wird FINDPATH zweimal aufgerufen, einmal mit den Parametern left peak(B'), w, B' und das zweite mal mit right peak(B'), base(B') und B'. Aus dem Ergebnis der Aufrufe von FINDPATH wird dann der richtige Pfad zusammengesetzt und als Ergebnis von OPEN zurückgegeben.

3.2.5.5 TOPOLOGICAL ERASE

Wenn von FINDPATH ein kürzester augmentierender Pfad gefunden wurde, dann wird das Matching um diesen Pfad erweitert. Alle Knoten und Kanten des Pfades werden dann von TOPOLOGICAL ERASE markiert, damit sie bei der weiteren Suche in dieser Phase nicht mehr gefunden werden können. Dadurch wird erreicht, daß alle Pfade, die in einer Phase gefunden werden, disjunkt sind.

3.2.5.6 Laufzeitbetrachtung zur inneren Schleife

Zum Aufbau des Breitensuchbaums wird jede Kante höchstens einmal durchlaufen. Dazu benötigt die Routine SEARCH höchstens $|E|$ Operationen. Die Erkennung von blossoms durch die Routine BLOSS-AUG und deren Verwaltung kann durch die doppelte Tiefensuche und den Einsatz von barrier und DCV auf höchstens $|V| + |E|$ weitere Operationen beschränkt werden. Die Routine FINDPATH durchläuft jede Kante höchstens zwei mal und benötigt deshalb maximal $2|E|$ weitere Operationen. Unter Berücksichtigung des Verwaltungsaufwands der durch TOPOLOGICAL ERASE gelöschten Kanten von höchstens $|V|$ Operationen, ergibt sich ein Gesamtaufwand pro Phase von maximal $(6|E| + 2|V|) \in O(|E|)$.

Durch Multiplikation mit dem in Abschnitt 3.2.4 erwähnten Aufwand der äußeren Schleife kann der Gesamtaufwand des MM-Algorithmus bestimmt werden. Dieser liegt in $O(\sqrt{|V|}|E|)$ und kann wegen $|E| \leq |V|^2$ mit $O(|V|^{2,5})$ abgeschätzt werden.

3.3 Kürzeste Wege in Planaren Graphen

3.3.1 Übersicht

In diesem Abschnitt wird ein Verfahren betrachtet, mit dem man die kürzesten Wege in einem planaren Graphen ausgehend von einer Quelle berechnen kann.

Nach einer Einführung in die Problemstellung wird der bekannte Algorithmus von Dijkstra vorgestellt. Der Algorithmus von Frederickson verringert die Laufzeit gegenüber dem Algorithmus von Dijkstra von $O(n \log n)$ auf $O(n\sqrt{\log n})$, indem er ein divide and conquer-Verfahren verwendet. Dazu werden einige Grundlagen erläutert; ein Separator Theorem, eine Technik, Graphen in Regionen zu unterteilen und eine Topologiebaumtechnik. Anschließend werden der verbesserte Algorithmus und seine Laufzeit betrachtet.

3.3.2 Einleitung

In Graphen interessiert man sich oft für kürzeste Wege zwischen zwei oder mehr Knoten. Dabei gibt es grundsätzlich zwei Problemstellungen: das single source shortest path- (SSSP) und das all pairs-Problem. Bei dem SSSP-Problem sucht man ausgehend von einem Quellknoten s alle kürzesten Wege zu den von s erreichbaren Knoten. Bei dem all pairs-Problem sucht man die kürzesten Wege zwischen allen miteinander verbundenen Knoten in einem Graphen. Die Graphen können jeweils gerichtet oder ungerichtet sein.

Üblicherweise wird zur Lösung des SSSP-Problems das Verfahren von Dijkstra verwendet, das auf planaren Graphen bei Verwendung eines heap einen Aufwand von $O(n \log n)$ hat.

Der Algorithmus von Frederickson [6] hat für das gleiche Problem bei planaren Graphen nur einen Aufwand von $O(n\sqrt{\log n})$. Dies wird durch ein divide and conquer-Verfahren erreicht, bei dem der Graph in Regionen unterteilt wird. Innerhalb der Regionen wird das Verfahren von Dijkstra für die Ermittlung der kürzesten Wege benutzt. Mit Hilfe einer Topologiebaumtechnik werden die kürzesten Wege innerhalb der Regionen auf den ganzen Graphen übertragen.

Im folgenden beschränke ich mich auf ungerichtete planare Graphen. Ich benutze n für die Anzahl der Knoten ($|V|$) und m für die Anzahl der Kanten ($|E|$), wenn nicht explizit etwas anderes vereinbart wird.

3.3.3 Planare Graphen

Ein planarer Graph ist ein Graph $G = (V, E)$, für den eine Abbildung in den zweidimensionalen Raum existiert, so daß sich keine Kanten schneiden. Ein bekannter nichtplanarer Graph ist ein K_5, der vollständig verbundene Graph mit fünf Knoten (siehe Abbildung 3.28).

Planare Graphen besitzen Eigenschaften, die man nutzen kann, um effizientere Algorithmen auf ihnen zu formulieren als auf allgemeinen Graphen.

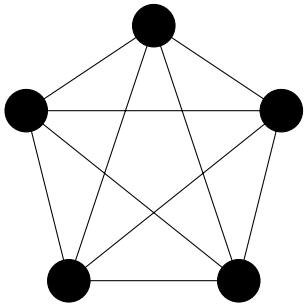


Abb. 3.28: Beispiel eines nichtplanaren Graphen: der K_5

1. Die Anzahl der Kanten steigt linear mit der Anzahl der Knoten
   $m \leq 3n \Leftrightarrow 6$
2. Man kann einen planaren Graphen G in einen planaren Graphen G' überführen, so daß jeder Knoten maximal den Grad drei hat und
   $|V'| = n'$ mit $n' \leq 6n \Leftrightarrow 12$

Konstruktionsverfahren "transform" für einen ungerichteten planaren Graphen mit den Eigenschaften aus Punkt 2		
INPUT:	$G = (V, E)$	der ursprüngliche Graph
OUTPUT:	$G' = (V', E')$	ein Graph bei dem gilt: $\forall v \in V' : deg(v) \leq 3$

$transform(G) \equiv$
  $V' \leftarrow V;\ E' \leftarrow E;$
  foreach $v \in V$ **do**
    if $deg(v) > 3$ **then**
      $V' := (V' \Leftrightarrow \{v\}) \cup \{v_1, .., v_n\},\ n = deg(v);$
      foreach $w_i \in V,\ (v, w_i) \in E$ **do**
        $E' := (E' \Leftrightarrow \{(v, w_i)\}) \cup \{(v_i, w_i), (v_i, v_{i+1})\};$
      od;
    fi;
  od
  return $\leftarrow\ G' = (V', E')$

3.3.4 Single Source Shortest Path-Algorithmus

Die Algorithmen für das Finden der kürzesten Entfernungen von einer Quelle s zu allen anderen Knoten in einem Graphen G nennt man single source shortest path-Algorithmen. Zumeist wird der Algorithmus von Dijkstra verwendet,

der auf planaren Graphen einen Aufwand von $O(n \log n)$ hat. Dabei tragen die Kanten ein Gewicht, das einer Entfernung entspricht. Es kann sich dabei um eine echte Entfernung, den Zeitaufwand oder kompliziertere Werte handeln, die verschiedene Widerstände zu Kosten verrechnen. Die kürzeste Entfernung zwischen den Knoten u und v ist also die Länge des Pfades von u nach v , dessen Kantengewichte sich zu der geringsten Entfernung summieren.

3.3.4.1 Dijkstra

Der Algorithmus von Dijkstra verwaltet drei Mengen von Knoten: bekannte Knoten, Randknoten und unbekannte Knoten. Die bekannten Knoten B sind die Knoten, zu denen die kürzeste Entfernung bereits bekannt ist. Die Randknoten sind die Knoten in der Umgebung der bekannten Knoten $R = U(B) \Leftrightarrow B$, wobei die Umgebung $U(v) = \{w \in V | (v, w) \in E\}$ ist. Die unbekannten Knoten sind die restlichen Knoten. Schritt für Schritt wird ausgehend von der Quelle s die Menge der bekannten Knoten um einen Randknoten v erweitert, für den gilt: die Entfernung von s zu v ist minimal. Die Entfernung zwischen zwei Knoten wird durch ein Gewicht an der Kante zwischen diesen Knoten angegeben $\omega(v, w)$. Alle Knoten $w \in U(v) \Leftrightarrow B$ werden zu Randknoten. Der Algorithmus terminiert, wenn die Menge der Randknoten leer ist.

Algorithmus von "Dijkstra": Die Mengen der bekannten Knoten B und der Randknoten R werden Schritt für Schritt erweitert, bis die kürzesten Entfernungen zu allen Knoten bekannt sind, die von s aus erreichbar sind. Dazu wird die kürzeste Entfernung $\rho(v)$ nach jedem Durchgang auf den neuesten Stand gebracht.		
INPUT:	$G = (V, E, \omega)$	kantengewichteter, ungerichteter und zusammenhängender Graph, wobei V =Menge der Knoten, E =Menge der Kanten und $\omega : E \to \mathbb{R}$ ordnet jeder Kante (v, w) die Entfernung zwischen v und w zu.
	s	die Quelle
OUTPUT:	$\forall v \in V : \rho(v)$	$\rho(v)$ =minimale Entfernung von s nach v

$Dijkstra(G, s) \equiv$
  $B \leftarrow s;\ R \leftarrow U(s);\ \rho(s) \leftarrow 0$
  $\forall v \in V,\ v \neq s :\ \rho(v) \leftarrow \infty;$
  $\forall v \in R :\ \rho(v) \leftarrow \omega(s, v);$
  while $R \neq \emptyset$ **do**
    $suche\ v \in R\ mit\ \rho(v) = min\{\rho(w) | w \in R\};$
    $B := B \cup \{v\};$
    $R := R \cup U(v) \Leftrightarrow B;$
    $\forall w \in R :\ \rho(w) := min\{\rho(w), \rho(v) + \omega(v, w)\}$
  od
  return $\leftarrow\ \forall v \in V : \rho(v)$

Hieraus ergibt sich ein Aufwand von $O(n^2)$, den man durch Verwenden eines Fibonnacci-heap auf $O(m + n \log n)$ reduzieren kann. Bei planaren Graphen ergibt sich durch die Linearität der Kanten bei Benutzung eines heap $O(n \log n)$.

3.3.4.2 Frederickson

Der Algorithmus von Frederickson verbessert die Laufzeit des Algorithmus von Dijkstra, indem er ein divide and conquer-Verfahren anwendet. Der Graph wird in mehrere überlappende Regionen aufgespaltet. Man erreicht Knoten innerhalb einer Region nur über solche Überlappungsknoten (*äußere* Knoten). Kennt man kürzeste Wege von einer Quelle s zu diesen *äußeren* Knoten, kann man leicht die kürzesten Wege zu den noch verbleibenden Knoten herausfinden.

3.3.5 Vorbereitungen zum Algorithmus von Frederickson

Der Algorithmus läßt sich grob in zwei Phasen unterteilen. Das preprocessing und die search phase. Im preprocessing wird der Graph $G = (V, E, \omega)$ in Regionen unterteilt und das all pairs-Problem für die *äußeren* Knoten dieser Regionen berechnet. In der search phase berechnet man nun die kürzesten Wege von der Quelle s zu den *äußeren* Knoten (main phase) und anschließend die restlichen kürzesten Wege ausgehend von der Quelle und den äußeren Knoten (mop-up).

Für das Aufspalten des Graphen benötigt man ein Separator Theorem und geeignete Regionen. Für das Ermitteln kürzester Wege verwendet Frederickson den Algorithmus von Dijkstra. Die Zusammenführung der Teilergebnisse erfolgt mit Hilfe topologiebasierter Bäume (Topologiebäume). Im weiteren werden die Techniken vorgestellt, die die Grundlage für den Algorithmus von Frederickson bilden.

3.3.5.1 Das Separator Theorem

Ein $f(n)$-Separator Theorem, $n \in \mathbb{N}$, für eine Graphenklasse S ist ein Theorem der folgenden Form:
$\exists$ Konstanten $\alpha < 1, \beta > 0$, für die die Knotenmenge eines Graphen $G = (V, E)$ aus S in drei disjunkte Mengen A, B, C partitioniert werden kann, so daß gilt:

- $\neg\exists(a, b) \in E \; mit \; a \in A \; und \; b \in B$
- $|A| \leq \alpha n, \; |B| \leq \alpha n, \; |C| \leq \beta f(n)$

Man erhält dadurch also zwei durch C voneinander getrennte Teilgraphen. Dies ist in vielen Fällen sehr hilfreich, weil man dadurch Bereiche einengen kann, auf denen Algorithmen laufen, oder für die man etwas beweisen möchte. Der Algorithmus von Frederickson benutzt das folgende Separator Theorem, um einen Graphen in Regionen zu unterteilen.

Im $\sqrt{n}$-Separator Theorem von Lipton und Tarjan[13] "separate" ergeben sich $\alpha = \frac{2}{3}$, $\beta = 2\sqrt{2}$. Der Algorithmus hat lineare Laufzeit. Er wird in dem Artikel von Lipton und Tarjan[13] genauer beschrieben.		
INPUT:	G=(V,E)	Graph mit Knoten- und Kantenmenge
OUTPUT:	A,B,C	die drei Mengen, wobei $\|A\| \leq \frac{2}{3}n$, $\|B\| \leq \frac{2}{3}n$ und $\|C\| \leq 2\sqrt{2}\sqrt{n}$

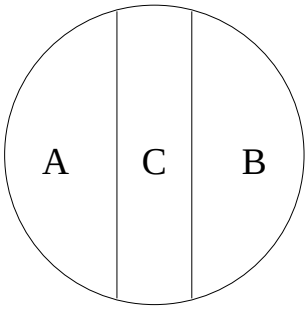


Abb. 3.29: Ergebnis des Separator Theorems

3.3.5.2 Ein Divide And Conquer-Verfahren für SSSP

Um den Aufwand, den der Algorithmus von Dijkstra braucht, zu verringern, nimmt man ein divide and conquer-Verfahren. Der Graph wird in eine Menge

von Regionen $\{R_1, ..., R_l\}$ aufgespalten. Dies geschieht durch ein relativ einfaches Verfahren.

Regionen

Man kann einen Graphen $G = (V, E)$ in mehrere Regionen $\{R_1, ..., R_l\}$ aufspalten, die Knoten enthalten. Die Regionen können sich überlappen, so daß Knoten in mehreren Regionen enthalten sind. *Äußere* Knoten sind Knoten v, für die gilt:

$$\exists i, j \in \{1, ..., l\}, i \neq j : v \in R_i \wedge v \in R_j.$$

Innere Knoten sind Knoten w, für die gilt:

$$\forall i, j \in \{1, ..., l\} : (w \in R_i \wedge w \in R_j) \Rightarrow i = j.$$

Im Beispiel in der Abbildung 3.30 ist $\{v_1, v_2, v_3, v_4\}$ die Menge der *äußeren* Knoten.

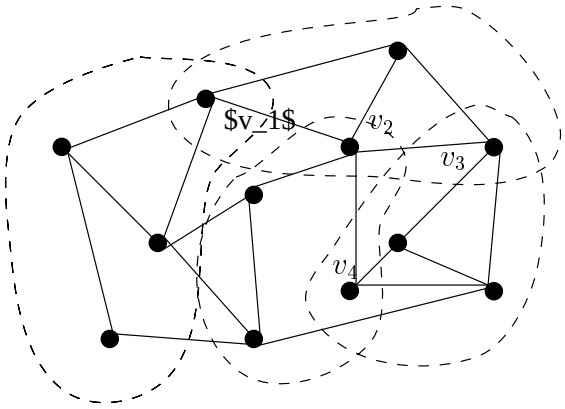


Abb. 3.30: Aufteilung eines Graphen in Regionen

Eine r-Division

Eine r-Division ist die Aufspaltung eines Graphen $G = (V, E)$ in l Regionen $\{R_1, ..., R_l\}$, so daß gilt:

- $\bigcup_{i=1}^{l} R_i = V$
- $\forall i \in \{1, ..., l\} : |R_i| \leq r$

Eine geeignete r-Division

Eine geeignete r-Division ist eine Aufspaltung des Graphen G in l Regionen $\{R_1, ..., R_l\}$, so daß gilt:

- $\{R_1, ..., R_l\}$ ist eine r-Division von G
- es gibt $O(\frac{n}{\sqrt{r}})$ *äußere* Knoten
- $l \leq \Theta(\frac{n}{r})$

Eine solche geeignete r-Division kann in $O(n \log n)$ gefunden werden[6].

Ein Verfahren zur Ermittlung einer geeigneten r-Division

Das Verfahren für eine geeignete r-Division "rDivision" transformiert den Graphen $G = (V, E)$ in einen Graphen $G' = (V', E')$ (siehe Kapitel 3.3.3). Dazu wird rekursiv das Separator Theorem auf Knotenmengen angewandt, die zu groß sind. Am Ende werden noch einige Mengen zusammengefaßt.		
INPUT:	$G = (V, E)$	der Ausgangsgraph
	r	die Größe für die r-Division
OUTPUT:	M	eine Menge von Knotenmengen, wobei jede Knotenmenge einer Region entspricht

$rDivision(G, r) \equiv$
  $G' \leftarrow transform(G)$;
  $M \leftarrow \{V'\}$;
  while $\exists R \in M,\ mit\ |R| \geq r$ **do**
    $(A, B, C) := separate(R)$;
    $C' := \{v \in C | \neg\exists w \in (A \cup B) : (v, w) \in E'\}$;
    $C'' := C \Leftrightarrow C'$;
    Suche Zusammenhangskomponenten $A_1, A_2, ..., A_q$ *in*
      $(A \cup B \cup C')$;
    Ist ein Knoten v *in* C'' *nur mit Knoten aus* A_i *verbunden :*
      $C'' := C'' \Leftrightarrow \{v\};\ A_i := A_i \cup \{v\}$;
    $M := (M \Leftrightarrow R) \cup \{A_1, ..., A_q\}$;
  od;
  Vereinige alle Paare von Mengen $R_i, R_j \in M,\ für\ die\ gilt :$
    $\exists v \in (R_i \cap R_j) \wedge |R_i| \leq \frac{r}{2} \wedge |R_j| \leq \frac{r}{2}$
  Vereinige alle Paare von Mengen $R_i, R_j \in M,\ für\ die\ gilt :$
    $|R_i| \leq \frac{r}{2},\ |R_j| \leq \frac{r}{2}\ \wedge$ *(beide enthalten eine oder mehrere*
    Kanten in die Menge $R_k\ \vee$ *beide enthalten eine oder*
    mehrere Kanten in die Mengen $R_k, R_l,\ wobei\ k, l \neq i, j)$
  $\forall v \in C'',\ für\ die\ gilt : \exists w \in R, (v, w) \in E :\ R := R \cup \{v\}$
  return $\leftarrow M$

Äquivalenzklassen *äußerer* Knoten

Wenn *äußere* Knoten in den gleichen Schnittmengen von Regionen liegen, kann man sie zu Äquivalenzklassen zusammenfassen. Nummeriert man die Regionen des Graphen, lassen sich die Knoten in den Äquivalenzklassen lexikographisch anordnen. Das wird für das Verfahren verwendet, das mittels eines Topologiebaumes kürzeste Wege propagiert.

3.3.5.3 Topologiebäume

Um die Strukturierung eines Graphen $G = (V, E, \omega)$ in Regionen für die kürzeste Wegesuche ausnutzen zu können, verwendet der Algorithmus von Fredericksson einen Topologiebaum. Dies ist ein ausgeglichener Binärbaum, der die *äußeren* Knoten der Regionen in lexikographischer Ordnung der Regionennummern enthält. Im Beispiel aus Abbildung 3.31, kann man die *äußeren* Knoten folgendermaßen anordnen : $\{v_5\}, \{v_1, v_2, v_3, v_4\}, \{v_8, v_9\}, \{v_6, v_7\}$.

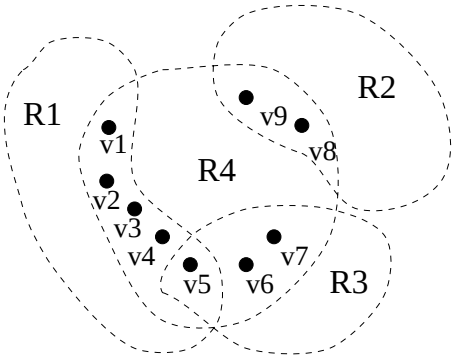


Abb. 3.31: Regionen mit äußeren Knoten

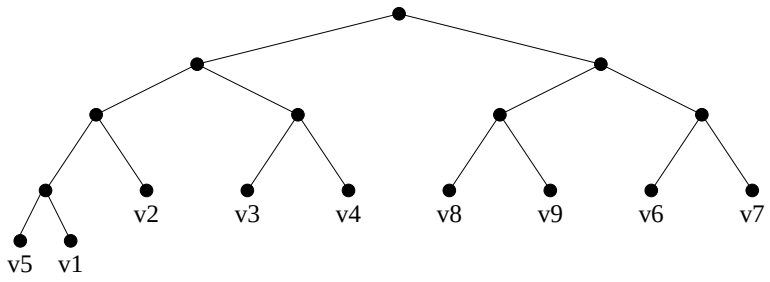


Abb. 3.32: Topologiebaum

Hieraus ergibt sich dann der in Abbildung 3.32 dargestellte ausgeglichene Binärbaum. Die inneren Knoten des Baumes werden mit der minimalen kürzesten Entfernung zwischen den darunter liegenden Blättern und der Quelle s markiert.

An einem Beispiel wird jetzt gezeigt, wie man innerhalb einer Region mit der Topologiebaumtechnik das all pairs-Problem lösen kann. Gegeben sei ein ungerichteter Graph $G = (\{a, b, c, d\}, \{(a, b), (a, c), (a, d), (b, c), (c, d)\}, \omega)$ mit Kantengewichten wie in Abbildung 3.33.

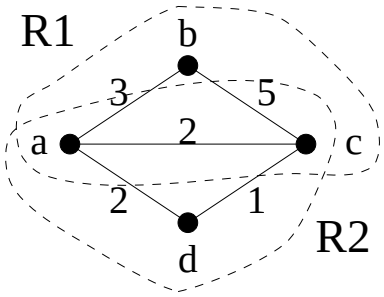


Abb. 3.33: Beispielgraph mit Kantengewichten

Führt man nun innerhalb der Regionen $R1$ und $R2$ für jeden *äußeren* Knoten den SSSP-Algorithmus von Dijkstra durch, kann man den kürzesten Weg von a nach c in beiden Regionen bestimmen. Es kommt der Baum aus Abbildung 3.34 heraus.

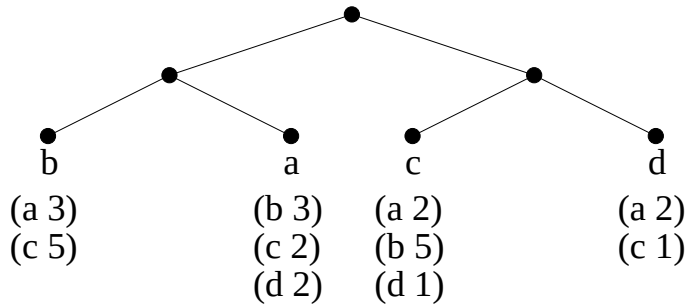


Abb. 3.34: Ergebnis nach SSSP mit Dijkstra

Nun sucht man für die restlichen Wege eine Verbindung von dem Ausgangsknoten zum Zielknoten über mindestens einen *äußeren* Knoten. In diesem Fall geht man von b aus zunächst nach a mit der Entfernung 3. Dort findet man eine Verbindung zu d mit der Entfernung 2. Der Weg zu c ist genauso lang, scheidet daher direkt aus. Jeder andere Pfad scheidet auch aus, da von a aus nur Wege mit der Entfernung 2, wie bei dem bereits gefundenen Pfad, oder mehr fortführen. Es ergibt sich also ein Baum wie in Abbildung 3.35, der alle kürzesten Wege zwischen den Knoten der Regionen R_1 und R_2 enthält.

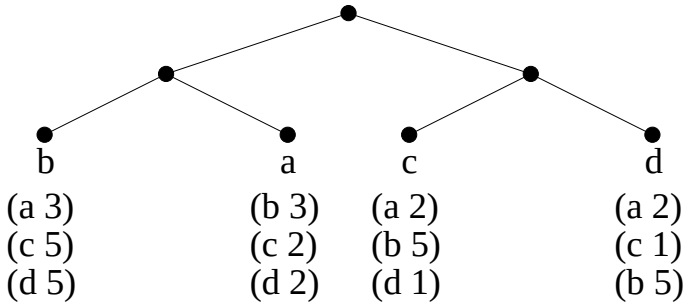


Abb. 3.35: Endergebnis nach Topologiebaumtechnik

3.3.6 Der Algorithmus von Frederickson

Dieser Algorithmus "Frederickson" benutzt das Topologiebaumverfahren "toptree"auf Regionen und den SSSP-Algorithmus von Dijkstra. Er kann grob in zwei Phasen unterteilt werden. Im *preprocessing* werden die kürzesten Wege zwischen *äußeren* Knoten ermittelt. In der *search phase* sucht man die kürzesten Wege von der Quelle s zu allen anderen Knoten.		
INPUT:	$G = (V, E, \omega)$	der Graph mit gewichteten Kanten
	s	die Quelle
OUTPUT:	$\forall v \in V : \rho(v)$	die kürzeste Entfernung von s nach v
	$\forall v \in V : p(s, v)$	der kürzeste Weg von s nach v

3.3.6.1 Das preprocessing

Der Graph wird durch eine r_1-Division in l Regionen $\{R_1, ..., R_l\}$ unterteilt (Ebene 1). Diese Regionen werden mit einer r_2-Division in $k_i, i \in \{1, .., l\}$ Regionen aufgegliedert (Ebene 2)

$$\{R_{1\,1}, ..., R_{1\,k_1}, ..., R_{l\,1}, ..., R_{l\,k_l}\}.$$

Für jeden *äußeren* Knoten der Ebenen 1 und 2 wird der Algorithmus von Dijkstra innerhalb der Regionen $\{R_{1\,1}, ..., R_{l\,k_l}\}$ durchgeführt. Mit der Topologiebaumtechnik ermittelt man die kürzesten Wege im Graphen zwischen allen *äußeren* Knoten der Ebene 1.

3.3.6.2 Die search phase

Diese Phase läßt sich in den *main thrust* und das *mop-up* unterteilen. Im *main thrust* benutzt man die Topologiebaumtechnik, um die kürzesten Wege zwischen der Quelle s und den *äußeren* Knoten der Ebene 1 zu finden. Im *mop-up* findet man nun die noch fehlenden Wege mithilfe eines modifizierten Dijkstra-Algorithmus. Hierbei führt man in den Regionen $\{R_1, ..., R_l\}$ den SSSP-Algorithmus für jeden *äußeren* Knoten der Ebene 1 durch, nachdem man diesen kurzzeitig mit einer Entfernungsmarke belegt hat, die der kürzesten Entfernung von ihm zur Quelle s entspricht.

3.3.6.3 Der Algorithmus

transform(graph) liefert einen Graphen für die r-Division (Kapitel 3.3.3)
rDivision(region,r) hat als Ergebnis eine Menge von Regionen
toptree(set-of-nodes,set-of-edges) erzeugt einen Topologiebaum für set-of-nodes
outer(set-of-regions) selektiert die *äußeren* Knoten der Regionen
dijkstra(node,region) führt einen Dijkstra-Algorithmus für node innerhalb von region aus und trägt die Ergebnisse in den Topologiebaum ein
mark(set-of-nodes) markiert die Knoten mit ihrer Entfernung zur Quelle s
sort(set-of-nodes,set-of-regions) erzeugt eine lexikographische Ordnung
es gilt: $outer(LevelOneRegions) \subset outer(LevelTwoRegions)$

$Frederickson \equiv$
preprocessing :
$G' \leftarrow transform(G),\ n := |V'|$
$LevelOneRegions \leftarrow rDivision(V', \log n)$
$LevelTwoRegions \leftarrow \emptyset,\ i := 1$
foreach $R \in LevelOneRegions$ **do**
$LevelTwoRegions := LevelTwoRegions \cup rDivision(R, \log\log n^2)$
od
$toptree(sort(outer(LevelOneRegions), LevelOneRegions), E')$
foreach $R \in LevelTwoRegions$ **do**
foreach $v \in (outer(LevelTwoRegions) \cap R$ **do**
$dijkstra(R, v)$
od
od
main thrust :
$toptree(sort(outer(LevelOneRegions) \cup s, LevelOneRegions), E')$
mop $\Leftrightarrow$up :
$mark(outer(LevelOneRegions))$
foreach $R \in LevelOneRegions$ **do**
foreach $v \in outer(LevelOneRegions) \cap R$ **do**
$dijkstra(R, v)$
od
od
return $\leftarrow\ \forall v \in V : \rho(v)\ und\ p(s, v)$

3.3.6.4 Der Aufwand

- Topologiebaumtechnik nach r-Division: $O(n + \frac{n}{\sqrt{r}} \log n)$
- Dijkstra-SSSP mit heap: $O(n \log n)$
- $r_1 = \log n$
- $r_2 = \log \log n^2$

Der Aufwand des Algorithmus ergibt sich folgendermaßen:

- Die erste rDivision wird durch einen Algorithmus von Frederickson mit Aufwand $O(n \log r_1 + \frac{n}{\sqrt{r_1}} \log n)$ ersetzt[6].
  $= O(n \log \log n + n\sqrt{\log n})$
- Die folgenden rDivision-Aufrufe benötigen $O(r_1 \log r_1) * \Theta(\frac{n}{r_1})$
  $= O(n \log \log n)$
- Dijkstra auf den LevelTwoRegions benötigt $O(r_2 \log r_2 * \frac{n}{\sqrt{r_2}})$
  $= O(n \log \log n \log \log \log n)$
- Die Topologiebaumtechnik benötigt $O(r_1 + (\frac{r_1}{\sqrt{r_2}}) \log r_1) * O(\frac{n}{\sqrt{r_1}})$
  $= O(r_1) * O(\frac{n}{\sqrt{r_1}})$
  $= O(n\sqrt{\log n})$
- Der main thrust benötigt $O(n + (\frac{n}{\sqrt{r_1}}) \log n)$
  $= O(n\sqrt{\log n})$
- Das mop-up benötigt $O(r_1 \log r_1) * \Theta(\frac{n}{r_1})$
  $= O(n \log \log n)$

Der Gesamtaufwand des Algorithmus ergibt sich aus seiner größten Komponente. Das ist in diesem Fall $O(n\sqrt{\log n})$.

3.4 Softwareengineering und Objektorientierte Entwicklung

3.4.1 Einleitung

Bei der Entwicklung großer Softwaresysteme gewinnen ingenieurtechnische Ansätze mehr und mehr an Bedeutung. Grund dafür ist die wachsende Notwendigkeit von Qualitätssicherung und Planbarkeit sowie die zunehmende Komplexität der Computersysteme. Durch Softwareengineering versucht man hier Lösungsansätze zu finden.

Dabei werden an ein Projekt folgende Anforderungen gestellt:

- Personen-, Zeit- und Finanzbedarf sollen planbar und minimal sein
- Das Endprodukt soll eine gesicherte Qualität haben
- Die Weiterentwicklung soll einfach möglich sein (Wartbarkeit)

Für den Entwicklungsprozeß von Software existieren verschiedene Modelle. Eines davon, das Wasserfallmodell, wird in Kapitel 3.4.2 näher erläutert. Besonders wichtig für Planbarkeit und Qualitätssicherung sind dabei die frühen Phasen wie Analyse und Spezifikation. Bei herkömmlichen Projekten sind diese mit dem Entwurf und der Implemtierung fest verbunden, so daß nicht unterschieden werden kann, welche Entscheidungen auf welcher Ebene getroffen wurden. Die Trennung der Phasen führt zu einer Dokumentation des Projektfortschritts.

3.4.1.1 Der Entwicklungsprozeß

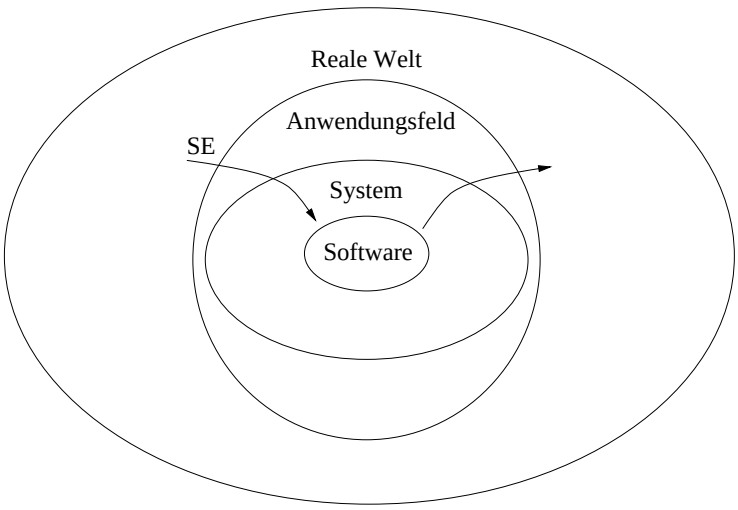


Abb. 3.36: Schichtenmodell

Ein Softwaresystem kann nicht für sich alleine entwickelt werden. Es ist eingebettet in verschiedene Schichten, die es wie Zwiebelschalen umgeben (Abb. 3.36).

Da ist zunächst das System, auf und in dem die Software läuft, also der Rechner, die Peripheriegeräte, das Netzwerk usw. Das System wird im Anwendungsfeld eingesetzt, in dem sich Personen, Arbeitsabläufe, Daten und Informationen befinden [11]. Das Anwendungsfeld schließlich ist eingebettet in die „reale Welt“, die die Software mit ihren Gegebenheiten beeinflußt (z.B. Stromausfall).

Aus diesem Schichtenmodell ergeben sich nun die verschiedenen Phasen des Entwicklungsmodells. Dabei arbeitet sich der Prozeß von außen an den Kern des Systems, die Software, heran, um diese dann wieder in die umgebenden Schichten zu integrieren. Die Analysephase findet im Anwendungsfeld statt und erforscht die Beziehungen zwischen dem System und der es betreffenden Welt. Während der Spezifikation wird das genaue Systemverhalten definiert. Im Entwurf konkretisiert man nun, auf welche Weise dieses Verhalten im Programm realisiert wird. Die Implementierung schließlich führt zum ausführbaren Programm, welches dann in Integration und Test wieder in die äußeren Schichten eingebettet wird. Dieser Verlauf wird durch den Pfeil in Abbildung 3.36 verdeutlicht.

3.4.1.2 Die Badewannenkurve

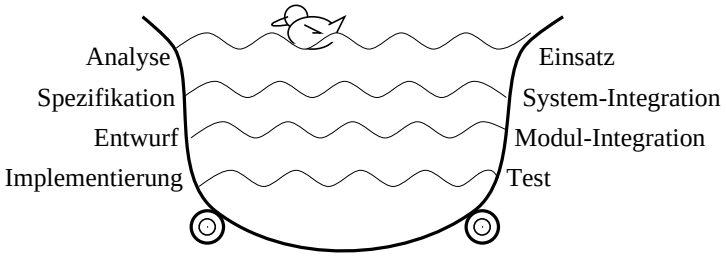


Abb. 3.37: Die Badewannenkurve

In Abbildung 3.37 ist zu erkennen, wie die frühen Phasen im Entwicklungsprozeß mit den späten zusammenhängen, da sie in der selben Schicht stattfinden. Jeweils zusammengehörige Phasen haben die Eigenschaft, daß Fehler, die in der einen auftreten, wenn nicht sofort, dann erst in der anderen entdeckt werden. Fehler der Implementierung werden im Test erkannt. Fehler beim Entwurf tauchen in dem Moment auf, wo man versucht, die einzelnen Module zu integrieren. Zur Beseitigung muß nun die Implementierungs- und Testphase erneut durchlaufen werden. Fehler bei der Spezifikation werden in der Regel erst bei der Systemintegration deutlich, wenn das System nicht mit dem Anwendungsfeld zusammenpaßt. Und grundsätzliche Fehler bei der Analyse merkt man meist erst im Einsatz. Hier müßen dann oft sämtliche Phasen erneut durchlaufen werden.

Aufgrund seiner Form nennt man dieses Modell die „Badewannenkurve“. Es verdeutlicht, daß Fehler immer teurer werden, je später sie erkannt werden. Als zweite Konsequenz erkennt man die Wichtigkeit der frühen Phasen: ihre

Ergebnisse sind nicht unmittelbar testbar wie z.B. Quelltext, ihre Fehler sind aber besonders kritisch für das Projekt. Methoden aus dem Softwareengineering wie z.B. technische Reviews (siehe Kapitel 3.4.4.4) können hier helfen, diese Fehler frühzeitig zu entdecken [14].

3.4.2 Das Wasserfallmodell

Die verschiedenen Phasen, die beim Entwicklungsprozeß ablaufen, wurden schon 1970 von Royce [18] in einem Modell beschrieben, das sich aus anderen Ingenieurswissenschaften herleitet. Aufgrund der kaskadenähnlichen Folge der verschiedenen Phasen wird es als das „Wasserfallmodell“ bezeichnet (siehe Abbildung 3.38).

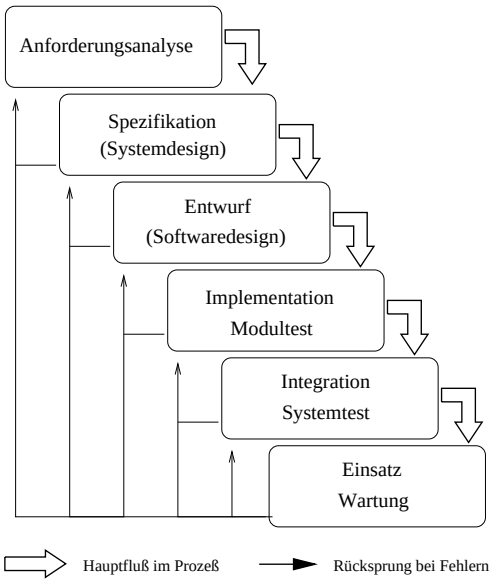
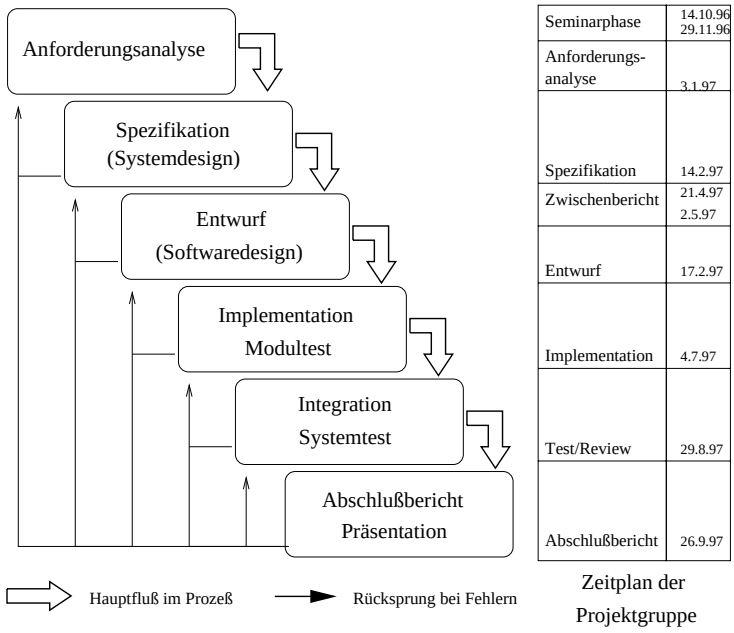


Abb. 3.38: Das Wasserfallmodell nach Royce

In dieser Darstellung sieht man auch das Hauptproblem beim Wasserfallmodell. Es ist ein Aktivitätenmodell. Bei Fehlern, die in einer Phase erkannt werden, kann in frühere Phasen zurückgesprungen werden. Das macht es jedoch schwierig, feste Termine einzuhalten, die für die Planung eines Projekts unbedingt notwendig sind. Durch die Einführung von „Meilensteinen“ kann dieses Problem abgeschwächt werden. Meilensteine sind Abschlußdokumente und auch -termine, die eine Phase beenden. Auf die Bedeutung von Meilensteinen wird später in 3.4.4.2 eingegangen.

Im Rahmen des Projekts Fahrgemeinschaften kommt - wie in einer Projektgruppe an der Fakultät Informatik üblich - die letzte Phase, Einsatz und Wartung, nicht zum Zuge. Um das Modell den Gegebenheiten anzupassen, kann man die letzte Phase durch „Endbericht und Abschlußpräsentation“ ersetzen. Damit

lassen sich die Phasen gut auf den Zeitplan der Projektgruppe abbilden (Abb. 3.39). Dieser Zeitplan enthält nun feste Meilensteine und Termine, bis zu denen alle Rücksprünge in frühere Phasen beendet sein müssen.



Seminarphase	14.10.96 29.11.96
Anforderungs-analyse	3.1.97
Spezifikation	14.2.97
Zwischenbericht	21.4.97 2.5.97
Entwurf	17.2.97
Implementation	4.7.97
Test/Review	29.8.97
Abschlußbericht	26.9.97

Zeitplan der Projektgruppe

Abb. 3.39: Das Wasserfallmodell für eine Projektgruppe

Im folgenden werden nun die einzelnen Phasen des Wasserfallmodells erläutert.

3.4.2.1 Anforderungsanalyse

Die Anforderungsanalyse dient der genauen Klärung des Problems. Sie findet komplett im Anwendungsfeld in Interaktion mit dem Kunden statt. Sie durchläuft folgende Phasen:

1. Machbarkeit und Notwendigkeit
   Bevor mit einem kostspieligen Projekt begonnen wird, sollte zunächst festgestellt werden, ob es überhaupt notwendig ist. Vielleicht gibt es ähnliche Produkte wie das zu entwickelnde schon auf dem Markt. Oder der Rechnereinsatz an sich ist für die Lösung des Problems nicht angebracht. Wenn sich die Notwendigkeit des Produkts ergibt, wird geprüft, ob es durchführbar ist und ob die notwendigen Resourcen zur Verfügung stehen. In dieser Phase können viele Projekte schon abgebrochen werden, bevor sie nach teuren Entwicklungsversuchen scheitern.

2. Analyse des Anwendungsfelds
   Bei der Analyse des Anwendungsfelds kommt es darauf an, die Welt des Kunden zu verstehen. Wichtig ist, daß zunächt der Ist-Zustand genau festgehalten wird. In der Regel soll dieser durch ein neues System nur verbessert, aber nicht völlig verändert werden. Deswegen liegt hier ein großer Teil der Arbeit. Es entstehen Modelle, die den bisherigen und den gewünschten Zustand beschreiben.

3. Erstellen des Anforderungskatalogs
   Nun werden die Wünsche des Kunden erfaßt und im Anforderungskatalog aufgeführt. Dabei muß er unterstützt werden, damit er sich das neue System vorstellen kann. Dies kann durch Prototypen oder Szenarien geschehen. Man muß versuchen, vermeintliche von echten Anforderungen zu trennen. Nicht immer ist dies möglich. Im Zweifelsfall entscheidet der Kunde über die Relevanz einer Anforderung.

   In der Regel gibt es auch nicht nur einen Kunden, sondern eine Reihe von Menschen, die unterschiedliche Erwartungen an das System stellen. Deswegen muß zu jeder neuen Anforderung genau dokumentiert werden, woher sie stammt und wie relevant sie in bezug auf das System ist. Das Zielsystem muß von vielen verschiedenen Blickwinkeln aus beleuchtet werden, wie z.B. Datenfluß, Bedienung, Schnittstellen, Sicherheit oder Wartung. Dies für zu detaillierteren Systemmodellen.

4. Klassifizieren der Anforderungen
   Der Anforderungskatalog wird nun geordnet. Die Anfordrungen werden nach verschiedenen Gesichtspunkten klassifiziert, wie z.B. funktionale Anforderungen, Anforderungen an die Schnittstellen, an die Benutzungsoberfläche oder an die Sicherheit.

5. Konflikte lösen
   Bei sich widersprechenden Anforderungen müssen Lösungen oder Synthesen gefunden werden. Dazu ist möglicherweise Rücksprache mit dem Kunden erforderlich.

6. Prioritäten setzen
   Nun werden die Anforderungen mit Prioritäten versehen, um einen Spielraum für den Fortgang des Projekts zu bekommen. Wichtige Funktionalität wird zuerst entwickelt, weniger wichtige ist optional.

7. Evaluierung der Anforderungen auf

   - Konsistenz - sind noch Konflikte vorhanden?
   - Allgemeingültigkeit - wollen das auch alle so?
   - Vollständigkeit - wurden wichtige Entscheidungen vergessen?
   - Realisierbarkeit - sind die Anforderungen überhaupt machbar?

   Wurden in einem dieser Punkte Defizite festgestellt, so wird eine der vorherigen Phasen zurückgesprungen, um diese auszugleichen.

8. Abnahme durch den Kunden
   Das Ergebnis dieser Phase ist der Meilenstein Anforderungsanalyse. Er enthält die klassifizierten und evaluierten Anforderungen des Kunden und sollte nun von ihm abgenommen werden.

3.4.2.2 Spezifikation

Die Spezifikation ist eine geordnete Menge von Anforderungen. Das äußere Systemverhalten - und nur das äußere! - wird detailliert beschrieben. Sie liegt damit auf der Schnittstelle zwischen dem System und dem Anwendungsfeld [11].

Es gibt es prinzipiell zwei Möglichkeiten zu spezifizieren: die formale und die informale Spezifikation.

1. Formale Spezifikation Das Problem der natürlichen Sprache ist, daß sie oft mehrdeutig und unvollständig ist. Bei der formalen Spezifikation wird daher eine eindeutig definierte Notation verwendet. Dies hat den Vorteil, daß man die Korrektheit formal beweisen kann. Es gibt auch Werkzeuge, um aus einer formalen Spezifikation Programmcode zu erzeugen. Formale Spezifikationen erfordern eine spezielle Notation, die zunächst erlernt werden muß. Dies kann man höchstens einem Entwickler, nicht aber dem Kunden zumuten. Da die Spezifikation aber die Grundlage für den Entwurf ist, sollte sie jeder lesen können. Jede formale Spezifikation muß deswegen durch eine informale ergänzt werden.

   Bei sicherheitsrelevanten Systemen ist die formale Spezifikation jedoch von immanenter Bedeutung. Bei einem Programm, das einen Airbus steuert, darf es einfach keinen Programm- und/oder Flugzeugabsturz durch eine fehlerhafte Spezifikation geben. Hier ist die Beweisbarkeit der Korrektheit eine lebenswichtige Eigenschaft.

   Ideal ist deshalb eine Kombination aus formaler und informaler Spezifikation, wobei alles informal und die sicherheitsrelevanten Teile zusätzlich noch formal spezifieziert werden.

2. Informale Spezifikation

   Eine informale oder natürlichsprachliche Spezifikation ist also unumgänglich. Auch hier kann ein gewisser Formalismus verwendet werden. Folgt man Standards, so kann man sicher gehen, daß man keine wichtigen Teile vergißt. Ein solcher Standard findet sich bei IEEE [10].

Die Spezifikation setzt sich also zusammen aus einer informalen und evtl. formalen Beschreibung des äußeren Systemverhaltens. Bei späteren Änderungen muß es aktualisiert werden. Sie stellt eine Art Vertrag dar. Das und nur das, was in der Spezifikation beschreiben wird, muß auch entwickelt werden. Das Systemverhalten wird so beschrieben, daß es nachher mit dem fertigen Produkt verglichen werden kann: Wurden die Anforderungen erfüllt? Sie ist die Schnittstelle zwischen dem Anwendungsfeld und der Implementation. Als Grundlage für den Entwurf befreit sie auch den Entwickler von wichtigen Entscheidungen über das Systemverhalten.

3.4.2.3 Entwurf

Beim Entwurf wird nun, ausgehend von der Spezifikation, das System in immer kleinere Komponenten zerlegt, bis hin zu den grundlegenden Datenstrukturen und Algorithmen. Ein guter Entwurf fördert die Arbeitsteilung: da die Schnittstellen zwischen den Subsystemen festgelegt werden, können die Einzelteile unabhängig voneinander entwickelt und getestet werden. Der Entwurf läuft in folgenden Phasen ab:

1. Architektur
   Das System wird in einem Grobentwurf analysiert. Es werden Subsysteme identifiziert und Zusammenhänge und Schnittstellen erkannt.

2. Spezifikation der Subsysteme
   Die einzelnen Subsysteme und ihr Datenfluß werden spezifiziert.

3. Schnittstellen
   Die Schnittstellen der Subsysteme werden spezifiziert. Dabei ist auf Kapselung und schmale Schnittstellen zu achten.

4. Modulentwurf
   Die Subsysteme werden in einzelne Module gegliedert, denen Aufgaben zugeordnet werden. Die Schnittstellen zwischen den Modulen werden definiert.

5. Datenstrukturen
   Die den Modulen zugrundeliegenden Datenstrukturen werden identifiziert.

6. Algorithmen
   Die Algorithmen, die auf den Datenstrukturen laufen, werden spezifiziert.

3.4.2.4 Implementierung

Da beim Entwurf schon der Aufbau des Systems beschrieben wurde, können die Einzelteile nun unabhängig voneinander entwickelt und getestet werden. Wichtig ist, daß es Richtlinien für die Programmierung gibt, in denen die Wahl von Bezeichnern, die Versionskontrolle, die Größe von Modulen, etc festgelegt werden. Wie das konkret aussieht, ist nebensächlich; nur einheitlich sollte es sein. Ein guter Programmierstil zeichnet sich dadurch aus, daß er nicht als solcher erkennbar ist, das bedeutet, daß von dem Programmcode nicht auf den Entwickler geschlossen werden kann.

3.4.2.5 Integration und Test

Nach der Implementierung des Systems werden die einzelnen Module und Subsysteme nach und nach zusammengesetzt oder integriert und dann systematisch getestet. Ein systematischer Test besteht aus einem Testdatensatz und einem Sollresultat, das aus der Spezifikation abgeleitet wird. Nach Durchführung des Tests wird das Ergebnis mit dem Sollresultat verglichen. Stimmen sie *nicht* überein, so ist der Test *positiv* verlaufen, es wurde ein Fehler gefunden. Erst nach der ersten Testreihe sollten diese Fehler korrigiert werden. Denn eine sofortige Korrektur könnte entweder neue Fehler erbringen oder spätere Fehler verdecken. Wenn keine Fehler mehr gefunden werden, kann das System in den Einsatz gehen. Im Fall einer Projektgruppe wird dann der Endbericht und die Abschlußpräsentation vorbereitet.

3.4.3 Besonderheiten bei objektorientierter Entwicklung

Das folgende Kapitel gibt eine grobe Einführung in die objektorientierte Entwicklung nach Booch [2].

3.4.3.1 Das Objektmodell

Der objektorientierten Entwicklung liegt eine bestimmte Sicht auf die Dinge zugrunde, das Objektmodell. Dieses kann von Sprache zu Sprache variieren. Es können jedoch folgende Eigenschaften identifiziert werden, die man z.B. bei den Sprachen Smalltalk oder C++ auch wiederfindet.

Anders als bei herkömmlicher Programmierung die die kleinste Einheit nicht der Algorithmus, sondern das Objekt. Ein Objekt ist eine Einheit, die aus Daten und Algorithmen auf diesen Daten besteht. Es besitzt eine gewisse Integrität und ist für seine Daten selbst verantwortlich. Objekte sind Instanzen von Klassen, die das Objektverhalten beschreiben. Eine Klasse kann man als die Idee eines Objekts beschreiben, während das Objekt als Instanz einer Klasse eine konkrete Materialisierung dieser Idee darstellt. Die Hauptelemente des Objektmodells sind folgende:

1. Abstraktion.
   Es werden jeweils nur die für das Problem konkret interessanten Eigenschaften eines Objekts betrachtet, relativ zur Perspektive des Betrachters.

2. Kapselung.
   Die Details der Implementierung werden vor der Außenwelt verborgen. Das Objekt liefert nach außen nur Methoden zur Manipulation seiner Daten, verbirgt jedoch deren konkrete Realisierung.

3. Modularität.
   Verschiedene Objekte können zu Modulen zusammengefaßt werden, die für sich wiederum das Prinzip der Abstraktion und der Kapselung innehaben. Dadurch kann das System auf immer höherer Ebene abstrahiert und beschrieben werden.

4. Hierarchie.
   Die Klassen stehen in einer hierarchischen Beziehung zueinander. Niedere Klassen werden von höheren Klassen abgeleitet und erben von diesen alle Eigenschaften. Diese können sie ergänzen oder überschreiben.

5. Typisierung.
   Daten haben Typen. Daten unterschiedlichen Typs können nicht ohne weiteres miteinander kombiniert werden, sondern bedürfen besonderer Konvertierung. Auch Klassen werden als Typ behandelt und bieten oft selbst unterschiedliche Zugriffe. Eine Klasse Integer könnte zum Beispiel eine Methode anbieten, die den Wert des Objekts als Real-Zahl zurück gibt. Dies folgt wiederum dem Prinzip der Kapselung: Die konkrete Implementierung der Integerzahl bleibt dadurch verborgen.

6. Nebenläufigkeit.
   Das Objektmodell beinhaltet eine Parallelität bei der Ausführung. Verschiedene Objekte können gleichzeitig unabhängig voneinander agieren oder sich auch gegenseitig beeinflußen.

7. Persistenz.
   Objekte haben eine Lebensdauer. Diese kann von wenigen Taktzyklen wie z.B. einer Schleifenvariablen, die nur kurz instanziiert und dann wieder freigegeben wird, bis über die Laufzeit der Anwendung hinaus reichen wie bei Datenbankobjekten.

3.4.3.2 Die Phasen bei der objektorientierten Entwicklung

Analyse Bei der Analyse betrachtet man die Anforderungen aus der Sicht der Klassen und Objekte. Man sammelt sie und klassifiziert sie zu Einheiten, die später Objekte werden könnten.

Spezifikation Die objektorientierte Spezifikation unterscheidet sich nicht von der „normalen“, da die Spezifikation nur das äußere Systemverhalten, nicht aber die Implementierung beschreibt. Ein Fenster wird einfach geschlossen, egal, ob es die Methode eines Objektes Fenster oder einfach eine entsprechende Prozedur war.

Entwurf Beim objektorientierten Entwurf wird das Problem nicht in Algorithmen, sondern in Objekte, Klassen und ihre Beziehungen zueinander zerlegt. Sehr hilfreich ist eine Notation, die die Zusammenhänge übersichtlich beschreibt. Eine mächtige Methode findet sich bei Booch [2]. Dabei können die Klassen und ihre Beziehungen voneinander graphisch dargestellt und einfach um zusätzliche Erkenntnisse erweitert werden.

Programmierung Grundsätzlich kann in jeder Sprache objektorientiert programmiert werden. Manche legen es jedoch näher als andere. Die Konzepte des Objektmodells sollten ein Bestandteil der Programmierung sein. Ohne Vererbung unterscheidet es sich z.B. nicht vom Programmieren mit abstrakten Datentypen. Bekannte OO-Sprachen sind Smalltalk oder C++. Inzwischen bieten aber viele andere Sprachen objektorientierte Zusätze.

3.4.4 Techniken der Dokumentation

Der Stellenwert der Dokumentation in einem Projekt ist meist untergeordnet. Sie kostet eine Menge Zeit und hält den Entwicklungsprozeß auf. Aber ohne Dokumentation gibt es keine Sicherheiten, keine Vereinbarungen, auf die man sich verlassen kann. Da man aus früheren Fehlern nicht lernen kann, gibt es auch keinen Fortschritt im Prozeß. Dokumentation gehört genauso zur Software wie der Programmcode. Gerade in den frühen Phasen wie Anforderungsanalyse oder Spezifikation kommt man ohne eine Dokumentation nicht aus, da es sonst keine Dokumente über den Stand und den Fortgang des Projekts gibt.

3.4.4.1 Prozeßdokumentation

Für jede Phase muß der Projektstand und die Ergebnisse der Phase dokumentiert werden. Dazu gehören Protokolle der Sitzungen, Untergruppenberichte und Vereinbarungen. In der Regel wird nur dann regelmäßig und zuverlässig dokumentiert, wenn es kein zu hoher Aufwand ist. Deswegen braucht man für alle diese Dokumente Formulare, die das Verfassen erleichtern und ihnen auch ein einheitliches Aussehen geben. Die mindeste Information, die ein Dokument enthalten muß, ist das Datum und den Verfasser, damit man es in den zeitlichen Ablauf des Projekts einordnen kann.

3.4.4.2 Meilensteine

Meilensteine sind sowohl Termine als auch Abschlußdokumente der verschiedenen Phasen. Sie sind jeweils der Ausgangspunkt für die folgende Phase und beeinflußen diese nachhaltig. Änderungen an Meilensteinen, sobald sie abgenommen wurden, müssen deshalb wohlüberlegt und selten sein. Die Meilensteine eines Projekts heißen Anforderungsanalyse, Spezifikation, Entwurf, Zwischenbericht, Endbericht und Programmcode.

3.4.4.3 Dokumentation von Programmcode

Viele verstehen unter Dokumentation nur die Kommentare im Programmcode. Dies ist jedoch nur ein kleiner Teil davon. Nichtsdestotrotz sind Kommentare außerordentlich wichtig, nicht nur, wenn mehrere Menschen am selben Code arbeiten. Man versuche nur, sich vier Wochen nach dem Verfassen noch an die Absichten hinter einem Stück Programm zu erinnern.

Gute Kommentare wiederholen nicht das Offensichtliche, sondern erläutern die Absichten.

- schlechtes Beispiel:

  ```
  x:=1;           // x bekommt den Wert Eins
  ```

  Was im Kommentar steht, kann jeder aus der Programmzeile entnehmen. Solche Kommentare blähen den Code nur unnötig auf. Außerdem ist der Variablenname alles andere als sprechend.

- gutes Beispiel:

  ```
  zaehler:=1;     // Der Zaehler wird initialisiert
  ```

  Hier offenbart der Kommentar, welche Absicht der Autor mit dieser Zeile verfolgt.

3.4.4.4 Reviews

Technische Reviews dienen zur Überprüfung von Dokumenten und Programmcode. Gerade bei den frühen Phasen sind sie unumgänglich, da mit einem Textdokument kein systematischer Test gemacht werden kann.

Werden technische Reviews eingeführt, so findet man am Anfang viele kleine, z.B. Rechtschreib- oder Formfehler. Mit der Zeit werden die Autoren jedoch besser und es ist eine Qualitätsverbesserung festzustellen: Es werden die strukturellen, die wirklich „teuren“ Fehler gefunden.

Personen und Rollen

- Autor. Nimmt am Review teil, muß sich jedoch nicht rechtfertigen. Es wird das Produkt, nicht sein Autor gereviewed.
- Sitzungsleitung. Lädt zum Review ein und verschickt das anstehende Dokument.
- Protokollant. Schreibt alles mit, was an Fehlern gefunden wird und lässt es dem Autor zukommen.
- Manager oder Auftraggeber. Nimmt am Review nicht teil. Das technische Review ist eine interne Angelegenheit der Entwickler zur Verbesserung des Produkts.
- Gutachter. Die Gutachter sind meist andere Entwickler, die den Prüfling auf Fehler untersuchen.

Vorbereitung Um ein technisches Review durchzuführen, benötigt man eine Version des zu prüfenden Dokuments, die während der Reviewphase nicht mehr verändert wird. Ein Review auf einem veraltenden Prüfling ist nutzlos.

Dieses Dokument verschickt nun die Sitzungsleitung an die Teilnehmer des Reviews zusammen mit der Einladung und einer Aufgabenverteilung. Dies muß rechtzeitig vor dem Termin erfolgen, damit die Gutachter nun anhand den ihnen zugeteilten Aspekten den Prüfling untersuchen können. Dabei registrieren sie auch Fehler, die nicht in ihrem Bereich liegen. Durch die Konzentration auf bestimmte Aspekte werden jedoch mehr Fehler gefunden.

Ablauf Ein technisches Review beginnt damit, daß der Sitzungsleiter feststellt, ob die Gutachter erschienen sind und ihre Arbeit erledigt haben. Ist dies nicht der Fall, kann das Review abgebrochen werden.

Beim eigentlichen Review wird das Dokument absatzweise durchgegangen. Jeder Gutachter sagt dabei, welche Fehler er unter seinem Aspekt gefunden hat. Der Protokollant notiert dies. Nach dem Review stellt die Runde noch ein Ergebnis aus. Mögliche Ergebnisse sind

- Keine Beanstandungen. Das Dokument wird so, wie es ist, angenommen.
- Kleine Änderungen, kein weiteres Review. Nach der Einarbeitung der erwähnten Änderungen durch den Autor wird das Dokument ohne weiteres Review angenommen.
- Große Änderungen, weiteres Review. Der Autor arbeitet die Änderungen ein. Danach wird ein weiteres Review angesetzt.
- Gravierende Beanstandungen. Das Dokument wird nicht angenommen. Es wird empfohlen, es von Grund auf neu zu schreiben.

3.4.5 Vereinbarungen

Um in einem Projekt Techniken des Softwareengineerings anzuwenden, ist es umbedingt notwendig, daß die Teilnehmer Vereinbarungen treffen. Es ist nicht wichtig, was konkret vereinbart wird, sondern daß etwas vereinbart wird. Diese Vereinbarungen enthalten dann Absprachen, Formulare, Standards, Entscheidungen und Style Guides. Sie sind die Referenz für die einzelnen Teilnehmer und müssen stets aktualisiert werden, wenn sich etwas geändert hat oder eine neue Absprache dazugekommen ist.

3.5 Constraint Programmierung

3.5.1 Einleitung

Immer häufiger werden zum Lösen von Problemstellungen elektronische Geräte eingesetzt. Angefangen bei einfachen Küchengeräten über Telefonanlagen bis hin zu hochentwickelten Industrierobotern, die entsprechend ihrem Einsatzgebiet ausgesucht und eingesetzt werden. Aber dennoch gilt für jedes Gerät, die Lösung zur vorhandenen Problemstellung so einfach wie möglich zu programmieren. Dabei stellt sich die Frage, auf welche Art und Weise kann überhaupt programmiert werden ?

Eine Art ist die herkömmliche Programmiermethode. Das bedeutet, man gibt eine genaue Handlungsanweisung an, wie das Problem zu lösen ist.

Beispiel:

```
begin
    führe A aus;
    führe B aus;
    führe C aus;
    if D = true then
        führe E aus;
    else F
end
```

Allerdings kann hier in einigen Fällen nicht das gesamte Problemfeld abgedeckt werden. Vor allem dann nicht, wenn sich zur Laufzeit die Verhältnisse ändern. Diese waren zu Anfang in dieser Weise nicht bekannt oder sind einfach vergessen worden. Probleme ergeben sich auch, falls mehrere Lösungen zulässig sind. Zum Beispiel besitzt ein Industrieroboter unterschiedlich angeordnete Gelenke, die zum Lösen der Problemstellung entsprechend angesteuert werden müssen. Dabei kommt es nicht auf die Position des einzelnen Gelenks an, sondern auf die Gesamtposition des Roboters. Folglich sind mehrere Lösungen möglich und richtig.

Wie man sieht sind dieser Programmiermethode Grenzen gesetzt. Damit die oben genannten Probleme dennoch gelöst werden können, wurde die Constraint Programmierung entwickelte.
Die Constraint Programmierung besitzt folgende Eigenschaft: Man gibt nicht mehr an, wie ein Problem gelöst werden soll, sondern beschreibt das Problem mit Bedingungen, die erfüllt sein müssen bzw. sollen.
Constraint Programmierung bedeutet weiterhin, daß der Programmierer nicht mehr selbst die Übertragung von der Beschreibung der Bedingungen zur tatsächlichen Lösung zu machen braucht. Dies übernimmt ein System, das Algorithmen zur Auflösung von Constraint einsetzt.

Weitere Einzelheiten der Constraint Programmierung werden nun in den

folgenden Abschnitten vorgestellt und erklärt, wobei auf den Auflösungsalgorithmus genauer eingegangen wird.

3.5.2 Constraint Programmierung

Bereits in der Einleitung wurde die Grundidee der Constraint Programmierung vorgestellt: Man gibt nicht mehr an, wie eine Aufgabe gelöst werden soll, sondern welche Bedingung nach der Bearbeitung der Aufgabe gelten sollen. Durch diese Eigenschaft entstand eine neue Programmmiertechnik, für die ein entsprechender Programmierablauf eingehalten werden sollte. Solch ein Ablauf wird nun im folgenden allgemein vorgestellt.

3.5.2.1 Allgemeiner Programmierablauf

Bevor man anfängt zu programmieren, muß die Problemstellung analysiert und detailliert aufgeschrieben werden. Danach definiert man alle denkbaren Constraints, die das Problem komplett beschreiben. Wird nur eine Teilmenge der Constraints benötigt, muß eine Auswahl getroffen werden, die wirklich zum Lösen der Problemstellung beitragen. Nachdem die Constraints ausgewählt wurden, wird ein Auflösungsalgorithmus gestartet, der als Ergebnis eine Lösung oder ein entsprechendes Verhalten berechnet. Danach wird eine Lösung ausgegeben bzw. ein Verhalten ausgeführt.

Schematischer Ablauf:

1. Problemanalyse
2. Definition der Constraints
3. Auswahl einer Teilmenge von Constraints
4. Auflösungsalgorithmus
5. Lösung bzw. entsprechendes Verhalten

3.5.2.2 Constraints

Während der grobe Ablauf der Constraint Programmierung nur erwähnt, daß entsprechende Constraints definiert werden müssen, soll hier kurz eine Definition der Constraints zum Verständnis gegeben werden.
Constraints sind Bedingungen, die als Relationen über Variablen definiert werden. Ein Constraint C mit Variablen $p_1, \ldots, p_n$ legt eine beliebige Relation R zwischen den einzelnen p_i, $i = 1, \ldots, n$ fest. Wenn für alle Variablen eine entsprechende Wertzuweisung gefunden wurde, ist ein Constraint erfüllt.
Beispielsweise wird der numerische Constraint Summe als $Summe(a, b, c)$ dargestellt. Zur Verarbeitung dieses Constraints ist jedoch eine speziellere Darstellung

der Relation notwendig:

$$\begin{aligned} Summe(a,b,c) \quad \Leftrightarrow \quad & (c = a + b) \\ & (b = c \Leftrightarrow a) \\ & (a = c \Leftrightarrow b) \end{aligned}$$

Diese Darstellung hat den Vorteil, daß für jede Variable der Relation ein eigener Ausdruck vorliegt und so die weitere Verarbeitung ohne zusätzliches Wissen über andere Relationen möglich ist.

3.5.2.3 Anwendungen

Im folgenden werden zwei Anwendungsgebiete vorgestellt, die sich für die Constraint Programmierung besonders gut eignen. Dabei werden die Constraints konkretisiert und zum anderen wird gezeigt, wie bzw. wo sie in Programmen eingesetzt werden können.
Die folgenden Beispiele stammen aus den zwei Anwendungsgebieten:

1. Regel- und Steuerungstechnik
2. Berechnungstechnik

1. Regel- und Steuerungstechnik

Zu diesem großen Gebiet der Regel- und Steuerungstechnik gehört unter anderem der Bereich der Roboterprogrammierung. Roboterprogrammierung basiert zum größten Teil auf einer Bewegungsbeschreibung, die mit Hilfe von Bewegungsbahnen in einem Konfigurationsraum oder einem Zustandsraum beschrieben wird. Meist werden bei Industrieroboter diese Bewegungsbahnen von Hand spezifiziert. Das ist einfach, intuitiv und ausreichend für einige Anwendungen.

Gehmaschine

Für das Programmieren einer Gehmaschine, die sich dynamisch in drei Dimensionen fortbewegen kann, bereitet dieses herkömmliche Vorfahren große Schwierigkeiten. Hierfür müßte nach immer wiederkehrende Bewegungsabläufe der einzelnen Gelenke gesucht werden. Anschließend werden diese Abläufe geplant und in einer entsprechender Reihenfolge aufgerufen. Das ist sehr kompliziert und komplex.

Diese Aufgabe kann mit der Constraint Programmierung eleganter gelöst werden. Dazu müssen allerdings entsprechende Constraints für das gewünschte Verhalten definiert werden, die etwa so aussehen können:

$Constraint_1 =$ Fuß sollte während der Schwingphase des Beines keinen Kontakt zum Boden haben.

$Constraint_2$= Die Fußplazierung muß so gewählt werden, daß ein ausgeglichenes Gleichgewicht vorliegt.

Damit die Gehmaschine richtig laufen kann, müssen zur Laufzeit die Constraints ständig auf Erfüllung kontrolliert werden. Stellt sich heraus, daß die beiden Bedingungen nicht ausreichend sind, kann durch Hinzufügen eines neuen Constraints das existierende Programm erweitert werden.

$Constraint_3$= Das Becken muß über einer bestimmten Höhe sein.

Mobile Roboter

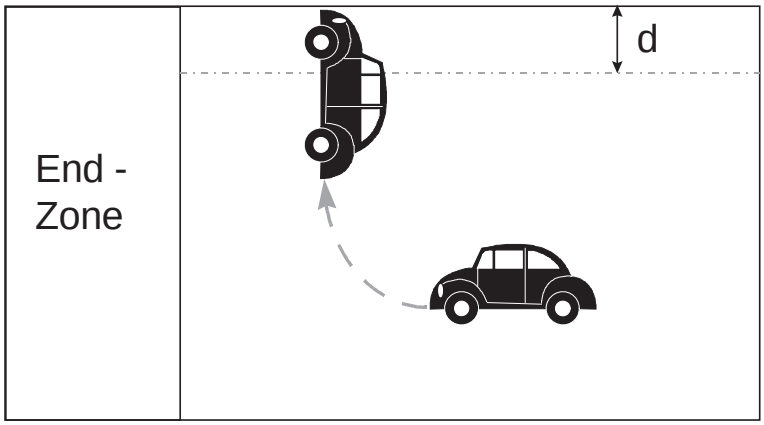


Abb. 3.40: Versuchsanordnung

Mit einem weiteren Beispiel der Roboterprogrammierung, den mobilen Robotern, können folgende nützliche Eigenschaften der Constraints charakterisiert werden. Dazu werden funkgesteuerte mobile Roboter in einer Versuchsanordnung betrachtet, bei denen die Richtung und die Geschwindigkeit über eine entsprechende Vorrichtung eingestellt werden können (Abbildung 3.40).
Die Constraints, die die Fahrt eines solchen mobilen Roboters vollständig beschreiben, lassen sich untergliedern in:

1. Constraints, die zu Beginn des Programms initialisiert werden.
    1. Fahr-Constraint
    2. Geschwindigkeits-Constraint
    3. Richtungs-Constraint
    4. Endzone-Contraint
       Der mobile Roboter wird so gesteuert, daß er in Richtung Endzone fährt.
2. Constraints, die zur Laufzeit generiert werden.

1. Wand und Hindernis-Constraint
   Der mobile Roboter befindet sich in einem abgeschlossenen Raum. Kommt es nun zu einer Berührung zwischen dem Roboter und einer Kante, so wird ein Constraint generiert. Dieser Constraint veranlaßt eine virtuelle Kante w, die mit Abstand d_w entlang der realen Kante in Richtung des freien Raumes eingefügt wird. Dadurch gelangt der Roboter in Zukunft nicht mehr an die reale Kante. Dem Roboter wurde angelernt, wo eine Kante ist.
   Diese Lerneigenschaft mit Hilfe von Constraints macht sich die Künstliche Intelligenz zu nutze, da nur wenige Programmiersprachen diese Eigenschaft des Lernens komfortabel unterstützen.

Fazit

In den Beispielen wurde gezeigt, daß Generierung von Constraints zur Laufzeit und Ergänzungen, wie im Beispiel Gehmaschine durch das Constraint 3, einfach realisiert und ohne Probleme in das System integriert werden können. Mit anderen Programmiermethoden wäre dies bedingt möglich, jedoch wenn überhaupt, nur sehr umständlich.
Durch die Kombination dieser vielfältigen Eigenschaften der Constraints ist es durchaus möglich, komplizierte Problemstellungen in der Regel-und Steuerungstechnik sehr einfach und vorallem sehr schnell zu programmieren.

2. Berechnungstechnik

Ein weiteres Aufgabengebiet der Constraint Programmierung ist das Lösen von Berechnungsaufgaben. Auch hier wünscht man sich eine einfache Beschreibung der Aufgabe und eine schnelle Lösung.

Viele Berechnungsconstraints können wie elektrische Schaltkreise (Abbildung 3.41) dargestellt werden. Bildhaft kann man sich einen Constraint als Blackbox vorstellen. Im Gegensatz zu einem elektrischen Schaltkreis gibt es jedoch keine vordefinierte Berechnungsrichtung. Dadurch kann ein beliebig ausgewählter Anschluß eines Constraints aus den übrigen Anschlüssen berechnet werden. Beispielsweise besitzt der Addierer *add-1* aus Abbildung 3.41 drei Anschlüsse: A,B und C. Wenn nun zwei der drei Anschlüsse Werte besitzen, wobei Anschluß B durch die Konstante immer einen Wert besitzt, wird der Wert für den dritten Anschluß so berechnet, daß die Summe der Werte der Anschlüsse A und B gleich dem Wert von Anschluß C ist, d.h. wenn der Wert von Anschluß C gegeben und A gesucht ist, wird subtrahiert: $A = C \Leftrightarrow B$.

Die Anschlüsse von Constraintinstanzen können miteinander verbunden werden, so daß Constraintnetze entstehen (Abbildung 3.41). Alle miteinander verbundenen Anschlüsse besitzen den gleichen Wert. Ist das nicht der Fall, so befindet sich das Netz in einem Konfliktzustand und es muß eine Auflösung des Konfliktes erfolgen.
Wie solch eine allgemeine Auflösung im einzelnen funktioniert wird im nächsten Abschnitt erklärt.

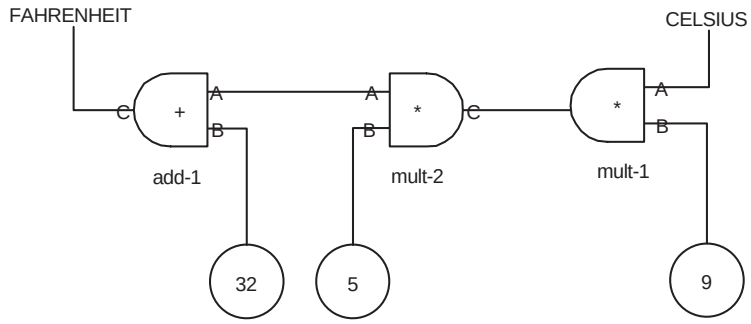


Abb. 3.41: Temperaturkonverter

3.5.3 Auflösungsalgorithmus

In den vergangenen Abschnitten wurden die Eigenschaften der Constraints vorgestellt, was Constraints sind und wie bzw. wo sie eingesetzt werden können. Dabei wurde auf die Beschreibung der Problemstellung mit Constraints eingegangen und nicht wie das System eine bzw. mehrere Lösungen berechnet. Das WIE wird nun in diesem Abschnitt näher beschrieben.

Aus Abschnitt 3.5.2.1 ist bekannt, daß nach der Auswahl der Constraints ein Auflösungsalgorithmus gestartet wird, der die Lösung oder ein Verhalten berechnet. Der hier vorgestellte Auflösungsalgorithmus bezieht sich nur auf die zweite vorgestellte Anwendung: Berechnungstechnik.
Der nun betrachtete Auflösungsalgorithmus (vgl. [19]) stellt ein inkrementelles Verfahren dar, das durch schrittweises Einsetzen der Constraints gelöst wird.

3.5.3.1 Vorbemerkungen

Im nächsten Abschnitt werden zur Beschreibung des Auflösungsalgorithmus Begriffe verwendet, die im folgenden definiert werden.

- **Constraint Hierarchie**

  Eine Constraint Hierarchie H ist ein 5-Tupel $(Vars, Cons, D, level, d)$, wobei gilt:

  1. $Vars = \{v_1, v_2, \ldots, v_n\}$ ist eine Menge aus Variablen
  2. $Cons = \{c_1, c_2, \ldots, c_m\}$ ist eine Menge aus Constraints
  3. D ist ein endlicher Definitionsbereich aller Variablen
  4. $level$ ist eine Funktion, die die Priorität der einzelnen Constraints ausgibt
     $level : Cons \to \mathbb{N}_0; \quad c_i \mapsto n,$
     wobei $level(c_i) = n$ auch als $c_i@n$ geschrieben wird

5. d ist eine Funktion, die jeder Variable einen Definitionsbereich zuordnet
   $d : Vars \rightarrow D; \quad v_i \mapsto D_i \subseteq D$

- **Constraint Speicher**

  Ein Constraint Speicher S ist eine Menge $S \subseteq Cons$, die bezüglich der Priorität unterteilt werden kann.

  $S = S_{[i]} \vee S_{[<i]} \vee S_{[>i]}$ wobei i eine Priorität ist

- **Konfiguration einer Hierarchie**

  Eine Konfiguration Φ, der Hierarchie H, besteht aus drei verschiedenen Constraint Speichern $\langle AS \bullet RS \bullet US \rangle$, wobei AS der Active Speicher, RS der Relaxed Speicher und US der Unexplored Speicher ist.
  Der Speicher AS beinhaltet Constraints, die aufgrund des eingeschränkten Definitionsbereichs erfüllt sind. Der Speicher RS enthält aus dem Speicher AS stammende Constraints, die zur Zeit der Aktivierung einen Konflikt hervorgerufen haben. Der Speicher US ist ein Zwischenspeicher. Er enthält nur die Constraints, die zum Ausführen der Regeln notwendig sind. Ansonsten füllt der Speicher US seinen Inhalt mit immer nur einem Constraint aus der Menge $Cons$ auf.

- **Konflikt**

  Man spricht von einem Konflikt, wenn der Definitionsbereich einer Variablen die leere Menge ist. Dazu werden nur die Variablen der Constraints im Speicher AS betrachtet.

- **Besser Prädikat**

  Gegeben seien zwei Konfiguration Φ und Φ'. Φ ist besser als Φ' (Schreibweise $\Phi \leq \Phi'$), gdw

  1. $\Phi = \Phi'$ oder
  2. $(\exists k > 0, \forall i < k), \sharp RS_{[i]} = \sharp RS'_{[i]}$ und $\sharp RS_{[k]} < \sharp RS'_{[k]}$

3.5.3.2 Beschreibung des Algorithmus

Im folgenden wird nun der Algorithmus mit seinen Regeln und Abbruchkriterien vorgestellt und näher erläutert.

- **Final Konfiguration**

  Eine Konfiguration Φ ist eine Final Konfiguration (Schreibweise$FC(\Phi)$) genau dann, wenn

  1. $\forall c \in AS$: c ist erfüllt
  2. $\forall c \in Cons : c \in AS$ oder $c \in RS$

- **Promising Konfiguration**

  Eine Konfiguration Φ ist eine Promising Konfiguration (Schreibweise$PC(\Phi)$) genau dann, wenn

  1. $\forall c \in AS$: c erfüllt ist
  2. $\neg FC(\Phi') \leq PC(\Phi)$ (Es gibt keine bessere Final Konfiguration)

- **Dead End**

  Ein Dead End bedeutet, daß weder die Forward Regel noch die Backward Regel angewendet werden kann. Für die Forward Regel bedeutet dies, es liegt bereits eine Final Konfiguration vor, die besser ist als die aktuelle Konfiguration. Während mit Hilfe der Backward Regel keine schlechtere Konfiguration $succ(\Phi_{conf})$ bestimmt werden kann, da im Aktiv Speicher nur noch Constraints mit der Priorität 0 enthalten sind, die nicht deaktiviert werden können.

Algorithmus

begin
  Durchlauf = 1;
  repeat
    if (Speicher US = leer) **then**
      füge ein Constraint aus der Cons Menge in den
      Speicher US;
    if (aktuelle Konfiguration = Promising Konfiguration) **then**
      führe die Forward Regel aus;
      schränke die Definitionsbereiche der Variablen entsprechend
      der $c_i \in AS$ ein;
    if (mindestens ein Definitionsbereich einer Variablen = $\emptyset$) **then**
      führe die Backward Regel aus;
    Durchlauf = Durchlauf + 1;
  until (Konfiguration = Final) oder (Konfiguration = Dead End)
end

- **Forward Regel**

  Die Forward Regel entfernt aus dem Speicher US ein Constraint und fügt es in den Aktiv Speicher ein (Aktivierung eines Constraints), sofern die

aktuelle Konfiguration eine Promising Konfiguration ist. Sei also Φ die aktuelle Konfiguration und c das aktivierte Constraint.

Forward Regel

$$\frac{PC(\Phi) \qquad \exists c \in US}{\Phi \Leftrightarrow \langle AS \cup \{c\} \bullet RS \bullet US \setminus \{c\}\rangle}$$

- **Backward Regel**

  Im Falle eines Konfliktes sucht die Regel nach einer alternativen Promising Konfiguration. Dazu werden Constraints mit Priorität größer 0 aus dem Speicher AS in den Speicher RS verschoben (Deaktivierung eines Constraints) und in einem früheren Duchlauf dorthin verschobene Constraints, können wieder in den Speicher AS reaktiviert werden. Auch hier wird der Definitionsbereich entsprechend den nun vorhandenen Constraints im Speicher AS angepaßt. Im folgenden sei Φ, die aktuelle Konfiguration, gegeben, aus der die Konfliktkonfiguration Φ_{conf} abgeleitet wird. Daraus ergibt sich dann eine schlechtere Konfiguration $succ(\Phi_{conf})$.

  Vorgehensweise der Backward Regel

  1. Zunächst legt die Backward Regel einen Konfliktspeicher $CS \subseteq AS$ an, in dem das aktivierte Constraint (das zuletzt in AS eingefügte) zusammen mit den Constraints des Aktiv Speichers, die den Definitionsbereich mindestens einer Variablen des aktivierten Constraints beeinflussen, enthalten sind.
  2. Aus diesem Konfliktspeicher wird eine Konfliktkonfiguration Φ_{conf} erstellt, die aus den drei Speichern $Active_{Konflikt}$, $Relaxed_{Konflikt}$ und $Unexplored_{Konflikt}$ besteht. Diese drei Speicher werden entsprechend der aktuellen Konfiguration Φ mit Constraints aus der Menge $Cons_{conf}$ belegt, d.h. Constraints, die in Konfiguration Φ im Speicher AS waren, bleiben auch in der Konflikt Konfiguration Φ_{conf} im Speicher AS. Die Menge $Cons_{conf}$ besteht aus der Vereinigung aller früheren Konfliktspeichern inklusiv der aktuellen.
  3. Die Konfiguration Φ_{conf} dient zur Festlegung einer schlechteren Konfiguration $succ(\Phi_{conf})$, aus der die Constraints zur De- und Reaktivierung bestimmt werden. Dabei können mehrere Möglichkeiten entstehen. Laut dem Besser Prädikat genügt es nun, ein oder mehrere Constraint aus dem Speicher AS in den Speicher RS zu verschieben bzw. umgekehrt. Den Inhalt des Speichers AS bezeichnet man im folgenden als $Activate$ und den Speicher RS als $Relax$.
  4. Wird ein Constraint deaktiviert, werden alle Nachfolge-Constraints gesucht. Nachfolge-Constraints sind Constraints, die zeitlich später in den Speicher AS eingefügt worden sind, als das deaktivierte Constraint. Weiterhin gilt, daß nur nach solchen Nachfolge-Constraints gesucht wird, die dieselben Variablen benutzen, wie das deaktivierte

Constraint. Dies hat zur Folge, daß der Definitionsbereich der Variablen für die Nachfolge-Constraints im Speicher AS abgeändert werden muß. Dazu werden die gefundenen Nachfolge-Constraints in der Menge $Reset$ abgelegt, wobei gilt $Reset \subseteq AS$.

Nachdem diese einzelnen Mengen bestimmt sind, kann die Nachfolgekonfiguration Φ' entsprechend den Berechnungsformeln berechnet werden.

Backward Regel

$$\frac{AS \vdash x \perp \qquad \exists \Phi_{conf} \subseteq \Phi \qquad \exists Reset \subseteq AS}{\Phi \Leftrightarrow\!\!\!\!\rightarrow \Phi'}$$

wobei für $\Phi' = \langle AS' \bullet RS' \bullet US' \rangle$ gilt:

1. $\langle Activate \bullet Relax \rangle \leftarrow succ(\Phi_{conf}, \leq) \setminus \Phi_{conf}$
2. $AS' \leftarrow AS \setminus (Relax \cup Reset)$
3. $RS' \leftarrow (RS \setminus Activate) \cup Relax$
4. $US' \leftarrow (US \setminus Relax) \cup Reset \cup Activate$

und $AS \vdash x \perp :\Leftrightarrow \exists c_i \in AS$: so daß c_i nicht erfüllt ist

3.5.3.3 Beispiel

Die abgebildete Auflösungstabelle (Tabelle 3.1) zeigt, wie der Algorithmus zur Bestimmung einer möglichen Lösung vorgeht. Dabei werden die verwendeten Regeln und die daraus entstehenden Konfigurationen mit entsprechendem Definitionsbereich angegeben. Zum Bestimmen einer alternativen Konfiguration bei Auftritt eines Konfliktes, werden aus der Tabelle 3.2 die notwendigen Mengen verwendet.
Für das Beispiel sei folgende Hierarchie H gegeben:

$$H = \big(\{x, y\}, \{c_1, c_2, c_3, c_4, c_5, c_6\}, \mathbb{Z}, level, d\big)$$

mit den Definitionsbereichen

$$d(x) = [1 \ldots 10] \quad d(y) = [1 \ldots 10]$$

und den Prioritäten der Constraints

$$level(c_i) = \begin{cases} 0 & für \quad i = 5, 6 \\ 1 & für \quad i = 1, 2 \\ 2 & für \quad i = 3, 4 \end{cases}$$

Die verwendeten Constraints besitzen folgende Definition:

$$
\begin{aligned}
c_1 &= x + y = 15 \\
c_2 &= 3 * x \Leftrightarrow y < 15 \\
c_3 &= x > y + 1 \\
c_4 &= x < 7 \\
c_5 &= x < 8 \\
c_6 &= y < 5
\end{aligned}
$$

Die dargestellten Beispieldurchläufe dienen zur Erklärung der Auflösungstabelle. Dazu werden zwei Teilberechnungen aus der gesamten Auflösungstabelle (Tabelle 3.1) genauer beschrieben.

1. Beispieldurchlauf
   Der Auflösungsalgorithmus startet mit dem Constraint c_1 im Speicher US (Durchlauf 0). Aufgrund der vorliegenden Promising Konfiguration, wird die Forward Regel ausgeführt. Dabei gelangt c_1 in den Speicher AS und der Definitionsbereich der beiden Variablen wird so eingeschränkt, daß das Constraint erfüllt ist (Durchlauf 1). Aus der Menge $Cons$ wird ein nächstes Constraint c_2 in den Speicher US eingefügt. Da auch hier eine Promising Konfiguration vorliegt, wird die Forward Regel angewendet und das Constraint wandert in den Speicher AS (Durchlauf 2). Dabei entsteht ein Konflikt, da es keinen geeigneten Definitionsbereich für die vorhandenen Variablen gibt, so daß die Constraints c_1 und c_2 erfüllt sind. Deshalb muß die Backward Regel angewendet werden. Hierzu werden entsprechende Speicher und Konfigurationen laut der Vorgehensweise der Backward Regel ermittelt. Aufgrund der Regel 2: $AS' \leftarrow AS \setminus (Relax \cup Reset)$ und der Regel 3: $RS' \leftarrow (RS \setminus Activate) \cup Relax$, wird das Constraint c_2 aus dem Speicher AS in den Speicher RS verschoben und der vorherige Definitionsbereich wieder hergestellt (Durchgang 3).
   Die weiteren Durchläufe werden analog behandelt.

2. Beispieldurchlauf
   Eine Besonderheit des Auflösungsalgorithmus entsteht ab Durchlauf 6. Zur Situation: c_1,$c_3 \in AS$, c_2,$c_4 \in RS$ und $c_5 \in US$.
   Durch die Forward Regel gelangt das Constraint c_5 in den Speicher AS und es entsteht ein Konflikt. Die Backward Regel muß angesetzt werden (Durchlauf 7). Da das Constraint c_5 die Priorität $level(c_5) = 0$ besitzt, kann c_5 nicht in den Speicher RS verschoben werden. Deshalb muß ein anderes Constraint aus dem Speicher AS gefunden werden, das in Speicher RS verschoben werden kann. In diesem Beispiel ist es das Constraint c_3. Da das Constraint c_5 zwar verantwortlich für den Konflikt war, aber nicht in den Speicher RS verschoben werden kann, wird c_5 nochmals in den Speicher US hinzugefügt, entsprechend der Regel 4 der Backwardregel (Durchlauf 8). Dadurch ergibt sich folgende Situation: $c_1 \in AS$, c_2, c_3, $c_4 \in RS$ und $c_5 \in US$. Die Definitionsbereiche werden entsprechend dem übrig gebliebenen Constraint im Speicher AS eingeschränkt.

Eine ähnlich Situation erfolgt ab Durchlauf 9 - 11. Allerdings werden nicht nur die Constraints mit der Priorität $level(c_i) = 0$ in den Speicher US verschoben, sondern auch die Constraints höherer Ordnung. Grund: Die Bestimmung einer schlechteren Konfiguration ist nur vom Speicher RS abhängig. Dadurch können auch Constraints $level(c_i) > 0$ wieder reaktiviert werden, falls das Besser Predikat erfüllt ist.

Durchlauf	Konfiguration $\Phi = \langle AS \bullet RS \bullet US \rangle$	$d(x)$	$d(y)$	Regel	Konfikt
0	$\{\} \bullet \{\} \bullet \{c_1\}$	$1 \ldots 10$	$1 \ldots 10$	fw	
1	$\{c_1\} \bullet \{\} \bullet \{\}$	$5 \ldots 10$	$5 \ldots 10$		
	$\{c_1\} \bullet \{\} \bullet \{c_2\}$	$5 \ldots 10$	$5 \ldots 10$	fw	
2	$\{c_1, c_2\} \bullet \{\} \bullet \{\}$	$\emptyset$	$\emptyset$	**bw**	$\surd$
3	$\{c_1\} \bullet \{c_2\} \bullet \{\}$	$5 \ldots 10$	$5 \ldots 10$		
	$\{c_1\} \bullet \{c_2\} \bullet \{c_3\}$	$5 \ldots 10$	$5 \ldots 10$	fw	
4	$\{c_1, c_3\} \bullet \{c_2\} \bullet \{\}$	9,10	5,6		
	$\{c_1, c_3\} \bullet \{c_2\} \bullet \{c_4\}$	9,10	5,6	fw	
5	$\{c_1, c_3, c_4\} \bullet \{c_2\} \bullet \{\}$	$\emptyset$	$\emptyset$	**bw**	$\surd$
6	$\{c_1, c_3\} \bullet \{c_2, c_4\} \bullet \{\}$	9,10	5,6		
	$\{c_1, c_3\} \bullet \{c_2, c_4\} \bullet \{c_5\}$	9,10	5,6	fw	
7	$\{c_1, c_3, c_5\} \bullet \{c_2, c_4\} \bullet \{\}$	$\emptyset$	$\emptyset$	**bw**	$\surd$
8	$\{c_1\} \bullet \{c_2, c_3, c_4\} \bullet \{c_5\}$	$5 \ldots 10$	$5 \ldots 10$	fw	
9	$\{c_1, c_5\} \bullet \{c_2, c_3, c_4\} \bullet \{\}$	$5 \ldots 7$	$5 \ldots 10$		
	$\{c_1, c_5\} \bullet \{c_2, c_3, c_4\} \bullet \{c_6\}$	$5 \ldots 7$	$5 \ldots 10$	fw	
10	$\{c_1, c_5, c_6\} \bullet \{c_2, c_3, c_4\} \bullet \{\}$	$\emptyset$	$\emptyset$	**bw**	$\surd$
11	$\{\} \bullet \{c_1, c_2\} \bullet \{c_3, c_4, c_5, c_6\}$	$1 \ldots 10$	$1 \ldots 10$	fw	
12	$\{c_5\} \bullet \{c_1, c_2\} \bullet \{c_3, c_4, c_6\}$	$1 \ldots 7$	$1 \ldots 10$	fw	
13	$\{c_5, c_6\} \bullet \{c_1, c_2\} \bullet \{c_3, c_4\}$	$1 \ldots 7$	$1 \ldots 8$	fw	
14	$\{c_5, c_6, c_3, \} \bullet \{c_1, c_2\} \bullet \{c_4\}$	$3 \ldots 7$	$1 \ldots 8$	fw	
15	$\{c_5, c_6, c_3, c_4\} \bullet \{c_1, c_2\} \bullet \{\}$	$3 \ldots 6$	$1 \ldots 8$	fw	

Tabelle 3.1: Auflösungstabelle

Zur Lösung der gestellten Aufgabe, können die Constraints c_5, c_6, c_3 und c_4 ohne gegenseitige Beeinflussung verwendet werden. Das Ergebnis, das mit Hilfe des Auflösungsalgorithmus gelöst wurde, besitzt folgende Konfiguration:

$$\Phi = \langle \{c_5, c_6, c_3, c_4\} \bullet \{c_1, c_2\} \bullet \{\} \rangle$$

Alle Constraints im Speicher AS sind erfüllt, da es eine gültige Wertzuweisung der Variablen gibt. Die Variablen x und y besitzen dabei folgende eingeschränkte Definitionsbereiche:

$$d(x) = [3 \ldots 6] \quad d(y) = [1 \ldots 8].$$

Die Constraints c_1, c_2 befinden sich im Speicher RS, da es für diese keine gültigen Definitionsbereiche der Variablen im Speicher AS gibt.

Durchlauf	Konflikt-speicher $CS_j \subseteq AS_j$	Konflikt-konfiguration $\Phi_{conf} \subseteq \Phi$	schlechtere Konfiguration $succ(\Phi_{conf})$	Reset Menge $Reset \subseteq AS$
2	$\{c_1, c_2\}$	$\Phi_{CS_2} = \{c_1, c_2\} \bullet \{\} \bullet \{\}$	$\{\} \bullet \{c_2\} \bullet \{\}$	$\{\}$
5	$\{c_1, c_3, c_4\}$	$\Phi_{CS_2 \cap CS_5} = \{c_1, c_3, c_4\} \bullet \{c_2\} \bullet \{\}$	$\{\} \bullet \{c_2, c_4\} \bullet \{\}$	$\{\}$
7	$\{c_1, c_3\}$	$\Phi_{CS_2 \cap CS_5 \cap CS_7} = \{c_1, c_3\} \bullet \{c_2, c_4\} \bullet \{\}$	$\{\} \bullet \{c_2, c_3, c_4\} \bullet \{\}$	$\{c_5\}$
10	$\{c_1\}$	$\Phi_{CS_2 \cap CS_5 \cap CS_7 \cap CS_{10}} = \{c_1\} \bullet \{c_2, c_3, c_4\} \bullet \{\}$	$\{, c_3, c_4\} \bullet \{c_1, c_2\} \bullet \{\}$	$\{c_5, c_6\}$

Tabelle 3.2: Mengen zur Bestimmung einer alternativ Konfiguration

3.5.4 Ausblick

3.5.4.1 Vor- und Nachteile

Die Constraint Programmierung zeichnet sich gegenüber anderen Programmiersprachen vor allem durch die folgenden Vorteile aus:

- Die Aufgabenbeschreibung kann intuitiv angegeben werden.
- Die Lösungsbeschreibung ist transparent.
- Das System kann durch einfaches Hinzufügen von Constraints erweitert werden.
- Constraints können zur Laufzeit generiert werden.
- Die Priorität eines Constraints kann durch Angabe von Werten festgelegt werden.
- Die Lösungsberechnung übernimmt ein Auflösungsalgorithmus.

Die Constraint Programmierung hat natürlich auch Nachteile:

- Die Constraint Programmierung ist nicht für schnelle Entscheidungen geeignet.
- Der Auflösungsalgorithmus kann sehr kompliziert und aufwendig zu programmieren sein.

Da das Bestimmen eines geeigneten Auflösungsalgorithmus eine Schwierigkeit der Constraint Programmierung ist, wird die Constraint Programmierung nur in bestimmten Fällen eingesetzt. Zum einen wenn der Auflösungsalgorithmus einfach bestimmbar ist und zum anderen wenn es sich lohnt einen komplexen Auflösungsalgorithmus zu programmieren, weil viele ähnliche Anwendungen damit laufen sollen. Aber dennoch wird in Zukunft eine Weiterentwicklung dieser Programmiermethode zu erwarten sein.

3.5.4.2 Fahrgemeinschaftssystem

Sicherlich hat jede Programmiermethode ihre Vor- und Nachteile. Der entscheidende Vorteil der Constraint Programmierung ist jedoch die Einfachheit, mit der eine Aufgabe gelöst werden kann. Man gibt nicht an, wie eine Aufgabe gelöst werden soll, sondern welche Bedingungen nach der Bearbeitung der Aufgabe gelten sollen. Dadurch lassen sich in vielen Anwendungsgebieten, komplizierte und komplexe Aufgaben lösen.
Ein mögliches komplexes Anwendungsgebiet ist das Berechnen von Fahrgemeinschaften. Eine Fahrgemeinschaft ist eine Gruppe von Personen, die mit einem Fahrzeug einen gemeinsamen Weg zurücklegen will. Zum Bilden einer Fahrgemeinschaft müssen verschiedene Bedingungen berücksichtigt werden:

1. Der **Umweg** einer FGM ist der nach der Wegstrecke berechnete zusätzliche Weg.
2. Die **Personenzahl** einer FGM gibt an, wieviele Personen in der Fahrgemeinschaft mitfahren.
3. Die **Eigenschaften** einer Person geben Auskunft über Geschlecht, Raucher oder Nichtraucher, Musikgeschmack usw.

Diese Bedingungen können mit Constraints einfach beschrieben werden.

$Constraint_{fgm1}=$ Umweg $\leq$ einer Schranke k

$Constraint_{fgm2}=$ Personenanzahl im Auto $\leq$ einer Schranke m

$Constraint_{fgm3}=$ Eigenschaften$(Person_i)$ = Eigenschaft$(Person_j)$

Der Auflösungsalgorithmus berechnet dann aufgrund der oben definierten Constraints eine mögliche Fahrgemeinschafteneinteilung.

Kapitel 4

Anforderungsanalyse

Die Anforderungsanalyse spiegelt die Eigenschaften einer aus der Sicht des Kunden optimalen Software zur Lösung seines Problems wider. Während der Anforderungsanalyse wird versucht, alle Anforderungen und Wünsche, die der Kunde an die Software hat, zu erfassen. Das Dokument enthält nur Anforderungen, die für den Kunden wichtig sind. Deshalb ist die Menge dieser Anforderungen weder vollständig, noch sind Aussagen über deren Realisierbarkeit getroffen worden. Es wird sich erst in der Spezifikation zeigen, welche der gewünschten Anforderungen in das Endprodukt eingehen.

Szenarien dienen dazu, die Anforderungen an das System darzustellen und zu vervollständigen. Sie beschreiben Situationen, in denen der zukünftige Benutzer mit dem System umgeht. In den folgenden Abschnitten wird jeweils ein solches Szenario erläutert.

4.1 Neuer Fahrgemeinschafts-Teilnehmer

Eine Person kommt in eine FGM-Zentrale und möchte als neues Mitglied in einer Fahrgemeinschaft aufgenommen werden. Dazu müssen zunächst Personalien, persönliche Eigenschaften (z.B. Auto, Führerschein, Raucher, usw.), aber auch Abneigungseigenschaften (Musikgeschmack, männlich/weiblich, Raucher/Nichtraucher, usw.) über eine Eingabemaske erfaßt werden. Zusätzliche Angaben, wie z.B. Person A sollte bzw. sollte nicht in der Fahrgemeinschaft sein, können ebenfalls erfaßt werden.

Nachdem alle notwendigen Angaben eingegeben sind, kann eine entsprechende FGM gesucht werden. Hierzu kann folgendermaßen vorgegangen werden:

1. Manuell

   Hier kann eine Person von Hand in eine bestehende oder neue Fahrgemeinschaft eingefügt werden. Dazu wird eine Liste aller Fahrgemeinschaften ausgegeben, die auch durch einen Filter eingeschränkt werden kann. Danach kann die Person einer FGM zugeordnet werden, indem sie über den

Index angewählt wird. Anschließend kann mit Hilfe einer Bewertungsfunktion überprüft werden, wie gut oder wie schlecht diese FGM ist. Vorübergehend kann erstmal ein Platz für diese Person reserviert werden, damit sich die betroffenen Teilnehmer dieser FGM informieren und dazu äußern können. Nach Überprüfung muß der Benutzer entscheiden, ob diese FGM in die Partition mitaufgenommen wird oder nicht.

2. Über ein 'Suchsystem'.

   Wird der Menüpunkt 'Suchsystem' aufgerufen, bekommt man für die neue Person eine Liste von Fahrgemeinschaften angezeigt, die für diese Person in Frage kommen. Dabei werden die Fahrgemeinschaften herausgesucht, die gut zu der Person passen, abhängig von der Bewertungsfunktion (z.B. ähnliche Start- und Zielorte). Danach kann die Person einer FGM zugeordnet werden, indem sie über den Index angewählt wird. Auch hier entscheidet der Benutzer, ob die gewählte FGM in die Partition mitaufgenommen wird oder nicht.

4.2 System-Aufbau

Normalerweise werden Fahrgemeinschaften in einer FGM-Zentrale von Hand (mit Fähnchen und Landkarten) ausgetüftelt. Natürlich will man auch hier eine Software-Lösung, die diese Arbeit abnimmt. Die Basis eines solchen Systems bilden die Personen- und Verkehrsdaten.

Personendaten können zum einen über ein spezifiziertes Datenformat (inkl. Eigenschaften) eingelesen bzw. über Tastatur und eine Eingabemaske eingegeben werden.

Verkehrsdaten können auf zwei Arten eingelesen werden:

1. GDF-Daten können eingelesen werden.
2. Mit Hilfe eines Graph-Editors (Graphlet, Leda, ...) können Verkehrsdaten manuell erstellt werden.

Das System besitzt eine weitere Möglichkeit, Daten einzugeben und zwar Partitionen von Personen, d.h. es können Fahrgemeinschaften mit allen Eigenschaften (Personen, wer ist Fahrer, Route usw.) eingegeben werden.

Der Benutzer kann die Daten mit Hilfe einer Visualisierung auf dem Bildschirm bzw. einem Ausdruck überprüfen. Dabei werden die Personendaten in einer Tabelle und die Verkehrsdaten mit Hilfe eines Editors, der automatisch aufgerufen wird, angezeigt.

4.3 Änderung eines Fahrgemeinschafts-Teilnehmers

Natürlich kommt es auch bei FGM-Teilnehmern vor, daß sie sich persönlich bzw. örtlich verändern, d.h. im Klartext für das System: Die Eigenschaften

eines FGM-Teilnehmers müssen geändert werden.

Nach Durchführung der Änderungen meldet das System, wie sich die Qualität der betroffenen Fahrgemeinschaft bezüglich der Bewertungsfunktion geändert hat.

4.4 Änderung einer Fahrgemeinschaft

Die Adresse eines FGM-Teilnehmers hat sich geändert. Um Konflikte auszuschließen, wird die FGM aufgelöst. Da alle Teilnehmer der bisherigen FGM auch in Zukunft noch an einer FGM teilnehmen wollen, müssen neue FGMs gefunden werden. Dazu wird festgestellt, welche Teilnehmer überhaupt diese FGM gebildet haben. Nun wird für jeden Teilnehmer eine Liste von Fahrgemeinschaften angezeigt, die bezüglich der Bewertungsfunktion gut zu ihm passen würden. Dabei kann jeder Teilnehmer einer neuen FGM zugeordnet werden, indem er über den Index angewählt wird. Vorübergehend kann erstmal ein Platz für diesen Teilnehmer reserviert werden, damit sich die betroffenen Teilnehmer dieser neuen FGM informieren und dazu äußern können. Die Liste der betroffenen Teilnehmer wird ausgegeben.

Damit der Benutzer aber zunächst ein Gefühl bekommt, wie gut diese Fahrgemeinschaft letztendlich ist, kann er eine Bewertungsfunktion darauf anwenden. Diese berechnet einen Wert und gibt ihn auf dem Bildschirm aus. Danach muß der Benutzer entscheiden, ob diese Fahrgemeinschaft in die Partition mitaufgenommen werden soll oder nicht.

Weiterhin kann der Benutzer Partitionen mit und ohne Änderung miteinander vergleichen. Dazu müssen die einzelnen Partitionen getrennt voneinander mit Hilfe einer Bewertungsfunktion bewertet werden. Das Ergebnis wird wie gewohnt ausgegeben. Hier muß der Benutzer entscheiden, welche Partition die aktuelle werden soll.

4.5 Eine neue Partition

Szenario

Da sich einiges geändert hat, z.B. viele neue Daten in das System integriert wurden, möchte der Benutzer eine völlig neue Partition erstellen.

Vorgehen

Zuerst sichert er die aktuelle Partition mit Hilfe eines Menupunkts “aktuelle Partition sichern“ auf einen Datenträger. Diese Partition kann später mit “Partition laden“ wiederhergestellt werden, ganz allgemein können mehrere Partitionen auf dem Datenträger verwaltet werden. Dann wählt er einen Menupunkt „Partition löschen“. Nach einer Sicherheitsabfrage werden alle aktuellen Fahrgemeinschaften zerschlagen, ausgenommen diejenigen, die auf jeden Fall erhalten bleiben

sollen (wurden vom Benutzer vorher markiert). Das System befindet sich in einem Zustand, als wären die Personendaten gerade erst eingelesen worden. Nun wählt der Benutzer einen Menupunkt „Partition erstellen“. Das System sucht nun mit heuristischen Algorithmen nach einer in bezug auf die eingestellte Bewertungsfunktion möglichst guten Lösung des Problems.

4.6 Die optimale Lösung

Szenario:

Das Fahrgemeinschaftensystem (FGM-System) hat einen kleinen Datenstamm an Verkehrs- und Kundendaten eingelesen. Der Benutzer möchte nun eine optimale Partition auf diesem Datenstamm berechnen.

Vorgehen:

Das FGM-System bietet dem Benutzer den Menüpunkt „optimale Lösung berechnen“. Er wählt diese aus und zusätzlich noch den Menüpunkt „verbose mode on“. Das System beginnt mit der Berechnung der optimalen Partition (bezogen auf die aktuelle Bewertungsfunktion) und gibt währenddessen Ausgaben auf den Bildschirm und in ein Logfile aus, die Aufschluß über den Fortgang der Berechnung geben. Da das Problem sicherlich NP-hart ist, und der Benutzer eine Mittagspause machen möchte, wählt er den Menüpunkt „Berechnung unterbrechen“ aus. FGM unterbricht die Berechnung und merkt sich den aktuellen Stand.

Nach der Mittagspause möchte der Benutzer die Berechnung fortsetzen. Er wählt den Menüpunkt „mit letzter Berechnung fortfahren“, FGM rechnet weiter und gibt nach einer gewissen Zeit die optimale Partition auf dem Bildschirm aus.

4.7 Inkrementelle Verbesserung von Partitionen

Es soll auch die Verbesserung von schon vorhandenen Partitionen möglich sein. Die Startpartition wird aus einer Datei eingelesen und durch Umsetzen von Personen inkrementell verbessert. Dadurch ergibt sich eine Partitionenfolge, die bei Erreichen einer bestimmten Qualität abgebrochen wird.

4.8 Festlegung der Bewertungsfunktion

Szenario:

Der Benutzer möchte die Bewertungsfunktion angeben.

Vorgehen:

FGM bietet ihm die Möglichkeit, in einer dokumentierten Notation Gewichte für die unten aufgeführten Bereiche zu verteilen. Die Gewichte sind Parameter einer fest vorgegebenen Bewertungsfunktion.

- Wegstrecke/Umwege: ähnliche Start-/Zielorte der Personen.
- Arbeitszeiten: ähnliche Arbeitszeiten der Personen.
- Eigenschaftsabneigungen: gewünschte Eigenschaften der Mitfahrer, die Menge der Eigenschaftsabneigungen ist erweiterbar.
- Explizite Zuneigungen: jemand will unbedingt mit einer bestimmten Person zusammenfahren.
- Explizite Abneigungen: analog

4.9 Kürzeste Wegestrecke

Szenario:

Der Benutzer hat seinen Arbeitstag beendet und möchte noch auf eine Party. Er möchte nun das FGM-System zur Berechnung des kürzesten Wegs einsetzen.

Vorgehen:

Er wählt den Menüpunkt „Kürzeste Wegesuche“ aus. Dann gibt er als Startort seinen Arbeitsplatz und als Zielort die Party als postalische Adresse an. Das FGM-System berechnet ihm den kürzesten Weg, zeigt seine Länge und eine Wegbeschreibung als Liste auf dem Bildschirm an. Auf Wunsch zeigt es den Weg mit Hilfe eines externen Graphen-Viewer an und gibt ihn in ein Postscriptfile aus.

4.10 Neuer Algorithmus

Szenario:

Friedhelm hat einen neuen Algorithmus erfunden, um einen kürzesten Weg zu finden. Diesen möchte er nun im FGM-System verwenden.

Vorgehen:

Er implementiert den Algorithmus in der imperativen, objektorientierten Sprache, in der auch der entsprechende Algorithmus des FGM-Systems geschrieben

ist. Dann ersetzt er das entsprechende Modul mit seinem Algorithmus. Er muß nur an wenigen, dokumentierten Stellen etwas ändern. Danach startet er das FGM-System, das nun bei der Bestimmung kürzester Wege den neuen Algorithmus benützt. Im Lieferumfang befinden sich Tools, mit denen er die Laufzeit verschiedener Module berechnen kann. Damit kann er nun die Effizienz seines neuen Algorithmus überprüfen.

4.11 Hilfesystem

Szenario:

Ein Benutzer, der bisher noch nie mit dem FGM-System gearbeitet hat, setzt sich an ein Terminal, auf dem es läuft.

Vorgehen:

Zu jedem Zeitpunkt der Benutzung des Fahrgeminschaftensystems gibt es unter den Menüpunkten einen, der "Hilfe" heißt. Nach der Auswahl dieses Menüpunkts wird ihm ein Text angezeigt, aus dem er entnehmen kann, wo er sich im System befindet, was er hier machen kann, wo er von hier aus hinkommt und von wo aus er hier hin gekommen sein kann.

4.12 Funktionale Anforderungen

4.12.1 Personen

1. Entstehung
   - Eingabe durch den Benutzer.
   - Automatische Generierung einer bestimmten Anzahl von Personen mit zufälligen Eigenschaften. Dabei können verschiedene Wahrscheinlichkeitsverteilungen für die Eigenschaften angegeben werden (Gleichverteilung oder Normalverteilung mit Parametern).
   - Eingabe über eine Schnittstelle.
2. Attribute
   - Name, Adresse
   - Geburtsdatum
   - Start- und Zielort
   - Arbeitszeiten
     - Beginn und Ende der Arbeitszeit als Zeitpunkte oder -intervalle (Gleitzeit)
     - Für die einzelnen Wochentage sind unterschiedliche Arbeitszeiten möglich.

- zu welcher Fahrgemeinschaft gehört eine Person
- gehört die Person fest oder vorläufig zu einer Fahrgemeinschaft
- Eigenschaften von Personen (als Teilmenge der Personenattribute)
  - männlich/weiblich
  - Raucher/Nichtraucher
  - Musikgeschmack
  - kann und will fahren
  - Komfortklasse des eigenen Autos
- Präferenzen von Personen
  - männlich/weiblich
  - Raucher/Nichtraucher
  - Musikgeschmack
  - geforderte Komfortklasse des Autos
- Menge der Eigenschaften und Präferenzen ist erweiterbar.
- Eigenschaften und Präferenzen einer Person sind änderbar.
- Für die Präferenzen kann jede Person Prioritäten angeben (z.B. Musik egal, aber auf keinen Fall Raucher als Mitfahrer).
- Beziehungen zwischen Personen
  - explizite Abneigung (jede Person kann eine Menge von Personen angeben, mit denen sie nicht zusammenfahren möchte)
  - explizite Zuneigung (Personen, mit denen man zusammenfahren möchte)

4.12.2 Fahrgemeinschaften

1. Entstehung

   - Eingabe durch den Benutzer, wobei zwei Fälle zu unterscheiden sind: Eingabe von kompletten Fahrgemeinschaften oder Hinzunahme von Personen in bestehende Fahrgemeinschaften.
   - Eingabe über eine Datei
   - Automatische Generierung aus einer Personenmenge P unter Berücksichtigung bereits bestehender Fahrgemeinschaften (Partition M über einer Teilmenge von P) und einer Bewertungsfunktion f.

2. Löschen

   - manuelles Löschen
   - Auflösen aller Fahrgemeinschaften

Anforderungen

- Anzahl der vom Fahrer angebotenen freien Plätze
- Fahrer in einer Fahrgemeinschaft
- Komfortklasse des Autos, mit dem die Fahrgemeinschaft fährt
- welche Personen gehören zu einer Fahrgemeinschaft (mit Status)
- Startort und Zielort einer Fahrgemeinschaft
- Route
- Zeitplan für das Aufnehmen und Absetzen von Personen
- Entstehungszeitpunkt der Fahrgemeinschaft
- Datum der letzten Änderung der Fahrgemeinschaft
- Markierung, ob diese Fahrgemeinschaft erhalten werden soll

4.12.3 Partitionen

Eine Partition ist eine Einteilung einer Personenmenge in Fahrgemeinschaften. Im FGM-System können mehrere Partitionen verwaltet werden, wobei immer eine Partition als die aktuelle Partition gilt.

Entstehung einer Partition Aus einer Personenmenge P und einer Bewertungsfunktion f wird eine Partition M berechnet, die optimal bezüglich f ist oder eine bestimmte Qualitätsschranke überschreitet. Dabei können verschiedene Algorithmen angewendet bzw. neue Algorithmen eingesetzt werden.

4.12.3.1 Änderung einer Partition

Eine neue Personenmenge P' wird anhand von einer Partition M über der Personenmenge P ($P' \cap P = \emptyset$) und einer Bewertungsfunktion auf die Fahrgemeinschaften von M aufgeteilt. Dabei kann man zwischen zwei Verfahren unterscheiden. Beim inkrementellen Verfahren werden die Personen aus P' einzeln in die bestehenden Fahrgemeinschaften eingefügt. Beim optimalen Verfahren wird eine Partition von P' berechnet und diese zur Partition M hinzugefügt.

4.13 Weitere Anforderungen

4.13.1 Anforderungen unter dem Aspekt Graphen

4.13.1.1 Eingabe von Verkehrsgraphen

Verkehrsgraphen können über einen noch auszuwählenden Grapheneditor (LEDA, GraphEd, Graphlet ...) eingegeben werden. Das Dateiformat für Graphen

wird dementsprechend festgelegt. Außerdem können GDF-Daten in dieses Graphenformat konvertiert werden. Die Detaillierung und Attributierung des Verkehrsgraphen hängt dabei von den zur Verfügung gestellten Daten ab. Als Kantenbeschriftungen sind Straßenlängen, Straßennamen und Straßenklassen (Autobahn, Bundesstraße...) vorgesehen. Zur Bestimmung der Fahrtzeit trägt jede Kante außerdem einen Widerstand (Maß für den Durchsatz), die Fahrtzeit ergibt sich dann aus Widerstand*Länge. Die Knoten tragen (x, y)-Koordinaten und Kreuzungsnamen.

4.13.1.2 Hierarchische Verkehrsgraphen

Aus dem flachen Verkehrsgraphen soll zur Beschleunigung der Wegsuche ein hierarchischer Verkehrsgraph aufgebaut werden. Dafür kommen zwei Modelle in Frage. Beim Levelgraph erhält man eine Hierarchie durch Ausblenden bestimmter Straßenklassen bzw. Kanten (z.B. Autobahn, Bundesstraße, Siedlungsstraße ...) auf den verschiedenen Ebenen. Bei einer Knotenhierarchie werden benachbarte Knoten in einem übergeordneten Level zu einem neuen Knoten zusammengefaßt. Die Knoten stellen je nach Level z.B. Kreuzungen, Stadtteile oder Ortschaften dar. Beim Aufbau des hierarchischen Graphen kann die Laufzeit gemessen werden.

4.13.1.3 Wegsuche auf Verkehrsgraphen

Es wird ein Algorithmus zur Bestimmung des kürzesten Weges zwischen zwei Orten A und B implementiert. Die Eingabe von Start- und Zielknoten erfolgt über Knotenkennungen oder Koordinaten. Außerdem kann der kürzeste Weg zwischen zwei Kanten bestimmt werden (Eingabe der Straßennamen). Die Wegsuche erfolgt auf dem hierarchischen oder flachen Verkehrsgraphen. Der berechnete Weg wird mit Hilfe des Grapheneditors visualisiert. Außerdem ist eine Laufzeitmessung möglich. Der Benutzer wird im verbose-Mode durch Bildschirmmeldungen über den aktuellen Stand der Berechnung informiert.

Der Algorithmus zur kürzesten Wegesuche soll als Teilmodul implementiert werden, so daß er auch unabhängig vom FGM-System eingesetzt werden kann. Außerdem soll dieses Modul leicht durch ein anderes ersetzbar sein.

4.13.1.4 Personengraph

Nach der Bestimmung der kürzesten Wege zwischen den Start- und Zielorten der Personen wird ein Personengraph aufgebaut. Die Personen werden als Knoten dargestellt, die Kantengewichte werden mit der Bewertungsfunktion bestimmt und stellen die abstoßende Kraft zwischen zwei Personen dar. Dieser Graph ist die Grundlage für Matchingverfahren.

4.13.1.5 Ausgabe über den Grapheneditor

Die graphische Darstellung des Verkehrsgraphen erfolgt über den Grapheneditor. Dieser dient nur zur Ausgabe, eine Interaktion des Benutzers zur Eingabe

von Start- und Zielorten ist nicht vorgesehen. Außerdem werden kürzeste Wege, Start- und Zielorte und die Einteilungen in Fahrgemeinschaften visualisiert. Dies hängt aber stark von den Fähigkeiten des verwendeten Editors ab.

4.13.1.6 Erweiterung

Später sollen Sammelpunkte für Fahrgemeinschaften möglich sein. Dazu muß der Verkehrsgraph eventuell um Parkplätze, Halteverbote und ähnliches erweitert werden. Außerdem sollte die Eingabe von Sammelpunkten möglich sein.

Kapitel 5

Spezifikation

5.1 Einführung

Dieser Abschnitt enthält die Spezifikation des Softwaresystems Mobidick (Mobil durch intelligentes Kombinieren) und wurde nach den IEEE-Richtlinien aus [9] erstellt. Es baut auf der Anforderungsanalyse auf und dient der Beschreibung des äußeren Systemverhaltens. Es ist damit die Grundlage für alle weiteren im Ablauf der Projektgruppe entstehenden Dokumente.

Die Spezifikation gliedert sich wie folgt: Kapitel 5.2 gibt einen allgemeinen Überblick über das System Mobidick. Kapitel 5.3 beschreibt die funktionalen Anforderungen, Kapitel 5.4 die Anforderungen an externe Schnittstellen. In Kapitel 5.5 werden die Leistungsanforderungen beschrieben und in Kapitel 5.6 die zukünftigen Erweiterungen. Im letzten Kapitel (5.7) sind alle Systemmeldungen aufgelistet.

5.2 Allgemeine Beschreibung

5.2.1 Umgebung des Produkts

Das Softwaresystem Mobidick soll unter dem Betriebssystem Solaris 2.5 laufen. Die Ausgabe des Programms erfolgt auf dem Bildschirm oder in eine Datei, die Eingabe über die Tastatur oder eine Maus. Weitere Peripherie wird nicht benötigt.

Zum System gehört ein Tool namens GDF2GRA, um Verkehrsdaten aus dem GDF-Format in das Mobidick-Verkehrsdatenformat zu konvertieren. Das GDF-Format ist in der Dokumentation zu GDF2GRA beschrieben. Personendaten können über ein spezielles Format aus Dateien eingelesen werden. Dieses Format ist in der Dokumentation zu Mobidick beschrieben.

Eine direkte Schnittstelle zum Drucker ist nicht vorgesehen. Die Programmausgaben auf dem Bilschirm erfolgen rein textuell. Eine Schnittstelle zu einem später zu entwickelndem Fenstersystem ist vorgesehen.

5.2.2 Informelle Beschreibung der Funktionalität

In diesem Abschnitt wird informell beschrieben, welche Funktionen von Mobidick bereitgestellt werden. Eine genaue Beschreibung kann Kapitel 5.3 entnommen werden. Die Interaktion zwischen Benutzer und Programm erfolgt über Menüs.

Zu Programmbeginn erscheint das Hauptmenü mit den Funktionen Dateien, Personen, Vermittlung, Wegsuche, Bewertungsfunktion, Hilfe und Ende. Diese Untermenüs enthalten die folgenden Funktionalitäten:

- Im Untermenü **Dateien** findet sich die notwendigen Funktionen zum Laden und Speichern der Stammdaten. Eine gewisse Konstistenzprüfung findet statt.
- Daten einzelner Personen innerhalb der Stammdaten können im Untermenü **Personenverwaltung** eingefügt, verändert oder gelöscht werden. Es besteht die Möglichkeit, zu Testzwecken einen zufälligen Personendatensatz zu generieren.
- Unter **Vermittlung** kann man Einteilungen in Fahrgemeinschaften berechnen. Es stehen verschiedene heuristische und optimale Algorithmen zur Verfügung. Eine Berechnung kann abgebrochen und zu einem späteren Zeitpunkt fortgesetzt werden.
- Die **Wegsuche** bietet die Möglichkeit, kürzeste Wege zwischen zwei Punkten in dem Verkehrsgraphen zu berechnen. Auch hier können verschiedene Algorithmen verwendet und die Rechenzeit gemessen werden.
- Die **Bewertungsfunktion** ist die Grundlage für die Einteilung von Personen in Fahrgemeinschaften. In diesem Menü kann sie verändert, aus einer Datei gelesen oder gespeichert werden.
- Die **Hilfefunktion** zeigt für jedes Menü einen jeweils passenden Hilfetext an.

5.2.3 Charakteristika der Benutzer und Benutzerinnen

In der jetzigen Version kann davon ausgegangen werden, daß die Benutzer und Benutzerinnen von Mobidick über durchschnittliche Erfahrungen im Umgang mit Rechnern verfügen. Für die Erweiterung von Mobidick um Algorithmen werden Erfahrungen mit der Entwicklung in C++ vorrausgesetzt.

5.3 Funktionale Anforderungen

Die funktionalen Anforderungen werden in Form von Use Cases oder Szenarien formuliert. Ähnliche oder verwandte Anforderungen werden zu einem Use Case zusammengefaßt. In einem Use Case wird das Zusammenspiel zwischen einem Akteur, in diesem Fall dem Benutzer, und dem System für einen konkreten Anwendungsfall beschrieben.

5.3.1 Start des Fahrgemeinschaftensystems

Das Programm wird von einem Kommandozeileninterpreter aus durch Eintippen des Namens Mobidick gestartet. Weitere Optionen sich nicht notwendig und werden vom Programm ignoriert. Nach dem Start wird dem Benutzer das Hauptmenü am Bildschirm angezeigt und die in den Voreinstellungen (s. Abschnitt 5.3.11) angegebenen Dateien werden geöffnet. Der Benutzer ist selbst dafür verantwortlich, daß nicht mehrere gleichzeitig gestartete Programme auf die gleiche Personendatei zugreifen.

5.3.2 Menüstruktur

Die Menüstruktur des Systems ist folgendermaßen aufgebaut:

1. Dateien
    1. Personendateien
        i. Neu
        ii. Laden
        iii. Speichern
        iv. Speichern unter
        v. Schließen
        vi. Importieren
        vii. Zurück
        viii. Hauptmenü
        ix. Hilfe
    2. Verkehrsgraph laden
    3. Fahrgemeinschaftseinteilung
        i. Umbenennen
        ii. Duplizieren
        iii. Löschen
        iv. Zurück
        v. Hauptmenü
        vi. Hilfe
    4. Bewertungsfunktionen
        i. Laden
        ii. Speichern
        iii. Speichern unter
        iv. Zurück
        v. Hauptmenü
        vi. Hilfe
    5. Zurück
    6. Hauptmenü

7. Hilfe

2. Personen
    1. Neue Person
    2. Person ändern
    3. Person löschen
    4. Personen generieren
    5. Personen anzeigen
    6. Personeneigenschaften ändern
    7. Zurück
    8. Hauptmenü
    9. Hilfe
3. Fahrgemeinschaften
    1. Neuer Teilnehmer
        i. Manuell eintragen
        ii. Suchsystem
        iii. Zurück
        iv. Hauptmenü
        v. Hilfe
    2. Teilnehmer fest eintragen
    3. Teilnehmer löschen
    4. Fahrgemeinschaft eingeben
    5. Fahrgemeinschaft ändern
    6. Fahrgemeinschaft auflösen
    7. Fahrgemeinschaft bewerten
    8. Fahrgemeinschaft anzeigen
    9. Fahrgemeinschaft markieren/unmarkieren
    10. Zurück
    11. Hauptmenü
    12. Hilfe
4. Vermittlung
    1. Einteilung auswählen
    2. Systemmeldungen (ein/aus)
    3. Einteilung berechnen
    4. Fortfahren mit letzter Berechnung
    5. Laufzeitmessung (ein/aus)
    6. Einteilung bewerten
    7. Einteilung anzeigen

8. Einteilung auflösen
9. Zurück
10. Hauptmenü
11. Hilfe

5. Bewertungsfunktionen
    1. Neu
    2. Ändern
    3. Auswählen
    4. Anzeigen
    5. Zurück
    6. Hauptmenü
    7. Hilfe
6. Wegsuche
    1. Einzelwegsuche
    2. Auswahl des Algorithmus
    3. Laufzeit
    4. Systemmeldungen (ein/aus)
    5. Zurück
    6. Hauptmenü
    7. Hilfe
7. Voreinstellungen
8. Hilfe
9. Ende

5.3.3 Datenmodell

Mit dem Mobidick-System können sogenannte Personendateien verwaltet werden. Diese bestehen aus einer Menge P von Personendaten, Einteilungen M_i und den Einteilungen zugeordneten Bewertungsfunktionen f_i (siehe Abbildung 5.1). Die Personendaten einer Person bestehen aus einer eindeutigen Identifikationsnummer, Angaben zur Arbeitszeit und den Eigenschaften der Person. Eine Einteilung besteht aus einer Menge von Fahrgemeinschaften und einer Bewertungsfunktion, mit der diese eingeteilt wurden. Fahrgemeinschaften sind Teilmengen der Personenmenge und alle Fahrgemeinschaften einer Einteilung sind paarweise disjunkt. Die Vereinigung aller Fahrgemeinschaften einer Einteilung ergibt eine Teilmenge der Personenmenge.

Es kann immer nur eine Personendatei geöffnet sein, d.h. vor dem Öffnen einer anderen Personendatei muß die bereits geöffnete geschlossen werden. Alle zu einer Personendatei gehörenden Daten werden gemeinsam in einer weiteren Datei

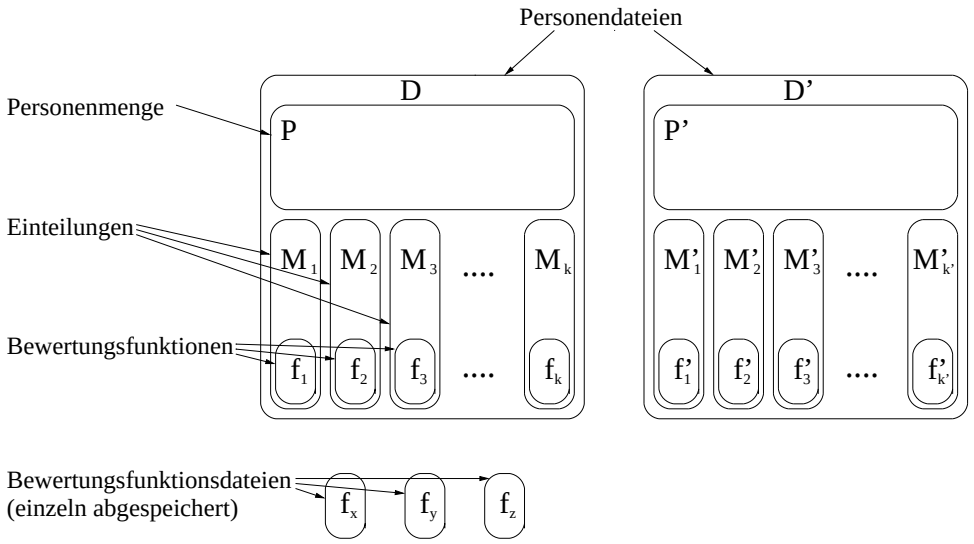


Abb. 5.1: Datenmodell

abgespeichert. Um dem Benutzer die Möglichkeit zu geben, eine Bewertungsfunktion aus einer Personendatei D in einer anderen Personendatei D' anzuwenden, kann er Bewertungsfunktionen einzeln in Dateien abspeichern. Solche unabhängig gespeicherten Bewertungsfunktionen können dann parallel zu einer Personendatei geöffnet werden (siehe Abbildung 5.1).

Eine in der Personendatei enthaltene und vom Benutzer bestimmte Einteilung wird als aktuelle Einteilung bezeichnen.

5.3.4 Beenden des Fahrgemeinschaftensystems

Zum Beenden des Programms muß der Befehl `Ende` aus dem Hauptmenü ausgewählt werden. Ist noch eine Personendatei geöffnet, dann wird der Benutzer gefragt „`Personendatei <Name> vor dem Beenden speichern? [J/n]`“. Für die Behandlung noch geöffneter Bewertungsfunktionen siehe Use Case 5.3.5.5.

5.3.5 Dateien

5.3.5.1 Neue Personendatei anlegen

Durch Auswahl des Menüpunkt `Datei-Personendateien-Neu` wird eine leere Personendatei angelegt. Nach Auswahl des Menüpunkts wird dem Benutzer eine Liste mit den im aktuellen Verzeichnis enthaltenen Personendateien angezeigt. Dann wird er aufgefordert den neuen Dateinamen anzugeben durch die Meldung „`Neuer Name:`“. Gibt der Benutzer einen schon existierenden Namen ein, so wird die Meldung „`Name existiert schon, trotzdem abspeichern und`

`vorhandene Datei überschreiben? [j/N]`“ angezeigt. Bei Eingabe von „j“ wird die unter diesem Namen existierende Datei überschrieben, bei „n“ wird die Eingabeaufforderung wiederholt.

An den Dateinamen wird vom System die Erweiterung `.per` angehängt, mit der die Datei als Personendatei gekennzeichnet wird.

Das System kann nur eine Personendatei im Hauptspeicher bereithalten. Wenn während der Auswahl von `Neu` schon eine Datei geöffnet ist, wird der Benutzer gefragt „`Personendatei <Name> vor dem Erstellen der neuen Datei speichern? [J/n]`“. Für die Behandlung noch geöffneter Bewertungsfunktionen siehe Use Case 5.3.5.5.

5.3.5.2 Personendatei laden

Nach Auswahl des Menüpunkts `Datei-Personendateien-Laden` wird eine Liste mit den im aktuellen Verzeichnis enthaltenen Personendateien angezeigt. Aus dieser Liste kann die gewünschte Datei ausgewählt werden.

Wenn während der Auswahl von `Laden` schon eine Datei geöffnet ist, wird der Benutzer gefragt „`Personendatei <Name> vor dem Laden der anderen Datei speichern? [J/n]`“. Für die Behandlung noch geöffneter Bewertungsfunktionen siehe Use Case 5.3.5.5.

Enthält die zu ladende Personendatei eine oder mehrere Einteilungen, dann werden nach dem Ladevorgang die erste Einteilung und deren Bewertungsfunktion als aktuell eingestellt.

5.3.5.3 Personendatei speichern

Durch Auswahl des Menüpunkt `Datei-Personendateien-Speichern` wird die aktuelle Personendatendatei unter dem ihr zugewiesenen Dateinamen abgespeichert.

5.3.5.4 Personendatei unter anderem Namen speichern

Nach Auswahl des Menüpunkts `Datei-Personendateien-Speichern unter` wird dem Benutzer eine Liste mit den im aktuellen Verzeichnis enthaltenen Personendateien angezeigt. Dann wird er aufgefordert den neuen Dateinamen anzugeben. Gibt der Benutzer einen schon existierenden Namen ein, so wird die Meldung „`Name existiert schon, trotzdem abspeichern und vorhandene Datei überschreiben? [j/N]`“ angezeigt. Bei Eingabe von „j“ wird die unter diesem Namen existierende Datei überschrieben, bei „n“ wird die Eingabeaufforderung wiederholt.

5.3.5.5 Personendatei schließen

Durch Auswahl des Menüpunkts `Datei-Personendateien-Schließen` wird die aktuelle Personendatei geschlossen. Dazu wird der Benutzer gefragt „`Datei vor`

`dem Schließen speichern? [J/n]`“. Wenn er sie unter einem anderen Namen abspeichern will, hat er die Möglichkeit, den Vorgang abzubrechen und dies durch Auswahl des Use Case 5.3.5.4 zu tun. Bei Abbruch des Vorgangs bleibt die aktuelle Datei unverändert im Speicher.

Sind noch Bewertungsfunktionen geöffnet, die seit der letzten Änderung nicht gespeichert wurden, so wird der Benutzer zu jeder Bewertungsfunktion gefragt „`Bewertungsfunktion <Name> vor dem Schließen speichern? [J/n]`“. Antwortet der Benutzer mit „j“, so wird die Bewertungsfunktion gespeichert, sonst gehen ihre Werte verloren.

5.3.5.6 Personendatei importieren

Durch das Importieren wird die aktuelle Personendatei um den Inhalt der auf dem Datenträger abgespeicherten Datei erweitert. Sei D die aktuelle Personendatei mit der Personenmenge P und den Fahrgemeinschaftseinteilungen $M_1, \ldots, M_k$ mit den zugehörigen Bewertungsfunktionen $f_1, \ldots, f_k$. D' sei die zu importierende Personendatei mit der Personenmenge P' und D'' die erweiterte Personendatei mit der Personenmenge P'; dann hat die Importierung folgende Auswirkungen:

- Die Personenmenge P'' ist die disjunkte Vereinigung der Personenmengen P und P', wobei die Personen-IDs in P' automatisch angepasst werden, damit keine ID doppelt vorkommt. Personen die in P und in P' enthalten sind werden in P'' nur einmal aufgeführt. Personen können durch den Namen und das Geburtsdatum eindeutig identifiziert werden.
- Die Menge der Fahrgemeinschaftseinteilungen in D'' enthält alle Einteilungen aus D und D'. Dabei werden die Personen-IDs der Einteilungen aus D' wie oben automatisch angepasst. Alle Personen aus P (P') sind in den Einteilungen aus D' (D) noch nicht vermittelt. Das bedeutet, daß alle Einteilungen aus D'' nur einen Teil der Personenmenge enthalten können. Will der Benutzer eine Fahrgemeinschaftseinteilung, die alle Personen berücksichtigt, so muß er entweder eine neue Einteilung berechnen oder eine der bestehenden Einteilungen erweitern (siehe 5.3.8.3).

Nach Auswahl des Menüpunkts `Datei-Personendateien-Importieren` wird der Benutzer durch die Systemmeldung „`Name:`“ aufgefordert, den Namen der zu importierenden Datei anzugeben.

Es können nur Dateien, die dem Personendateiformat entsprechen, importiert werden. Dieses Format wird im Enfwurf festgelegt.

5.3.5.7 Verkehrsdaten laden

Nach Auswahl des Menüpunkts `Datei-Verkehrsdaten Laden` wird eine Liste mit den im aktuellen Verzeichnis enthaltenen Verkehrsgraphen angezeigt. Aus dieser Liste kann die gewünschte Datei ausgewählt werden. Das System kann nur einen Verkehrsgraphen im Hauptspeicher bereithalten. Falls zum Zeitpunkt

des Lade-Befehls schon ein Graph geöffnet ist, so wird dieser vor Ausführung des Lade-Befehls geschlossen. Es können nur Verkehrsgraphen geladen werden, die dem vom System unterstützten Graphenformat entsprechen.

5.3.5.8 Einteilung umbenennen

Nach Auswahl des Menüpunkts `Datei-Einteilungen-Umbenennen` wird dem Benutzer eine Liste aller in der Personendatei enthaltenen Einteilungen angezeigt. Dann wird er aufgefordert, den neuen Einteilungsnamen anzugeben. Gibt der Benutzer einen schon existierenden Namen ein, so wird die Meldung „`Name existiert schon, trotzdem abspeichern und vorhandene Datei überschreiben? [j/N]`“ angezeigt. Bei Eingabe von „j“ wird die unter diesem Namen existierende Datei überschrieben, bei „n“ wird die Eingabeaufforderung wiederholt.

5.3.5.9 Einteilung duplizieren

Durch Auswahl des Menüpunkts `Datei-Einteilungen-Duplizieren` kann eine bestehende Einteilung dupliziert werden. Nach Auswahl des Menüpunkts wird dem Benutzer eine Liste aller in der Personendatei enthaltenen Einteilungen angezeigt und er wird aufgefordert, den Einteilungsnamen des Duplikats anzugeben. Gibt der Benutzer einen schon existierenden Namen ein, so wird die Meldung „`Name existiert schon, trotzdem abspeichern und vorhandene Datei überschreiben? [j/N]`“ angezeigt. Bei Eingabe von „j“ wird die unter diesem Namen existierende Datei überschrieben, bei „n“ wird die Eingabeaufforderung wiederholt. Beim duplizieren einer Einteilung werden alle Fahrgemeinschaften und die Bewertungsfunktion kopiert.

5.3.5.10 Einteilung löschen

Nach Auswahl des Menüpunkts `Datei-Einteilungen-Löschen` wird dem Benutzer eine Liste aller in der Personendatei enthaltenen Einteilungen angezeigt. Daraus kann er die zu löschende Einteilung auswählen. Nach der Auswahl wird er gefragt „`Einteilung wirklich löschen? [j/N]`“. Antwortet er mit „j“, so wird die ausgewählte Einteilung aus der Personendatei gelöscht.

5.3.5.11 Bewertungsfunktion laden

Mit diesem Menüpunkt kann eine Bewertungsfunktion, die zuvor in einer Datei abgespeichert wurde, in die Liste der Bewertungsfunktionen aufgenommen werden. Der Benutzer erhält so die Möglichkeit, Bewertungsfunktionen der Personendatei X abzuspeichern und dann auf eine Fahrgemeinschaftseinteilung der Personendatei Y anzuwenden.

Nach Auswahl des Menüpunkts `Datei-Bewertungsfunktionen-Laden` wird eine Liste mit den im aktuellen Verzeichnis enthaltenen Bewertungsfunktionsdateien angezeigt. Aus dieser Liste kann die gewünschte Datei ausgewählt werden.

5.3.5.12 Bewertungsfunktion speichern

Durch Auswahl des Menüpunkts `Datei-Bewertungsfunktionen-Speichern` wird die aktuelle Bewertungsfunktion unter dem ihr zugewiesenen Dateinamen abgespeichert.

5.3.5.13 Bewertungsfunktion unter anderem Namen speichern

Nach Auswahl des Menüpunkts `Datei-Bewertungsfunktionen-Speichern unter` wird dem Benutzer eine Liste mit den im aktuellen Verzeichnis enthaltenen Bewertungsfunktionsdateien angezeigt. Dann wird er aufgefordert, den neuen Dateinamen anzugeben. Gibt der Benutzer einen schon existierenden Namen ein, so wird die Meldung „`Name existiert schon, trotzdem abspeichern und vorhandene Datei überschreiben? [j/N]`“ angezeigt. Bei Eingabe von „j“ wird die unter diesem Namen existierende Datei überschrieben, bei „n“ wird die Eingabeaufforderung wiederholt.

5.3.6 Personen

5.3.6.1 Use Case: Neue Person eintragen

Durch Aufruf der Menüpunkte `Personen-Neue Person` wird der Benutzer durch mehrere Bildschirmmasken geführt. In der ersten Maske werden die Personeneigenschaften eingegeben. Auch bei den folgenden Bildschirmmasken bedeutet `(*)`, daß eine Wertangabe unbedingt notwendig ist. Wird bei solchen Feldern kein Wert angegeben, kann die Person zwar in den Personenbestand aufgenommen werden, wird aber mit dem Vermerk *Daten unvollständig* versehen. Felder, bei denen nichts eingegeben wurde, tragen den Wert `keine Angabe`.

1. `Name: (*)`
2. `Vorname: (*)`
3. `Geschlecht: (*)`
4. `Geburtsdatum:`
5. `Straße: (*)`
6. `Hausnummer: (*)`
7. `PLZ: (*)`
8. `Wohnort: (*)`
9. `Telefon:`
10. `email:`
11. `Raucher?:`
12. `Fahrer?:`

13. `Komfortklasse des Autos:`
14. `Baujahr:`
15. `Anzahl Plätze:`
16. `Musikgeschmack:`

Bei der Eingabe werden die Felder nacheinander abgefragt. Die Felder 13, 14 und 15 können freigelassen werden, sofern bei 12. `Fahrer?` „n“ eingegeben wurde, d.h. die neue Person kommt nicht als Fahrer in Frage. In diesem Fall sind die Angaben zum eigenen Auto unnötig. Bei 13. `Komfortklasse` wird zwischen `Kleinwagen, Mittelklasse` und `gehobener Klasse` unterschieden. Bei 15. `Anzahl Plätze` erhält man durch Drücken der Return-Taste den Defaultwert vier. Bei 16. `Musikgeschmack` können mehrere Musikrichtungen ausgewählt werden (`Klassik, Pop, Rock, Schlager`), außerdem gibt es noch die Alternativen `Ruhe` und `Egal`.

In der zweiten Maske werden die Daten für die gewünschte Fahrt eingegeben, für die eine Fahrgemeinschaft gesucht wird.

1. `Startort (Adresse): (*)`
2. `Startort (Kanten-ID):`
3. `Zielort (Adresse): (*)`
4. `Zielort (Kanten-ID):`
5. `Ankunftszeit (von): (*)`
6. `Ankunftszeit (bis): (*)`
7. `Rückfahrtzeit (von): (*)`
8. `Rückfahrtzeit (bis): (*)`
9. `Arbeitsdauer:`

Start- und Zielort können über Eingabe einer Adresse (Straße, Hausnummer, Ort) eingegeben werden. Dazu erscheinen nacheinander die Abfragen `Straße?`, `Hausnummer?` und `Ort?`. Drückt man bei `Ort` die Return-Taste, so erhält man den Defaultwert `Stuttgart`. Nach der Eingabe wird die Adresse daraufhin überprüft, ob sie im aktuellen Verkehrsgraphen vorhanden ist. Falls sie nicht vorhanden ist, erscheint die Fehlermeldung „`Fehler 3: Adresse nicht im Verkehrsgraphen vorhanden`“. Alternativ dazu ist die Eingabe einer Kanten-ID des Verkehrsgraphen möglich. Falls die ID nicht existiert, erscheint die Fehlermeldung „`Fehler 4: Kante nicht im Verkehrsgraphen vorhanden`“.

Wurde ein Adresse oder eine Kante angegeben, die nicht im Verkehrsgraphen vorhanden ist, so kann die betroffene Person trotzdem gespeichert werden. Sie erhält dann den Status *Daten unvollständig*.

Bei der Ankunftszeit kann ein Intervall eingegeben werden, das auch die Länge null haben darf. Bei falscher Eingabe erscheint die Fehlermeldung „`Fehler 5:`

`Obere Grenze kleiner untere Grenze`". Für die Rückfahrtzeit kann ebenfalls ein Intervall eingegeben werden oder alternativ die gewünschte Arbeitsdauer. Die Rückfahrtzeit errechnet sich dann aus Ankunftszeit plus Arbeitsdauer.

In der dritten Maske werden die Wünsche in bezug auf die Mitfahrer eingegeben. Jeder Wunsch trägt einen Gewichtsfaktor von 0 bis 10. 0 steht für völlig unwichtig, 10 für sehr wichtig. Durch Belegung mit dem Gewicht 0 werden sämtliche Wünsche abgeschaltet. Alle Gewichte haben den Defaultwert 0, zu Wünschen mit dem Gewicht 0 muß nichts eingegeben werden.

1. `abgelehnte Musikrichtungen:`
2. `Gewicht:`
3. `Geschlecht:`
4. `Gewicht:`
5. `Raucher:`
6. `Gewicht:`
7. `gewünschte Komfortklasse:`
8. `Gewicht:`
9. `persönliche Abneigung: Person hinzufügen`
10. `persönliche Abneigung: Person löschen`
11. `persönliche Zuneigung: Person hinzufügen`
12. `persönliche Zuneigung: Person löschen`
13. `Person in Datenbestand übernehmen`
14. `Zurück`
15. `Hauptmenü`
16. `Hilfe`

Bei 1. `abgelehnte Musikrichtungen` können eine oder mehrere Musikrichtungen aus der Menge `Klassik, Pop, Rock, Schlager` angegeben werden. Bei 3. `Geschlecht` kann man angeben, ob man nur mit Männern oder nur mit Frauen fahren will. Bei 5. `Raucher` wird festgelegt, ob man nur mit Rauchern oder nur mit Nichtrauchern fahren will. Bei 7. `gewünschte Komfortklasse` sind die drei oben erwähnten Komfortklassen als Eingabe möglich, die angegebene Komfortklasse ist als Mindestanforderung zu verstehen.

Durch Aufruf von 9. `persönliche Abneigung: Person hinzufügen` wird direkt zur Filterfunktion von Use Case 5.3.6.5 übergegangen. Dort wird über den Index der ausgegebenen Personenliste eine Person ausgewählt, die dann in die Liste der abgelehnten Personen eingefügt wird. Danach befindet man sich wieder in der ursprünglichen Bildschirmmaske, in der diese Liste auch angezeigt wird (Personen-IDs). Bei dem Versuch, eine Person wiederholt einzufügen, erscheint

die Fehlermeldung „`Fehler 6: Person bereits vorhanden`“. Der Menüpunkt 11. `persönliche Zuneigung: Person hinzufügen` verhält sich analog.

Bei Aufruf von 10. `persönliche Abneigung: Person löschen` kann eine Personen-ID aus der angezeigten Liste eingegeben werden, die betreffende Person wird dann aus der Liste entfernt. Ist die ID nicht in der Liste vorhanden, so erscheint die Fehlermeldung „`Fehler 7: Person nicht in Liste vorhanden`“. Der Menüpunkt 12. `persönliche Zuneigung: Person löschen` verhält sich analog.

Nach Aufruf von 13. `Person in Datenbestand übernehmen` erscheint die Frage „`Person wirklich übernehmen? [J/n]`“. Bei Eingabe von „n“ geschieht überhaupt nichts, man befindet sich immer noch im vorherigen Menü.

Der Benutzer kann mit Pfeil rechts und Pfeil links jederzeit zwischen den drei Bildschirmmasken wechseln.

5.3.6.2 Use Case: Person ändern

Durch Aufruf der Menüpunkte `Personen-Person ändern` wird direkt zur Filterfunktion aus Use Case 5.3.6.5 übergegangen. Dort kann der Benutzer eine Person über die Personen-ID, Name und Vorname oder ein anderes Kriterium suchen lassen. Aus den gefundenen Personen wählt er über den Index eine aus. Danach erscheinen dieselben Bildschirmmasken wie in Use Case 5.3.6.1 und die Änderungen können vorgenommen werden. Die Änderungen werden erst wirksam, wenn der Menüpunkt `Person in Datenbestand übernehmen` ausgewählt wird. Falls sich Startort, Zielort, Zeiten oder das Feld `Fahrer?` geändert haben, erscheint die Meldung „`Änderungen für Fahrgemeinschaftsbildung relevant`“ und anschließend die Frage „`Änderungen vornehmen und Person aus den betroffenen Fahrgemeinschaften löschen? [J/n]`“. Falls durch die Änderung die Auflösung einer Fahrgemeinschaft notwendig wird (Person kann nicht mehr fahren), erscheint die Frage „`Änderungen vornehmen und Fahrgemeinschaft auflösen? [J/n]`“. Nach Eingabe von „j“ wird eine Liste der Personen ausgegeben, die von dieser Auflösung betroffen sind.

5.3.6.3 Use Case: Person löschen

Durch Aufruf der Menüpunkte `Personen-Person löschen` wird direkt zur Filterfunktion aus Use Case 5.3.6.5 übergegangen. Die Auswahl einer Person geschieht wie in Use Case 5.3.6.2. Falls die Person Fahrer einer Fahrgemeinschaft war, erscheint die Meldung „`Person ist Fahrer, löschen führt zur Auflösung einer Fahrgemeinschaft`“. Danach erscheint die Frage „`Person wirklich löschen? [J/n]`“. Bei Eingabe von „j“ wird die Person aus dem Datenbestand gelöscht, eine Liste der von der Auflösung betroffenen Personen angezeigt und zum Menü `Personenverwaltung` zurückgekehrt, bei „n“ befindet man sich sofort wieder im Menü `Personenverwaltung`.

5.3.6.4 Use Case: Personen generieren

Durch Aufruf der Menüpunkte `Personen-Personen generieren` kann man für Testzwecke eine Personenmenge zufällig zu erzeugen. Da die alte Personenmenge vor der Generierung gelöscht wird, erscheint zunächst die Frage „`Haben Sie die alte Personenmenge gesichert? [j/N]`“. Bei Eingabe von „n“ befindet man sich sofort wieder im Menü Personenverwaltung und der Benutzer hat die Gelegenheit, die Personendatei wie in Use Case 5.3.5.5 zu schließen. Nach Eingabe von „j“ kann der Benutzer folgende Parameter einstellen:

1. `Personenzahl: (*)`
2. `Anteil Fahrer: (*)`
3. `Startorte (Gleichverteilung): (*)`
4. `Startorte (Rechteck):`
5. `Startort (fest):`
6. `Zielorte (Gleichverteilung): (*)`
7. `Zielorte (Abstand zum Startort):`
8. `Zielort (fest):`
9. `Ankunftszeit: (von) (*)`
10. `Ankunftszeit: (bis) (*)`
11. `Ankunftszeit: (Intervallängen) (*)`
12. `Rückfahrtzeit: (von) (*)`
13. `Rückfahrtzeit: (bis) (*)`
14. `Rückfahrtzeit: (Intervallängen) (*)`
15. `Personenmenge generieren`
16. `Zurück`
17. `Hauptmenü`
18. `Hilfe`

Bei den Startorten kann zwischen einer Gleichverteilung auf dem ganzen Verkehrsgraphen oder in einem rechteckigen Bereich (Angabe links, rechts, oben, unten in Gauß-Krüger-Koordinaten) gewählt werden. Im letzteren Fall werden die vier benötigten Werte nacheinander vom Benutzer abgefragt. Außerdem ist die Eingabe eines festen Startorts möglich.

Bei den Zielorten gibt es ebenso die Gleichverteilung und alternativ kann der Abstand zum Startort normalverteilt generiert werden. Dazu werden Mittelwert und Standardabweichung nacheinander vom Benutzer abgefragt.

Die Ankunftszeiten unterliegen einer Gleichverteilung auf dem durch `von` und `bis` gegebenen Intervall. Bei falscher Eingabe erscheint die Fehlermeldung „`Fehler 5: Obere Grenze kleiner untere Grenze`“. In diesem Intervall liegen die einzelnen Ankunftsintervalle der Personen, deren Länge wird in 11. `Ankunftszeit: (Intervallängen)` angegeben.

Für die Rückfahrtzeiten gilt das entsprechende.

Nach Aufruf von 15. `Personenmenge generieren` wird für jede Person ein Start-, ein Zielort, ein Ankunftsintervall und ein Rückfahrtsintervall generiert, wobei die Intervallängen für alle Personen gleich sind. Alle hier nicht aufgeführten Personeneigenschaften sind nicht vom Benutzer beeinflußbar und werden automatisch generiert. Nach der Berechnung erscheint die Frage „`Personenmenge übernehmen? [J/n]`“. Bei Eingabe von „j“ wird die generierte Personenmenge als aktuelle Personenmenge übernommen, bei „n“ befindet man sich wieder in obigem Menü und kann die Parameter verändern.

5.3.6.5 Use Case: Personen anzeigen

Durch Aufruf der Menüpunkte `Personen-Personen anzeigen` ist die Personensuche über eine Filterfunktion möglich. Dabei können folgende Suchkriterien angegeben werden:

1. `Name:`
2. `Vorname:`
3. `Personen-ID:`
4. `Startort: (Straßenname)`
5. `Startort: (Radius)`
6. `Zielort: (Straßenname)`
7. `Zielort: (Radius)`
8. `Ankunftszeit: (von)`
9. `Ankunftszeit: (bis)`
10. `Rückfahrtzeit: (von)`
11. `Rückfahrtzeit: (bis)`
12. `Status: (vermittelbar/als reserviert eingetragen/fest eingetragen/Daten unvollständig)`
13. `FGM-ID:`
14. `Fahrer?:`
15. `Zurück`
16. `Hauptmenü`

17. `Hilfe`

Beim Startort kann ein Straßenname und ein Radius angegeben werden, in diesem Bereich soll dann der Startort der Person liegen. Analog beim Zielort. Für Ankunfts- und Rückfahrtzeit können Intervalle angegeben werden, bei falscher Eingabe erscheint die Fehlermeldung „`Fehler 5: Obere Grenze kleiner untere Grenze.`“ In diesem Intervall müssen die tatsächliche Ankunfts- und Rückfahrtszeit der Fahrgemeinschaft liegen, nicht das von der Person angegebene Wunschintervall.

Beim Status wird zwischen folgenden Personengruppen unterschieden: Eine Person ist *vermittelbar*, falls sie bisher in keine Fahrgemeinschaft eingetragen wurde. Sie hat den Status *als reserviert eingetragen*, falls bereits ein Platz in einer Fahrgemeinschaft für sie reserviert wurde. Nimmt sie diesen Platz an, geht der Status über in *fest eingetragen*. Reichen die zu einer Person eingegebenen Daten für eine Vermittlung noch nicht aus, hat sie den Status *Daten unvollständig* (siehe Use Case 5.3.6.1). Das Feld `Fahrer?` entspricht dem gleichnamigen Feld in Use Case 5.3.6.1.

Bei den Suchkriterien können auch einzelne Eingabefelder freigelassen werden. Das System sucht dann nach allen Personen, die alle (UND-Verknüpfung) Kriterien erfüllen und gibt sie als Liste mit Index auf dem Bildschirm aus. Es werden tabellarisch angezeigt:

1. `Personen-ID:`
2. `Name:`
3. `Vorname:`
4. `Startort:`
5. `Zielort:`
6. `Ankunftszeitpunkt:`
7. `Rückfahrtszeitpunkt:`
8. `Status: vermittelbar/als reserviert eingetragen/fest eingetragen/Daten unvollständig`
9. `Fahrgemeinschafts-ID:`

Danach erscheint die Frage „`Tabelle in PostScript-Datei ausgeben? [j/N]`“. Durch Anwahl über die Indexnummer einer Person kann der Benutzer zwischen folgenden weitergehenden Informationen zu einer Person wählen:

1. `weitere Personeneigenschaften` (s. Use Case 5.3.6.1)
2. `Wünsche der Person` (s. Use Case 5.3.6.1)
3. `als Fahrer eingeteilt?`
4. `Datum der Eintragung der Person in das System`

Wird die Filterfunktion zur Auswahl einer bestimmten Person benutzt (z.B. in Use Case 5.3.6.2), führt die Angabe einer Indexnummer nicht zur Anzeige weitergehender Informationen. Der Vorgang ist dann mit der Auswahl abgeschlossen.

5.3.6.6 Use Case: Personeneigenschaften erweitern

Durch Aufruf der Menüpunkte `Personen-Personeneigenschaften erweitern` ist es möglich, eine weitere Personeneigenschaft hinzuzufügen. Durch Hinzunahme einer weiteren Personeneigenschaft tritt diese auch bei den Wünschen bezüglich der Mitfahrer auf. Für eine bereits vorhandene Personenmenge wird der Wert der neuen Eigenschaft zunächst offengelassen, dies entspricht dem Wert `keine Angabe`.

Es erscheint folgendes Menü:

1. `Name der neuen Eigenschaft:`
2. `Wertebereich:`
3. `eindeutiger Wert`
4. `Mehrfachauswahl`
5. `Eigenschaft hinzufügen`
6. `Zurück`
7. `Hauptmenü`
8. `Hilfe`

Bei 1. kann der Name der neuen Eigenschaft eingegeben werden. Dieser wird daraufhin überprüft, ob er nicht bereits schon für eine andere Eigenschaft verwendet wurde. Bei 2. können die einzelnen Werte in Form von Strings, getrennt durch Kommata eingegeben werden. Bei Mehrfacheingabe eines Wertes muß die Eingabe wiederholt werden. In 3. und 4. wird festgelegt, ob einer Person ein eindeutiger oder mehrere Werte aus dem Wertebereich zugeordnet werden. Diese Einstellung gilt dann auch für die Wünsche bezüglich der Mitfahrer. Durch Auswahl von 5. wird die neue Personeneigenschaft hinzugefügt.

5.3.7 Fahrgemeinschaften

5.3.7.1 Fahrgemeinschaften-Filter

Das Anzeigen von Fahrgemeinschaften kann durch Filter eingeschränkt werden. Dabei werden nach Index die Filterkriterien aufgeführt:

1. Anzahl der freien Plätze (min.)
2. Anzahl der Teilnehmer (max.)

3. Fahrgemeinschaften-ID
4. Komfortklasse des Autos (min.)
5. Startort (Punkt und Radius in km)
6. Zielort (Punkt und Radius in km)
7. Startzeit (Intervall)
8. Zielzeit (Intervall)
9. Markierung
10. Akzeptieren und Weiter

Die Änderungen nimmt man durch Anwählen des Index und Eingeben des neuen Wertes vor, wobei gewisse Werte ignoriert werden. Dies sind negative Werte, nicht existierende Komfortklassen, nicht existierende Punkte und Zeitintervalle. Zusätzlich gibt es den Punkt `Akzeptieren und Weiter`, der die Eingabe der Kriterien beendet.

5.3.7.2 Teilnehmer anzeigen

Die Teilnehmer einer bereits gewählten Fahrgemeinschaft werden gekürzt dargestellt. Dabei werden sie beginnend mit eins aufsteigend numeriert und die Attribute Vorname, Nachname, kann und will fahren, die Markierung (*vermittelbar*, *reserviert eingetragen*, *fest eingetragen*), sowie die ID der Person werden angezeigt.

5.3.7.3 Neuer Teilnehmer

Um einen neuen Teilnehmer in eine Fahrgemeinschaft der aktuellen Fahrgemeinschaftseinteilung aufzunehmen, müssen vorher bereits die Daten des Teilnehmers wie in Use Case 5.3.6.1 eingegeben worden sein. Man kann anschließend den neuen Teilnehmer manuell oder per Suchsystem eintragen lassen. Dazu wählt man den Menüeintrag `Fahrgemeinschaften-Neuer Teilnehmer-Manuell eintragen` oder den Menüeintrag `Fahrgemeinschaften-Neuer Teilnehmer-Suchsystem`.

manuell eintragen

Man möchte eine Person in eine bestehende Fahrgemeinschaft per Hand aufnehmen. Die betreffende Person wird wie in Use Case 5.3.6.5 beschrieben selektiert. Um nun die Fahrgemeinschaft zu finden, werden Fahrgemeinschaften angezeigt (siehe Use Case 5.3.7.10). Nun wählt man über den Index eine der Fahrgemeinschaften aus und beantwortet die Frage des Systems „`Person in Fahrgemeinschaft aufnehmen? [J/n]`“ mit „j“. Auf die neue Fahrgemeinschaft wird dann automatisch die Bewertungsfunktion angewendet und das Ergebnis präsentiert.

Nun beantwortet der Benutzer noch die Frage „`Fahrgemeinschaft übernehmen? [J/n]`“ mit „j“ und die neu entstandene Fahrgemeinschaft wird vom System übernommen. Der neue Teilnehmer wird als *reserviert eingetragen* markiert. Beantwortet man eine der beiden Fragen mit „n“, so wird wieder die Liste der Fahrgemeinschaften angezeigt.

Suchsystem

Man möchte eine Person in eine bestehende Fahrgemeinschaft eintragen und dabei die Hilfe des Systems in Anspruch nehmen. Dazu wird einem nach der Auswahl der Person wie in Use Case 5.3.6.5 beschrieben eine Liste von Fahrgemeinschaften angezeigt, die nach der aktuellen Bewertungsfunktion gut zu der Person passen würden (siehe Use Case 5.3.7.10). Nun wählt man über den Index eine der Fahrgemeinschaften aus und beantwortet die Frage des Systems „`Person in Fahrgemeinschaft aufnehmen? [J/n]`“ mit „j“. Auf die neue Fahrgemeinschaft wird dann automatisch die Bewertungsfunktion angewendet und das Ergebnis präsentiert.

Nun beantwortet der Benutzer noch die Frage „`Fahrgemeinschaft übernehmen? [J/n]`“ mit „j“ und die neu entstandene Fahrgemeinschaft wird vom System übernommen. Der neue Teilnehmer wird als reserviert markiert. Beantwortet man eine der beiden Fragen mit „n“, so wird wieder die Liste der Fahrgemeinschaften angezeigt.

5.3.7.4 Teilnehmer fest eintragen

Ein bereits in eine Fahrgemeinschaft eingetragener Teilnehmer, der noch *reserviert eingetragen* ist, wird nun *fest eingetragen*. Man wählt den Menüpunkt `Fahrgemeinschaften-Teilnehmer fest eintragen`. Es wird eine Liste von Fahrgemeinschaften angezeigt (siehe Use Case 5.3.7.10) die reserviert eingetragene Personen enthalten. Daraus wählt man dann die betreffende Fahrgemeinschaft über den Index aus.

Die Teilnehmer der Fahrgemeinschaft werden nach Use Case 5.3.7.2 angezeigt und man wählt die fest einzutragende Person über ihren Index an. Falls die Person bis jetzt *reserviert eingetragen* war, wird gefragt „`Teilnehmer fest eintragen? [J/n]`“. Sonst wird die Wahl übergangen und die Meldung „`Teilnehmer schon fest eingetragen!`“ ausgegeben. Wird die Frage mit „j“ beantwortet, wird der neue Teilnehmer als *fest eingetragen* markiert. Beantwortet man die Frage mit „n“, werden wieder die Teilnehmer der Fahrgemeinschaften angezeigt.

5.3.7.5 Teilnehmer löschen

Eine Person, die in eine Fahrgemeinschaft eingetragen ist, soll aus ihr gelöscht werden. Bei dieser Person wird dann nur die Markierung *fest eingetragen* oder *reserviert eingetragen* in *vermittelbar* abgeändert und sie wird aus der Fahrgemeinschaft ausgetragen. Dazu wählt man `Fahrgemeinschaften-Teilnehmer`

löschen. Es wird eine Liste von Fahrgemeinschaften angezeigt (siehe Use Case 5.3.7.10). Daraus wählt man dann die betreffende Fahrgemeinschaft über den Index aus.

Die Teilnehmer der Fahrgemeinschaft werden nach Use Case 5.3.7.2 angezeigt und man wählt die zu löschende Person über ihren Index an. Wird die Frage des Systems „`Person aus Fahrgemeinschaft löschen? [J/n]`“ mit „j“ beantwortet, wird die Person aus dieser Fahrgemeinschaft gelöscht.

Ist die zu löschende Person der Fahrer, so erscheint die Systemmeldung „`Vorsicht. Durch Loeschen des Fahrers wird die Fahrgemeinschaft aufgelöst.`“. Daraufhin wird gefragt „`Fahrer löschen? [j/N]`“. Wird diese Frage mit „j“ beantwortet, wird die Fahrgemeinschaft aufgelöst (siehe auch Use Case 5.3.7.8) und eine Liste mit den betroffenen Personen wird angezeigt. Ansonsten befindet man sich wieder bei der Liste der Personen.

5.3.7.6 Fahrgemeinschaft eingeben

Fahrgemeinschaften können auch manuell zusammengestellt werden. Die Personen, die hier eingetragen werden sollen, müssen bereits im System erfaßt sein. Der Punkt `Fahrgemeinschaften-Fahrgemeinschaft eingeben` wird gewählt. Nun wird zuerst der Fahrer nach Use Case 5.3.6.5 selektiert, wobei nur potentielle Fahrer angezeigt werden, die in keiner Fahrgemeinschaft eingetragen sind. Die Person wird über den Index angewählt und die Frage „`Person als Fahrer übernehmen? [J/n]`“ gestellt. Wird die Frage mit „j“ beantwortet, fährt man fort, bei „n“ zeigt man wieder die Liste der potentiellen Fahrer.

Solange freie Plätze vorhanden sind, wird gefragt „`Weiteren Teilnehmer eintragen [J/n]?`“. Wird die Frage mit „j“ beantwortet, wird wie bei der Auswahl des Fahrers eine Liste von Personen angezeigt, die noch keiner Fahrgemeinschaft zugeordnet sind. Man wählt wieder über den Index eine Person aus, die in die Fahrgemeinschaft übernommen wird. Wird die Frage mit „n“ beantwortet, oder sind die freien Plätze erschöpft, wird die Bewertungsfunktion auf die Fahrgemeinschaft angewendet und das Ergebnis präsentiert. Wird die Frage „`Fahrgemeinschaft übernehmen? [J/n]`“ mit „j“ beantwortet, wird sie in das System übernommen, sonst startet man wieder am Anfang dieses Use Case.

5.3.7.7 Fahrgemeinschaft ändern

Die Eigenschaften einer Fahrgemeinschaft sollen geändert werden. Dazu wird nach Auswahl des Menüpunktes `Fahrgemeinschaften-Fahrgemeinschaft ändern` eine Liste der Fahrgemeinschaften angezeigt (siehe Use Case 5.3.7.10). Daraus wählt man dann die betreffende Fahrgemeinschaft über den Index aus. Die änderbaren Eigenschaften der Fahrgemeinschaft werden mit Index angezeigt:

1. Anzahl der freien Plätze
2. Fahrer
3. Akzeptieren und Verlassen

Die zu ändernde Komponente wird über den Index angewählt. Die Anzahl der freien Plätze kann nicht größer werden als Autoplätze minus Teilnehmer. Wird der Fahrer gewählt, werden alle Personen in der Fahrgemeinschaft angezeigt, die potentielle Fahrer sind (siehe Use Case 5.3.7.2). Aus ihnen kann man über den Index einen neuen Fahrer auswählen. Ergibt sich durch die Wahl ein neuer Fahrer, wird die Frage „`Neuer Fahrer: Fahrtroute neu berechnen? [j/N]`“ ausgegeben und eine neue Wegberechnung durchgeführt, falls mit „j“ geantwortet wird. Bei „n“ wird der alte Fahrer beibehalten. Mit `Akzeptieren und Verlassen` werden die Änderungen übernommen.

5.3.7.8 Fahrgemeinschaft auflösen

Eine Fahrgemeinschaft aus der aktuellen Einteilung wird aufgelöst. Die Teilnehmer dieser Fahrgemeinschaft werden dabei nur in ihrer Markierung *fest eingetragen* oder *reserviert eingetragen* geändert, die auf *vermittelbar* gesetzt wird. Man wählt den Menüpunkt `Fahrgemeinschaften-Fahrgemeinschaft auflösen`. Eine Fahrgemeinschaft wird nach dem Anzeigen (Use Case 5.3.7.10) über ihren Index ausgewählt. Der Benutzer wird gefragt „`Fahrgemeinschaft auflösen? [J/n]`“. Wird die Frage mit „j“ beantwortet und war die Fahrgemeinschaft markiert, wird nachgefragt „`Fahrgemeinschaft ist markiert. Wirklich auflösen? [j/N]`“. Wird die Frage auch mit „j“ beantwortet, werden die Teilnehmer aus der Fahrgemeinschaft entfernt und die Fahrgemeinschaft aus dem System gelöscht. Sonst werden wieder die Fahrgemeinschaften angezeigt.

5.3.7.9 Fahrgemeinschaft bewerten

Die Qualität einer Fahgemeinschaft kann mit der aktuellen Bewertungsfunktion bewertet werden, indem man den Menüpunkt `Fahrgemeinschaften-Fahrgemeinschaft bewerten` aufruft. Eine Liste der Fahrgemeinschaften wird angezeigt (Use Case 5.3.7.10). Die zu bewertende Fahrgemeinschaft wird über ihren Index angewählt. Die Bewertungsfunktion wird auf diese Fahrgemeinschaft angewendet und das Ergebnis präsentiert. Man kann solange aus der Liste auswählen, die wieder angezeigt wird, bis man die Frage „`Weitere Fahrgemeinschaft bewerten? [J/n]`“ mit „n“ beantwortet.

5.3.7.10 Fahrgemeinschaft anzeigen

Eine Liste der Fahrgemeinschaften mit Einschränkung durch eine Filterfunktion soll angezeigt werden. Dazu wählt man den Menüpunkt `Fahrgemeinschaften-Fahrgemeinschaften anzeigen` und es wird der Filter wie in Use Case 5.3.7.1 aufgerufen. Anschließend wird eine Liste aller Fahrgemeinschaften zusammengestellt, die den Bedingungen des Filters genügen. Sie werden dabei aufsteigend sortiert und mit ID der Fahrgemeinschaft, einer Liste der Nachnamen der Teilnehmer und ihrer Markierung dargestellt, wobei der Fahrer besonders gekennzeichnet ist. Wählt man eine der Fahrgemeinschaften über ihren Index an, wird sie im Detail dargestellt. Die Liste der Fahrgemeinschaften und die detaillierte Anzeige können nach Use Case 5.4.1.4 durch Drücken der Taste „d“ in eine PostScript-Datei ausgegeben werden.

Fahrgemeinschaft im Detail anzeigen

Eine Fahrgemeinschaft wird detailliert angezeigt. Dabei werden folgende Attribute aufgeführt:

- Anzahl der freien Plätze
- Teilnehmer der Fahrgemeinschaft (Vorname, Nachname, Status, Fahrer)
- Komfortklasse des Autos
- Startort
- Zielort
- Entstehungszeitpunkt
- Datum der letzten Änderung
- Markierung (*markiert/unmarkiert*)

Desweiteren kann man folgende zwei Punkte anwählen und sich anzeigen lassen:

1. Fahrtroute (graphisch/Straßennamen)
2. Zeitplan

5.3.7.11 Fahrgemeinschaft markieren/unmarkieren

Man möchte Fahrgemeinschaften vor dem Auflösen schützen oder dafür sorgen, daß eine markierte Fahrgemeinschaft doch wieder aufgelöst werden darf. Nach der Auswahl des Menüpunktes `Fahrgemeinschaften-Fahrgemeinschaften markieren/unmarkieren` wird der Benutzer gefragt „`Markieren oder Unmarkieren? [M/u]`“. Nach der Wahl der Operation wird dem Benutzer eine Liste aller Fahrgemeinschaften angezeigt (siehe Use Case 5.3.7.10). Die Liste wird durch die Wahl der Option eingeschränkt. Hat der Benutzer „m“ gewählt, sieht er nur unmarkierte Fahrgemeinschaften, sonst nur markierte. Die zu verändernde Fahrgemeinschaft wird über ihren Index angewählt. Je nach Wahl der Option wird die Frage „`Fahrgemeinschaft markieren? [J/n]`“ oder „`Fahrgemeinschaft unmarkieren? [J/n]`“ gestellt. Wird „j“ gewählt und hatte die Fahrgemeinschaft vorher eine andere Einstellung, wird die neue eingesetzt. Danach sieht man wieder die Liste der Fahrgemeinschaften.

5.3.8 Vermittlung

5.3.8.1 Einteilung auswählen

Durch Auswahl des Menüpunkts `Vermittlung-Einteilung auswählen` kann man die aktuelle Einteilung wechseln. Es werden die Namen der zur aktuellen Personendatei gehörenden Einteilungen in einer numerierten Liste angezeigt.

Die aktuelle Einteilung ist dabei voreingestellt als solche markiert. Wird eine der Einteilungen ausgewählt, so wird diese zur aktuellen Einteilung. Wird der Vorgang abgebrochen oder die bisher aktuelle Einteilung gewählt, so bleibt sie auch die aktuelle. Beim wechseln der aktuellen Einteilung die Bewertungsfunktion der neuen aktuellen Einteilung zur aktuellen Bewertungsfunktion.

5.3.8.2 Systemmeldungen: ein/aus

Der Benutzer möchte über den aktuellen Stand der Berechnung durch Bildschirmmeldungen, informiert werden. Dazu muß der Menüpunkt `Vermittlung-Systemmeldungen: ein/aus` aufgerufen werden. Dieser verhält sich wie ein Wechselschalter (an oder aus). Bei jedem Aufruf ändert sich der Status und die aktuelle Einstellung wird am Bildschirm angezeigt. Danach gelangt man automatisch in das Untermenü Vermittlung zurück. Der Status hat keine Auswirkungen auf den Menüpunkt `Wegsuche-Systemmeldungen: ein/aus` (siehe Use Cases 5.3.10.5).

Wenn die Systemmeldungen eingeschaltet sind, werden bei der Berechnung einer Einteilung Informationen über den Stand und den Verlauf der Berechnung auf den Bildschirm und in eine Datei namens `Mobidick.log` ausgegeben.

5.3.8.3 Einteilung berechnen

Der Benutzer muß zur Berechnung einer Fahrgemeinschaftseinteilung einen von mehreren unterschiedlichen Algorithmen auswählen. Die Menge von Algorithmen ist in drei Algorithmenklassen eingeteilt. Es gibt heuristische, optimale und inkrementelle Algorithmen zur Berechnung einer Fahgemeinschaftseinteilung.

Bei den heuristischen und den optimalen Algorithmen kann der Benutzer Angaben zur Güte des Ergebnisses machen. Die möglichen Angaben beziehen sich auf die Anzahl der zu berechnenden Fahrgemeinschaften und auf die bestmögliche Bewertung, die eine Einteilung unter der aktuellen Bewertungsfunktion erreichen kann. Zur Eingabe der Güte werden dem Benutzer die Abfragen „`Güte bezüglich der Fahrgemeinschaftenanzahl? [<min ♯FGM> - <max ♯FGM>]`“ und „`Güte bezüglich der Bewertung? [<min Bewertung> - <max Bewertung>]`“ angezeigt. Der Benutzer kann dann je einen Wert innerhalb der angegebenen Grenzen eingeben. Gibt er keinen Wert sondern nur `Enter` ein, so werden die Werte `<min ♯FGM>` und `<max Bewertung>` als eingegeben angesehen. Die Kombination dieser Werte ergibt die beste mögliche Güte.

Der Wert `<min ♯FGM>` gibt an wieviele Fahrgemeinschaften eine Einteilung mindestens enthalten muß und wird aus der Größe der Personenmenge und der maximalen Fahrgemeinschaftsgröße berechnet. Die maximale Fahrgemeinschaftsgröße ist vier. Die maximale Anzahl der Fahrgemeinschaften (`<min ♯FGM>`) in einer Einteilung entspricht der Größe der Personenmenge. Die beste mögliche Bewertung (`<max Bewertung>`) und die schlechteste mögliche Berwertung (`<min Bewertung>`) werden passend zur aktuellen Bewertungsfunktion berechnet.

Heuristische Algorithmen sind Näherungsverfahren zur Einteilungsberechnung. Mit ihrer Hilfe soll es möglich sein, in kurzer Zeit eine gute, aber nicht unbedingt optimale Lösung zu finden. Bei der Berechnung einer Einteilung kann ein

heuristischer Algorithmus auf die Personendaten, die aktuelle Bewertungsfunktion, eine vom Benutzer anzugebende Güte des Ergebnisses und die kürzesten Wege zwischen den Personen untereinander und zu ihren Arbeitsplätzen zugreifen.

Sobald eine Einteilung gefunden wurde, die der Güte des Benutzers entspricht, wird der Algorithmus beendet und die Einteilung als Ergebnis zurückgegeben. Ist während der Berechnung abzusehen, daß die Heuristik auf den gegebenen Personendaten keine Einteilung mit der vom Benutzer geforderten Güte findet, so kann der Vorgang mit einer entsprechenden Fehlermeldung abgebrochen werden. Am Ende einer erfolgreichen Berechnung wird die gefundene Einteilung mit der aktuellen Bewertungsfunktion bewertet und das Ergebnis dem Benutzer angezeigt.

Optimale Algorithmen sind Verfahren zur Einteilungsberechnung, die die beste Einteilung bezüglich der aktuellen Bewertungsfunktion finden. Optimiert wird nach der Anzahl der Fahrgemeinschaften, wobei die in der Bewertungsfunktion (siehe 5.3.9.1) enthaltenen Randbedingungen eingehalten werden müssen. Bei der Einteilungsberechnung können die optimalen Algorithmen auf die gleichen Daten wie die heuristischen Algorithmen zugreifen.

Der Benutzer hat auch bei den optimalen Algorithmen die Möglichkeit, eine Güte für das Ergebnis anzugeben. Sobald eine Einteilung der entsprechenden Güte gefunden wurde wird der Algorithmus abgebrochen. Die dabei berechnete Einteilung ist aber nicht die optimale Einteilung sondern nur eine Einteilung mit einer Güte größer oder gleich der vom Benutzer geforderten Güte. Zur Berechnung der optimalen Einteilung muß die beste mögliche Güte vom Benutzer gefordert werden.

Inkrementelle Algorithmen sind Verfahren, die eine bestehende Fahrgemeinschaftseinteilung erweitern. Bei der Berechnung bleiben zunächst alle Fahrgemeinschaften bestehen und es wird versucht, die noch nicht vermittelten Personen in die Fahrgemeinschaften mit freien Plätzen einzufügen. Wenn dies nicht möglich ist, können auch neue Fahrgemeinschaften gebildet werden. Die Inkrementellen Verfahren bieten sich an, wenn die Personenmenge nur um wenige Personen erweitert wurde.

Wie unten beschrieben, kann der Benutzer nach Auswahl der Algorithmenklasse einen Algorithmus aussuchen. In dem Mobidick-System werden zu jeder Algorithmenklasse mindestens je ein Algorthmus implementiert. Die Dokumentation enthält Hinweise, wie ein neuer Algorithmus in den Quellcode des Systems eingefügt werden kann.

Der Menüpunkt `Vermittlung-Einteilung berechnen` führt in ein Untermenü, wo die Wahl zwischen verschiedenen Arten der Berechnung besteht:

1. heuristisch
2. optimal
3. inkrementell

Bei jeder Auswahl erscheint ein weiteres Menü der Form

1. Die [art] Einteilung berechnen
2. Algorithmus wechseln
3. Zurück
4. Hauptmenü
5. Hilfe

Dabei steht [art] für die vorher gewählte Art der Berechnung, also heuristisch, optimal oder inkrementell.

Mit dem ersten Menüpunkt startet man die entsprechende Berechnung. Nach Auswahl des zweiten Menüpunkts wird eine Liste mit Algorithmen angezeigt, die vom Typ [art] sind. Aus dieser Liste kann der Benutzer einen Algorithmus auswählen, wodurch dieser zum aktuellen Algorithmus seiner Art wird. Zu jeder Art gibt es einen aktuellen Algorithmus, der ausgeführt wird, wenn der Benutzer mit dem Menüpunkt eins eine Berechnung seiner Art startet. Die übrigen Menüpunkte entsprechen den Erwartungen (siehe 5.4.1).

Nach dem Start eines Algorithmus wird entschieden, ob eine neue Einteilung berechnet werden soll, oder ob eine schon bestehende Einteilung erweitert wird. Ist die gewählte Art inkrementell, so wird die aktuelle Einteilung erweitert. Handelt es sich um eine optimale oder heuristische Berechnung, so wird eine neue Einteilung geöffnet. Dazu wird der Benutzer gefragt „`Aktuelle Einteilung speichern? [J/n]`“. Lautet die Antwort „j“, so wird sie gespeichert, ansonsten wird sie verworfen. Nun wird eine neue Einteilung geöffnet mit der Frage „`Name der neuen Einteilung:`“. Existiert der eingegebene Name schon, so erscheint die Fehlermeldung „`Fehler 2: Dieser Name existiert schon!`“ und die Eingabeaufforderung wird wiederholt.

Alle Algorithmen übernehmen die markierten Fahrgemeinschaften der beim Aufruf des Algorithmus aktuellen Einteilung. Diese Fahrgemeinschaften dürfen bei der Berechnung nicht aufgelöst, sondern nur erweitert werden, falls noch Plätze frei sind. Der Benutzer kann durch manuelle Änderung der freien Plätze die Erweiterbarkeit einer Fahrgemeinschaft einschränken (siehe 5.3.7.7). Durch das markieren einer Fahrgemeinschaft kann der Benutzer erreichen, daß Fahrgemeinschaften aus der aktuellen Einteilung in der von einem Algorithmus neu berechneten Einteilung wieder enthalten sind.

Zur Einteilungsberechnung werden die kürzesten Wege zwischen allen Start- und Zielorten benötigt. Deshalb wird vor dem Start des Einteilungsalgorithmus überprüf ob diese bereits berechnet wurden. Falls dies nicht der Fall ist, werden die kürzesten Wege mit dem im Untermenü `Auswahl des Algorithmus` in der Wegsuche (siehe 5.3.10.3) eingestellten Algorithmus berechnet.

Drückt man die Taste Esc bei einer laufenden Berechnung, so erscheint die Meldung „`Die Berechnung kann nicht wieder aufgenommen werden.`“ und die Frage „`Wirklich abbrechen? [j/N]?`“. Wird die Frage mit „j“ beantwortet, so wird die Berechnung abgebrochen (und keine Daten zur Fortsetzung werden gespeichert). Beantwortet man die Frage mit „n“, so wird sie fortgesetzt.

Nun wird die Berechnung gestartet, die durch Drücken der Taste „u“ (wie unterbrechen) unterbrochen werden kann. In diesem Fall werden alle zur Fortsetzung

der Berechnung notwendigen Daten bei den zugehörigen Personendaten abgespeichert. Existieren schon Daten dieser Art, so werden sie überschrieben. Es kann somit immer nur die zuletzt abgebrochene Berechnung fortgesetzt werden (siehe Use Case 5.3.8.4). Ist die Laufzeitmessung eingeschaltet, so wird nach Abschluß der Berechnung die Laufzeit angezeigt.

5.3.8.4 Fortfahren mit letzter Berechnung

Durch Auswahl des Menüpunkts `Vermittlung-Fortfahren mit letzter Berechnung` kann man eine unterbrochene Berechnung fortsetzen. Wurde zuvor keine Berechnung unterbrochen, d.h. befindet sich keine Information über eine Berechnung in den Personendaten, so meldet das System „`Es wurde noch keine Berechnung durchgeführt`“ und kehrt ins Menü zurück. Ansonsten lädt es die notwendigen Daten für die Fortsetzung der Berechnung und fährt mit der letzten Berechnung fort. Diese kann wieder mit der Taste „u“ unterbrochen werden (siehe 5.3.8.3).

5.3.8.5 Laufzeitmessung (ein/aus)

Der Menüpunkt `Vermittlung-Laufzeitmessung (ein/aus)` funktioniert wie ein Wechselschalter. Bei Programmstart ist der Status per Default aus. Da zum Zeitpunkt der Spezifikation noch nicht bekannt ist, wie dieser Punkt realisiert werden kann, gibt es hier mehrere Möglichkeiten. Die erste Möglichkeit hat bei der Entwicklung höchste Priorität:

Volle Integration

Ist die Laufzeitmessung eingeschaltet, so wird bei weiteren Einteilungsberechnungen die Laufzeit nach Abschluß der Berechnung angezeigt. Dieser Punkt hat keine Auswirkungen auf das Verhalten der Laufzeitmessung im Menü „Wegsuche“ (siehe Use Case 5.3.10.4).

Externe Laufzeitmessung

Das System zeigt dem Benutzer einen Informationstext an, in dem detailliert beschrieben wird, mit welchem Werkzeug er eine Laufzeitmessung auf den Algorithmen durchführen kann.

5.3.8.6 Einteilung bewerten

Bei Auswahl des Menüpunkts `Vermittlung-Einteilung bewerten` wird die Bewertungsfunktion auf die aktuelle Einteilung angewendet und das Ergebnis dem Benutzer präsentiert.

5.3.8.7 Einteilung anzeigen

Nach Auswahl des Menüpunkts `Vermittlung-Einteilung anzeigen` erhält der Benutzer eine Liste der Personen mit ihrem Namen, die nach Fahrgemeinschaften sortiert sind. Paßt diese Liste nicht auf den Bildschirm, wird er nach jeder Seite gefragt, ob er noch mehr sehen will oder nicht (siehe 5.4.1.7).

5.3.8.8 Einteilung auflösen

Wählt der Benutzer den Menüpunkt `Vermittlung-Einteilung auflösen`, so wird er gefragt „`Möchten Sie alle bis auf markierte Fahrgemeinschaften auflösen? [j/N]`“. Antwortet er mit „j“, so wird noch gefragt: „`Aktuelle Einteilung speichern? [J/n]`“ Beantwortert er auch diese Abfrage mit „j“, so werden alle bis auf markierte Fahrgemeinschaften gelöscht. Die aktuelle Einteilung besteht nun nur noch aus den markierten Fahrgemeinschaften. Alle anderen Personen sind nun wieder zu vermitteln.

5.3.9 Bewertungsfunktionen

5.3.9.1 Aufbau der Bewertungsfunktion

Eingangsgrößen

Zunächst eine Auflistung sämtlicher Größen, die bei der Bewertung der Güte einer Fahrgemeinschaft eingehen:

1. Umweg des Fahrers (bei mehreren potentiellen Fahrern der kleinste), und zwar relativ zu seinem alten Weg (vor Bildung der Fahrgemeinschaft) (s.u.).
2. Arbeitszeiten: jede Person gibt ein Ankunfts- und ein Abfahrtsintervall an; in diesen Intervallen muß die tatsächliche Ankunfts- bzw. Abfahrtszeit am Arbeitsplatz liegen. Bei der Abfahrt kann auch die Arbeitsdauer angegeben werden; der früheste Abfahrtszeitpunkt errechnet sich dann aus Ankunftszeit plus Arbeitsdauer. In diesem Fall wird nur ein einseitig beschränktes Intervall betrachtet.
3. Wünsche der Personen (jeweils mit Gewicht):
   - abgelehnte Musikrichtungen
   - Geschlecht
   - Raucher
   - Komfortklasse des Autos
   - weitere

   Bei der Berechnung der Bewertungsfunktion wird für jede Person der Fahrgemeinschaft geprüft, ob ihre Wünsche mit den anderen Personen in Konflikt stehen. Dabei wird nur beachtet, ob ein Konflikt vorliegt oder nicht.

Wieviele Personen nicht wunschgemäß sind, wird dabei nicht betrachtet. Jeder Wunsch trägt ein Gewicht zwischen 0 und 10 (unwichtig bzw. sehr wichtig). Wünsche mit dem Gewicht 10 werden als absolut verbindlich betrachtet, d.h. es wird keine Fahrgemeinschaft gebildet, in der dieser verletzt würde.

4. Zu- und Abneigungen gegenüber bestimmten Personen:

   - Liste der erwünschten Personen
   - Liste der abgelehnten Personen

Bewertungsfunktion für Fahrgemeinschaften

Sei $P = \{p_1, \ldots, p_n\}$ die betrachtete Personenmenge, $M = \{f_1, \ldots, f_m\}$ eine Einteilung in Fahrgemeinschaften $f_i = \{p_{i_1}, \ldots, p_{i_j}\}$ mit $1 \leq j \leq 4$. $W = \{w_1, \ldots, w_k\}$ sei die Menge der Wünsche, $\gamma_1(p), \ldots, \gamma_k(p) \in \{0, 1, \ldots, 10\}$ die zugehörigen Gewichte für jede Person p. Die Bewertungsfunktion

$$\psi : M \to \mathbb{R}^+ \times [0,1] \times ([0,1] \cup \{\infty\}) \times [0,1]^2$$

bewertet eine Fahrgemeinschaft f_i mit einem Quintupel von Prozentzahlen:

$$\psi(f_i) = (U(f_i), Z(f_i), E(f_i), N_e(f_i), N_a(f_i))$$

Für jede Fahrgemeinschaft wird eine Mindestgüte gefordert, d.h.
$\psi(f_i) \leq (U_{max}, Z_{max}, E_{max}, N_{e,max}, N_{a,max}) \quad \forall f_i \in M$ mit
$U_{max} \in \mathbb{R}^+$, $Z_{max}, E_{max}, N_{e,max}, N_{a,max} \in [0,1]$. Die vier Teilfunktionen nun im einzelnen:

1. $U(f_i) = \min_{p \in f_i} \min_{\text{alle Wege } w \text{ mit Fahrer } p} U(p,w)$, wobei $U(p,w) = d(w) \Leftrightarrow d(s_p, z_p)$ den Umweg von Fahrer p beim Fahrgemeinschaftsweg w bezeichnet. $d(w)$ ist die Länge des Weges w und $d(s_p, z_p)$ ist die Weglänge zwischen Start- und Zielort von p.

   Alternativ zum Umweg des Fahrers könnte man auch die Differenz zwischen den aufsummierten Einzelwegen der Teilnehmer vor Bildung der Fahrgemeinschaft und dem Fahrgemeinschaftsweg betrachten, falls der ökologische Aspekt wichtiger ist. Obige Umwegdefinition soll dafür sorgen, daß der Umweg für den Fahrer sich in akzeptablen Grenzen hält.

2. $Z(f_i) = \frac{\#verfehlteIntervalle}{2 \cdot |f_i|} \in [0,1]$. Hier wird einfach gezählt, wie oft ein Ankunfts- oder Abfahrtsintervall nicht beachtet wurde (maximal $2 \cdot |f_i|$ mal).

3. $E : M \to [0,1] \cup \infty$ mit

$$E(f_i) = \begin{cases} (\sum_{p \in f_i} \sum_{j=1}^{k} \frac{\gamma_j(p)}{\gamma(p)} \cdot K(p,j,f_i)) \cdot \frac{1}{k \cdot |f_i|}, & \text{falls } \gamma_j(p) \cdot K(p,j,f_i) \neq 10 \\ & \forall p \in f_i, \forall j \in \{1, \ldots, k\} \\ \infty, & \text{sonst} \end{cases}$$

wobei $\gamma(p) = \sum_{j=1}^{k} \gamma_j(p)$ gilt . Mit $K(p, j, f_i)$ wird überprüft, ob der Wunsch w_j von Person p mit der Fahrgemeinschaft f_i in Konflikt steht ($K : P \times \{1, \ldots, k\} \times M \to [0, 1]$):

$$K(p, j, f_i) = \begin{cases} 1, & \text{falls Wunsch } w_j \text{ von } p \text{ in Konflikt mit } f_i \\ 0, & \text{sonst} \end{cases}$$

Für $E(f_i)$ ergibt sich der Wert ∞ , falls ein Wunsch mit Gewicht 10 verletzt wird. Diese Fahrgemeinschaft kommt dann auf jeden Fall nicht zustande.

4. $N_e(f_i) = 1 \Leftrightarrow \frac{1}{(|f_i|-1)\cdot|f_i|} \cdot \sum_{p \in f_i}$ *# erwünschte Personen aus Sicht von* p, $N_e : M \to [0, 1]$.
   Für jede Person p der Fahrgemeinschaft wird die Anzahl der Personen in f_i summiert, die von p erwünscht sind.

5. $N_a(f_i) = \frac{1}{(|f_i|-1)\cdot|f_i|} \cdot \sum_{p \in f_i}$ *# abgelehnte Personen aus Sicht von* p,
   $N_a : M \to [0, 1]$. Für jede Person p der Fahrgemeinschaft wird die Anzahl der Personen in f_i summiert, die von p abgelehnt werden.

Bewertungsfunktion für Einteilungen

Die Bewertung einer Einteilung M ergibt sich einfach durch Aufsummieren der Bewertungen der einzelnen Fahrgemeinschaften:

$$\psi(M) = (U(M), A(M), E(M), N_e(M), N_a(M))$$

mit

1. $U(M) = \frac{1}{|M|} \cdot \sum_{f_i \in M} U(f_i) \in \mathbb{R}^+$
2. $A(M) = \frac{1}{|M|} \cdot \sum_{f_i \in M} I(f_i) \in [0, 1]$
3. $E(M) = \frac{1}{|M|} \cdot \sum_{f_i \in M} E(f_i) \in [0, 1] \cup \{\infty\}$
4. $N_e(M) = \frac{1}{|M|} \cdot \sum_{f_i \in M} N_e(f_i) \in [0, 1]$
5. $N_a(M) = \frac{1}{|M|} \cdot \sum_{f_i \in M} N_a(f_i) \in [0, 1]$

Auch hier wir wieder eine Mindestgüte $\psi(M) \leq (U^M_{max}, Z^M_{max}, E^M_{max}, N^M_{e,max}, N^M_{a,max})$ und eine minimale Anzahl von Fahrgemeinschaften, d.h. $|M|$ minimal gefordert.

5.3.9.2 Neue Bewertungsfunktion anlegen

Durch Auswahl des Menüpunkt `Bewertungsfunktionen-neu` wird eine neue Bewertungsfunktion angelegt. Nach Auswahl des Menüpunkts wird dem Benutzer eine Liste mit allen geöffneten Bewertungsfunktionen angezeigt. Dann wird er aufgefordert den neuen Dateinamen anzugeben durch die Meldung „`Neuer Name:`“. Gibt der Benutzer einen schon existierenden Namen ein, so wird die Fehlermeldung „`Fehler 2: Dieser Name existiert schon!`“ angezeigt und die Eingabeaufforderung wiederholt.

An den Dateinamen wird vom System die Erweiterung `.fkt` angehängt, mit der die Datei als Bewertungsfunktion gekennzeichnet wird. Nach Eingabe des Namens kann der Benutzer wie in 5.3.9.3 die Parameter der neuen Bewertungsfunktion anpassen.

5.3.9.3 Bewertungsfunktion ändern

Nach Auswahl des Menüpunkts `Bewertungsfunktionen-ändern` können die Parameter der aktuellen Bewertungsfunktion verändert werden. Die Parameter werden mit Index präsentiert. Wählt man einen aus, wird der aktuelle Wert und der mögliche Bereich angezeigt. Nicht korrekte Werte werden nicht angenommen und es wird die Fehlermeldung „`Fehler 1: Eingabe enthält unzulässige Zeichen oder ungültigen Bereich!`“ angezeigt. Gibt man einen korrekten Wert ein, wird dieser übernommen. Zum Verlassen dieses Punktes wählt man den Menüeintrag `Akzeptieren und Verlassen`.

5.3.9.4 Bewertungsfunktion auswählen

Nach Auswahl des Menüpunkts `Bewertungsfunktionen-auswählen` wird eine Liste aller geöffneten Bewertungsfunktionen angezeigt. Aus dieser Liste kann der Benutzer eine Bewertungsfunktion auswählen, die dann zur aktuellen wird.

5.3.9.5 Bewertungsfunktion anzeigen

Nach Auswahl des Menüpunkts `Bewertungsfunktionen-anzeigen` werden die Parameter der aktuellen Bewertungsfunktion am Bildschirm angezeigt.

5.3.10 Wegsuche

5.3.10.1 Einzelwegsuche

Mit dem Menüpunkt `Wegsuche-Einzelwegsuche` kann der Benutzer eine Wegsuche von Straße A nach Straße B durchführen, ohne dabei eine Fahrgemeinschaft betrachten zu müssen. Nach Auswahl des Menüpunkts wird überpüft, ob überhaupt ein Verkehrsgraph geladen wurde. Wenn dies nicht der Fall ist, erscheint die Fehlermeldung „`Fehler 8: Kein Verkehrsgraph geladen.`“ und der Vorgang wird abgebrochen. Wenn ja, wird eine Bildschirmmaske angezeigt, in der die Start- und Zielstraße eingegeben werden muß. Dabei wird jeweils überprüft, ob der Straßenname mehrmals im System vorkommt. Ist dies der Fall, wird eine Liste aller Straßennamen mit entsprechender ID anzeigt und der Benutzer muß zum Namen noch zusätzlich die ID angeben. Wird nun die Frage „`Wegsuche starten? [J/n]`“ mit „j“ beantwortet, startet die Suche. Ist die Suche erfolgreich, erscheint der Text „`Wegsuche erfolgreich. Route gefunden`“ und die gefundene Route kann mit Hilfe einer Straßenliste oder einem Verkehrsgraphen angezeigt werden. Möchte man einen Ausdruck von dieser Route haben, so kann durch Drücken der Taste „d“ eine PostScript-Datei erstellt werden.

Wenn sich im System überhaupt kein oder kein gültiger Verkehrsgraph befindet, der zu den Straßen paßt, wird der gesamte Vorgang abgebrochen.

5.3.10.2 n-Wegesuche

Mit dem Menüpunkt `Wegsuche-n-Wegesuche` kann der Benutzer die Zeit anzeigen lassen, die der aktuelle Wegsuchealgorithmus zur Berechnung von allen kürzesten Wegen zwischen n zufällig gewählten Paaren aus Start- und Zielorten benötigt. Nach Auswahl des Menüpunkts wird überpüft, ob überhaupt ein Verkehrsgraph geladen wurde.

Wenn dies nicht der Fall ist, erscheint die Fehlermeldung „`Fehler 8: Kein Verkehrsgraph geladen.`“ und der Vorgang wird abgebrochen. Sonst wird der Benutzer aufgefordert die Zahl n einzugeben. Nach der Eingabe werden n Wege zwischen zufällig gewählten Start- und Zielorten berechnet und die dafür benötigte Laufzeit wird ausgegeben.

5.3.10.3 Auswahl des Algorithmus

Mit dem Menüpunkt `Wegsuche-Auswahl des Algorithmus` kann der Benutzer einen Wegsuchealgorithmus aus einer Liste über einen Index auswählen. Der ausgewählte Algorithmus wird bei der Wegsuche in 5.3.10.1 und in 5.3.8.3 verwendet.

5.3.10.4 Laufzeit

Der Menüpunkt `Wegsuche-Laufzeit` funktioniert wie ein Wechselschalter. Bei Programmstart ist der Status per Default aus. Ist die Laufzeitmessung eingeschaltet, so wird bei der nächsten Wegsuche die Laufzeit nach Abschluß der Berechnung angezeigt. Dieser Punkt hat keine Auswirkungen auf das Verhalten der Laufzeitmessung im Menü `Vermittlung` (siehe Use Case 5.3.8.5).

5.3.10.5 Systemmeldungen: ein/aus

Der Benutzer möchte über den aktuellen Stand der Berechnung, durch Bildschirmmeldungen, informiert werden. Dazu muß der Menüpunkt `Wegsuche-Systemmeldungen: ein/aus` einfach nur aufgerufen werden. Dieser verhält sich wie ein Wechselschalter. Bei jedem Aufruf ändert sich der Status und die aktuelle Einstellung wird am Bildschirm angezeigt. Danach gelangt man automatisch in das Untermenü zurück. Der Status hat keine Auswirkungen auf die 'Systemmeldung: ein/aus' im Menü `Vermittlung` (siehe Use Case 5.3.8.2).

5.3.11 Voreinstellungen

Nach Anwahl des Menüpunkts `Voreinstellungen` hat der Benutzer die Möglichkeit, Pfade und Namen der beim Systemstart zu öffnenden Dateien und

Default-Pfade anzugeben. Die möglichen Eingaben sind:

- Default-Pfad für Personendateien
- Name der zu öffnenden Personendatei
- Default-Pfad für Verkehrsgraphen
- Name des zu öffnenden Verkehrsgraphen
- Default-Pfad für Bewertungsfunktionen

Wenn keine Datei geöffnet werden soll, so wird kein Name sondern die Zeichenkette `leer` eingegeben. Die Zeichenkette `leer` als Pfad entspricht dem Pfad \. Beim Beenden der Eingabe wird der Benutzer gefragt: „`Änderungen speichern? [J/n]`“.

Beim Beenden des Mobidick-Systems werden die in den Voreinstellungen angegebenen Daten in der Datei voreinstellungen.mbd abgespeichert.

5.4 Anforderungen an externe Schnittstellen

5.4.1 Benutzungsschnittstelle

5.4.1.1 Menüpunkt Zurück

Wird dieser Menüpunkt gewählt, wird das aktuelle Menü verlassen und man befindet sich eine Ebene höher.

5.4.1.2 Menüpunkt Hauptmenü

Wird dieser Menüpunkt gewählt, wird das aktuelle Menü verlassen und man befindet sich im Hauptmenü.

5.4.1.3 Menüpunkt Hilfe

Wird dieser Menüpunkt aufgerufen, wird dem Benutzer ein Hilfetext präsentiert. Er enthält Informationen darüber, wo man sich in der Menüstruktur befindet, wie man hierher kam, wohin man von hier aus gehen kann, was man in diesem Menü alles anwählen kann und was dies bewirkt.

5.4.1.4 PostScript-Ausgabe

Einige angezeigte Daten kann man zum späteren Drucken in eine PostScript-Datei ausgeben lassen. Dazu drückt man, wenn die entsprechenden Daten angezeigt werden die Taste “d”. Nun bekommt man eine Liste aller Dateien mit der

Endung ".ps" in dem aktuellen Verzeichnis angezeigt und erhält eine Eingabeaufforderung für den Namen der Datei. Wählt man einen bereits existierenden Namen aus, wird man gefragt „`Datei existiert bereits. Überschreiben? [j/N]`". Antwortet man mit "j", wird die Datei überschrieben und man sieht wieder die Daten. Antwortet man mit "n", wird die Eingabeaufforderung wiederholt.

5.4.1.5 Markierter Eintrag

Eingestellte Auswahlmenüeinträge werden mit einem $*$ vor der Nummer markiert. Wird bei einer solchen Auswahl anstelle einer Nummer die Return-Taste gedrückt, wird der mit $*$ markierte Eintrag ausgewählt.

5.4.1.6 Escape

Durch Drücken der Escape-Taste wird die aktuelle Bearbeitung einer Bildschirmmaske abgebrochen und der Ausgangszustand wird wieder hergestellt. Danach gelangt man in das Menü zurück, aus dem die Aktion gestartet wurde.

5.4.1.7 Darstellung von Listen

Aufgrund der Länge kann die vollständige Darstellung einer Liste im aktuellen Fenster zu Problemen führen. Daher wird zunächst eine Fensterseite angezeigt. Beanwortet man nun die Frage „`Weiter? [J/n/=]`" mit „j", wird die nächste Fensterseite angezeigt. Wählt man die Alternative „n", wird die Darstellung abgebrochen und bei „=" werden die restlichen Seiten ohne Unterbrechung ausgegeben.

5.4.1.8 Abfragen

Der Benutzer kann im System die Abfrage auf zwei Arten beantworten. Zum einen durch Drücken der zugelassenen Zeichen, dabei wird nicht auf die Groß- und Kleinschreibung geachtet und zum anderen durch Drücken der Return-Taste. Die Return-Taste löst dabei die Aktion aus, die dem Großbuchstaben entspricht.

5.4.1.9 Dialoge

Es gibt insgesamt drei verschiedene Arten von Dialogen, die auf unterschiedliche Weise bedient werden.

Menü: Nachdem das System gestartet wird, erscheint eine allgemeine Bildschirmanzeige, die eine Reihe von Auswahleinträgen besitzt. Diese Einträge können in der Kommandozeile, die sich unten am Bildschirmrand befindet, über den entsprechenden Index angewählt werden.

Abfrage von Einzeldaten: Ruft man eine Bildschirmanzeige mit Werten zum ersten Mal auf, steht hinter jedem Eintrag ein Default-Wert. Will man diesen Default-Wert ändern, muß der entsprechende Index in der Kommandozeile angewählt werden. Danach kann der Wert eingegeben werden. Entspricht dieser dem Definitionsbereich, so wird der Wert übernommen und in die Bildschirmanzeige eingefügt. Ansonsten muß der Wert neu eingegeben werden.

Abfrage von kompletten Datensätzen: Ruft man eine Bildschirmanzeige auf, in der komplette Datensätze eingegeben werden müssen, so erfolgt eine automatische Abfrage der einzelnen Felder. Am Ende wird der komplette Datensatz präsentiert und durch Beantworten der Frage: „`Alles Korrekt? [J/n]`“ mit „j“, wird der Datensatz ins System aufgenommen. Wenn nein, werden die Abfragen nochmals durchgegangen. Unterschied zu voher ist, daß der Wert vom vorherigen Durchlauf in der Bildschirmanzeige steht und nur noch dort geändert werden muß, wo ein Fehler ist.

5.4.2 Hardwareschnittstellen

5.4.2.1 Drucken

Folgende Daten können in einer PostScript-Datei abgelegt werden:

- Ergebnis der Suche mit Personenfilter
- Ergebnis der Suche mit Fahrgemeinschaftenfilter
- Ergebnis der Wegsuche (textuell)
- Einteilung als Personenliste

5.4.2.2 Sekundärspeicher

Das System Mobidick verwendet folgende Dateien:

- Personendaten mit zugehörigen Einteilungen: Endung `.per`
- Verkehrsgraphen: Endung `.gra`
- Parametereinstellung für Bewertungsfunktion: Endung `.fkt`
- Logfile für Systemmeldungen bei Weg- und Einteilungsberechnung: Endung `.log`
- Die Datei `voreinstellungen.mbd`, in der die Voreinstellungsparameter enthalten sind.

Die entsprechenden Dateiformate werden im Entwurf festgelegt. Bei allen Lade- und Speicheroperationen müssen Name und Pfad frei wählbar sein.

5.4.3 Softwareschnittstellen

Das System Mobidick kommuniziert mit einem externen Graphenviewer, dies ist in 5.5.4 beschrieben.

5.5 Leistungsanforderungen

Das System Mobidick ist für den Einbenutzerbetrieb ausgelegt und läuft auf *einem* Terminal.

5.5.1 Dateien

Das System soll mit beliebig vielen Einteilungs-, Verkehrsgraphen- und Bewertungsfunktionsdateien umgehen können. Bei den Verkehrsgraphen sollte das Stadtgebiet von Stuttgart verarbeitet werden können.

5.5.2 Daten im Hauptspeicher

Es wird immer nur ein Verkehrgraph im Hauptspeicher gehalten. Personen-, Einteilungs- und Bewertungsfunktionsdaten können solange angelegt werden, bis der Speicher voll ist.

5.5.3 Antwortzeiten

Die folgenden Funktionen sollen interaktiv und dementsprechend schnell sein:

- Personenfilter
- Filter für Fahrgemeinschaften
- Bewertung einer Fahrgemeinschaft
- Bewertung einer Einteilung
- Suchen einer Fahrgemeinschaft beim manuellen Einfügen
- Wegsuche auf dem hierarchischen Graphen

Für die optimale, heuristische und inkrementelle Berechnung von Einteilungen muß mit längeren Berechnungszeiten (Stunden, Tage) gerechnet werden.

5.5.4 Entwurfseinschränkungen

Das System Mobidick muß auf dem Rechner tagetes des Rechnerpools der Abteilung Formale Konzepte unter Solaris 5.4 laufen. Als Programmiersprache wird C++ verwendet, und zwar die aktuelle Version des gnu-Compilers.

Da bereits ein Graphenviewer zur Anzeige von Verkehrsgraphen in C++ implementiert wurde, muß dieser in die Entwurfsüberlegungen einbezogen werden. Da dieser als eigenständiges Programm bestehen bleiben soll, muß eine Kommunikationsschnittstelle zwischen beiden Programmen festgelegt werden. Zur Anzeige von kürzesten Wegen müssen die entsprechenden Daten an den Graphenviewer übergeben werden.

Zunächst wird nur eine auf Text basierende Benutzungsoberfläche implementiert, eine graphische Oberfläche erscheint für einen Prototypen zu aufwendig.

An Verkehrsdaten liegt uns bisher der Stadtplan von Stuttgart im GDF-Format vor (GDF-Version 2.1). Für die Umwandlung der GDF-Daten in eine Verkehrsgraphendatei wurde ein Perlskript implementiert. Dafür wurde ein vorläufiges Verkehrsgraphenformat festgelegt. Für das Konvertierungsprogramm wird noch eine Dokumentation erstellt, in der Quell-, Zielformat und die Umwandlung beschrieben werden.

5.5.5 Attribute

Da es sich um einen Prototyp handelt, wird nichts zur Verfügbarkeit und zur Sicherheit ausgesagt. Die Wartbarkeit wird durch noch festzulegende Codierungs- und Entwurfsrichtlinien erreicht. Die Portabilität wird durch folgende Maßnahmen erleichtert:

- ausschließliche Verwendung von überall verfügbaren C++-Bibliotheken
- separate Module für die Programmteile, die auf Bildschirm und Dateien zugreifen.

5.6 Zukünftige Erweiterungen

Wochentage: Jeder Teilnehmer kann für jeden Wochentag eine andere Arbeitszeit angeben. Beispiel: Mo 9.00-17.00 Uhr, Di 8.00-16.00 Uhr, Mi 10.00-13.00 Uhr.

Hin- oder Rückfahrt: Der Teilnehmer kann angeben, ob er mit der Fahrgemeinschaft beide Wege oder nur einen Weg fahren will.

Statistische Angaben: Der Benutzer kann statistische Daten über das FGM-System erfahren. Beispielweise die Auslastung der Fahrzeuge, die durchschnittliche Personenzahl einer FGM und weitere mehr.

Zwischenpunkte bei der Wegsuche: Der Teilnehmer kann zusätzliche Straßennamen angeben, die bei der Wegsuche berücksichtigt werden und auf jeden Fall in der Route enthalten sind.

Verknüpfungen: Der Benutzer möchte eine Liste aller reservierten Personen. Danach wählt er eine Person aus. Dabei besteht die Möglichkeit die entsprechende FGM der Person, mit allen anderen Teilnehmern, anzeigen zu lassen.

Sammelpunkte: Man möchte Orte angeben, an denen sich Teilnehmende einer FGM treffen, um von dort loszufahren, oder sich nach dem Ankommen zu zerstreuen.

Graphische Benutzungsoberfläche: Das System besitzt eine graphische Benutzungsoberfläche (Fenster, Pulldown-Menüs usw.) und kann mit Hilfe einer Maus gesteuert werden.

5.7 Systemmeldungen

5.7.1 Meldungen

- „`Änderungen für Fahrgemeinschaftsbildung relevant`“ (5.3.6.1)
- „`Person ist Fahrer, löschen führt zur Auflösung einer Fahrgemeinschaft`“ (5.3.6.1)
- „`Teilnehmer schon fest eingetragen!`“ (5.3.7.4)
- „`Vorsicht. Durch Loeschen des Fahrers wird die Fahrgemeinschaft aufgelöst`“ (5.3.7.5)
- „`Die Berechnung kann nicht wieder aufgenommen werden`“ (5.3.8.3)
- „`Es wurde noch keine Berechnung durchgeführt`“ (5.3.8.4)
- „`Wegsuche erfolgreich. Route gefunden`“ (5.3.10.1)

5.7.2 Fragen

- „`Personendatei <Name> vor dem Beenden speichern? [J/n]`“ (5.3.4)
- „`Neuer Name:`“ (5.3.5.1)
- „`Personendatei <Name> vor dem Erstellen der neuen Datei speichern? [J/n]`“ (5.3.5.1)
- „`Name existiert schon, trotzdem abspeichern und vorhandene Datei überschreiben? [j/N]`“ (5.3.5.1, 5.3.5.4, 5.3.5.8, 5.3.5.9, 5.3.5.13)
- „`Personendatei <Name> vor dem Laden der anderen Datei speichern? [J/n]`“ (5.3.5.2)
- „`Datei vor dem Schließen speichern? [J/n]`“ (5.3.5.5)

- „Bewertungsfunktion <Name> vor dem Schließen speichern? [J/n]“ (5.3.5.5)
- „Name:“ (5.3.5.6)
- „Einteilung wirklich löschen? [j/N]“ (5.3.5.10)
- „Person wirklich übernehmen? [J/n]“ (5.3.6.1)
- „Änderungen vornehmen und Person aus den betroffenen Fahrgemeinschaften löschen? [J/n]“ (5.3.6.1)
- „Änderungen vornehmen und Fahrgemeinschaft auflösen? [J/n]“ (5.3.6.1)
- „Person wirklich löschen? [J/n]“ (5.3.6.1)
- „Haben Sie die alte Personenmenge gesichert? [j/N]“ (5.3.6.1)
- „Personenmenge übernehmen? [J/n]“ (5.3.6.1)
- „Tabelle in PostScript-Datei ausgeben? [j/N]“ (5.3.6.1)
- „Person in Fahrgemeinschaft aufnehmen? [J/n]“ (5.3.7.3, 5.3.7.3)
- „Fahrgemeinschaft übernehmen? [J/n]“ (5.3.7.3, 5.3.7.3, 5.3.7.6)
- „Teilnehmer fest eintragen? [J/n]“ (5.3.7.4)
- „Person aus Fahrgemeinschaft löschen? [J/n]“ (5.3.7.5)
- „Fahrer löschen? [j/N]“ (5.3.7.5)
- „Person als Fahrer übernehmen? [J/n]“ (5.3.7.6)
- „Weiteren Teilnehmer eintragen? [J/n]“ (5.3.7.6)
- „Neuer Fahrer: Fahrtroute neu berechnen? [j/N]“ (5.3.7.7)
- „Fahrgemeinschaft auflösen? [J/n]“ (5.3.7.8)
- „Fahrgemeinschaft ist markiert. Wirklich auflösen? [j/N]“ (5.3.7.8)
- „Weitere Fahrgemeinschaft bewerten? [J/n]“ (5.3.7.9)
- „Markieren oder Unmarkieren? [M/u]“ (5.3.7.11)
- „Fahrgemeinschaft markieren? [J/n]“ (5.3.7.11)
- „Fahrgemeinschaft unmarkieren? [J/n]“ (5.3.7.11)
- „Güte bezüglich der Fahrgemeinschaftenanzahl? [<min ♯FGM> - <max ♯FGM>]“ (5.3.8.3)
- „Güte bezüglich der Bewertung? [<min Bewertung> - <max Bewertung>]“ (5.3.8.3)
- „Aktuelle Einteilung speichern? [J/n]“ (5.3.8.3)

- „`Name der neuen Einteilung:`“ (5.3.8.3)
- „`Wirklich abbrechen? [j/N]?`“ (5.3.8.3)
- „`Möchten Sie alle bis auf markierte Fahrgemeinschaften auflösen? [j/N]`“ (5.3.8.8)
- „`Wegsuche starten? [J/n]`“ (5.3.10.1)
- „`Änderungen speichern? [J/n]`“ (5.3.11)
- „`Datei existiert bereits. Überschreiben? [j/N]`“ (5.4.1.4)
- „`Weiter? [J/n/=]`“ (5.4.1.7)
- „`Alles Korrekt? [J/n]`“ (5.4.1.9)

5.7.3 Fehler

- „`Fehler 1: Eingabe enthält unzulässige Zeichen oder ungültigen Bereich!`“ (5.3.9.3)
- „`Fehler 3: Adresse nicht im Verkehrsgraphen vorhanden`“ (5.3.6.1)
- „`Fehler 4: Kante nicht im Verkehrsgraphen vorhanden`“ (5.3.6.1)
- „`Fehler 5: Obere Grenze kleiner untere Grenze`“ (5.3.6.1)
- „`Fehler 6: Person bereits vorhanden`“ (5.3.6.1)
- „`Fehler 7: Person nicht in Liste vorhanden`“ (5.3.6.1)
- „`Fehler 8: Kein Verkehrsgraph geladen`“ (5.3.10.1)

Kapitel 6

Grobentwurf

In diesem Kapitel wird der Stand des Entwurfes zum Zeitpunkt des Zwischenberichts dargestellt. Ausgehend von einem Grobentwurf wurden Teilgebiete mit Verantwortlichkeiten festgelegt. Den groben Aufbau des Systems ersieht man aus Abb. 6.1.

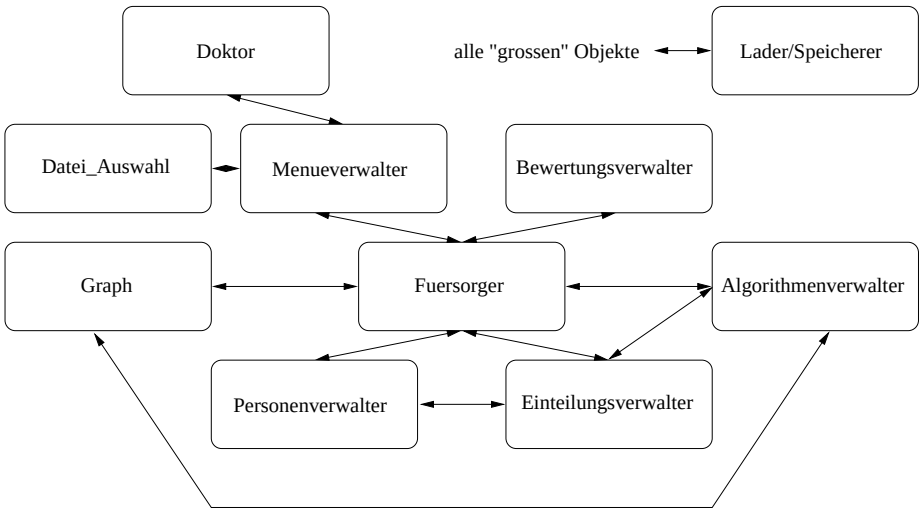


Abb. 6.1: Der Grobaufbau des Systems

Die Teilgebiete und die Verantwortlichen sind:

- Menüverwalter und Doktor : Daniela Nicklas
- Algorithmenverwalter und Bewertungsverwalter : Herbert Heid und Volker Scholz
- Personenverwalter und Einteilungsverwalter : Thomas Schäffer
- Fürsorger, Datei-Auswahl und Lader/Speicherer : Alexander Porrmann

6.1 Menüverwalter und Doktor

Die Kontrollfluß während des Programmlaufes liegt hauptsächlich beim Menüverwalter. Dies ermöglicht ein leichtes Austauschen der textuellen durch eine graphische Oberfläche. Der Doktor enthält Warnungen und Fehlermeldungen im Klartext, diese werden von den Menüs auf dem Bildschirm ausgegeben. Bei einem Fehler wird ein entsprechender Fehlercode als Rückgabewert geliefert, den das Menü dann behandelt, indem es den Text aus dem Doktor ausliest, ausgibt und sich dem Fehler entsprechend verhält. Im Doktor können auch die Systemmeldungen abgelegt werden.

6.2 Algorithmenverwalter und Bewertungsverwalter

Hier liegt der Knackpunkt des Entwurfes. Die Algorithmen zur Wegsuche und zur Berechnung von Einteilungen müssen verwaltet werden und ein Zugriff auf die entsprechenden Daten gewährleistet sein. Hauptpunkt ist hier, sich eine passende Form für die Algorithmen und den Ablauf der Berechnungen zu überlegen. Das Ausprogrammieren der einzelnen Algorithmen wird dann später von Untergruppen übernommen. Der Bewertungsverwalter übernimmt die Verwaltung der Bewertungsfunktionen zur Bewertung von Fahrgemeinschaften und Einteilungen.

6.3 Personenverwalter und Einteilungsverwalter

Diese beiden Verwalter wachen über die Daten des Systems. Abgespeichert werden hauptsächlich Personen- und Einteilungsdaten. Dies ist wohl der am wenigsten komplexe, aber arbeitsaufwendigste Teil.

6.4 Fürsorger, Datei-Auswahl und Lader/Speicherer

Der Fürsorger ist die Schnittstelle zwischen dem Menüverwalter und dem Rest des Systems. Er verteilt die Nachrichten und speichert einige Systemeinstellungen. Das gewährleistet die Möglichkeit, später ein Framework für die graphische Oberfläche zu verwenden, ohne die Systemstruktur zu ändern. Die Datei-Auswahl ist ein Hilfswerkzeug für den Filedialog. Der Lader/Speicherer bedient sich der Voreinstellungen und stellt einen konsistenten Anfangszustand beim Programmstart her.

Anhang A

Glossar

Arbeitszeiten Anfangs- und Endzeit der Arbeit bzw. Intervalle bei Gleitzeit.

Attribute einer Person sind Eigenschaften, Präferenzen, Verwaltungsdaten, die mit einer Person verbunden sind.

automatisches Hilfesystem kontextsensitive Hilfe, die in jeder Situation zur Verfügung steht. In unserem Fall durch den Menüpunkt `Hilfe`.

Benutzer meint Benutzer oder Benutzerin des FGM-Systems.

Bewertungsfunktion h bewertet eine Einteilung, bzw. eine Fahrgemeinschaft. Sie verwendet den Umweg, Präferenzen und Neigungen.

Datenstatus einer Person ist vollständig oder unvollständig.

Eigenschaften der Person gewichtete, erweiterbare Eigenschaften, mindestens: Geschlecht, Raucher oder Nichtraucher, Musikgeschmack.

Eingabeschnittstelle die Schnittstelle zur interaktiven Eingabe von Daten und Dateischnittstelle.

Einteilung siehe Fahrgemeinschaftseinteilung.

entkoppelt vom System als Modul in anderen Programmen einsetzbar

Erweiterbarkeit des Systems heißt klare Festlegung der Algorithmenschnittstellen zum einfachen Austausch.

explizite Zu- und Abneigung Funktionen, die einer Person eine Menge von Personen zuordnen $Z(P_1) = \{P_4, P_7, P_3\}$, $A(P_1) = \{P_2, P_3\}$, wobei $Z(P) \cap A(P) = \emptyset$. Zuneigung hat man zu den Personen, mit denen man auf jeden Fall in einer Fahrgemeinschaft mitfahren möchte. Abneigung analog.

Fahrer einer Fahrgemeinschaft ist die Person, die das Auto fährt, mit dem die Teilnehmer der Fahrgemeinschaft reisen.

Fahrgemeinschaft ist eine Gruppe von Personen, die mit einem Fahrzeug einen gemeinsamen Weg zurücklegt.

Fahrgemeinschaftseinteilung ist die Aufteilung des Personenstammes gemäß einer Bewertungsfunktion in Fahrgemeinschaften, wobei nicht alle Personen eingeteilt werden müssen.

FGM Abkürzung für Fahrgemeinschaft.

FGM-System das System, das zum Finden und Verwalten von Fahrgemeinschaften von der Projektgruppe erstellt wird.

Generierung von Personen Zufällige, wahrscheinlichkeitsverteilte Generierung von Personendaten, bei der einzelne Verteilungen angegeben werden können.

Grapheneditor dient der Visualisierung des Verkehrsgraphen, stellt Ausschnitte dar.

GDF Geographic Data File, digitales Format zur Darstellung von Verkehrsdaten (Ausgangsinformation unseres Systems).

heuristische Partitionierung Die Personen werden entsprechend der Bewertungsfunktion in Fahrgemeinschaften aufgeteilt. Dies geschieht mit Hilfe von Heuristiken.

hierarchische Graphen Level- oder Regionengraphen zur effizienten kürzeste Wege-Berechnung.

Intervall bezüglich der Gleitzeit ein Zeitraum, in dem die Arbeitszeit beginnt oder endet, z.B. $[8^{00}\text{-}9^{30}]$.

Kommentare Ausgabe von Programmlaufinformationen, falls Systemmeldungen eingeschaltet sind.

kürzeste Entfernung nach Wegstrecke kürzeste Entfernung zwischen zwei Knoten bzw. Kanten.

kürzeste Weg-Suche Algorithmen zum Finden kürzester Wege zwischen ausgewählten Knoten bzw. Kanten.

Laufzeitmessung Messung der effektiven Laufzeiten bestimmter Komponenten des Systems.

Levelgraph Graph aus mehreren Ebenen, bei dem z.B. verschiedene Straßentypen auf verschiedenen Leveln liegen (Level 0: Feldwege, Level 1: Gemeindestraßen, Level 2: Bundesstraßen, Level 3: Autobahnen).

Markierung von Fahrgemeinschaften kann gesetzt oder nicht gesetzt sein. Ist eine Fahrgemeinschaft markiert, muß man sie zum Auflösen freigeben. Ist eine Fahrgemeinschaft nicht markiert, kann sie problemlos aufgelöst werden.

Präferenzen Zu- und Abneigungen hinsichtlich bestimmter Eigenschaften, die in die Bewertungsfunktion eingehen.

Programmausgabe bezieht sich auf die Anfragen, die das System beantworten kann, die Ausgabe von Personendaten und das Anzeigen des Verkehrsgraphen.

Regionengraph Graph auf mehreren Ebenen, wobei Knoten Regionen entsprechen und Teilmengen von Knoten der nächst niedrigeren Ebene repräsentieren.

reserviert heißt ein Platz, wenn eine Person einer Fahrgemeinschaft zugeordnet ist, jedoch noch nicht geklärt ist, ob sie dort auch wirklich mitfährt.

Start- und Zielort der FGM der Start- und Zielknoten des Fahrers bzw. die Start- und Zielkoordinaten des Fahrers. Der Fahrer wohnt meistens an einer Straße (Kante). Zum einfacheren Umgang kann man die Koordinaten jedoch auf Knoten reduzieren. Die Knoten sind dann die den Koordinaten am nächsten liegenden Straßengraphknoten.

Start- und Zielort der Person der Start- und Zielknoten einer Person bzw. die Start- und Zielkoordinaten einer Person.

Status einer Person kann *fest eingetragen*, *reserviert eingetragen* oder *vermittelbar* sein.

Systemmeldungen Modus, in dem das System laufen kann und dann Kommentare zum aktuellen Stand an den Benutzer ausgibt.

Teilnehmer ist eine Person, deren Daten im System FGM erfaßt sind.

Teilnehmer einer Fahrgemeinschaft ist eine Person, die einer Fahrgemeinschaft zugeordnet ist. Ihr Status ist dabei *reserviert eingetragen* oder *fest eingetragen*.

Umweg einer FGM ist der nach Wegstrecke berechnete zusätzliche Weg. Er berechnet sich aus der Länge des kürzesten Weges des Fahrers einer Fahrgemeinschaft und dem Fahrtweg, den der Fahrer mit der Fahrgemeinschaft zurückzulegen hat.

Unterbrechung des Programmlaufes Bei der Berechnung der Einteilung kann der Benutzer das System stoppen. Anschließend kann an der abgebrochenen Stelle fortgefahren werden.

Verkehrsgraph ist der einer Straßenkarte entsprechende Graph mit Kantengewichten.

Wegstrecke ist die von einem Knoten zu einem anderen Knoten im Verkehrsgraphen zurückzulegende Wegstrecke in km oder m.

Anhang B

Konvertierung von Verkehrsdaten

B.1 Einleitung

Zur Berechnung kürzester Wege bei der Zusammenstellung von Fahrgemeinschaften benötigt man reale Straßenverkehrsdaten. Mittlerweile liegen diese auch in elektronischer Form vor und werden z.B. in Autocopiloten (Navigationshilfen) verwendet. Vom Institut für Photogrammetrie wurde uns ein GDF-Datensatz für das Stadtgebiet von Stuttgart zur Verfügung gestellt (Stand 1993). GDF steht für Geographic Data File und ist ein europäischer Standard für Straßenverkehrsdaten, der von Autoherstellern erarbeitet wurde. Zur weiteren Verwendung im Projekt wurden die relevanten Teile aus den Daten extrahiert und in ein eigenes Graphenformat gebracht.

Zunächst erfolgt eine Beschreibung des Quellformats (GDF) und des Zielformats (Graphenformat). Danach wird kurz die Implementierung des Konverters erläutert und auf die Visualisierung der Daten eingegangen. Im letzten Abschnitt erfolgt eine Auflistung der wichtigsten Recordformate in GDF.

B.2 GDF-Format

Die zur Verfügung gestellten Daten lagen in GDF-Version 2.1 vor, für die schon eine umfangreiche Dokumentation existiert (siehe [5]). Nach ein paar allgemeinen Bemerkungen werden hier nur die für die Konvertierung relevanten Teile beschrieben.

GDF liegt als ASCII-File vor und besteht aus verzeigerten Records, die wichtigsten Unterklassen sind Roadelement Records (Straßensegmente), Junction Records (Kreuzungen), Edge (Kanten), Node (Knoten) und XY-coordinate Records (Koordinateninformation). Abb. B.1 zeigt die Verzeigerung dieser Hauptrecordtypen. Roadelements und Junctions sind den geometrischen Ele-

menten (Edges und Nodes) übergeordnet, die sich wiederum beide auf XY-Koordinaten beziehen. Straßensegmente können mit Attributen (s.u.) und Na-

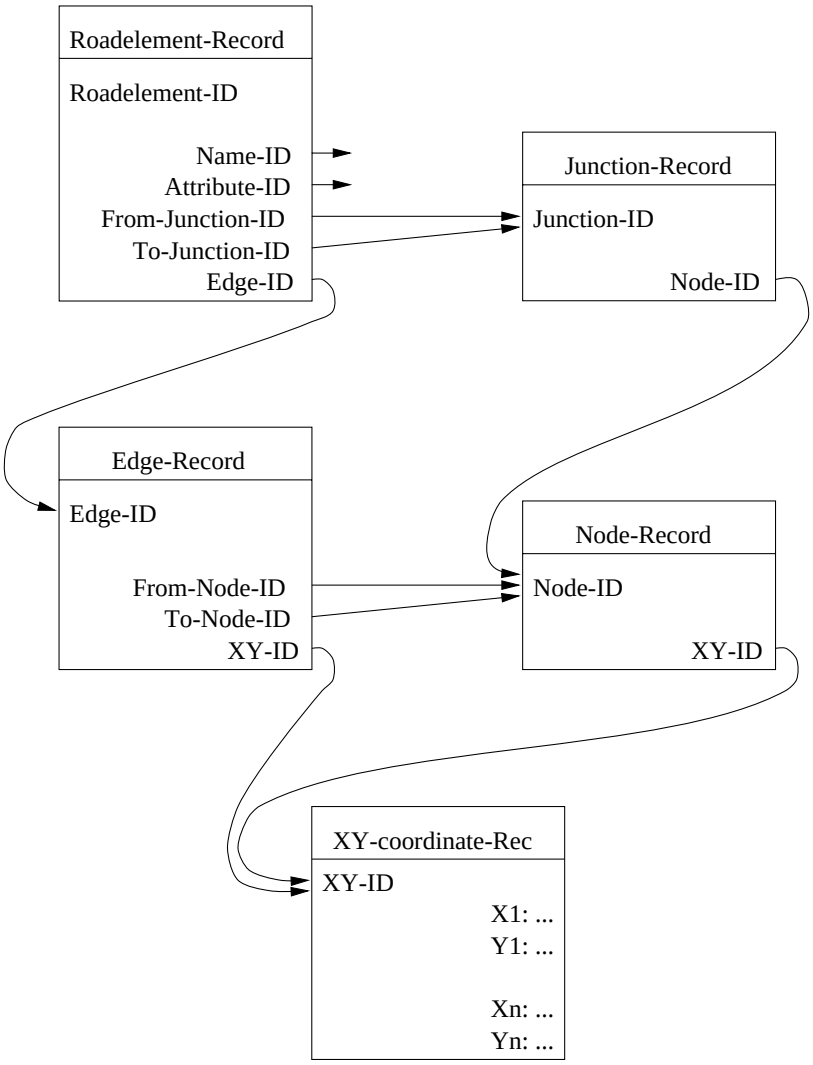


Abb. B.1: Die wichtigsten Recordtypen und ihre Verzeigerung

men versehen werden; dazu dienen Segmented Attribute Records und Name Records. Zur Erfassung von Abbiegeverboten gibt es schließlich Prohibited Turn Records, eine Unterklasse der Relationship Records.

In Abschnitt B.6 sind die Formate dieser Recordtypen aufgelistet, jedes Recordfeld ist durch einen Index mit einer Längenangabe versehen, dabei steht * für ein Feld beliebiger Länge. {}* zeigt an, daß ein Feld auch wiederholt auftreten kann, in solchen Fällen wird vorher die Anzahl der Wiederholungen in einem NUM-Feld angegeben. Ziffernfolgen deuten darauf hin, daß der Inhalt des betreffenden Feldes fest ist. Die wichtigsten Recordfelder werden für jeden

Recordtyp kurz erläutert.

Im GDF-Format sind nur Zeilen mit einer maximalen Länge von 80 Zeichen (plus eines oder zwei für den Zeilenumbruch) zugelassen, lange Records werden also umgebrochen und mit einer Umbruchmarkierung versehen.

Folgende Attribute der Straßensegmente wurden aus den Daten extrahiert:

- Straßenname
- Einbahnstraßen
- Functional Road Class: hier werden die Straßen nach Wichtigkeit klassifiziert, laut Dokumentation gibt es folgende Straßenklassen:
  - Klasse 0: Autobahn
  - Klasse 1: Bundesstraße
  - Klasse 2: Hauptstraße
  - Klasse 3: Nebenstraße
  - Klassen 4+5: befestigter Fahrweg
  - Klassen 6-8: undokumentiert, u.a. auch Fußgängerzonen
- Straßennummer (z.B. B10, E52 falls vorhanden)

Abbiegeverbote liegen als dreistellige Relation zwischen Einfahrtsstraße, Kreuzung und Ausfahrtsstraße vor.

B.3 Graphenformat

Als vorläufiges Graphenformat wurde drei Dateiformate für die Knoten, Kanten und Abbiegeverbote des Graphen festgelegt. Die Knoten werden mit ID, x- und y-Koordinate abgespeichert:

node_id	x-coord	y-coord

Die Kanten werden ebenfalls als einfache Kantenliste mit den entsprechenden Attributen abgelegt:

edge_id	source_id	target_id	length	road class	direction	road no.	name

Source_id und target_id beschreiben die inzidenten Knoten und die Richtung der Kante edge_id, mittels direction werden die möglichen Durchfahrtsrichtungen durch diese Kante angegeben (1=beide, 2=positive, 3=negative Richtung, bezogen auf die Kantenrichtung). Length gibt die Länge der Kante in Metern an, die restlichen Felder stehen für Straßenklasse, Straßennummer und Straßennamen.

Die Abbiegeverbote werden als dreistellige Relation zwischen Kreuzung, Einfahrts- und Ausfahrtsstraße abgespeichert:

junction_id	from_edge_id	to_edge_id

B.4 Umwandlung

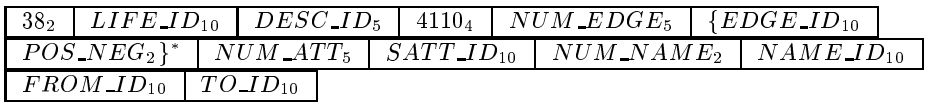
Zur Umwandlung wurden Perl-Skripte implementiert, mit denen die Verzeigerung in assoziativen Arrays im Hauptspeicher aufgebaut wurde. Zur Darstellung des Graphen in der Ebene war noch eine kartographische Projektion nötig, deswegen wurde die in GDF vorliegenden geographischen Koordinaten (Länge, Breite) in Gauß-Krüger-Koordinaten umgerechnet (konforme Abbildung, siehe [7]). Somit trägt jeder Knoten des Graphen eine x- und y-Koordinate in der Einheit Meter.

B.5 Visualisierung der Daten

Zur Darstellung des Verkehrsgraphen im Graphenformat wurde von Dirk Farin das Programm GraphView implementiert. Dieses erlaubt die Darstellung beliebiger Ausschnitte des Verkehrsgraphen, die Straßenklassen haben unterschiedliche Farben und können selektiv ein- und ausgeschaltet werden. Die Suche nach Straßennamen ist ebenfalls möglich.

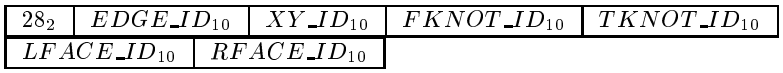
B.6 Recordformate

- Roadelement (Line Feature Record):



  - $LIFE\_ID$: ID des Roadelements
  - NUM_EDGE: Anzahl der Edges, die zu diesem Roadelement gehören
  - $EDGE\_ID$, POS_NEG: Zeiger auf Edges, relative Orientierung der Edge zu Roadelement
  - NUM_ATT: Anzahl der zugehörigen Attribute Records
  - $SATT\_ID$: Zeiger auf Attribute Record
  - NUM_NAME, $NAME\_ID$: Anzahl Namen, Zeiger auf Name Record
  - $FROM\_ID$, TO_ID: Zeiger auf angrenzende Junctions (from, to)

- Edge Record:



  - $EDGE\_ID$: ID dieser Edge
  - XY_ID: Zeiger auf XY-coordinate Record, der die Zwischenpunkte dieser Edge enthält

  - $FKNOT\_ID$, $TKNOT\_ID$: Zeiger auf angrenzende Nodes (from, to)

- New Node Record:

25_2	$KNOT\_ID_{10}$	XY_ID_{10}	$FACE\_ID_{10}$	$STATUS_2$

  - $KNOT\_ID$: ID dieses Nodes
  - XY_ID: Zeiger auf XY-coordinate Record, der Nodekoordinaten enthält

- XY-coordinate Record:

22_2	XY_ID_{10}	G_TYPE_1	Q_PLAN_2	$DESC\_ID_5$	NUM_COORD_5
$\{X\_COORD_{10}$	$Y\_COORD_{10}\}^*$				

  - XY_ID: ID dieses XY-coordinate Records
  - NUM_COORD: Anzahl Koordinaten in diesem Record
  - X_COORD, Y_COORD: geographische Länge, Breite

- Junction (Point Feature Record):

37_2	$POINT\_ID_{10}$	$DESC\_ID_5$	4120_4	NUM_KNOT_5	$\{KNOT\_ID_{10}\}^*$
NUM_ATT_5	$\{SATT\_ID_{10}\}^*$	NUM_NAME_2	$\{NAME\_ID_{10}\}^*$		

  - $POINT\_ID$: ID dieser Junction
  - NUM_KNOT: Anzahl der Nodes, die zu dieser Junction gehören
  - $KNOT\_ID$: Zeiger auf zugehörigen Node
  - NUM_ATT, $SATT\_ID$: Anzahl Attribute Records, Zeiger auf Attribute Record
  - NUM_NAME, $NAME\_ID$: Anzahl Namen, Zeiger auf Name Record

- Segmented Attribute Record:

43_2	$SATT\_ID_{10}$	$FROM_5$	TO_5	NUM_ATT_5	ATT_TYPE_2
$DESC\_ID_5$	ATT_VAL_{10}				

  - $SATT\_ID$: ID diese Attribute Records
  - NUM_ATT: Anzahl der Attribute, die in diesem Record abgelegt sind
  - ATT_TYPE, ATT_VAL: Attributtyp (Attribute Type Code), Attributwert (Attribute Value Code)
  - Attribute Type Codes mit zugehörigen Attribute Value Codes:
  - DF = Direction of Traffic Flow, 1 = both, 2 = positive, 3 = negative, 4 = none
  - RN = Route Number (z.B. B10, E52)
  - FC = Functional Road Class, Klassen 0-8 (s.o.)

- Name Record:

41_2	$NAME\_ID_{10}$	$DESC\_ID_5$	LAN_CODE_3	$TEXT_*$

- $NAME\_ID$: ID dieses Name Records
- LAN_CODE: Sprache, GER = Deutsch
- $TEXT$: Freitextfeld beliebiger Länge

- Prohibited Turn (Relationship Record):

49_2	REL_ID_{10}	2101_4	$DESC\_ID_5$	3_1	2_1	$FEAT\_ID_{10}$	1_1	$FEAT\_ID_{10}$
2_1	$FEAT\_ID_{10}$	NUM_ATT_5	$SATT\_ID_{10}$	NUM_NAME_2	$NAME\_ID^*_{10}$			

  - REL_ID: ID dieses Relationship Records
  - 3x $FEAT\_ID$: Zeiger auf Roadelement from, Junction, Roadelement to (Abbiegeverbot wird durch Einfahrtsstraße, Kreuzung und Ausfahrtsstraße beschrieben)

Literaturverzeichnis

[1] J. L. Bentley and R. A. Finkel. *Quad trees - A Data Structure for Retrieval on Composite Keys*, chapter 1-9. AA, Acta Informatica 4, 1974.

[2] Grady Booch. *Objektorientierte Analyse und Design.* Addison Wesley, 1994.

[3] Friedhelm Buchholz. Komplexität des Fahrgemeinschaften-Problems. Studienarbeit 1327. Universität Stuttgart Institut für Informatik, 1994.

[4] Friedhelm Buchholz. Entwurf eines Systems zur Vermittlung von Fahrgemeinschaften, Diplomarbeit 1226. Universität Stuttgart Institut für Informatik, 1995.

[5] H. Claussen et al. GDF 2.1 Draft Standard. unpublished, 1992.

[6] Greg N. Frederickson. Fast Algorithms for Shortest Paths in Planar Graphs. *SIAM Journal on computing, IEEE*, Vol. 16(No. 6), December 1987.

[7] Walter Großmann. *Geodätische Rechnungen und Abbildungen in der Landesvermessung.* Konrad-Wittwer-Verlag, 3rd edition, 1976.

[8] John E. Hopcroft and Richard M. Karp. An $n^{2,5}$ algorithm for maximum matching in bipartite graphs. *SIAM Journal on computing, IEEE*, pages 225–231, 1973.

[9] IEEE Computer Society (Hrsg.). IEEE guide for software requirements specification. *IEEE Std 830-1984*, 1984.

[10] IEEE. *Software Engineering Standards Collections*, 1994.

[11] Michael Jackson. *Software Requirements & Specifications.* Addison-Wesley, 1995.

[12] D. T. Lee and C. K. Wong. *Worst-Case Analysis for Region and Partial Region Searches in Multidimensional Binary Search Trees and Balanced Quad Trees*, pages 23–29. AA, Acta Informatica 9, 1977.

[13] Richard J. Lipton and Robert Endre Tarjan. A Separator Theorem for Planar Graphs. *SIAM J. APPL. MATH.*, Vol. 36(No. 2), April 1979.

[14] Jochen Ludewig. Grundlagen des Software Engineerings. Fachschaft Informatik, 1997. Skript zur Vorlesung im Sommersemester '97.

[15] K. Mehlhorn. *Data Structures and Algorithms 3: Multi-dimensional Searching and Computational Geometry*, pages 24–55. Springer-Verlag, 1984.

[16] Silvio Micali and Vijay V. Vazirani. An $O(\sqrt{|V|}|E|)$ algorithm for finding maximum matching in general graphs. *IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 17–27, 1980.

[17] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction*, pages 67–77 and 189–199. Springer-Verlag, 1985.

[18] W. W. Royce. *Managing the development of large software systems: concepts and techniques.* Proc. IEEE WESTCON, Los Angeles, 1970.

[19] Vijay A. Saraswat. Principles and practice of constraint programming - the newport papers. *MIT Press, Cambrigde*, 1995.

[20] T. Ottmann und P. Widmayer. *Algorithmen und Datenstrukturen*, pages 232–242. Reihe Informatik Band 70. BI-Wissenschaftsverlag, 1993.