Institut für Parallele und Verteilte Systeme
Abteilung Verteilte Systeme
Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Diplomarbeit Nr. 3102

Platzierungsoptimierung für vertrauliche Verwaltung der verteilten Positionsinformationen

Björn Schembera

Studiengang:	Informatik
Prüfer:	Prof. Dr. rer. nat. Dr. h. c. Kurt Rothermel
Betreuer:	M. Sc. Pavel Skvorzov
begonnen am:	2. November 2010
beendet am:	4. Mai 2011
CR-Klassifikation:	C.2.4, G.1.6, I.2.8, K.4.1

Abstract

Die vorliegende Arbeit beschäftigt sich mit der Sicherheit der Privatsphäre bei Location-based Services. Solche Dienste erlauben es den Nutzern mobiler Geräte wie Smartphones, Informationen zur Umgebung zu erhalten (z.B. welche Restaurants sich in der Nähe befinden). Dabei muss die Positionsinformation des Benutzers stets bekannt sein. Hier wird das Anliegen nach Privatsphäre ein zentrales: Wer kann zu welcher Zeit wie auf Positionsdaten zugreifen? Es besteht das Problem, dass der Speicherort dieser Positionsinformation (Provider) kompromittiert sein oder es sich um einen nicht vertrauenswürdigen Provider handeln kann. In diesem Fall ist es Dritten technisch möglich, unautorisiert über Positionsinformation zu verfügen. Dieses Sicherheitsrisiko kann minimiert werden, indem die Positionsinformation zu Teilen (Shares) auf autonome Provider verteilt wird, so dass sich die exakte Position nur aus allen Teilen rekonstruieren lässt. Der Ansatz ist somit eine Verteilung der Positionsinformationen. Dieses bereits entwickelte System wird im Rahmen dieser Diplomarbeit so erweitert, dass abhängig von den Parametern (Vertrauen, Risiko, usw.) unterschiedlich viele Teile auf dem jeweiligen Provider platziert werden – dadurch können unterschiedliche Sicherheitsbedenken bei diesen Providern ausbalanciert werden und eine wenigstens angemessene, d.h. konstante Verschlechterung der Privatsphäre bei zunehmender Zahl kompromittierter Server erreicht werden. Keine Instanz des Systems ist überproportional kritischer Punkt im Bezug auf Privatsphäre.
Hierzu wurde das System zunächst um eine Trust Database erweitert, in der die Vertrauenswerte der Provider gehalten und verwaltet werden. In der vorliegenden Diplomarbeit werden Lösungsansätze für die optimale Sicherheit hinsichtlich verschiedener Szenarien bei verteilter Positionsinformation erörtert. Ein Szenario ist ein Zuweisungsproblem aus der Klasse der kombinatorischen Optimierung und ist im Allgemeinen schwer zu handhaben. Hierfür werden algorithmische Lösungsansätze erarbeitet und evaluiert.

Danksagung

Zunächst möchte ich meinem Betreuer *Pavel Skvorzov* für die gute Zusammenarbeit danken. Er fand immer die Zeit, Probleme auch im Detail zu diskutieren.
Für das sorgfältige Probelesen und die vielen konstruktiven Anmerkungen danke ich *Kaja* und *Amir* herzlich.
Schließlich möchte ich natürlich *meinen Eltern* danken, für die fortwährende Unterstützung während des Studiums und dafür, dass sie mir 1997 mein erstes Programmierbuch schenkten.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Verzeichnis der Algorithmen

Abkürzungsverzeichnis

$\mathcal{NP}$	Komplexitätsklasse der nichtdeterministisch-polynomiellen Zeit
$\mathcal{P}$	Komplexitätsklasse der deterministisch-polynomiellen Zeit
ABGAP	Agent Bottleneck Generalized Assignment Problem
AP	Assignment Problem, Zuweisungsproblem
BaGAP	Balanced Generalized Assignment Problem
BAP	Bottleneck Assignment Problem
BGAP	Bottleneck Generalized Assignment Problem
BOP	Balanced Optimization Problem
GAP	Generalized Assignment Problem, Generalisiertes Zuweisungsproblem
LBS	Location-based Service, Standortbezogener Dienst
LS	Location Server
POI	Point of Interest
TBGAP	Task Bottleneck Generalized Assignment Problem

Variablenverzeichnis

Variable	Erklärung
m	Anzahl der Provider
n	Anzahl der Shares
I	Menge aller Provider $I = \{1, 2, \cdots, m\}$
J	Menge aller Shares $J = \{1, 2, \cdots, n\}$
i	Zählvariable für Provider, $i \in I$
j	Zählvariable für Shares, $j \in J$
r_i	Risiko des Providers i
$r_{i(start)}$	Initiales, frei wählbares Risiko des Providers i, wobei $r_{i(start)} \in [0, 1]$
s_j	Gewichtung der Share j
S_i	Sharerisiko aller einem Provider i zugeordneter Shares
n_i	Zahl der Shares auf dem Provider i
R_i	Fragilität eines Providers i (spezifisches Sicherheitsrisiko)
Δ	Die Differenz zwischen größtem und kleinstem Fragilitätswert
c_{ij}	Kosten für eine Zuweisung von Objekt/Share i auf Objekt/Provider j, $c_{ij} \in \mathbb{N}$
x_{ij}	Zuweisung von Objekt/Share i auf Objekt/Provider j, $x_{ij} \in \mathbb{B}$
X	Zuweisungsmatrix $X = (x_{ij})$
C	Kostenmatrix $C = (c_{ij})$
X_{GA}	Als Genom codierte Zuweisung in Form eines Vektors
t	Der Generationenzähler der evolutionären Algorithmen

1 Einleitung

1.1 Motivation

In seinem bahnbrechenden Aufsatz „The Computer for the 21st Century“ [Wei91] schrieb Mark Weiser 1991 von seiner Vision des *Ubiquitous Computing* – einer Vorstellung, wonach der wesentliche Fortschritt der Informationstechnologie im 21. Jahrhundert darin bestünde, dass die Computerisierung zunehmend unsichtbar und somit in das Alltägliche Einzug halten werde. Als technologische Grundlage für diese Entwicklung macht Weiser günstige, energieeffiziente und kleine Endgeräte, ein Kommunikationsnetzwerk zwischen diesen Geräten sowie eine Software-Anwendung aus. Vor allem die ersten beiden Aspekte finden sich heute in der maßenhaften Verbreitung von Smartphones realisiert – zumindest teilweise, weil die Geräte zwar ubiquitär[1], d.h. allgegenwärtig sind, aber nach wie vor bewusst genutzt werden und noch nicht in den „Hintergrund getreten“ sind, wie es der Kern seiner Vision vorsieht. 1991 schreibt Mark Weiser:

> *„Heutzutage haben Computer keinerlei Wissen von ihrer Position und Umgebung. Wenn ein Computer lediglich wüßte, in welchem Raum er sich befände, könnte er sein Verhalten in spezifischer Weise anpassen, ohne dass auch nur eine Spur künstlicher Intelligenz nötig wäre.“* [Wei91] (eigene Übersetzung)

In Zukunft soll der *Kontext*, in dem ein Computer genutzt wird, das Programmverhalten beeinflussen[2]. Unter Kontext lässt sich abstrakt ein Zusammenhang zwischen verbundenen Teilen verstehen, welche die Situation eines Individums – oder genereller einer Entität – charakterisieren, also die Gesamtheit ihrer physischen, physiologischen und sozialen Gegebenheiten [RDD+03] [DK06]. In der vorliegenden Arbeit wird Kontext als physischer Zusammenhang zwischen dem Raum und einem Individuum verwendet. Kontext entspricht im Folgenden also der *Position* eines Individuums oder einer Entität. Mittels der Position als Kontext soll sich das Verhalten des Computersystems an die momentane Umgebung anpassen oder Informationen dazu liefern. 1991 war man noch weit davon entfernt, die Positionsinformationen von Geräten einfach bestimmen zu können. Heute, 20 Jahre später, ist es technisch kein Problem mehr, die Positionen selbst von mobilen Geräten zu orten:

[1] Der Begriff „Ubiquität“ kommt eigentlich aus dem Bereich der Theologie und bezeichnete die Allgegegenwart Gottes, wobei Gott als physikalisch nicht lokalisierbar geglaubt wird [RM98].

[2] Daher wird auch von *context-based-* oder *context-aware computing* gesprochen [REF+06] [DK06] [RDD+03].

Eine Vielfalt von Möglichkeiten ist vorhanden und sogar die Bestimmung von Positionen innerhalb von Räumen ist handhabbar.

Der Forschungsbereich, in den diese Diplomarbeit eingebettet ist, ist das *Nexus*-Projekt[3] des Sonderforschungsbereichs 627 „Umgebungsmodelle für mobile kontextbezogene Anwendungen“ [REF+06]. Dort soll durch Kontextinformationen ein Umgebungsmodell der Welt geschaffen werden, das Mark Weisers oben skizzierten Vision nahe kommt.
Eine zentrale Stellung innerhalb dieses Projekts nehmen *Location-based Services* ein. Location-based Services sind Dienste, die einem mobilen Nutzer Informationen bezogen auf seinen gegenwärtigen Standort liefern. Bei diesen Informationen kann es sich beispielsweise um eine Liste aller Sehenswürdigkeiten in einem gewissen Umkreis des Nutzers handeln.
Die Position des Nutzers muss der Instanz, die die Positionen verwaltet, stets bekannt sein, um solche Dienste anbieten zu können. Aus diesem Grund handelt es sich bei Location-based Services um eine sensible Technologie in Bezug auf die Privatsphäre der Nutzer. Eine Aufgabe von Location-based Services ist es demnach, die Privatsphäre so gut wie möglich im Rahmen der technischen Möglichkeiten zu schützen. Dabei gibt es mehrere Verfahren, die von vertrauenswürdigen Instanzen zur Lagerung der Positionsinformation ausgehen. Das Manko dieser Verfahren ist allerdings die Annahme der Vertrautheit, denn die Instanz kann einerseits von außen kompromittiert sein, andererseits kann es auch von innerhalb des Systems zu unautorisiertem Zugriff kommen, z.B. durch von vornherein nicht vertrauenswürdige Firmen oder totalitäre staatliche Institutionen.
Um diesen Schwächen beizukommen, wurden innerhalb des Nexus-Projekts Verfahren entwickelt, die mit mehreren unabhängigen Instanzen zur Speicherung der Positionsinformation, genannt *Provider*, arbeiten [DSR11]. Das Risiko soll verringert werden, indem die exakte Positionsinformation in mehrere Teile aufgeteilt wird. Diese Teile werden auch als *Shares* bezeichnet. Die Shares werden dann auf die Provider aufgeteilt. Sicherheit der Privatsphäre soll erreicht werden, weil es unwahrscheinlich ist, dass alle Server gleichzeitig kompromittiert sind. Selbst wenn einer dieser Server gehackt würde, ließe sich nur eine ungenaue Position rekonstruieren.

[3] Projektwebsite: http://www.nexus.uni-stuttgart.de, Zugriff 16.4.2011

1.2 Aufgabenstellung

Im Rahmen dieser Diplomarbeit soll das Systemmodell in [DSR11], welches mit mehreren Location Servern arbeitet, zunächst so erweitert werden, dass mit jedem Provider, der Positionsinformationen speichert, ein Risikowert assoziiert wird. Außerdem können die Shares gewichtet und beide Fälle kombiniert werden. Da jede Zuweisung einer Share auf einen Provider einen anderen Faktor liefert, um den die Genauigkeit der Position verfeinert wird, kommt es auf die getroffenen Zuweisungen an, wie genau die exakte Position bei kompromittierten Providern rekonstruiert werden kann.
Konkret müssen dabei einem Provider, der ein hohes Sicherheitsrisiko aufweist, relativ weniger Positionsinformationen zugewiesen werden als einem, der ein niedriges Sicherheitsrisiko hat. Eine gleichförmige Verteilung von Sicherheitsrisiken zwischen den Servern ist erwünscht und Balance von entscheidender Bedeutung. Die Aufgabe ist folglich, die Positionsinformationen so auf die Provider zu verteilen, dass die Sicherheitsrisiken ausbalanciert werden. Dieser optimale Zustand ist in Ausdruck (1.1) formalisiert.

$$R_1 = R_2 = \cdots = R_m \tag{1.1}$$

Sind die Shares ungewichtet, soll damit erreicht werden, dass risikoreiche Provider, deren Kompromittierung wahrscheinlicher ist, weniger Shares bekommen als vertraute Provider. Sind die Shares gewichtet, wird durch den Balancezustand eine wenigstens *angemessene Verschlechterung* (engl. *graceful degradation*) der Privatsphäre bei zunehmender Zahl der kompromittierten Server erreicht, was in Abbildung 1.1 dargestellt und folgendermaßen motiviert ist: Bei k kompromittierten Servern kann eine nur um einen konstanten Faktor höhere Genauigkeit der Position ermittelt werden – so gibt es keinerlei Flaschenhals im System, der die Sicherheit überproportional beeinträchtigt, falls jener Server kompromittiert wird.

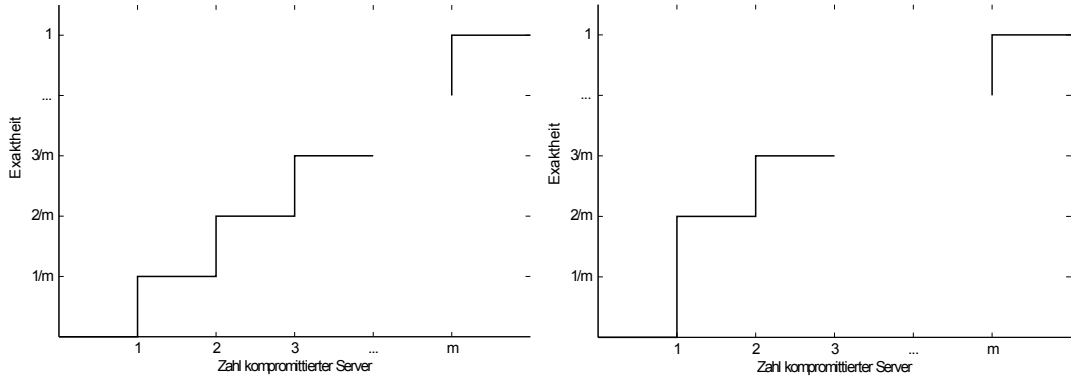

Abbildung 1.1: Angemessene konstante Verschlechterung der Privatsphäre.

1.3 Szenarien

Die Aufgabenstellung wird auf zwei Szenarien bezogen, wobei diese im Folgenden kurz eingeführt werden. In jedem Szenario ist die Zahl der Provider als gegeben anzunehmen.

Szenario 1 In Szenario 1 wird davon ausgegangen, dass die Zahl der Shares und somit auch deren Gewichtungen *frei wählbar* sind.

Szenario 2 In Szenario 2 wird davon ausgegangen, dass neben der Anzahl der Provider auch die Anzahl der Shares *vorgegeben* ist. Damit sind auch die Gewichtungen der Shares als gegeben zu betrachten.

Teilszenarien Jedes Szenario differenziert sich noch wie oben angedeutet in Teilszenarien, wonach entweder Provider (b), Shares (c) oder beide (d) gewichtet sind. Auch der Fall, dass beide ungewichtet (a) sind, wird der Vollständigkeit halber betrachtet. In Tabelle 1.1 sind die Szenarien überblicksartig dargestellt.

Anzahl der Shares					
Szenario 1: nicht vordefiniert			**Szenario 2: vordefiniert**		
Fall	**Providerrisiken**	**Sharerisiken**	Fall	**Providerrisiken**	**Sharerisiken**
1-a	gleich	gleich	**2-a**	gleich	gleich
1-b	gewichtet	gleich	**2-b**	gewichtet	gleich
1-c	gleich	gewichtet	**2-c**	gleich	gewichtet
1-d	gewichtet	gewichtet	**2-d**	gewichtet	gewichtet

Tabelle 1.1: Übersicht über die unterschiedlichen Szenarien

1.4 Der Gang der Untersuchung

Im folgenden zweiten Kapitel werden die theoretischen Grundlagen dieser Arbeit beschrieben sowie die verwandten Arbeiten vorgestellt. Dabei wird einerseits auf die *Location-based Services* eingegangen, die den Rahmen dieser Arbeit bilden. Andererseits wird die *kombinatorische Optimierung* erläutert, die das Fundament für unsere Problemstellung bildet. Es wird auch auf verwandte Problemstellungen sowie auf spezielle und generelle Lösungsmöglichkeiten eingegangen.
Im konzeptuellen Kapitel 3 wird zunächst das *Systemmodell* vorgestellt, was eine bisherige Entwicklung [DSR11] um eine Trust Database erweitert. Dann wird die *Problemstellung* selbst expliziert und die *Anforderungen* an das System werden spezifiziert.
Die eigentliche Problemstellung, nämlich die Platzierung der Shares auf den Servern derart, dass Sicherheitsrisiken ausbalanciert werden, wird anhand der beiden Szenarien konkret behandelt. Für Szenario 2, in dem neben der Anzahl der Provider auch die Anzahl der Shares und deren Gewichte als fest angenommen werden müssen, wird die Problemstellung als Zuweisungsproblem formalisiert und es wird gezeigt, dass es sich dabei um ein NP-schweres Problem handelt. Schließlich werden die Unterklassen des Problems behandelt und algorithmische Lösungsansätze erarbeitet.
In Kapitel 4 folgt die Evaluation der im dritten Kapitel vorgeschlagenen Konzepte.
Das letzte Kapitel fasst die Ergebnisse der Arbeit kritisch zusammen, stellt Anknüpfungspunkte vor und liefert einen Ausblick.

2 Theoretische Grundlagen und verwandte Arbeiten

In diesem Kaptitel werden die theoretischen Grundlagen, die für diese Arbeit von Bedeutung sind, dargestellt sowie auf die verwandten Arbeiten eingegangen. Dabei teilt sich dieses Kapitel in zwei Hauptabschnitte.
Zunächst werden in Abschnitt 2.1 *Location-based Services* vorgestellt, die den Hintergrund der vorliegenden Arbeit bilden. Dabei wird besonderes Augenmerk auf die Sicherheit der Privatsphäre bei Location-based Services gelegt. Location-based Services werden definiert und abgegrenzt, um dann auf die einzelnen Ansätze zur Wahrung der Privatsphäre in Teil 2.2 näher einzugehen.
Abschnitt 2.3 erläutert das theoretische Fundament unserer Problemstellung, nämlich die *kombinatorische Optimierung*. Diese wird zunächst definiert und das Problem der Komplexität bei kombinatorischen Fragestellungen expliziert. Dann werden verschiedene algorithmische Lösungsansätze vorgestellt sowie deren Vor- und Nachteile besprochen. Schließlich wird auf Zuweisungsprobleme, einer speziellen Unterklasse der kombinatorischen Optimierung eingegangen, da sich die Problemstellung dieser Diplomarbeit als Zuweisungsproblem formalisieren lässt.

2.1 Location-based Services

Unter *Location-based Service* (*LBS*) versteht man einen mobilen Dienst, der dem Nutzer standortsbezogene Informationen liefert [VTV+01] [DSR11] [ACD+07]. Ein LBS kann nach Virrantaus et al. [VTV+01] wie folgt definiert werden:

> *„LBS sind Dienste, auf die von mobilen Geräten über ein mobiles Netzwerk zugegriffen wird und die von der Fähigkeit Gebrauch machen, Positionen dieser Geräte bestimmen zu können.“* [VTV+01] (eigene Übersetzung)

Der LBS soll dem Nutzer bezogen auf die Position eines Nutzers Dienste verschiedener Art anbieten. Hierbei kann es sich um einen Dienst handeln, der dem Nutzer angeforderte Informationen über einen interessanten (räumlichen) Punkt (*Point of Interest*, (*POI*)) zukommen lässt, wenn er z.B. auf der Suche nach Hotels in seinem Umkreis ist oder über die

gegenwärtige Verkehrssituation auf seiner Route informiert werden will.
Der Kontext besteht in dieser Form im Wesentlichen aus der Position eines Nutzers – sie ist der zentrale Faktor. Ein LBS besteht aus folgenden Teilen [DSR11] [VTV+01] [RM03]:

- **Mobiles Gerät:** Dieses Gerät wird vom Nutzer geführt und steht im Mittelpunkt des Systems, da hier Dienste angefordert und bezogen werden. Hierbei handelt es sich heutzutage um ein *Smartphone*, also ein Handy mit erweiterter Funktionalität wie bspw. einem GPS-Empfänger.
- **Positionierungssystem:** Mittels dieses Systems kann das mobile Gerät seinen Aufenthaltsort bestimmen. Möglichkeiten zur Ortsbestimmung sind z.B. GPS oder das Mobilfunksystem, welches die ID der momentanen Mobilfunkzelle liefert (die Cell ID).
- **Location Server (LS):** Dieser Teil ist für die Verwaltung der Positionsdaten der mobilen Geräte zuständig, des Weiteren für die Bereitstellung dieser Daten an die ortsbezogene Anwendung. Auf dem LS ist zumindest der momentane Aufenthaltsort, evtl. sogar ein Bewegungsprofil der jeweiligen Endgeräte abgelegt. Außerdem werden auch Positionen von statischen Objekten verwaltet.
- **Ortsbezogene Anwendung:** Dienst bzw. die eigentliche Anwendung, die dem Nutzer bezogen auf seine Position, die im LS nachgeschlagen wird, Informationen liefert.

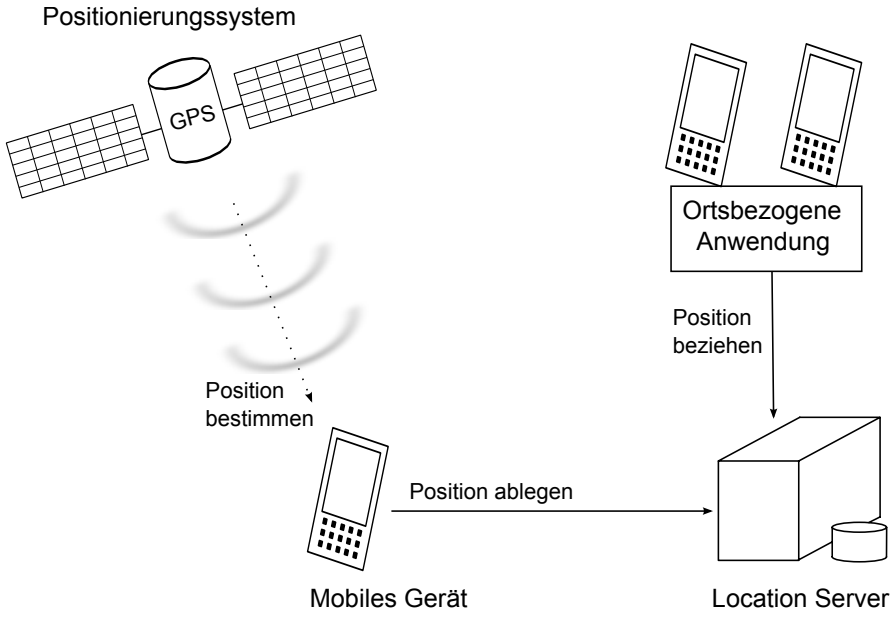

Abbildung 2.1: Die prinzipielle Systemarchitektur bei LBS, bestehend aus *mobilem Gerät*, *Positionierungssystem*, *Location Server* und *Ortsbezogener Anwendung*.

Die Architektur eines LBS ist in Abbildung 2.1 schematisch dargestellt. Zwischen den Geräten muss natürlich ein Kommunikationsnetzwerk bestehen, um die Informationen austauschen

zu könnnen. Da diese Schicht für unsere Problemstellung unerheblich ist, wird auf eine Erläuterung verzichtet[1].
Einfache LBS könnten auch ohne die Location Server auskommen, indem die Position direkt vom mobilen Gerät an den LBS weitergegeben wird und das mobile Gerät die gewünschte Informationen bezogen auf seinen gegenwärtige Position direkt erhält. Sobald aber die Positionen von mehreren Objekten eine Rolle für den LBS spielen, ist ein Location Server notwendig, der die Objektpositionen verwaltet. Nur mittels eines LS lassen sich beispielsweise räumliche Anfragen über mehreren bewegten Objekten effizient realisieren.

2.2 Sicherheit der Privatsphäre in Location-based Services

Im vorhergehenden Abschnitt wurden einige Beispiele für LBS genannt, z.B. ein unterstützender Dienst, der interessante Punkte findet. Dieser ist reaktiv und vom Nutzer autorisiert. Dem gegenüber stehen proaktive Diente, die im Hintergrund ablaufen und auf Ereignisse reagieren, wie dem Besuchen eines Einkaufszentrums, so dass standortbezogene Werbung geschaltet werden kann.
Da den Nutzern bei LBS aufgrund ihrer Position Dienste angeboten werden und ihre Position gewissen Instanzen bekannt ist, handelt es sich um eine sensible Technologie bzgl. der Privatsphäre. Duckham und Kulik [DK06] erweitern die klassische Definition der Privatsphäre für Informationshandhabung nach Westin [Wes67] und kommen zu einer Definition der Privatsphäre der Position:

> *„Privatsphäre ist das Anrecht von Individuen, Gruppen oder Institutionen, selbst bestimmen zu können, wann, wie und in welchem Ausmaß Information über ihre Position an andere weitergegeben wird.“* [DK06] (eigene Übersetzung)

Dieser Definition folgend bedeutet Privatsphäre die Fähigkeit, selbstbestimmt mit der Positionsinformation umzugehen, ihre Verarbeitung potentiell verweigern zu können und sie vor unautorisiertem Zugriff zu schützen. Es geht um die selbstbestimmte Kontrolle der eigenen Positionsinformation.
Neben dem oben dargestellten problematischen Fall der unautorisierten bzw. unangeforderten Bereitstellung von Informationen wie Werbung gibt es noch wesentlich kritischere Fälle: So hängt die Position immer mit der Sicherheit der Person zusammen. Wissen Dritte über meinen Standort Bescheid, kann dies Konsequenzen für mein persönliches Wohlergehen haben [DK06]. Darüber hinaus – und dies ist der wirklich heikle Punkt – lassen sich durch das Bewegungsprofil Rückschlüsse auf das Sozialverhalten, den Gesundheitszustand sowie die persönlichen Interessen ziehen [DK06] [DF03]. Ist bekannt, an welchem Ort sich eine Person zu einer gewissen Zeit aufhält, und findet an diesem Ort zu dieser Zeit beispielsweise eine

[1] Der Leser sei auf die Standardliteratur verwiesen [Tan03].

Kundgebung statt, so können über die Positionsinformation Rückschlüsse auf die politische Einstellung dieser Person getroffen werden. Ebenso könnten Personen von Dritten aufgrund der Positionsinformation gestalkt[2] werden. Dobson und Fisher bezeichnen diese Form der potentiellen Unterhöhlung der Privatsphäre durch LBS auch als *Geoslavery* und zeichnen eine düstere Vision der Entwicklung [DF03], die es zu beachten gilt.

Dem Schutz der Privatsphäre bei LBS muss auf verschiedenen Ebenen nachgekommen werden.
Zunächst muss diese Technologie als in gesellschaftliche Prozesse eingebettet verstanden werden, so dass sich kritisch mit den Konsequenzen auseinandergesetzt werden kann. Dazu ist eine Sensibilierung der Massen, aber auch eine kritische Haltung der Wissenschaft gegenüber dem eigenen Forschungsgegenstand notwendig.
Neben den gesellschaftlichen Maßnahmen zum Schutz der Privatsphäre gibt es ein rechtliches Rahmenwerk, welches den Schutz der Privatsphäre in der Bundesrepublik Deutschland (BRD) regelt. So besagt § 98 des Telekommunikationsgesetzes, dass eine Nutzung von Positionsinformationen, denen nicht zugestimmt wurde, untersagt ist. Wer in der BRD diesem Grundsatz, der Teil der informationellen Selbstbestimmung ist, zuwiderhandelt, macht sich strafbar.

Diese beiden Konzepte sind jedoch für Schutz der Privatsphäre nicht hinreichend, da der rechtliche Rahmen den technologischen Entwicklungen zeitlich hinterherhinken kann. Außerdem kann der rechtliche Rahmen bewusst oder unbewusst hintergangen und ausgehebelt werden, was ein jüngster Vorfall bei Smartphones der Firma *Apple* zeigt, die alle Positionsdaten im Hintergrund unautorisiert mitschreiben[3]. Daher muss dem Problem der Wahrung der Privatsphäre bei LBS auch in technischer Hinsicht beigekommen werden. Die folgenden Abschnitte beleuchten einzelne technische Sicherungsmaßnahmen für Location Server im Detail.

[2] Stalking beschreibt, dass einer Person willentlich und beharrlich gegen ihren Willen nachgestellt, sie verfolgt und/oder belästigt wird. http://de.wikipedia.org/wiki/Stalking, Zugriff 16.4.2011

[3] http://www.heise.de/tp/artikel/34/34601/1.html, Zugriff 24.4.2011. Zwar ist in diesem Fall das mobile Gerät kompromittiert, was unserer Annahme widerspricht, denn diese Diplomarbeit geht von vertrauenswürdigen mobilen Geräten aus. Der Fall soll jedoch zeigen, dass selbst renomierten Firmen unter Umständen nicht vertraut werden kann.

2.2.1 Zugriffskontrolle

Die Methode der *Zugriffskontrolle* ist aus anderen Forschungsfeldern der Computerwissenschaften, wie z.B. der Betriebssysteme oder Datenbanken, bekannt und kann auch auf Location-based Services angewendet werden [DK06] [RPB09]. Schutz der Privatsphäre soll erreicht werden, indem Zugriffe von unautorisierten Instanzen abgewiesen und verboten werden können. Dazu wird eine Zugriffsliste geführt, welche Aktion auf einer Ressource von welcher Person oder Entität ausgeführt werden darf.
Im Gegensatz zum klassischen Anwendungsfall wie beispielsweise bei Betriebssystemen wird in vernetzen und mobilen Szenarien keine zentrale Verwaltung angestrebt, was den Verwaltungsaufwand stark erhöht. Außerdem muss in diesem Fall von einer vertrauenswürdigen Verwaltung der Zugriffsliste ausgegangen werden.

2.2.2 Verschlüsselung

Wie die Zugriffskontrolle ist die *Verschlüsselung* ein Konzept, das nicht speziell auf Location-based Services zugeschnitten ist. Allerdings lässt es sich auf LBS anwenden [DK06] [RPB09], indem die Positionsinformationen verschlüsselt auf dem Location Server abgelegt werden. Dieses Verfahren entspricht von der Sicherheit her immer dem gegenwärtigen Stand der Verschlüsselungsverfahren. Dürr und Skvorzov [DSR11] kritisieren an der Verschlüsselung der Positionsinformationen, dass dadurch räumliche Anfragen über Positionsinformationen, wie sie bei LBS vorkommen und erwünscht sind, unmöglich werden.

2.2.3 k-Anonymity

Dies ist eines der bekanntesten Konzepte zur Sicherung der Privatsphäre. Die Idee hinter *k-Anonymity* ist es, die eigene Position von der Position $k-1$ weiterer Nutzer des LBS ununterscheidbar zu machen [DK06] [RPB09]. Man spricht hierbei auch von einem anonymisierenden Verfahren, weil die Personen durch eine Gruppe anonymisiert werden [SDfMb09]. Dazu wird die eigene Position zunächst mit der von $k-1$ benachbarten Nutzern zu einem *Cluster* gruppiert. Dann wird die Positionsanfrage auf diesem Cluster ausgeführt. Die Gruppierung, Anonymisierung und Rück-Personalisierung wird durch einen anonymisierenden Server vollzogen, dem vertraut werden muss – er kennt die Identität des echten anfragenden Nutzers. Dies stellt eine generelle Schwachstelle des Konzepts dar [DK06] [DSR11]. Das Verfahren ist in 2.2 schematisch dargestellt.
Des Weiteren ist es keine triviale Aufgabe, eine gute Gruppierung im Hinblick auf Sicherheit als Ununterscheidbarkeit von k Personen zu finden. Man denke hier daran, 40 Personen zu gruppieren, entweder in einer Konzerthalle oder in der Wildnis. Im ersten Fall lässt sich die Gruppierung leicht finden, im zweiten Fall ist dies problematisch, wodurch die Qualität der Sicherheit stark schwanken kann.

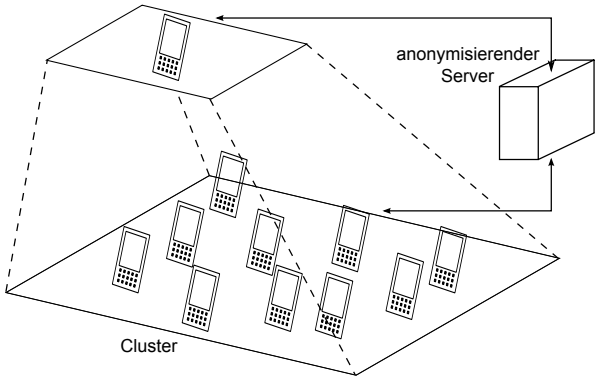

Abbildung 2.2: Schematische Darstellung von k-Anonymity mit $k = 10$.

2.2.4 Räumliche Verschleierung

Bei der *räumlichen Verschleierung* (engl. *Obfuscation*) wird die Position eines Nutzers künstlich unscharf bzw. ungenau gemacht [DK06] [DSR11] [ACD+07], was in Abbildung 2.3 dargestellt ist. Die Position entspricht einer spezifisch verteilten Aufenthaltswahrscheinlichkeit innerhalb eines Radius r. Der Nutzer ist dadurch nicht exakt lokalisierbar und es lassen sich je nach Stufe der Verschleierung nur vage Rückschlüsse auf seine exakte Position treffen. Vom Nutzer kann gesteuert werden, wieviele Informationen an welchen ortsbezogenen Dienst herausgegeben werden. So ist es möglich, feingranular Dienste zu autorisieren oder eigene Abstufungen zu definieren: Jedem Dienst werden gerade genug Informationen wie benötigt zugänglich gemacht. Ein weiterer Vorteil dieses Verfahrens ist, dass keine vertrauenswürdigen Instanzen zu Speicherung angenommen werden müssen. Darüber hinaus können trotzdem Anfragen über Positionen getätigt werden, was bei einem anonymisierenden Verfahren von vornherein ausscheidet. Allerdings ist die Exaktheit eines LBS beim Lokalisieren von POIs nicht mehr gegeben.

2.2.5 Koordinatentransformation

Das Verfahren der *Koordinatentransformation* versucht, Privatsphäre in LBS zu erreichen, indem die reale Position auf ein anderes Koordinatensystem abgebildet wird [Guto6]. Dabei ist die Funktion, die die Transformation realisiert, nur dem Nutzer selbst (oder einer Gruppe, in der sich die Mitglieder gegenseitig vertrauen) bekannt. Alle Anfragen werden in diesem transformierten Koordinatenraum berechnet – nur der Inhaber der Funktion zieht aus dieser Information einen Nutzen, wodurch seine reale Position geschützt ist. Die Koordinatentransformation ist in Abbildung 2.4 dargestellt und zeigt, wie ein Punkt in zwei verschiedenen Koordinatensystemen repräsentiert wird und nur mittels einer Transformationsfunktion $\vec{d}_{B,A}$ bestimmt werden kann.

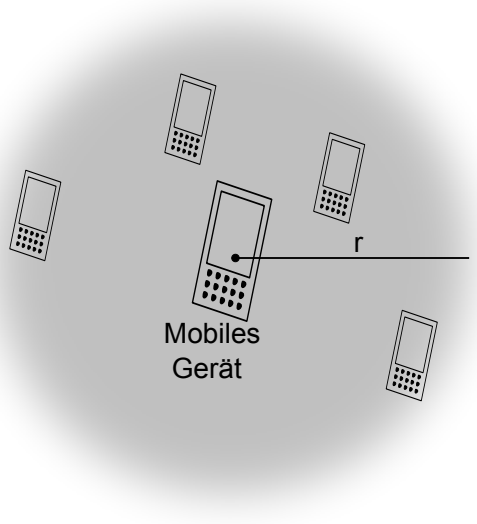

Abbildung 2.3: Darstellung der räumlichen Verschleierung der Position. Die exakte Position wird unscharf und entspricht einer Aufenthaltswahrscheinlichkeit.

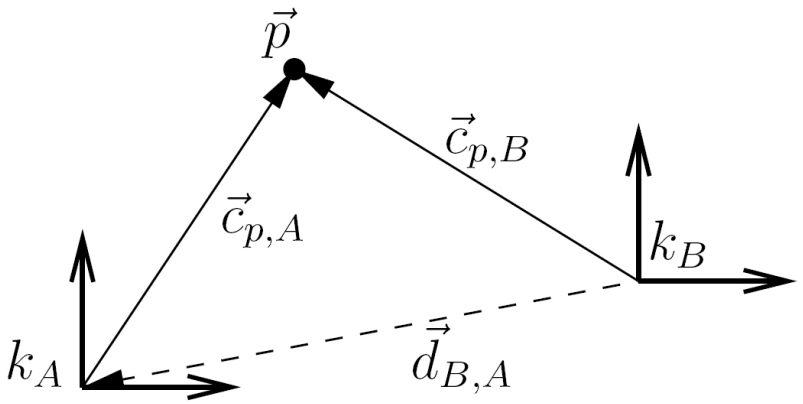

Abbildung 2.4: Koordinatentransformation schematisch dargestellt: Repräsentation des Punktes $\vec{p}$ in zwei Unterschiedlichen Koordinatensystemen k_A und k_B [Guto6].

2.2.6 Verteilung der Positionsinformation

Dürr und Skvorzov kombinieren die Koordinatentransformation mit der räumlichen Verschleierung und kommen zu einem neuen Ansatz für nicht vertrauenswürdige Instanzen, der mit mehreren unabhängigen Location Servern arbeitet [DSR11].
Bei diesem Verfahren wird die Position so in Teile zerlegt, die *Shares* genannt werden, dass die exakte Position nur rekonstruiert werden kann, wenn über alle Teile verfügt wird. Mit jedem Teil der Positionsinformation kann die Position genauer rekonstruiert werden. Dies ist in Abbildung 2.5 dargestellt. Die Shares sind als Vektoren zu verstehen, wobei jeder Vektor die Genauigkeit um ein gewisses Δ_ϕ erhöht. Diese Genauigkeit kann als Wahrscheinlichkeitsverteilung innerhalb eines Kreises verstanden werden, dessen Mittelpunkt durch den Vektor geliefert wird. Die vektorielle Addition von einigen Shares liefert somit eine ungenaue Position einer Entität, die sich überall innerhalb des jeweiligen Kreises befinden kann. Nur wenn alle Shares addiert werden, kann die exakte Position rekonstruiert werden. Da vektorielle Addition eine kommutative Rechenoperation ist, können die Vektoren in beliebiger Reihenfolge addiert werden, was in Abbildung 2.5 verdeutlicht wird.

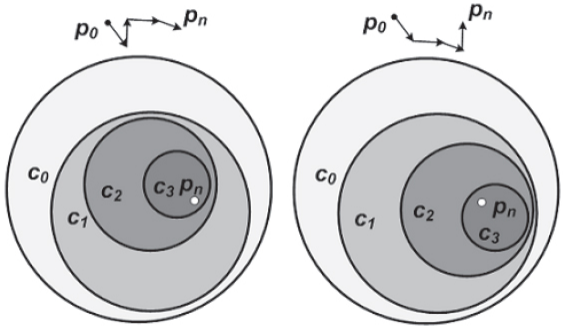

Abbildung 2.5: Darstellung der Genauigkeitsvergrößerung bei Share-Fusion. p_j gibt die Position nach Fusion der j-ten Share an, c_j die Genauigkeit [DSR11].

Diese Teile der Positionsinformation werden auf unabhängige Location Server, im Folgenden auch *Provider* genannt, aufgeteilt, so dass bei einem kompromittierten Server oder einem internen unautorisierten Zugriff die Position nur mit ungenügender Genauigkeit rekonstruiert werden kann. In diesem Konzept kann somit von nicht-vertrauenswürdigen Instanzen ausgegangen werden, was gegenüber dem k-Anonymity-Konzept sowie der Zugriffskontrolle ein Vorteil ist. Dabei ist der Grundgedanke, dass es unwahrscheinlich ist, dass viele oder alle Server gleichzeitig gehackt wurden und sich die Positionsinformation somit immer nur ungenau rekonstruieren lässt. Auf dieser Systemarchitektur aufbauend wird später in Abschnitt 3.1 eine Erweiterung vorgeschlagen, um der Aufgabenstellung beizukommen, wobei auch die originale Systemarchitektur im Detail erläutert wird.

2.2.7 Taxonomie der Sicherheitsverfahren

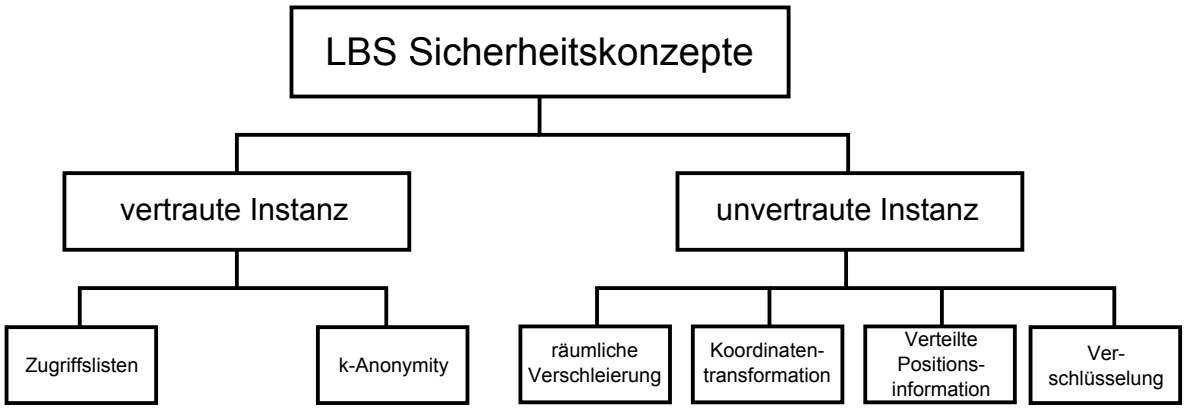

Abbildung 2.6: Taxonomie der Sicherheitskonzepte bei LBS.

In Abbildung 2.6 ist eine überblicksartige Taxonomie der vorgestellten Sicherheitskonzepte dargestellt. Hier werden die Konzepte nach der Vertrauenswürdigkeit der Instanzen zur Lagerung bzw. Verwaltung der Positionsinformation geordnet. Die Verfahren der räumlichen Verschleierung, der Koordinatentransformation, der verteilten Positionsinformation und der Verschlüsselung gehen dabei von nicht-vertrauten Instanzen aus, wohingegen k-Anonymity und Zugrifflisten Server annehmen, denen vertraut werden muss.

2.3 Kombinatorische Optimierung

Bisher wurde nur auf den technologischen Rahmen, in dem sich die Problemstellung bewegt, eingangen. Nun wird die Problemstellung selbst eingeordnet werden: Die Platzierung von Positionsinformationen auf nicht-vertrauten Servern zur Ausbalancierung von Sicherheitsrisiken kann am allgemeinsten als ein Problem der Kombinatorischen Optimierung kategorisiert werden.

2.3.1 Definition und Eigenschaften

Kombinatorische Optimierung bedeutet, aus einer Vielzahl von möglichen Lösungen (den möglichen *Kombinationen*) eine optimale im Bezug auf ein gewisses Kriterium (das *Optimalitätskriterium*) auszuwählen [VK08] [Hu82]. Bekannte Probleme dieser Art sind das *Problem des Handlungsreisenden*[4] [VK08], das *Behälterproblem*[5] [VK08, MT90] oder *Rucksackprobleme*[6] [MT90]. Beim Problem des Handlungsreisenden geht es beispielsweise darum, die kürzeste Rundreise zu finden, die alle Städte genau einmal besucht und wieder beim Ausgangsort endet.

Formal ist ein kombinatorisches Optimierungsproblem wie folgt definiert [VK08]:
Sei $\mathcal{L}$ die Menge aller möglichen Lösungen und die Formel (2.1)

$$f : \mathcal{L} \to \mathbb{R} \tag{2.1}$$

eine Funktion, die jeder möglichen Lösung Kosten zuweist (die *Kostenfunktion*). Die Lösung eines kombinatorischen Optimierungsproblems besteht nun darin, eine optimale Lösungsinstanz opt aus der Menge aller Lösungen $\mathcal{L}$ zu finden, die bezüglich der Kostenfunktion besser, d.h. kleiner ist als alle anderen Lösungen. Formal ausgedrückt lautet dies:

$$\exists opt \in \mathcal{L} \;\; \forall l \in \mathcal{L} : f(opt) \leq f(l) \tag{2.2}$$

Ziel der kombinatorischen Optierung ist es, den Wert für opt zu finden.
Übertragen auf unser Problem entspricht die Menge $\mathcal{L}$ allen möglichen Belegungen von Shares auf Providern. Die Kostenfunktion entspricht der Risikobewertung bzw. dem Grad der Balance. Dann gilt es, aus allen möglichen Lösungen die optimale im Bezug auf das geringste Risiko auszuwählen, was bei uns der besten Balance entspricht.

[4] engl. *Travelling Salesman Problem, TSP*
[5] engl. *Bin Packing*
[6] engl. *Knapsack Problems*

Für eine ausreichende Problembeschreibung im Hinblick auf Lösungsmöglichkeiten ist diese grobe Kategorisierung aber noch nicht ausreichend – diese kann erst in Kapitel 3 entwickelt werden.

Das Behälterproblem Zunächst wird das *Behälterproblem* betrachtet [VK08], welches einerseits ein typisches Beispiel für kombinatorische Optimierung ist und andererseits eine gewisse Verwandtschaft einem unserer Teilprobleme hat, wie sich noch zeigen wird.
Bei diesem Optimierungsproblem geht es darum, Objekte so auf Behälter aufzuteilen, dass kein Behälter überläuft und die Zahl der benutzten Behälter minimiert wird.
Formal ist eine Menge von n Objekten mit unterschiedlichen Gewichten $a_1, a_2, ...a_n$ gegeben sowie eine Behältergröße b. Für alle einzelnen Gewichte $a_i, i \in \{1, 2, ...n\}$ gilt $a_i \leq b$. Die Aufgabe ist es, eine Zuordnung f von den Objekten $A = \{1, 2, ...n\}$ zu den Behältern B zu finden, dass die Anzahl der Behälter minimal wird, also:

$$f : \{1, 2, ..., n\} \rightarrow \{1, 2, ...k\} \tag{2.3}$$

so dass k minimal und

$$\sum_{i:f(i)=j} a_i \leq b \tag{2.4}$$

gilt, was bedeutet, dass die Behältergröße nicht überstiegen werden darf. Dieses Problem ist in Abbildung 2.7 visualisiert.

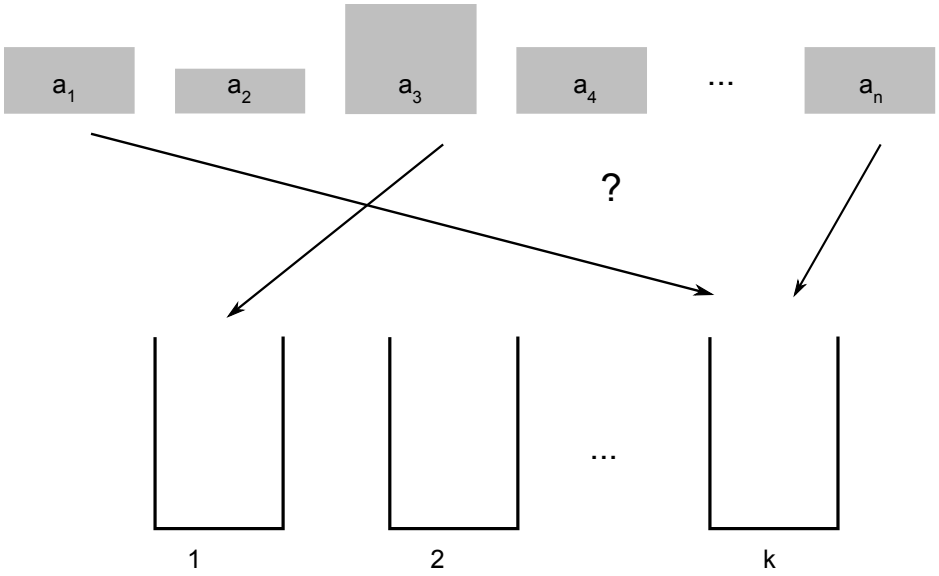

Abbildung 2.7: Schematische Darstellung des Behälterproblems.

Algorithmus 2.1 First-Fit Algorithmus nach [VK08]

1: **for** $i = 1$ to n **do**
2: $\quad f(i) \leftarrow \min_{j \in \mathbb{N}} \{\sum_{h<i:f(h)=j} a_h + a_i \leq b\}$
3: **end for**
4: $k \leftarrow \max_{i \in \{1,\cdots,n\}} f(i)$

Eine naive Methode würde alle Möglichkeiten durchprobieren und dann die beste auswählen. Diese würde aber n^n viele Abbildungen bzw. Möglichkeiten erzeugen, was schon bei einer kleinen Größe von n unzumutbar viele Lösungen erzeugt und damit ineffizient ist.
Ein bekannter Approximationsalgorithmus zur Lösung dieses Problems ist der *First-Fit Algorithmus*, welcher in Algorithmus 2.1 dargestellt ist. Der Algorithmus weist jedes Element $a_i, i \in \{1, \cdots, n\}$ dem ersten Behälter j zu, in dem noch genügend Platz ist. Ist in keinem der Behälter noch Platz, so wird ein neuer geöffnet.

Bei dem hier betrachteten Behälterproblem handelte es sich um ein Optimierungsproblem. Der Vollständigkeit halber sei erwähnt, dass es neben dem jeweiligen Optimierungsproblem noch ein entsprechendes *Entscheidungsproblem* gibt, welches danach fragt, ob das Problem für eine gegebene Probleminstanz gelöst werden kann oder nicht. Im Falle des Behälterproblems geht es um die Frage, ob n Elemente auf k Behälter verteilt werden können, ohne dass die Bedingung (2.4) verletzt wird.
Der Wichtigkeit für unsere Untersuchung entsprechend wurde der Fokus aber auf Optimierungsprobleme gelegt.

Kombinatorische Probleme sind häufig NP-schwer [CLRS04] [HMU03]. *NP-Schwere* eines Problems betrifft den Zusammenhang eines Problems mit der Komplexitätsklasse $\mathcal{NP}$ und besagt, dass es mindestens so schwer lösbar ist wie alle anderen Probleme in dieser Komplexitätsklasse. Ist ein Problem NP-schwer, so gibt es vermutlich[7] keinen deterministischen Algorithmus, der das Problem in Polynomialzeit lösen kann; für diese Problemklasse sind bisher nur deterministische Algorithmen mit exponentieller Komplexität bekannt[8].
Zum Nachweis, dass ein Problem A NP-schwer ist, muss ein Problem B, von dem schon bekannt ist, dass es NP-vollständig (oder NP-schwer) ist, in polynomialer Zeit auf A reduziert werden können. Eine *Reduktion* ist ein Algorithmus oder ein Verfahren, der Instanzen des einen Problems in die des anderen Problems transformiert, wobei jeweils die

[7] Unter der Annahme, dass $\mathcal{P} \neq \mathcal{NP}$. Die Frage, ob dies wirklich gilt, ist eine der fundamentalsten ungelösten Fragen der Informatik [VK08] [CLRS04] [HMU03].
[8] Bei Eingabelänge n hat der Algorithmus eine Zeitkomplexität von $\mathcal{O}(c^n)$

gleichen Lösungen für gleiche Eingaben herauskommen müssen[9]. Zum Beweis muss eine Funktion gefunden werden, die dies für alle Instanzen erledigt. Die Reduktionsvorschrift selbst muss polynomiale Zeitkomplexität besitzen.
NP-Vollständigkeit schließlich besagt, dass ein Problem NP-schwer ist und wirklich in der Komplexitätsklasse $\mathcal{NP}$ liegt. Bei NP-vollständigen Problemen ist die Verifikation, dass ein gegebener Lösungskandidat tatsächlich eine Lösung ist, in Polynomialzeit möglich, wohingegen das Ermitteln von Lösungen exponentielle Zeit beansprucht.
Martello und Toth [MT90] setzen NP-Schwere mit Unlösbarkeit gleich, wohlwissend, dass es hier nicht um die absolute Unmöglichkeit der Lösungssuche geht, sondern um das Finden von Lösungen in annehmbarer Zeit, womit polynomielle Zeit gemeint ist. Ein polynomieller Algorithmus wird als *effizient* bezeichnet.

2.3.2 Algorithmische Lösungsansätze

Die Aufgabenstellung innerhalb der kombinatorischen Optimierung ist es, Algorithmen zur Lösung dieser Probleme zu finden. Mit den Ausführungen des vorhergehenden Abschnitts sollen jetzt prinzipielle Lösungsansätze diskutiert werden. Wie dargelegt wurde, sind diese Probleme oft NP-schwer und machen damit eine Entscheidung zwischen Exaktheit der Lösung und Geschwindigkeit der Lösungserzeugung notwendig.

Vollständige Enumeration

Die *vollständige Enumeration* bezeichnet das Aufzählen und Bewerten aller Lösungen bei folgender Auswahl der Besten. Die Schwachstelle hier ist die große Anzahl der möglichen Lösungen, die exponetiell mit der Problemgröße steigt. Zwar braucht die Berechnung schon unverhältnismäßig lange, kann aber bei heutiger Rechnerleistung für kleine Problemgrößen in Erwägung gezogen werden, da die exakte Lösung gefunden wird.

Branch-and-Bound

Branch-and-Bound ist ein an Backtracking angelehntes Verfahren für Optimierungsprobleme [LW66] [Hu82].
Backtracking arbeitet sich durch einen Entscheidungsbaum und versucht, sukzessive Lösungskandidaten zu einer Gesamtlösung zu konstruieren. Ist bei einer Auswahlentscheidung

[9] Dies betrifft Entscheidungsprobleme, die Lösungen in Antworten der Form *ja* oder *nein* ausgeben. Dies kann auf Optimierungsprobleme übertragen werden, indem es um die zulässigen Lösungen des Optimierungsproblems geht, die nach Reduktion gleich sein müssen [VK08]: Eine Lösung muss sowohl für Problem A als auch für Problem B optimal sein.

jedoch absehbar, dass keine gültige Lösung mehr konstruiert werden kann, wird der komplette folgende Zweig verworfen. Gelangt die Suche in eine Sackgasse, geht der Algorithmus einen Schritt bis zum nächsten Entscheidungsknoten zurück, um dort die noch nicht besuchten Knoten nach dem selben Muster zu bearbeiten.
Branch-and-Bound erweitert diese strukturierte Suche im Entscheidungsbaum auf Optimierungsprobleme, indem neben der Gültigkeit noch die Optimalität der Lösungskandidaten als Entscheidungsfaktor miteinbezogen wird. Steht der Algorithmus vor einer Verzweigung (engl. *branch*), wird geprüft ob das Weiterverfolgen in einer Richtung zu einem nicht-optimalen Wert führt, indem mit einer Schranke (engl. *bound*) verglichen wird. Nur wenn der Wert unterhalb der Schranke liegt, wird in dieser Richtung weiterverfolgt – dann könnte eine bessere Lösung konstruiert werden. Ist dies für keine der vorhandenen Verzweigungen der Fall, muss einen Schritt zurückgegangen werden.
Das Verfahren ist exakt, d.h. es findet immer eine Lösung und kann als eine strukturierte Aufzählung aller Lösungskandidaten bezeichnet werden, bei der so früh wie möglich bereits ungültige oder nicht-optimale Lösungen ausgeschlossen werden. Dies senkt die Komplexität zwar, die Abarbeitungsdauer des Entscheidungsbaum liegt aber dennoch in $\mathcal{O}(c^n)$, da im schlechtesten Fall alle Knoten besucht werden müssten. Die praktische Effizienz von Branch-and-Bound hängt stark von der konkreten Problemstellung ab. Mit der Problemstellung hängt ebenso zusammen, wie sich eine Grenze bestimmen lässt und wieviele Zweige schon frühzeitig ausgeschlossen werden können.

Heuristiken

Als *Heuristik* wird ein problemspezifisches Lösungsverfahren bezeichnet, das eine Näherungslösung von NP-schweren Problemen in effizienter Zeit bestimmt. Heuristiken für bekannte NP-schwere Probleme finden sich in [VK08].

Metaheuristiken

Eine *Metaheuristik* ist ein Verfahren zur näherungsweisen Bestimmung von Lösungen bei Optimierungsproblemen, das nicht problemspezifisch, sondern allgemein anwendbar ist [Wei02]. Es folgt ein Überblick über einige Verfahren und eine genauere Betrachtung der *Evolutionären Algorithmen*, da ein in Kapitel 3 erarbeiteter Lösungsvorschlag auf diesem Konzept beruht.

Lokale Suche Bei der *lokalen Suche* [Egl90] wird zunächst eine Startlösung bestimmt, was zufällig oder in Form einer Heuristik geschehen kann. Dann wird von dieser Startlösung ausgehend die Nachbarschaft sondiert, d.h. die Lösungen, die numerisch nahe zur Startlösung sind. Ist eine Lösung in der Nachbarschaft besser, wird mit dieser fortgefahren. Das Problem ist allerdings, dass das Verfahren in lokalen Maxima stecken bleiben kann, da es

beim ersten lokalen Maximum dieses als Lösung akzeptiert. Dieses Verfahren wird auch *Bergsteigeralgorithmus* (engl. *Hill Climbing*) genannt.
Das Verfahren der *Simulierten Abkühlung* [Egl90] wird allgemein unter der lokalen Suche kategorisiert. Dieses Konzept beruht auf der Nachbildung eines Abkühlungsvorgangs in der Physik, welcher der Gesetzmäßigkeit folgt, dass ein System im idealen Zustand $\mathcal{Z}$ proportional ist zu $e^{-\frac{Energie(\mathcal{Z})}{temp}}$ ist, wobei $Energie(\mathcal{Z})$ den Energiezustand des Systems und $temp$ die momentane Temperatur bezeichnet. Übertragen auf Optimierungsprobleme bedeutet dies, dass je geringer die Temperatur ist, desto weniger wahrscheinlich wird die Auswahl eines Nicht-Optimums. Die Temperatur entspricht der Wahrscheinlichkeit für Akzeptanz von nicht-optimalen Zuständen.
Die vorgestellten Verfahren sind schnell, jedoch ist vor allem die reine lokale Suche ungenügend, da sie bei lokalen Optima stecken bleiben kann. Simulierte Abkühlung wird von Weicker [Wei02] als „launisch“ und stark vom Optimierungsproblem abhängig bezeichnet.

Evolutionäre Algorithmen *Evolutionäre Algorithmen* bilden Optimierungsprobleme als evolutionäre Prozesse nach [Wei02] [Nis97]. Sie können als Erweiterung der lokalen Suche begriffen werden. Im Gegensatz dazu werden mehrere Lösungskandidaten erzeugt und miteinander kombiniert, außerdem sorgen Mutationen dafür, dass möglichst der komplette Wertebereich abgedeckt wird. Darum spricht man bei evolutionären Algorithmen auch von populationsbasierten Verfahren, da sie mit mehr als einem Individuum arbeiten. In der Literatur finden sich des Weiteren die Verfahren *Evolutionsstrategien*, *evolutionärens Programmieren* und *genetische Algorithmen*, wobei die Unterschiede im Detail liegen und die Übergänge fließend sind. In der folgenden Ausführung konzentriere ich mich auf die evolutionären Algorithmen.
Für die Anwendung eines evolutionären Algorithmus auf ein Problem muss dieses zunächst als *Genom* codiert werden. Die Lösungskandidaten werden meist als Bit-Vektoren, manchmal aber auch als ganzzahlige Vektoren dargestellt. Potentielle Lösungskandidaten werden als *Individiuen* bezeichnet. Zunächst werden initiale Lösungskandidaten bestimmt, was entweder komplett randomisiert oder basierend auf einer Heuristik geschehen kann. Die Gesamtheit aller Individuen zu einem gewissen Zeitpunkt bezeichnet man auch als *Population* oder *Generation*. Individuen der anfangs erzeugten Startpopulation werden nun miteinander gekreuzt, wobei es unterschiedliche Kreuzungsoperationen gibt. Je nach Kreuzungsoperator werden verschiedene Teile der Genome miteinander kombiniert. Schließlich kommt es noch zu einer zufälligen Veränderung an einzelnen Individuen, was als *Mutation* bezeichnet wird. Diese Operation ist wichtig, um alle Werte im Bereich potentiell abdecken zu können. Die aus einer Population erzeugten Individuen werden nun bewertet, was durch eine *Fitnessfunktion* geschieht – sie bewertet die Güte eines Individuums im Bezug auf das Optimierungsziel. Dann werden die am besten bewerteten Individuen ausgewählt und der Zyklus beginnt von Neuem. Dieser Zyklus ist in Abbildung 2.8 dargestellt und endet, wenn entweder eine bestimmte Anzahl von Generationen oder ein Abbruchkriterium erreicht wurde.

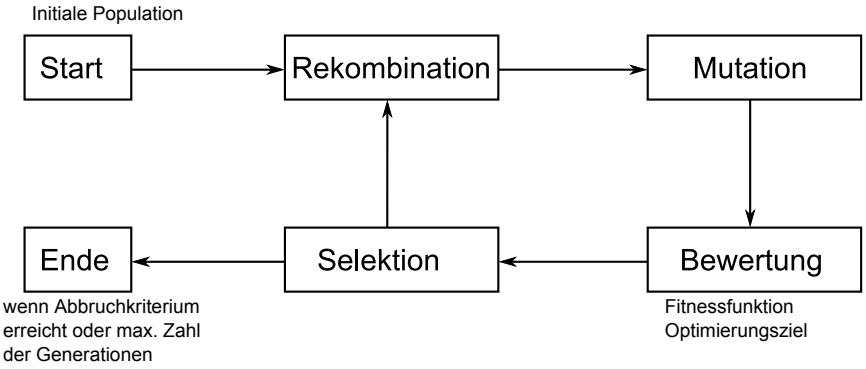

Abbildung 2.8: Zyklus von evolutionären Algorithmen nach [Wei02].

Über evolutionäre Algorithmen lässt sich sagen, dass sie ein gutes Approximationsverfahren für kombinatorische Optimierungsprobleme darstellen, da sie durch den Mutationsoperator in der Wertelandschaft „springen“ und so zuverlässig ein globales Maximum finden können. Des Weiteren sind die Parameter gut anpassbar und durch geschickte Wahl der initialen Population verbessert sich die Laufzeit und die Qualität. Prinzipiell stoßen sie schnell in die Nähe der optimalen Lösung vor, benötigen dann aber Zeit, um sich weiter zum wirklichen Optimalwert vorzutasten. Der Nachteil liegt in der erhöhten Laufzeit gegenüber der lokalen Suche, weil die Operationen komplexer sind. Die Laufzeit von evolutionären Algorithmen hängt vom Abbruchkriterium ab: Wird über eine feste Anzahl von Generationen iteriert, ist die Laufzeit sogar konstant und wird maßgeblich durch den Aufwand der Kombinationsoperatoren bestimmt.

Schwarmintelligenz Das Verfahren der *Schwarmintelligenz* (engl. *Swarm Intelligence*) arbeitet mit einfach gearteten Einzelindividuen, deren Interaktion über die Umwelt vermittelt einem komplexen Verhalten des Gesamtsystems emergiert um Probleme zu lösen [MRF+03]. Ein typischer Vertreter sind hierbei die *Ameisenalgorithmen*, die das kollektive Verhalten von Ameisen zur Futtersuche nachbilden, um kürzeste Wege zu finden [DS03].

2.3.3 Zuweisungsprobleme

Zuweisungsprobleme stellen eine spezielle Unterkategorie der kombinatorischen Optimierung dar [BDM09] und müssen im Folgenden besprochen werden, da sich unser Problem als Zuweisungsproblem formulieren lässt.
Bei dieser Problemklasse geht es darum, wie eine gewisse Anzahl von Objekten anderen Objekten unter gewissen Kritierien zuwiesen wird [BDM09]. Statt abtrakt von *Objekten* spricht man auch von *Tasks* oder *Jobs*, die *Maschinen, Agenten* oder *Arbeitern* zugewiesen werden. Die Zuweisung ist mathematisch eine Abbildung zwischen zwei Mengen $\mathcal{U}$ und $\mathcal{V}$.

Eine solche Zuweisung ist eine Permutation der Anordnung und wird im Folgenden als *Zuweisungsmatrix* (*Assignment Matrix*) in Formel (2.5) dargestellt,

$$X = (x_{ij}) = \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nn} \end{pmatrix} \tag{2.5}$$

wobei

$$x_{ij} = \begin{cases} 1, & \text{wenn Objekt } i \text{ dem Objekt } j \text{ zugewiesen ist} \\ 0, & \text{sonst} \end{cases} \tag{2.6}$$

Werden beispielsweise drei Objekte $\mathcal{U} = \{a, b, c\}$ drei anderen $\mathcal{V} = \{a', b', c'\}$ zugewiesen, sieht eine mögliche Zuordnung wie folgt aus:

$$X = (x_{ij}) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \tag{2.7}$$

Die durch Matrix (2.7) bestimmte Zuweisung ist in Abbildung 2.9 grafisch dargestellt.

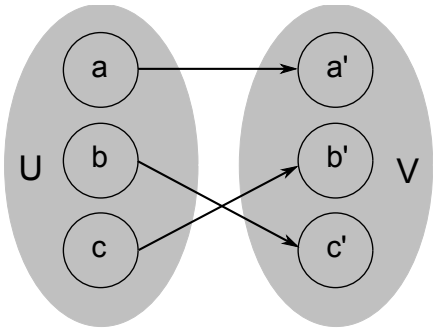

Abbildung 2.9: Exemplarische Zuweisung nach Zuweisungsmatrix (2.7) zwischen zwei Mengen $\mathcal{U} = \{a, b, c\}$ und $\mathcal{V} = \{a', b', c'\}$ mit jeweils drei Elementen

Zuweisungen werden durch eine Menge von linearen Gleichungen charakterisiert, die *Nebenbedingungen* genannt werden. Diese geben beispielsweise Auskunft darüber, wie viele Objekte der einen Menge $\mathcal{U}$ maximal einem Objekt der Menge $\mathcal{V}$ zugewiesen sind und vice versa. Diese Nebenbedingungen hängen von der konkreten Problemstellung ab.
Darf im obigen Beispiel einem Objekt der Menge $\mathcal{U}$ ein Element der Menge $\mathcal{V}$ zugewiesen werden (2.8) sowie einem Element aus der Menge $\mathcal{V}$ genau eines (2.9), so lauten die Nebenbedingungen

$$\sum_{i=1}^{n} x_{ij} = 1 \quad \forall j \in \{1, 2, \cdots, n\} \tag{2.8}$$

$$\sum_{j=1}^{n} x_{ij} = 1 \quad \forall i \in \{1, 2, \cdots, n\} \tag{2.9}$$

und

$$x_{ij} \in \mathbb{B} \tag{2.10}$$

Außerdem darf die Zuweisungsmatrix nur aus bool'schen Werten bestehen, was Nebenbedingung (2.10) widerspiegelt.

Eine Zuweisung wird *gültige Zuweisung* genannt, wenn sie alle entsprechenden Nebenbedingungen erfüllt. Beispiel 2.9 ist somit eine gültige Zuweisung in Bezug auf die genannten Nebenbedingungen (2.8), (2.9) und (2.10). Es gibt $n!$ gültige Zuweisungsmatrizen, wenn alle dieser Nebenbedingungen erfüllt sind.

Allein über die Zuweisungsmatrix lässt sich allerdings kein Optimierungsproblem formulieren – es fehlt das Kriterium, nach dem überhaupt optimiert werden soll. Dazu wird mit jeder Zuweisungsoperation ein Zahlwert assozziert, der Kosten, Gewicht, Größe, aber evtl. auch Gewinn usw. repräsentiert.
In der Regel spricht man aber von *Zuweisungskosten*, die in einer *Kostenmatrix* C repräsentiert werden:

$$C = (c_{ij}) = \begin{pmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ c_{n1} & c_{n2} & \cdots & c_{nn} \end{pmatrix} \tag{2.11}$$

wobei c_{ij} die Kosten (oder den Gewinn) darstellt, die entstehen, wenn Objekt i dem Objekt j zugewiesen wird.
Eine Multiplikation $X_\phi \cdot C$ einer Zuweisungsmatrix mit der Kostenmatrix ergibt die Kosten aller getätigten Zuweisungen der Permutation X, wobei es sich hierbei um eine modifizierte Multiplikation und nicht um die Standardmultiplikation bei Matrizen handelt.

Um eine Optimierungsaufgabe formulieren zu können, wird noch ein Kriterium benötigt: die *Zielfunktion*. Die Zielfunktion entspricht dem Optimierungsziel, das je nach Aufgabenstellung unterschiedlich geartet ist. Handelt es sich bei der Zielfunktion um eine lineare Funktion, so spricht man von *linearer Optimierung*.

Klassische Zuweisungsprobleme

Mit dieser Vorarbeit lässt sich das *klassische Zuweisungsproblem* (*Assignment Problem, AP*) formulieren. Spezifisch für *klassische* Zuweisungsprobleme ist, dass die Zahl der Jobs und der Agenten gleich ist. Hierbei geht es darum, n Jobs so n Agenten zuzuweisen, dass die Gesamtkosten der Zuweisungen minimal werden. Die Zielfunktion, welche optimiert wird, ist demnach eine Funktion, welche die Kosten aller Zuweisungen bestimmt und in Gleichung (2.12) dargestellt ist.
Formal lässt sich dieses klassische Zuweisungsproblem wie folgt fassen:

$$\text{minimiere} \quad \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} \tag{2.12}$$

Unter den Bedingungen

$$\sum_{i=1}^{n} x_{ij} = 1 \quad \forall j \in \{1, 2, \cdots, n\} \tag{2.13}$$

$$\sum_{j=1}^{n} x_{ij} = 1 \quad \forall j \in \{1, 2, \cdots, n\} \tag{2.14}$$

$$x_{ij} \in \mathbb{B} \tag{2.15}$$

Die praktische Relevanz dieser Problemstellung liegt auf der Hand: So stellt beispielsweise die Zuweisung von n Jobs auf n Maschinen, wobei jede einzelne Job-Maschine-Zuweisung andere „Kosten“ c_{ij} verursacht (hier z.B. Zeit, die benötigt wird, um ein Job zu bearbeiten), eine solche praktische Anwendung dar.

Das Erzeugen aller Lösungen würde die Generierung von $n!$ möglichen Zuweisungsmatrizen bedeuten, was schon für kleine Werte von n unverhältnismäßig hohen Rechenaufwand bedeutet. Algorithmische Lösungsansätze sind die *Ungarische Methode* [BDM09] oder der *Simplex-Algorithmus* [CLRS04]. Diese Algorithmen drücken die Komplexität auf eine polynomielle Zeit. Auf die Algorithmen wird hier jedoch nicht näher eingegangen, da sie nicht unserer eigentlichen Problemstellung entsprechen.

Abgeleitet von diesem klassischen Zuweisungsproblem gibt es eine Fülle weiterer Zuweisungsprobleme, denen spezifisch ist, dass die Anzahl der zugewiesenen Objekte gleich ist. Als umfassende Referenz hierfür sei [Pen05] genannt. Im Folgenden werden zwei dieser Zuweisungsprobleme näher betrachtet, die unserer Problemstellung ähneln.

Bottleneck Assignment Problem (BAP)

Ein Zuweisungsproblem, welches von der Problemstellung her in unsere Richtung weist, ist das *Bottleneck Assignment Problem* (BAP) [Pen05], [BDM09]. Hierbei geht es nicht um die Miniermung der Gesamtkosten (-summe), sondern um die Minimierung des Maximalwertes – wie kann eine Zuweisung gefunden werden, die den kritischen Wert, den „*Flaschenhals*“minimiert. Formal ausgedrückt lautet die Zielfunktion:

$$\text{minimiere} \quad \max_{I,J}\{c_{ij}x_{ij}\} \tag{2.16}$$

Unter den Nebenbedingungen (2.13), (2.14) und (2.15) wie beim klassischen Zuweisungsproblem, wobei $\mathcal{I}$ die Menge aller Agents und $\mathcal{J}$ die Menge aller Jobs ist. Bei dieser Variante ist die Mächtigkeit der beiden Mengen gleich. Praktisch relevant ist die Problemstellung, wenn es darum geht, kritische maximale Zuweisungskosten so klein wie möglich werden zu lassen.
Algorithmische Lösungsansätze in Polynomialzeit finden sich in [BDM09]. Die effiziente Lösbarkeit dieses Problems rührt von der Tatsache her, dass die beiden Mengen von Objekten die gleichen Kardinalitäten haben.

Balanced Optimization Problem (BOP)

Ein weiteres klassisches Zuweisungsproblem ist das *Balanced Optimization Problem* (BOP), das *Balancierte Zuweisungsproblem* [MPTW84, BDM09, Pen05], das unserer Problemstellung ähnelt, da auch hier ein Differenz zwischen kleinstem und größtem Wert minimiert werden soll („Balanced“). Jedoch ist zu betonen, dass es sich um ein klassisches Zuweisungsproblem handelt, also die Mengen von Agents und Jobs gleich groß sind, was für die Problemstellung unserer Arbeit nicht zutrifft. Der prinzipiellen Nähe zu unserer Problemstellung wegen soll diese Unterklasse der APs trotzdem aufgeführt werden.
Im Gegensatz zum BAP (siehe (2.16)) wird hier nicht nur der Maximalwert, sondern auch der Minimalwert in die Betrachtung miteinfließen. Es soll die Differenz zwischen diesen beiden Werten minimiert werden, was die Zielfunktion (2.17) ausdrückt. Im Hinblick auf die Praxis ist diese Modellierung überall dort adäquat, wo „Kosten“ angeglichen werden sollen. Ein Beispiel in [MPTW84] veranschaulicht diesen Fall mittels einer Reiseagentur, die verschiedene Reisegruppen auf verschiedene Rundreisen schickt, wobei die Kosten hier

jeweils der beanspruchten Zeit entsprechen. Ziel ist es nun, die Reisegruppen so auf die Rundreisen zu schicken, dass am Ende alle möglichst zum gleichen Zeitpunkt wieder an der Sammelstelle ankommen. Mathematisch formalisiert werden kann dieser Fall wie folgt:

$$\text{minimiere} \quad \max_{I,J}\{c_{ij}x_{ij}\} - \min_{I,J}\{c_{ij}x_{ij}\} \tag{2.17}$$

unter den Nebenbedingungen (2.13), (2.14) und (2.15) wie beim klassischen Zuweisungsproblem.
Martello et al. schlagen in [MPTW84] einen Algorithmus zur Lösung des Problems vor, der in [BDM09] erweitert wird, so dass eine polynomielle Komplexität von $\mathcal{O}(n^4)$ erreicht wird, wobei n die Zahl der Agents und Jobs ist. Der Algorithmus baut auf der Lösung für das BAP auf und versucht, sich durch langsame Erhöhung des Minimalwertes an ein Optimum heranzutasten. Details finden sich in [BDM09].

Generalisierte Zuweisungsprobleme

Nachdem die klassischen Zuweisungsprobleme betrachtet wurden, deren Charakteristikum eine gleiche Zahl von Agents und Jobs war, kommt die Untersuchung nun zu generalistierten Zuweisungsproblemen [Pen05, MT90].
Ebenso wie beim klassischen Zuweisungsproblem geht es beim *generalisierten Zuweisungsproblem* (*Generalized Assignment Problems*, *GAP*) um die Minimierung der Gesamtkosten (-summe) aller Zuweisungen. Im Unterschied zum klassischen Zuweisungsproblem ist hier die Beschränkung aufgehoben, wonach die Zahl der Agents und Jobs gleich sein muss. Diese Erweiterung auf $n \geq m$ Jobs findet sich in der im Vergleich zum AP veränderten Zielfunktion (2.18) und in der Nebenbedingung (2.20) wieder. Dies ist für unsere Problemstellung von hoher Bedeutung, da die Zahl der Shares in der Regel größer ist als die Zahl der Provider.
Sei $I = \{1, \cdots, m\}$ die Menge aller Agents und $J = \{1, \cdots, n\}$ die Menge aller Jobs, so kann der Sachverhalt formal wie folgt dargestellt werden:

$$\text{minimiere} \quad \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij} \tag{2.18}$$

Unter den Bedingungen

$$\sum_{i=1}^{m} x_{ij} = 1 \quad \forall j \in \{1, 2, \cdots, n\} \tag{2.19}$$

$$\sum_{j=1}^{n} x_{ij} \geq 1 \quad \forall i \in \{1, 2, \cdots, m\} \tag{2.20}$$

$$x_{ij} \in \mathbb{B} \tag{2.21}$$

Dabei bedeutet die erste Nebenbedingung (2.19), dass einem Job j immer nur genau ein Agent i zugewiesen sein darf, ein Job somit nur von einem Agent bearbeitet wird. Die zweite Bedingung (2.20) besagt, dass einem Agent auch mehrere, aber mindestens ein Job zugewiesen werden muss. Die dritte Bedingung (2.21) gibt Auskunft über die Belegung der Variablen der Zuweisungsmatrix, welche nur aus bool'schen Werten bestehen darf.
Hier gibt es m^n gültige Zuweisungsmatrizen. Beim Generalisierten Zuweisungsproblem handelt es sich um ein NP-vollständiges Problem, was Martello und Toth durch Reduktion auf das multiple Rucksackproblem in [MT90] zeigen. Da dieses Problem von unserer Problemstellung entfernt ist, weil die Zielfunktion eine andere ist, sei hier eine Referenz [CVW92] genannt, die einen Überblick über Algorithmen zur Lösung von GAPs enthält.

Bottleneck Generalized Assignment Problem

Wie zum klassischen Zuweisungsproblem gibt es auch zur generalisierten Variante eine Ausformulierung, bei der es um die Minimierung eines Flaschenhalses geht. Diese Version wird als *Bottleneck Generalized Assignment Problem* (BGAP) bezeichnet [MN88, MN92, MT95, Peno5]. In der Literatur finden sich zwei spezifische Ausformulierungen dieses Problems, welche als *Task BGAP* und *Agent BGAP* bezeichnet werden und auf die im Folgenden eingegangen wird.

Task Bottleneck Generalized Assignment Problem Das TBGAP entspricht, analog zum klassischen Fall, der Übertragung der Problemstellung auf eine Zielfunktion, die der Minimierung eines Flaschenhalses dient, wenn die Zuweisungen für sich betrachtet den kritischen Wert darstellen können. Formal ausgedrückt lautet dies:

$$\text{minimiere} \quad \max_{I,J}\{c_{ij}x_{ij}\} \tag{2.22}$$

unter den gleichen Nebenbedinungen (2.19), (2.20) und (2.21) wie beim GAP.
Sowohl Martello und Toth [MT95] als auch Mazzola und Neebe [MN88, MN92] zeigen, dass es sich um ein NP-schweres Problem handelt – deterministische Algorithmen benötigen somit zur exakten Lösung wahrscheinlich[10] einen exponentiellen Rechenaufwand mit der Eingabegröße (Paramter m und n) des Problems.

Mazzola und Neebe [MN92] präsentieren eine exakte Lösung: Indem zunächst eine Heuristik angewandt wird, die manchmal die exakte Lösung liefert. Ist dies nicht der Fall wird ein Branch-and-Bound-Verfahren angewendet, um schließlich zur exakten Lösung zu kommen, was den Algorithmus prinzipiell ineffizient macht. Die Evaluation soll jedoch

[10] Wenn $\mathcal{P} \neq \mathcal{NP}$ gilt [HMU03].

zeigen, dass dieser Aufwand vernachlässigbar ist und der Algorithmus in den meisten Fällen trotzdem gute Performanz zeigt.
Martello und Toth stellen in [MT95] einen exakten Branch-and-Bound und auch einen Approximationsalgorithmus vor, wobei sich der exakte Branch-and-Bound-Algorithmus wie der Algorithmus vom Mazzola und Neebe [MN92] von einer Grenze aus vortastet und dann mit einer Branch-and-Bound-Strategie sukzessive die exakte Lösung konstruiert.

Agent Bottleneck Generalized Assignment Problem Die Problemstellung des TBGAP ist für unser Vorhaben weniger von Bedeutung, da bei uns der kritische Punkt nicht einzelne Zuweisungen sind, sondern alle Zuweisungen auf jeweils einen Provider – es geht darum, die Zuweisungen auf einen Provider, der evtl. als Flaschenhals wirkt, zu minimieren. Diese Minimierung eines kritischen Agents kann mathematisch modelliert wie folgt ausgedrückt werden:

$$minimiere\ (\max_{I}(\sum_{j=1}^{n} c_{ij}x_{ij})) \tag{2.23}$$

Dabei gelten die gleichen Bedingungen (2.19), (2.20) und (2.21) wie beim GAP. Der Teil $(\sum_{j=1}^{n} c_{ij}x_{ij})$ Zielfunktion (2.23) repräsentiert die Summe aller auf einem Agent platzierten Jobs. Mazzola und Neebe bezeichnen dieses Problem darum als *Agent Bottleneck Generalized Assignment Problem* (ABGAP) und erwähnen es in [MN88] lediglich, ohne konkreter darauf einzugehen.
Neben der genannten Referenz tritt das Problem auch unter dem Namen *Imbalanced Time Minimizing Assignment Problem* (ITMAP) bei Arora und Puri [AP97] auf[11]. Zur Lösung schlagen die Autoren eine lexikografische Suche vor[12], welche wie die anderen Verfahren von einer initialen Grenze ausgeht, die Schritt für Schritt zu einer optimalen Lösung ausgebaut wird. Der Evaluation der Autoren folgend liefert ihr Algorithmus aufgrund der guten initialen Grenze in der Praxis schnell einen Optimalwert. Ein in der Originalversion vorhandener Fehler wird in [MCA01] beseitigt sowie ein weiteres Verfahren für ITMAP vorgeschlagen.

[11] „Imbalanced“ im Namen ITMAP bezieht sich hier nicht wie in unserem Sinne auf einen Balancezustand zwischen den Agents/Providern, sondern soll bedeuten, dass die Zahl der Agents nicht gleich der Zahl der Jobs ist. Im Folgenden wird die Benennung ITMAP verworfen und auch ABGAP genannt, da es sich um die gleichen Probleme handelt und sich so Unklarheiten vermeiden lassen.

[12] Im Allgemeinen bedeutet lexikographische Suche bei Optimierungsproblemen, dass Lösungskandidaten als Wörter kodiert werden. Dabei werden Wörter sukzessive konstruiert. Wenn absehbar ist, dass ein Wort keine optimale Lösung mehr darstellen kann, weil schon ein Teilwort (Partialwort) schlechter ist, wird der komplette Zweig verworfen. Dieses Verfahren ist somit eine abgewandelte Form von Branch-and-Bound (im Worst Case m^n Schritte.

3 Konzepte zur Platzierungsoptimierung der verteilten Positionsinformationen

Nachdem auf die verwandten Arbeiten eingegangen und die theoretische Fundierung erläutert wurde, wird jetzt das Systemmodell vorgestellt: Die bestehende Architektur nach [DSR11] wird um eine *Trust Database* erweitert, welche die Vertrauenswerte der Provider hält und verwaltet.
Dann werden in Abschnitt 3.2 die Anforderungen an das Systems thematisiert, welche die maximale Sicherheit der Privatsphäre bei LBS gewähren sollen.
In Abschnitt 3.3 werden die konkreten Platzierungsstrategien für die verschiedenen Szenarien besprochen: Es wird einerseits auf den Fall eingegangen, dass die Zahl der Shares nicht vordefiniert ist. Sie kann also von Nutzer-Seite festgelegt werden. Damit ist die Problemstellung, für welche Anzahl von Shares das Risiko optimal verteilt werden kann.
Im anderen Fall wird die Zahl der Shares als fest vorgegeben angenommen und es wird gefragt, wie hier optimale Sicherheit erreicht werden kann. Bei dieser Problemstellung handelt es sich um ein Zuweisungsproblem, welches entsprechend formalisiert wird und von dem gezeigt wird, dass es sich um ein NP-schweres Problem handelt. Schließlich werden Lösungsansätze für dieses Problem erarbeitet. Dazu folgt in Abschnitt 3.3.2 eine Diskussion der Ergebnisse und Lösungsansätze aus Kapitel 2, wobei abgewogen wird, welcher Lösungsansatz im konzeptuellen Teil am besten geeignet ist, um den Anforderungen beizukommen.

3.1 Systemmodell

Wie sich in Abschnitt 2.2 zeigte, ist der Schutz der Privatsphäre bei Location-based Services ein zentrales Anliegen. Durch die Sensibilität dieser Technologie müssen Wege gefunden werden, um den Schutz der Privatsphäre zu gewähren.
Verfahren wie k-Anonymity oder Zugriffslisten gehen von einer vertrauenswürdigen Instanz zur Speicherung der Positionsinformation aus. Diese Annahme ist jedoch zu kritiseren, denn ein Location Server kann von außen kompromittiert werden. Darüber hinaus ist es möglich, dass intern ein unberechtiger Zugriff erfolgt, um z.B. Nutzerdaten mit der Positionsinformation zu verknüpfen und für Werbung zu missbrauchen.

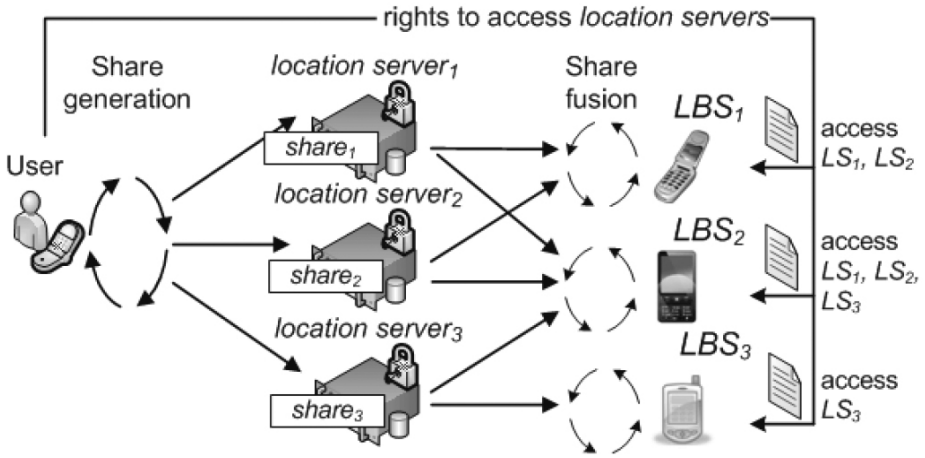

Abbildung 3.1: Darstellung der Systemarchitektur von Dürr und Skvorzov [DSR11].

Dürr und Skvorzov schlagen aus diesem Grund in [DSR11] ein System vor, welches mit mehreren unabhängigen Location Servern arbeitet und das Prinzip der räumlichen Verschleierung der Positionsinformation mit der Koordinatentransformation verbindet. Dieses Konzept wird auch als *Verteilung der Positionsinformation* bezeichnet und wurde im Detail in Abschnitt 2.2.6 erläutert. Hierbei wird die exakte Positionsinformation in Teilvektoren, auch *Shares* genannt, zerlegt und auf den unabhängigen Location Servern, genannt *Provider* platziert. Die Idee hinter diesem System ist, dass die exakte Position nur rekonstruiert werden kann, wenn alle Provider gehackt werden, was einen unwahrscheinlichen Fall darstellt. Einzelne Teile der Positionsinformation können die exakte Position nur ungenau rekonstruieren. Das Systemmodell nach 2.2.6 ist in Abbildung 3.1 dargestellt und besteht aus folgenden Teilen:

Mobiles Gerät Dabei handelt es sich um ein Handy, ein Smartphone oder ein sonstiges mobiles Gerät, welches in der Lage ist, die Position des Nutzers zu bestimmen, beispielsweise via GPS oder durch GSM-Ortungsverfahren. Die Abtastung der Position erfolgt regelmäßig. Eine Softwareanwendung auf diesem Gerät, dem vertraut wird, zerlegt die exakte Position in Teile, die Shares. Dieser Prozess wird *Share generation* genannt. Zur Generierung der Shares werden in [DSR11] zwei Algorithmen vorgeschlagen und evaluiert, auf die an dieser Stelle jedoch nicht näher eingegangen wird.

Location Server / Provider In diesem Systemmodell gibt es im Gegensatz zu bisherigen Ansätzen[1] mehrere Provider, denen nicht vertraut werden muss. Sie verwalten die Positionsinformation des mobilen Gerätes, wobei jeder Location Server nur einen Teil, also eine gewisse Anzahl von Shares bekommt.
Es ist hervorzuheben, dass bei diesem Ansatz die LS unabhängig voneinder sein müssen.
Jeder LS stellt sowohl den mobilen Nutzern wie auch dem LBS Schnittstellen zur Verfügung. Der Nutzer kann mittels dieser Schnittstelle seine Position hochladen oder einen gewissen LBS zur Nutzung der gehaltenen Positionsinformation autorisieren. Dies geschieht, indem jeder Provider eine Zugriffsliste[2] hält, in der die Berechtigungen gesetzt werden können. Zur Seite des LBS bietet die Schnittstelle eine Möglichkeit zur Abfrage der Positionsinformation.

Ortsbezogene Anwendung Diese Anwendung ist ein Dienst, der dem Nutzer bezogen auf seine Position Informationen anbietet. In Abbildung 3.1 ist dieser Dienst mit LBS_i bezeichnet. Diesem Dienst müssen Positionsinformationen zugänglich gemacht werden, die er erhält, indem die LS abgefragt werden. Auf der Seite des LBS müssen die einzelnen Shares wieder zusammengefügt werden, was mittels einem *Share Fusion*-Algorithmus geschieht [DSR11].
Dann kann die Anwendung dem Nutzer Dienste wie POI-Finder usw. anbieten. Eine Neuerung im Vergleich zu anderen Konzepten[3] ist, dass verschiedenen Diensten unterschiedliche Genauigkeitsstufen der Position zugänglich gemacht werden können. Die Dienste sind also feingranular regulierbar. Diese Regulation wird durch die Zugriffslisten auf den LS erreicht.

3.1.1 Trust Database

Momentan wird im Konzept von Dürr und Skvorzov [DSR11] von Providern ausgegangen, die alle das gleiche Sicherheitsrisiko besitzen, was eine Einschränkung darstellt, die in der Praxis nicht zutrifft.
Dieser Einschränkung kann beigekommen werden, indem mit jedem Provider ein Vertrauens- oder Risikowert assoziiert wird. Dies bedeutet, dass die Zahl der platzierten Shares auf dem jeweiligen Provider von Bedeutung ist: Auf einem Provider mit hohem Risiko sollten relativ wenige Shares platziert werden, da bei seiner Kompromittierung, die als wahrscheinlicher angesehen werden muss, eine genauere Position rekonstruiert werden kann. Um die Modellierung um diesen Aspekt zu erweitern, muss eine weitere Instanz ins System eingeführt werden, die Vertrauens- oder Risikowerte der LS speichert und verwalten kann. Diese Instanz

[1] siehe Abschnitt 2.2
[2] siehe Abschnitt 2.2.1
[3] siehe Abschnitt 2.2

wird als *Trust Database* bezeichnet. Das um die Trust Database erweitere Systemmodell ist in Abbildung 3.2 dargestellt.

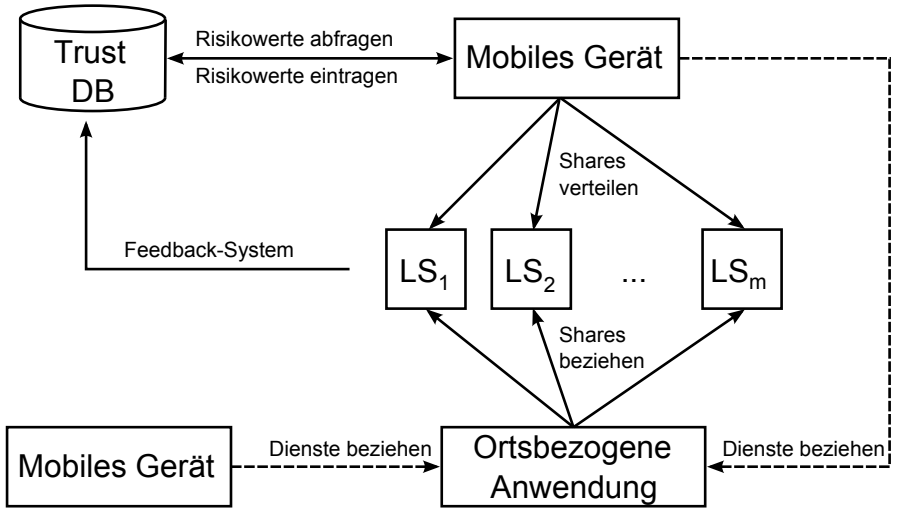

Abbildung 3.2: Um die *Trust Database* erweiterte Systemarchitektur.

In der Trust Database wird zu jedem LS ein Vertrauenswert bzw. ein Risikowert gehalten, der seine Wahrscheinlichkeit repräsentiert, kompromittiert zu sein. Die Trust Database ist eine Datenbank, die in einer Tabelle zu jedem Location Server einen absoluten Risikowert speichert. Dies ist in Tabelle 3.1 schematisch dargestellt.
Die Werte können auf verschiedene Weisen erzeugt werden, so z.B. durch ein automatisiertes Feedback-System oder auch durch subjektive Bewertungen des Nutzers. Die Vertrauenswerte können aus verschiedenen Werten wie Zuverlässigkeit, subjektiven Vertrauenswerten, Angriffshistorie oder Reputation der Betreiber gebildet werden, wobei hierauf nicht weiter eingegangen wird. Das angesprochene Feedback-System kann als Teil in dieser Trust Database integriert sein, aber auch außerhalb dieser liegen.

Location Server	Risikowert
1	r_1
2	r_2
3	r_3
...	...
m	r_m

Tabelle 3.1: Schematische Darstellung der Tabelle der Trust Database mit Risikowerten

Das Vertrauen muss in einer gewissen Weise modelliert werden, um algorithmisch damit arbeiten zu können. Gutscher [Guto7] entsprechend kann ein Vertrauensmodell in vier Komponenten unterteilt werden, die kurz erläutert werden.

Die Vertrauenswerte müssen modelliert werden. Darunter wird die Repräsentation der Vertrauenswerte verstanden, die als bool'sche oder skalare Werte oder in Form einer diskreten Verteilungsfunktion angegeben werden können.
Ein weiterer Teil ist die Berechnung der Vertrauenswerte: Hier wird festgelegt, wie die unterschiedlich repräsentierten Vertrauenswerte potentiell miteinander verrechnet werden können.
Neben diesen beiden Teilen werden noch Relationen und Schlussfolgerungsregeln als Teile eines Vertrauensmodells genannt. Damit ist gemeint, wie die Beziehung zwischen den Entitäten (bspw. „A vertraut B") modelliert wird und welche Schlüsse sich daraus ziehen lassen (bspw. „Wenn A vertraut B und B vertraut C, dann vertraut auch A der Instanz C").

Für diese Arbeit ist vor allem von Bedeutung, wie die Vertrauenswerte repräsentiert werden. Dabei werden die initialen Vertrauenswerte $t_{i(start)}$ als skalare Werte wie in Ausdruck (3.1) formalisiert, wobei i den Index eines Providers repräsentiert.

$$t_{i(start)} \in [0, 1] \tag{3.1}$$

$t_{i(start)}$ bedeutet dann, wie sehr einem Provider i auf einer Skala von 0 bis 1 vertraut wird. Risiko wird dann als Gegenereignis zum Vertrauen definiert:

$$r_{i(start)} = 1 - t_{i(start)} \tag{3.2}$$

Somit gilt auch $r_{i(start)} \in [0, 1]$, wobei diese Werte frei wählbare Risikowerte sind. Dann müssen alle Werte $r_{i(start)}$ auf Werte r_i normalisiert werden, so dass nach der Normalisierung $\sum_I r_i = 1$ gilt:

$$r_{i(start)} \overset{Norm.}{\rightarrow} r_i \tag{3.3}$$

Dies hat zum Ergebnis, dass es sich somit immer um relative Werte handelt, die in Bezug zueinander stehen. Konkret wird ein Sicherheitsrisiko von 1 unter den Providern aufgeteilt werden. Mit dieser Annahme lassen sich relative Aussagen über das Risiko der Provider treffen, die leicht quantifizierbar sind. Hätten alle Provider das gleiche Risiko, würde 1 zu gleichen Teilen aufgeteilt werden usw.
Da es sich algorithmisch aber als vorteilhaft erweist, auf natürlichen Zahlen zu operieren, sollten diese Werte noch skaliert werden. Dazu wird ein Wert r_i zunächst auf eine natürliche Zahl abgebildet, was durch eine Multiplikation mit einem Faktor wie 100 oder 1000 geschehen kann, je nachdem, welche Genauigkeit repräsentiert werden soll[4]. Dann kann der kleinste dieser Werte als Faktor 1 gesetzt werden, auf den sich alle anderen Werte beziehen.

[4]Dazu muss ferner gelten, dass $r_i \in \mathbb{Q}$, weil sich die Skalierung auf ganzzahlige Werte sonst nicht durchführen ließe.

Seien beispielsweise 5 Provider mit den Risikowerten $r_1 = 0,25$, $r_2 = 0,5$, $r_3 = 0,1$, $r_4 = 0,05$ und $r_5 = 0,1$ gegeben, dann sind die entsprechenden auf natürliche Zahlen skalierten Werte $r_{1(skal)} = 25$, $r_{2(skal)} = 50$, $r_{3(skal)} = 10$, $r_{4(skal)} = 5$ und $r_{5(skal)} = 10$. Dies kann noch weiter skaliert werden auf $r_{1(skal)} = 5$, $r_{2(skal)} = 10$, $r_{3(skal)} = 2$, $r_{4(skal)} = 1$ und $r_{5(skal)} = 2$, wobei jeweils die gleichen Risikorelationen zwischen den Providern repräsentiert werden. Eine Trust Database mit den Werten aus diesem Beispiel ist in Tabelle 3.2 dargestellt.

Location Server	Risikowert
1	5
2	10
3	2
4	1
5	2

Tabelle 3.2: Beispielhafte Darstellung der Tabelle der Trust Database

Die Rücktransformation geschieht, indem alle ganzzahligen Risikowerte r_i addiert werden. Dann werden die jeweiligen Risikowerte durch diese Summe geteilt, was einen Wert $r_i \in [0, 1]$ liefert. Diese Rechenvorschrift ist in Formel (3.4) dargestellt.

$$r_i = \frac{r_{i(skal)}}{\sum_I r_{i(skal)}} \tag{3.4}$$

Die beispielhafte Rücktransformation für den ersten Provider ist in Formel (3.5) dargestellt:

$$r_1 = \frac{5}{20} = 0,25 \tag{3.5}$$

3.2 Problemstellung und Anforderungen

Die Aufgabe ist es, die optimale Platzierung von Shares auf Providern zu finden.
Optimalität der Platzierung ist diejenige Belegung von Shares auf Providern, die im Falle von ungewichteten Shares die gegebenen Sicherheitsrisiken ausgleicht. Dies führt dazu, dass risikoreiche Provider weniger Shares zugewiesen bekommen als vertraute. Ist es wahrscheinlich, dass ein Server gehackt wird, liefert seine tatsächliche Kompromittierung nur wenige Shares, was die Sicherheit des Gesamtsystems erhöht.
Sind die Shares gewichtet, soll dadurch eine *angemessene Verschlechterung* (engl. *graceful degradation*) der Privatsphäre bei fortschreitender Zahl kompromittierter Provider erreicht werden. Bei jedem weiteren Server, der gehackt wird, verschlechtert sich die Privatsphäre nur um einen konstanten Faktor.
Die *Fragilität* R_i eines Providers i bezeichnet sein spezifisches Sicherheitsrisiko, das er ins System nach Aufnahme von Shares einbringt und wird berechnet als Multiplikation des Risikowertes eines Providers mit den Risikowerten der auf ihm platzierten Shares, also

$$R_i = r_i \cdot S_i \tag{3.6}$$

wobei $S_i = \sum_{\text{j auf i zugeordnet}} s_j$ die Summe der Risiken der Shares s_j auf dem platzierten Provider i darstellt. Sind die Shares ungewichtet ($= 1$), wird der Risikowert eines Providers mit der Anzahl der Shares multipliziert.

Der Zielzustand ist das Gleichgewicht der Fragilität aller m Provider; dieser Zielzustand ist in Formel (3.7) ausgedrückt.

$$R_1 = R_2 = \cdots = R_m \tag{3.7}$$

Die Problemstellung ist es nun, diesen Balance-Zustand durch eine entsprechende Platzierung so weit wie möglich herzustellen. Dabei ist es nicht immer möglich, ein komplett austariertes System zu erhalten.
Der Balance-Zustand wird durch eine möglichst geringe Differenz zwischen minimaler und maximaler Fragilität auf den Servern erreicht, weil dadurch die Risiken so gut wie möglich austariert werden und eine Balance wie in Formel (3.7) erreicht werden kann.
Eine erste Formalisierung der Problemstellung, die im folgenden Verlauf dieser Arbeit konkretisiert wird, lautet somit:

$$\text{minimiere} \quad (R_{max} - R_{min}) \ \forall \ \text{Provider} \tag{3.8}$$

wobei $(R_{max} - R_{min})$ im Folgenden auch als Δ bezeichnet wird.

Hierdurch wird erreicht, dass gegebene Sicherheitsrisiken der Server relativ ausbalanciert werden, wenn es sich um ungewichtete Shares handelt.

Im Falle von gewichteten Shares wird eine angemessene Verschlechterung der Privatsphäre erreicht, wodurch es keine Schwachstelle im System gibt, die das Sicherheitsrisiko überproportional gefährdet (keinen sog. Flaschenhals). Dieser Zusammenhang ist in Abbildung 3.3 dargestellt, wobei Abbildung 3.3(a) den Fall für eine optimale Lösung, Abbildung 3.3(b) für eine nicht-optimale Lösung mit einem Flaschenhals im System zeigt.
Ein weiteres Argument für die angemessene Verschlechterung ist, dass der Angreifer keinen Vorteil darin hat, einen Server mit hohem Risikowert oder mit schlechter Reputation anzugreifen: Die erfolgreiche Kompromittierung liefert den möglichst gleichen Teil an Positionsinformationen, wie wenn jeder beliebige andere Server gehackt würde.

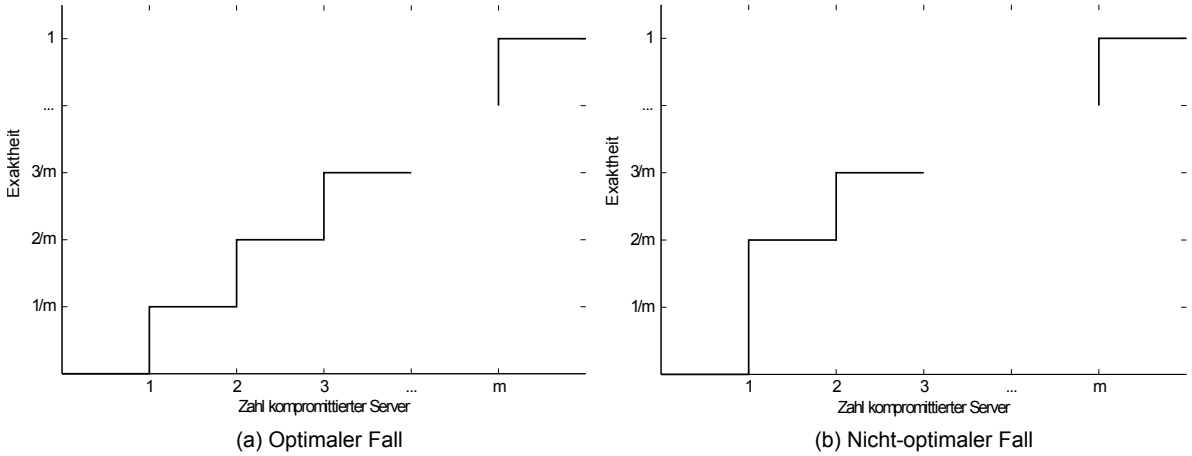

Abbildung 3.3: (a) Optimaler Fall der Verteilung, wenn Exaktheit der Positionsrekonstruktion konstant ansteigt.
(b) Nicht-optimaler Fall der Verteilung, da bei dem ersten kompromittierten Server sich eine genauere Position rekonstruieren ließe.

Sicherheit der Privatsphäre kann somit interpretiert werden als die Fähigkeit, die Fragilitäten konstant zu halten, um Sicherheitsrisiken auszubalancieren, keine überproportionalen Schwachstellen im System zu haben und eine angemessene Verschlechterung der Privatsphäre im Fall von kompromittierten Servern zu erreichen.

Neben der eigentlichen Optimierung ist auch die Zeit zur Berechnung einer Platzierung ein kritischer Faktor, da die Platzierung auf leistungsschwächeren mobilen Geräten berechnet wird. Darüber hinaus muss beachtet werden, dass bei jeder Positionsänderung die Shares evtl. neu platziert werden müssen. Will man von einer harten Grenze ausgehen, kann die Abtastrate von GPS (1Hz) herangezogen werden. Allerdings haben andere Methoden zur Positionsbestimmung eine weitaus geringere Abtastrate, wodurch obige Abschätzung relativiert wird [DSR11]. Somit ist eine kurze Berechnungsdauer (im Sekundenbereich) nach der Optimalität der Platzierung eine weitere Anforderung an das System.

3.3 Platzierungsstrategien

Nachdem das um die Trust Database erweiterte Systemmodell vorgestellt wurde, kann jetzt auf die eigentliche Aufgabe, nämlich die Erlangung des Balance-Zustands, eingegangen werden. Dabei gliedert sich der Abschnitt in 2 Szenarien:
In *Szenario 1* ist die Anzahl und damit auch die Gewichtung der Shares nicht von vornherein festgelegt, so dass der Optimalzustand über eine Justierung der Anzahl erreicht werden kann.
Szenario 2 gibt die Zahl der Shares vor, was dazu führt, dass es sich um ein Zuweisungsproblem handelt.
Zu jedem Szenario gibt es Unterklassen, die als Übersicht in Tabelle 3.3 dargestellt sind. In jedem Unterabschnitt sind zur Übersicht noch einmal die Fälle tabellarisch dargestellt und der jeweils betrachtete Fall grau hinterlegt.

Anzahl der Shares					
Szenario 1: nicht vordefiniert			**Szenario 2: vordefiniert**		
Fall	**Providerrisiken** r_i	**Sharerisiken** s_j	Fall	**Providerrisiken** r_i	**Sharerisiken** s_j
1-a	gleich	gleich	**2-a**	gleich	gleich
1-b	gewichtet	gleich	**2-b**	gewichtet	gleich
1-c	gleich	gewichtet	**2-c**	gleich	gewichtet
1-d	gewichtet	gewichtet	**2-d**	gewichtet	gewichtet
Ansatz					
Justierung der Anzahl			**Zuweisungsproblem**		

Tabelle 3.3: Übersicht über die unterschiedlichen Szenarien

Ziel der Optimierung ist es, um noch einmal die Anforderungen 3.2 zu rekapitulieren, die Fragilitäten der Provider anzugleichen, also einen Zustand der Balance zu erreichen:

$$R_1 = R_2 = \cdots = R_m = R = konst.$$

.

3.3.1 Szenario 1: Nicht vordefinierte Anzahl von Shares

Variable	Erklärung
n	Anzahl der Shares
m	Anzahl der Provider
r_i	Risiko des Providers i
s_j	Gewichtung der Share j
S_i	Sharerisiko aller einem Provider i zugeordneter Shares
n_i	Zahl der Shares auf dem Provider i
R_i	Fragilität eines Providers i
R	Relatives Gesamtrisiko bzw. Skalierungsfaktor

Tabelle 3.4: Erklärung der Variablen in Szenario 1

Zunächst wird der Fall betrachtet, dass die Anzahl der Shares n nicht von vornherein festgelegt ist. Damit sind auch die Sharegewichte nicht fest vorgegeben. Jedoch ist von einer beliebigen, aber festen Anzahl von Providern m auszugehen. Nun muss erörtert werden, für welche Anzahl von Shares das Sicherheitsrisiko minimal wird und optimal verteilt ist, so dass jeder Provider das möglichst gleiche, möglichst geringe Sicherheitsrisiko darstellt. Hierbei müssen vier Unterfälle betrachtet werden, die sich darin unterscheiden, ob die Shares und Provider jeweils gewichtet sind oder nicht. Die Semantik der in diesem Szenario benutzten Variablen wird in Tabelle 3.4 dargestellt.

Provider ungewichtet / Shares ungewichtet (Szenario 1-a)

Szenario 1: nicht vordefinierte Anzahl von Shares		
Fall	**Providerrisiken** r_i	**Sharerisiken** s_j
1-a	gleich	gleich
1-b	gewichtet	gleich
1-c	gleich	gewichtet
1-d	gewichtet	gewichtet

In der ersten Unterklasse von Szenario 1 sind sowohl die Providerrisiken als auch die Risiken der Shares gleich, d.h. $r_1 = r_2 = ... = r_m = r = 1$ und $s_1 = s_2 = ... = s_n = s = 1$.
Soll der Balancezustand (Formel (3.7) erreicht werden, folgt aus $r \cdot s \cdot n_1 = r \cdot s \cdot n_2 = \cdots = r \cdot s \cdot n_m$, dass jeder der m Provider eine gleiche Anzahl

$$n_1 = n_2 = \cdots = n_m \tag{3.9}$$

Shares bekommt. Dies ist nur möglich, wenn die Gesamtzahl der Shares $n = k \cdot m$ ein ganzzahliges Vielfaches der Anzahl der Provider ist, also $n = m$, $n = 2 \cdot m$, $n = 3 \cdot m$, ... bzw. wenn gilt $n \bmod m = 0$. Hätte ein Provider mehr Shares als ein anderer, so würden Dritte bei dessen Kompromittierung über mehr Shares verfügen, woraus sich eine exaktere Position rekonstruieren ließe.
Zu Bestimmung der optimalen Zahl n ist lediglich ein Schritt nötig, womit die Laufzeitkomplexität $\mathcal{O}(1)$ beträgt.

Provider gewichtet / Shares ungewichtet (Szenario 1-b)

Szenario 1: nicht vordefinierte Anzahl von Shares		
Fall	**Providerrisiken** r_i	**Sharerisiken** s_j
1-a	gleich	gleich
1-b	gewichtet	gleich
1-c	gleich	gewichtet
1-d	gewichtet	gewichtet

Haben die Provider unterschiedliche gegebene Risikowerte r_i, sind die Shares ungewichtet, d.h. es gilt $s_1 = s_2 = ... = s_n = s = 1$ und ist nach der Anzahl der Shares gefragt, für die das Sicherheitskrisiko minimal und ausbalanciert wird, so stellt sich der Sachverhalt wie folgt dar: Wenn jedem Provider i seinem eigenen Risiko r_i entsprechend umgekehrt proportional viele Shares n_i zugeordnet werden, kann ein konstantes Risiko erreicht werden, das für alle Provider gleich ist – das eigene Gewicht eines Providers wird durch die Zahl der Shares „ausbalanciert". Die Herleitung erfolgt aus Formel (3.7), wonach alle Fragilitäten im System gleich sein müssen: Aus $r_1 \cdot s \cdot n_1 = r_2 \cdot s \cdot n_2 = \cdots = r_m \cdot s \cdot n_m$ ergibt sich dann die gesuchte Anzahl von Shares auf einem Provider n_i als

$$n_i = \frac{1}{r_i} \tag{3.10}$$

bzw. $n_i = \frac{R}{r_i}$, weil mit ganzen Zahlen operiert werden muss.

Die gesuchte Gesamtzahl der Shares, für die sich Optimalität einstellt, ist dann:

$$n = \sum_{i=1}^{m} n_i = \sum_{i=1}^{m} \frac{R}{r_i} = R \cdot \sum_{i=1}^{m} \frac{1}{r_i} \tag{3.11}$$

R skaliert die Werte und kann als „relatives Gesamtrisiko" interpretiert werden – entscheidend ist, dass dieses R auf allen Providern gleich ist. Das Risiko kann nur dann gleich verteilt werden, wenn R gerade und durch alle möglichen Gewichte r_i teilbar ist. Dies könnte erreicht

werden, wenn $R = r_1 \cdot r_2 \cdot r_3 \cdots r_m$ gelten würde – dann wäre R aber nicht die minimale Lösung. Das R, welches sich durch alle r_i teilen lässt und dabei minimal ist, kann mittels des *kleinsten gemeinsamen Vielfachen* aller Werte r_i bestimmt werden:

$$R = kgV(r_1, r_2, \cdots, r_m) \tag{3.12}$$

Indem R dem kleinsten gemeinsamen Vielfachen entspricht, können die Shares so auf die Server verteilt werden, dass ein gleichverteiltes Risiko über allen Providern erreicht werden kann, selbst wenn diese unterschiedliche Risikowerte besitzen. Wird nun ein Provider, für den es wahrscheinlicher ist, kompromittiert zu werden, auch wirklich gehackt, gehen hier nur wenige Shares verloren. Dies folgt aus dem konstanten Gesamtrisiko über allen Providern. Die Laufzeit dieser Berechnung ist hauptsächlich durch die Bildung von $n_i = \frac{1}{r_i}$ bestimmt. Hierzu sind m Schritte nötig, weil einmal über alle Provider iteriert werden muss. Das kgV lässt sich mittels des euklidischen Algorithmus bestimmen, welcher eine logarithmische Komplexität mit Ziffernlänge besitzt. Da für unsere Problemstellung allerdings nur die Zahl der Shares und der Provider von Bedeutung ist, ergibt sich auch hier die Laufzeitkomplexität $\mathcal{O}(m)$.

Provider ungewichtet / Shares gewichtet (Szenario 1-c)

Szenario 1: nicht vordefinierte Anzahl von Shares		
Fall	**Providerrisiken** r_i	**Sharerisiken** s_j
1-a	gleich	gleich
1-b	gewichtet	gleich
1-c	gleich	gewichtet
1-d	gewichtet	gewichtet

In diesem Fall sind die Providergewichte gleich, d.h. $r_1 = r_2 = \cdots = r_m = r = 1$ und die Sharegewichte unterschiedlich. Da in Szenario 1 die Zahl n der Shares frei wählbar ist, sind auch die Gewichtungen der Shares frei wählbar und bedeuten so einen zusätzlichen Freiheitsgrad. Bezeichne S_i das Gesamtgewicht aller dem Provider i zugeordneten Shares: $S_i = \sum_{\text{j auf i zugeordnet}} s_j$. Dann gilt für die Provider nach der Balanceformel (3.7), dass $r \cdot S_1 = r \cdot S_2 = \cdots = r \cdot S_m$. Hieraus folgt

$$S_1 = S_2 = \cdots = S_m \tag{3.13}$$

Optimalität wird erreicht, indem genauso viele Shares wie Provider erzeugt werden und alle die gleiche Gewichtung besitzen. S_i kann interpretiert werden als Gewichtung einer Share, die evtl. noch weiter aufteilbar ist:

$$S_i = 2 \cdot (\frac{S_i}{2}) = \cdots = k \cdot (\frac{S_i}{k}) \tag{3.14}$$

Dabei kann diese *Grundshare* mit Gewicht S_i in k Shares aufgeteilt werden, wenn $S_i \mod k = 0$ gilt. Dies bedeutet, dass ein Provider i eine Share mit Gewichtung S_i haben kann, zwei mit jeweiliger Gewichtung $\frac{S_i}{2}$ oder allgemein n_i Shares mit Gewichtung $(\frac{S_i}{n_i})$.
Von der Abschätzung der Laufzeit her betrachtet ist dieser Fall analog zu Fall 1-a: Es reicht ein Vergleich, um die Zahl der nötigen Shares zu bestimmen, ihre Gewichtung wird gleich gewählt. Die Laufzeitkomplexität hierfür ist $\mathcal{O}(1)$.

Provider gewichtet / Shares gewichtet (Szenario 1-d)

Szenario 1: nicht vordefinierte Anzahl von Shares		
Fall	**Providerrisiken** r_i	**Sharerisiken** s_j
1-a	gleich	gleich
1-b	gewichtet	gleich
1-c	gleich	gewichtet
1-d	gewichtet	gewichtet

Wird die Anzahl der Shares gesucht und ist die Aufgabe, das Risiko über allen Providern gleich zu verteilen, wobei die gegebenen Providerrisiken frei wählbare, aber feste Werte r_1, $r_2, \cdots, r_m$, die Sharerisiken wie in Fall 1-c frei wählbar sind und gilt $S_i = \sum_{\text{j auf i zugeordnet}} s_j$, so lässt sich der Sachverhalt nach der Balance-Bedingung (3.7) formalisieren durch:

$$\begin{aligned} \text{1. Provider: } r_1 \cdot S_1 &= R \\ \text{2. Provider: } r_2 \cdot S_2 &= R \\ &\cdots \\ \text{m. Provider: } r_m \cdot S_m &= R \end{aligned}$$

Aus $r_1 \cdot S_1 = r_2 \cdot S_2 = \cdots = r_m \cdot S_m$ folgt, dass das Sicherheitsrisiko minimal wird, wenn genauso viele Shares wie vorhandene Provider erzeugt werden und jede Share eine Gewichtung bekommt, die umgekehrt proportional zum Providerrisiko ist:

$$S_i = \frac{1}{r_i} \tag{3.15}$$

bzw. $S_i = \frac{R}{r_i}$, weil mit ganzen Zahlen operiert werden muss, wobei R den Skalierungsfaktor bezeichnet.

Dadurch wird erreicht, dass das Risiko r_i eines Servers ausbalanciert wird. Dies ist nur möglich, wenn $R \bmod r_i = 0$. Es muss noch der Wert für R bestimmt werden, für den dies möglich ist. Analog zu Fall 2 lässt sich wie in Formel (3.12) das kleinste gemeinsame Vielfache verwenden, um R zu bestimmen.
Die Share S_i kann wieder wie im vorhergehenden Fall als Grundshare verstanden werden, die evtl. weiter aufgeteilt werden kann in:

$$S_i = 2 \cdot (\frac{S_i}{2}) = \cdots = n_i \cdot (\frac{S_i}{n_i}) \tag{3.16}$$

Dabei kann jede Grundshare in n_i Shares aufgeteilt werden, wenn $S_i \bmod n_i = 0$ gilt.

Die Komplextiät dieser Berechnung liegt in $\mathcal{O}(m)$: Zur Bestimmung der Zahl zu erzeugender Shares ist zwar nur genau ein Vergleich notwendig, die Berechnung der jeweiligen Gewichtung erfolgt aber, indem einmal über alle Providerrisiken iteriert wird.

Übersicht der Ergebnisse für Szenario 1

Da sich die Optimalität der Platzierung hier jeweils über die Justierung der Anzahl der Shares sowie deren Gewichtungen ergibt, lassen sich die Ergebnisse als geschlossene Formeln ausdrücken. Die Ergebnisse sind in Tabelle 3.5 noch einmal zusammenfassend dargestellt.

Fall	**Providerrisiken** r_i	**Sharerisiken** s_j	**Ergebnis: Zahl zu erzeugender Shares**
1-a	gleich	gleich	$n = m$
1-b	gewichtet	gleich	$n = \sum_{i=1}^{m} \frac{R}{r_i}$
1-c	gleich	gewichtet	$n = m$ mit gleichen Gewichten
1-d	gewichtet	gewichtet	$n = m$ mit Gewichtung $S_i = \frac{R}{r_i}$

Tabelle 3.5: Übersicht über die Ergebnisse in Szenario 1.

3.3.2 Szenario 2: Vordefinierte Anzahl von Shares

Das Problem stellt sich anders dar, wenn die Anzahl der Shares n beliebig, aber vordefiniert ist. Dann können zwar Aussagen darüber getroffen werden, für welche Anzahl der Shares sich ein optimales Sicherheitsrisiko einstellt. Die Anzahl der Shares n und deren Gewichtungen s_j müssen jedoch als gegeben betrachtet werden; die Shares müssen auf die m gegebenen Provider verteilt werden, nämlich so, dass das Sicherheitsrisiko zwischen den Providern ausbalanciert ist, was in Abschnitt 3.2 spezifiziert wurde. Die Anzahl der Shares ist in diesem Fall also ein fester Parameter. Mit den Überlegungen aus Teil 2.3 dieser Arbeit wird ersichtlich, dass es sich um ein Zuweisungsproblem handelt, das zunächst allgemein formalisiert wird, um dann konkret auf die Unterfälle einzugehen.

Formalisierung der Problemstellung als Zuweisungsproblem

Wie bei den Zuweisungsproblemen aus Kapitel 2 wird eine Zielfunktion benötigt, die unter gewissen Bedingungen minimiert werden soll. Diese Paramenter ergeben sich durch die Aufgabenstellung, die besagt, dass das Sicherheitsrisiko auf allen Providern möglichst gleich (klein) sein muss. Die weiteren Bedingungen ergeben sich aus dem Systemmodell, wonach jeder Provider mindestens eine Share zugewiesen bekommen sowie eine Share genau einem Provider zugeordnet sein muss. Hierbei ist $I = \{1, 2, \cdots, m\}$ die Menge aller Provider, $J = \{1, 2, \cdots, n\}$ die Menge aller Shares, wobei $m \leq n$ gilt. Die Kosten für eine Zuweisung einer Share j auf einen Provider i werden als c_{ij} bezeichnet, wobei gilt, dass $c_{ij} \in \mathbb{N}, c_{ij} \geq 0$. Diese Kosten entsprechen je nach Unterklasse dem Risiko des Providers, der Gewichtung der Shares oder dem Produkt aus beiden, was im Folgenden dargestellt ist:

$$\begin{aligned}
&\text{Szenario 2-a:} && c_{ij} = 1 \\
&\text{Szenario 2-b:} && c_{ij} = r_i \\
&\text{Szenario 2-c:} && c_{ij} = s_j \\
&\text{Szenario 2-d:} && c_{ij} = r_i \cdot s_j
\end{aligned}$$

Die Zuweisungsmatrix X wird gebildet aus bool'schen Werten $x_{ij} \in \mathbb{B}$, die eine Zuweisung einer Share j auf den Provider i darstellen, wenn sie den Wert 1 bzw. *wahr* besitzen. Formal ausgedrückt heißt die Optimierungsaufgabe nun:

$$\textit{minimiere } ((\max_{I}(\sum_{j=1}^{n} c_{ij}x_{ij})) - (\min_{I}(\sum_{j=1}^{n} c_{ij}x_{ij}))) \tag{3.17}$$

Unter den Bedingungen

$$\sum_{i=1}^{m} x_{ij} = 1 \quad \forall j \in \{1, 2, \cdots, n\} \tag{3.18}$$

$$\sum_{j=1}^{n} x_{ij} \geq 1 \quad \forall i \in \{1, 2, \cdots, m\} \tag{3.19}$$

$$x_{ij} \in \mathbb{B} \tag{3.20}$$

Bedingung (3.18) besagt dabei, dass jede Share genau einem Provider zugewiesen sein muss. Bedingung (3.19) bedeutet, dass einem Provider auch mehrere Shares gleichzeitig zugewiesen sein können. Die letzte Bedingung (3.20) stellt die Anforderung an die Zuweisungsmatrix dar, wonach diese Matrix aus bool'schen Werten gebildet werden muss.

Im Folgenden nenne ich dieses Problem in Anlehnung an die verwandten Zuweisungsprobleme aus Abschnitt 2.3.3 *Balanced Generalized Assignment Problem* (*BaGAP*).

Komplexität des Problems: NP-schwer

Das eben formalisierte Problem muss noch einer Untersuchung bzgl. seiner Komplexität unterzogen werden. Dazu müssen die Darstellungen aus Abschnitt 2.3.1 angeführt werden: Handelt es sich um ein NP-schweres Problem, muss dies durch eine Reduktion gezeigt werden, die ein anderes NP-vollständiges oder NP-schweres Problem auf das BaGAP polynomiell reduziert.

Beweis Vom ABGAP ist bekannt, dass es NP-schwer ist [MN88] [AP97]. Die Zielfunktion des ABGAP wird folgendermaßen dargestellt:

$$minimiere \quad (\max_{I}(\sum_{j=1}^{n} c_{ij} x_{ij})) \tag{3.21}$$

Während beim ABGAP (3.21) nur der Maximalwert der Gesamtsumme auf einem Provider minimiert wird, geht es beim BaGAP (3.17) um die Minimierung der Differenz zwischen Maximalwert und Minimalwert der Gesamtsummen auf dem jeweiligen Provider. Der Beweis erfolgt durch die Angabe eines Reduktionsverfahrens, welches

$$ABGAP \preceq_p BaGAP \tag{3.22}$$

leistet. Solch ein Reduktionsverfahren kann realisiert werden, indem zunächst ein zusätzlicher Provider $m+1$ hinzugefügt wird, dessen Risiko $r_{m+1} = T$ sehr hoch angesetzt wird. Außerdem wird eine Share eingefügt, deren Gewichtung $s_{n+1} = 0$ ist. Damit wird die Kostenmatrix C um eine Zeile, die dem neuen Provider entspricht sowie um eine Spalte, welche die neue Share repräsentiert, erweitert, was in Matrix (3.23) dargestellt ist.

$$C = \begin{pmatrix} c_{11} & c_{12} & \cdots & c_{1n} & 0 \\ c_{21} & c_{22} & \cdots & c_{2n} & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ c_{m1} & c_{m2} & \cdots & c_{mn} & 0 \\ s_1 \cdot T & s_2 \cdot T & \cdots & s_n \cdot T & 0 \end{pmatrix} \tag{3.23}$$

Indem das System sowohl um einen Provider mit einem hohen Risikowert als auch eine Share mit Gewicht 0 erweitert wird, wird sichergestellt, dass die Zuweisungen gemäß Nebenbedingung 3.19 gültig bleiben. Würde lediglich ein Provider mit Risiko 0 eingeführt, so würden alle Shares auf ihn zugewiesen, denn diese Zuweisungen würden in jedem Fall das optimale $\Delta = 0$ liefern, wobei die Zuweisung nicht mehr gültig wäre.
Durch eine derart aufgebaute Kostenmatrix wird erreicht, dass alle ursprünglichen Shares s_j mit $j \in \{1, \cdots, n\}$ nicht auf den neuen Provider zugewiesen werden, weil er bei Zuweisung zu hohe Kosten liefert, die unserem Ziel, die Differenz zu minimieren, von vornherein widersprechen. Hierbei reicht es schon, wenn $s_j \cdot T$ der doppelte Maximalwert aller Zuweisungskosten ist, da bei Auswahl dieser Zuweisung die Differenz immer größer als der Maximalwert der ursprünglichen Kosten wäre. Somit könnte eine Zuweisung auf diesen Provider niemals auf ein optimales Δ führen.
Die Zuweisung des neuen Providers $m + 1$ wird garantiert zu Share $n + 1$ erfolgen, denn jede andere Zuweisung für Provider $m + 1$ würde zu hohe Kosten verursachen und damit unserem Ziel, den Δ-Wert zu minimieren, widersprechen. Also gilt $x_{(m+1)(n+1)} = 1$.
Mit dieser Zuweisung wird der Minimal-Term des BaGAPs (Formel 3.17) für jede Probleminstanz garantiert 0 und ist daher nicht mehr relevant. Will man nun ein BaGAP lösen, gilt es, ein ABGAP zu lösen. Somit ist BaGAP NP-schwer. ■

Diskussion der Lösungsansätze

Indem das BaGAP als Zuweisungsproblem formalisiert wurde, gehört es zur Klasse der kombinatorischen Optimierungsprobleme. Diese Eigenschaft macht eine Abwägung notwendig zwischen einer exakten, aber mitunter ineffizenten Lösung und einer Näherungslösung. Dabei muss jedoch beachtet werden, dass eine exakte Lösung hinfällig ist, wenn eine unverhältnismäßig lange Zeit darauf gewartet werden muss. Diese Abwägung hängt von der konkreten Problemstellung ab.

Die in Abschnitt 2.3.2 dargelegten Lösungsansätze sind generelle Verfahren für kombinatorische Optimierungsprobleme. Grundsätzlich lässt sich sagen, dass die vollständige Enumeration vermieden werden sollte, da sie theoretisch wie praktisch exponentiell viele Schritte mit der Eingabelänge zur Lösung eines Problems benötigt.

Branch-and-Bound hängt stark von der Problemstellung ab: Kann aus dem Problem eine Strategie abgeleitet werden, zuverlässig eine Schranke zu bestimmen, mit der frühzeitig möglichst viele Zweige abgeschnitten werden können? Hierbei gilt für unser Problem, dass der theoretische Optimalwert zur Erzeugung der Balance $\Delta = 0$ ist. Die Nachbildung der Austarierung in einem Entscheidungsbaum ist jedoch keine triviale Aufgabe, denn es muss eine Schranke bestimmt werden, durch die schlechtere Lösungen von vornherein ausgeschlossen werden. Würde immer entlang dem Weg im Baum gegangen, der das minimale Δ liefert, wird das globale Optimum unter Umständen nicht erreicht. Dies rührt von der Tatsache her, dass es sich bei der Zielfunktion, die in Formel (3.17) formalisiert wurde, nicht um eine lineare Funktion handelt und darum ein lokaler Optimalwert nicht zu einem globalen führen muss. Es ist die Eigenart unserer Zielfunktion mit fortlaufender Zahl der getätigten Zuweisungen um den theoretischen Optimalwert von 0 zu zirkulieren. Dies rührt von der Tatsache her, dass in unserer Zielfunktion kein Wert fix gesetzt werden kann (im Gegensatz zum ABGAP z.B., wo immer $(\sum_J c_{ij}x_{ij}) = 0$ gilt). Ein Grenze würde also von vornherein zu unspezifisch sein; die Senkung der Komplexität durch Branch-and-Bound wird als gering eingeschätzt, weswegen auf diesen Ansatz verzichtet wird.
Die metaheuristischen Verfahren bieten im Allgemeinen keine Garantie, dass der optimale Wert gefunden wird, sind aber oft die einzige Möglichkeit, auch große Probleminstanzen behandeln zu können. Gegenüber problemspezifischen Heuristiken sind sie weniger schwer zu erarbeiten, dafür aber auch weniger genau. Unter den metaheuristischen Lösungsansätzen sollte die reine lokale Suche nicht verwendet werden, da sie beim ersten gefundenen Maximum stehen bleibt.
Evolutionäre Algorithmen als Vertreter der populationsbasierten Suchalgorithmen erweitern das Paradigma der lokalen Suche auf mehrere Lösungskandidaten. Durch die Mutationsoperation wird sichergestellt, dass ein großer Wertebereich abgedeckt wird. Evolutionäre Algorithmen sind zwar im Vergleich zum Bergsteigeransatz langsamer, lassen sich aber besser anpassen sowie mit problemspezifischen Heuristiken unterstützen. Weicker [Wei02] empfiehlt lokale Suchalgorithmen wie simulierte Abkühlung für gutmütige Probleme, wohingegen evolutionäre Algorithmen angewandt werden sollten, wenn viele lokale Optima bzw. eine zerklüftete Wertelandschaft dem Problem inhärent sind. Da es sich bei unserer Zielfunktion, wie oben schon dargelegt, um solch eine „schwierige“ Funktion handelt, wurde sich für die evolutionären Algorithmen zu Lösung entschieden, da durch ihren Mutationsoperator sichergestellt wird, dass die Wertelandschaft ausreichend abgedeckt ist. Eine ausführliche Darlegung des Konzepts der evolutionären Algorithmen findet sich in Abschnitt 3.3.2 dieses Kapitels.

Nicht für alle Teilszenarien war es notwendig, die diskutierten Lösungsansätze ins Feld zu führen. In den folgenden Abschnitten werden die konkreten Lösungsansätze für die Teilszenarien dargestellt. Dabei ist zu Beginn jedes Abschnitts noch einmal die Übersichtstabelle dargestellt, wobei der aktuell behandelte Fall jeweils grau hinterlegt ist.

Provider ungewichtet / Shares ungewichtet (Szenario 2-a)

Szenario 2: vordefinierte Anzahl von n Shares		
Fall	**Providerrisiken r_i**	**Sharerisiken s_j**
2-a	gleich	gleich
2-b	gewichtet	gleich
2-c	gleich	gewichtet
2-d	gewichtet	gewichtet

Dieser Fall ist einfach zu handhaben, da jede Zuweisung den gleichen Wert liefert. Somit macht es keinen Unterschied im Hinblick auf die Optimalität, welche Share welchem Provider zugeordnet wird. Es ist lediglich zu beachten, dass die Zahl der Shares gleichförmig auf den Server verteilt wird. Algorithmisch kann dies mittels eines Rundlauf-Verfahrens (engl. *Round-Robin*) geschehen, wobei Provider für Provider jeweils eine Share zugewiesen wird. Die Platzierung ist sogar in konstanter Zeit bezüglich der Shareanzahl und Provideranzahl zu erreichen: Die Rechenoperation n mod m liefert die Zahl der Provider, die eine Share mehr bekommen als die restlichen, was in konstanter Zeit $\mathcal{O}(1)$ geschieht. Bei diesem Teilszenario handelt es sich natürlich nicht mehr um ein NP-schweres Problem, da die Kostenmatrix derart degeneriert ist, dass es sich nicht mehr um ein Zuweisungsproblem handelt.

Provider gewichtet / Shares ungewichtet (Szenario 2-b)

Szenario 2: vordefinierte Anzahl von n Shares		
Fall	**Providerrisiken r_i**	**Sharesrisiken s_j**
2-a	gleich	gleich
2-b	gewichtet	gleich
2-c	gleich	gewichtet
2-d	gewichtet	gewichtet

Haben die Provider unterschiedliche gegebene Risikowerte r_i, sind die Gewichtungen der Shares j gleich und ist die Anzahl der Shares n sowie der Provider m vorgegeben, so handelt es sich um ein Zuweisungsproblem, wie wir es in Kapitel 2.3.3 kennengelernt haben. Im Gegensatz zu den bereits genannten Problemen degeneriert hier die Kostenmatrix C allerdings zu einem Kostenvektor, da alle Spalten gleich sind:

$$C = (c_{ij}) = \begin{pmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ c_{m1} & c_{m2} & \cdots & c_{mn} \end{pmatrix} = \begin{pmatrix} c_1 & c_1 & \cdots & c_1 \\ c_2 & c_2 & \cdots & c_2 \\ \vdots & \vdots & \vdots & \vdots \\ c_m & c_m & \cdots & c_m \end{pmatrix} = \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_m \end{pmatrix} = \begin{pmatrix} r_1 \\ r_2 \\ \vdots \\ r_m \end{pmatrix} \tag{3.24}$$

Beispiel Seien $m = 3$ Provider mit den Gewichtungen $r_1 = 2$, $r_2 = 5$ und $r_3 = 1$ gegeben und sollen $n = 7$ gleichgroße Shares auf sie verteilt werden, so würde die Kostenmatrix wie in Ausdruck 3.25 aussehen.

$$C = \begin{pmatrix} 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 5 & 5 & 5 & 5 & 5 & 5 & 5 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 2 \\ 5 \\ 1 \end{pmatrix} \tag{3.25}$$

Die Kosten c_{ij} entsprechen direkt den Risikowerten r_i der Provider; somit gilt $c_{ij} = c_i = r_i$, weil die Sharegewichte gleich und damit nicht relevant sind. Es macht keinen Unterschied, ob die Share j oder die Share l ($j \neq l$) auf ein und demselben Provider i platziert wird, der Zuwachs wäre der gleiche – nur zwischen den Providern gibt es Unterschiede. Dies bedeutet, dass die Wahl eines Providers nicht vom „Zeitpunkt“ abhängt, da eine spätere Zuweisung auf diesen Provider keinen schlechteren Wert liefert. Die Optimalität der Zuweisung von Share j hängt somit nur von den zuvor getroffenen Zuweisungen $X = (x_{ij})$ ab.

Wie lässt sich ein Algorithmus angeben, der diese Zielfunktion tatsächlich minimiert? Die Ausnutzung der speziellen Problemstruktur erlaubt es, einen Algorithmus zu konstruieren, der die Kostenmatrix zunächst einmal abscannt, d.h. jede Share betrachtet und aufgrund der vorherigen Ergebnisse eine Auswahl trifft. Dies führt aber unter gewissen Bedinungen, die später noch erklärt werden, zu einem nicht-optimalen Ergebnis. Es bleibt nichts anderes übrig, als am Ende des Algorithmus einige Schritte zurückzugehen, um die optimale Lösung in jedem Fall zu erreichen.
Das Vorgehen nimmt somit prinzipiell Anleihen an der Methode der dynamischen Programmierung, die gerade für Optimierungsprobleme oft eine passende Wahl darstellt – die gesuchte optimale Lösung kann aus den optimalen Lösungen der vorhergehenden Lösungen konstruiert werden. Im Laufe des Erstellungsprozesses konnte der erste Verdacht, dass sich das Problem mit reiner dynamischer Programmierung lösen ließe, nicht erhärten. Dies hätte eine sehr hohe Effizienz des Algorithmus bedeutet. Um auf die Evaluation in Kapitel 4 vorzugreifen, lässt sich sagen, dass bei Auslassen der Backtracking-Schritte sich dennoch eine geringe Fehlerrate einstellt.

Algorithmus 3.1 Exakter Algorithmus für Szenario 2-b

```
1:  for j = 1 to n do                                          // Äußere Schleife
2:      maxSum ← −∞
3:      max ← −∞
4:      min ← ∞
5:      for i = 1 to m do                                      // Innere Schleife
6:          curSum[i] ← c_i · Σ_{k=1}^{j−1} x_ik + c_i         // Zuwachs für jeden Provider Speichern
7:          curMin ← calcMiniMum
8:          curMax ← calcMaximum
9:          if curMin < min then
10:             min ← curMin
11:         end if
12:         if curMax > max then
13:             max ← curMax
14:         end if
15:         curDelta[i] = max − min                            // Deltas für diese Share speichern
16:     end for
                                                               // Jetzt Auswahlstrategie anwenden
17:     if multMinDeltas = false AND multSums = false then
18:         assignSmallestDelta(X)                             // Es gibt nur ein minimales Δ
19:     else if multMinDeltas = true AND multSums = false then
20:         assignSmallestSum(X)                               // Wenn es mehrere gleiche Δ gibt
21:     else if multMinDeltas = true AND multSums = true then
22:         assignBiggestBrick(X)    // Wenn es mehrere gleiche Δ und Summen gibt gibt
23:     end if
24: end for
25:
26: if BacktrackingEnabled = true then
27:     Backtrack(m-1)
28: end if
```

Ein Algorithmus, der das Problem exakt löst, wird in Algorithmus 3.1 dargestellt und funktioniert wie im Folgenden erklärt.

- Es werden alle Shares nacheinander überprüft, was durch die äußere Schleife mit der Zählvariablen j geschieht, die sich von Zeile 1 bis Zeile 24 erstreckt.
- Für jede Share werden so sukzessive die Platzierungen auf allen Providern überprüft, was mittels der verschachtelten, inneren Schleife (Zeilen 5 bis 16) mit der Zählvariablen i geschieht.

- In dieser Schleife wird zunächst jeder Zuwachs berechnet (Zeile 6), würde die momentan betrachtete Share j auf den i-ten Provider zugewiesen werden und ebenso, welcher Δ-Wert sich dadurch einstellen würde (Zeilen 7 bis 15). Diese Informationen werden in der Feld-Variablen `curSum[i]` (der Zuwachs) sowie `curDelta[i]` (das Δ) gespeichert. Die Variable `maxSum` hält die maximale (Zeilen-)Summe für den momentanen Schritt j, d.h. den maximalen Wert in diesem Schritt und die Variablen `max` und `min` das Maximum bzw. Minimum zur Berechnung des Δ-Wertes, würde Share j auf den Provider i zugewiesen.

Nachdem diese Informationen zusammengetragen wurden, kann die vorläufige Zuweisung vorgenommen werden; dazu ist eine Selektionsentscheidung nötig, die in den Zeilen 17 bis 23 abläuft:

- **minΔ**
  Es wird die Share j dem Provider zugewiesen, dessen Zuweisungskosten den Δ-Wert am wenigsten ansteigen lassen. Dies entspricht unserem Ziel, nämlich den Δ-Wert so wenig wie möglich ansteigen zu lassen.
- **minΣ**
  Gibt es mehrere gleiche Δ-Werte, so wird aufgrund des minimalsten Maximumwertes entschieden, wobei für jeden Provider überprüft wird, welchen Zuwachs seine Selektion ergeben würde und mit dem bisherigen Maximalwert verglichen wird – ergibt die Zuweisung der Share auf den momentan untersuchten Provider einen kleineren Zuwachs, so wird dieser ausgewählt und alle anderen werden deselektiert.
- **BiggestBrick**
  Sind neben den Δ-Werten auch die Summenwerte gleich, so wird das Element mit den größten Zuweisungskosten ausgewählt. Dies erweist sich für einige Sonderfälle als wichtig, denn im Hinblick auf eine spätere Entscheidung kann die Auswahl eines größeren Elementes besser sein. Da sowohl Δ- als auch Summenwerte gleich sind, macht es bezogen auf unser Optimierungsziel keinen Unterschied, ob wir ein kleineres oder größeres Element benutzen. Die Begründung erfolgt im Beweis.

Abschließend müssen noch $m-1$ Schritte Backtracking durchgeführt werden (Zeilen 26-28). Warum dies überhaupt und warum genau $m-1$ Schritte nötig sind, soll im folgenden Abschnitt zum Korrektheitsbeweis geklärt werden.

Die Laufzeit des exakten Algorithmus ist maßgeblich durch seine Backtracking-Schritte bestimmt. Hierbei müssen die $m-1$ letzten Shares noch auf die m Provider verteilt werden, was zu einer Laufzeitkomplexität wie in Formel (3.26) führt.

$$\mathcal{O}(m^{m-1}) \tag{3.26}$$

Die Laufzeit des Algorithmus wird somit nur bestimmt durch die Zahl der Provider.
Zwar ist seine Laufzeit somit immer noch exponentiell, jedoch ist sie wesentlich gedrückt im Vergleich zur naiven Methode, die in $\mathcal{O}(m^n)$ liegt, um n Shares auf m Provider zu verteilen. Dies folgt, weil per Annahme immer $n \geq m$ gilt.

Der Algorithmus kann aber auch ohne Backtracking als Heuristik mit einer gewissen Fehlerrate verwendet werden.
Dies kann einerseits wichtig sein, um für große Werte von m noch Ergebnisse zu bekommen, die durch Backtracking nicht effizient lösbar wären. Andererseits wird die vorgestellte Heuristik auch von anderen Verfahren genutzt, beispielsweise von evolutionären Algorithmen für den Fall 2-d, dass sowohl Shares wie auch Provider gewichtet sind, um Teile der Startpopulation zu erzeugen.
Daher soll eine Laufzeitabschätzung von Algorithmus 3.1 mit deaktiviertem Backtracking durchgeführt werden. Wie bereits erklärt, berechnet der Algorithmus die Platzierung jeder Share auf jedem Provider sukzessive und hält vorherige Ergebnisse in einer Tabelle, ähnlich der dynamischen Programmierung. Durch diesen Scan über die Kostenmatrix kommt der Algorithmus auf eine Laufzeit von

$$\mathcal{O}(n \cdot m) \tag{3.27}$$

weil für jede der n Shares alle m Providerrisiken berechnet werden müssen, um dann die optimalste in Bezug auf das Optimierungsziel auszuwählen.

Begründung der Korrektheit des Verfahrens

Im Folgenden soll ein Korrektheitsbeweis für den oben angegebenen Algorithmus erarbeitet werden. Da dieser an die Methode der dynamischen Programmierung angelehnt ist, muss aus der Struktur des Problems heraus erkennbar sein, in welchen Fällen er geeignet ist und wann es zu problematischen Fällen kommt, die mit Backtracking behandelt werden müssen. Die für dynamische Programmierung geforderte *Optimale Teilstruktur eines Problems* [CLRS04] bedeutet, dass einmal getroffene Entscheidungen optimal bleiben für nachfolgende. Eine optimale Lösung wird also sukzessive durch optimale Teillösungen konstruiert, ohne an den getroffenen Entscheidungen noch einmal etwas verändern zu müssen. Zwar kann für das vorliegende Problem von einer optimalen Teilstruktur im strengen Sinne keine Rede sein, denn in einigen Fällen müssen mittels Backtracking noch einige Schritte korrigiert werden, die aber streng begrenzt sind; jedoch hilft die Idee der dynamischen Programmierung bei der Begründung der Korrektheit.

Für den hier betrachteten Fall (2-b) muss zunächst eine Voraussetzung aufgestellt werden, die den Unterfall wiedergibt:

$$c_{ij} = c_{ij+1} \quad \forall j \in \{1, \cdots, n\} \tag{3.28}$$

Dies entspricht unserer Problemstruktur insofern, dass die Providerrisiken unterschiedlich, die Sharerisiken jedoch gleich gewichtet sind. Somit sind die Kosten für die Zuweisung aller Shares auf einen gewissen Provider gleich. Diesen Sachverhalt spiegelt Voraussetzung 3.28 wider. Nachdem die Voraussetzungen geklärt wurden, kann jetzt eine Behauptung aufgestellt werden:

Wenn die Voraussetzung (3.28) gilt, dann ist Algorithmus 3.1 korrekt.

Gemäß der im vorhergehenden Abschnitt dargelegten Selektionsstrategie wählt der Algorithmus immer das minimalste Δ aus. Im Folgenden wird die Selektionsstrategie genau betrachtet und begründet, wieso diese Selektionsstrategie korrekt ist:

- **Es gibt genau *ein* minimales Δ**:
  Die Zuweisung, für die sich ein minimales Δ einstellt, wird selektiert. Betrachtet man, wie sich die Δ-Werte mit laufendem n verhalten, ist feststellbar, dass sie von einem balancierten Zustand ausgehend evtl. in einen unbalancierten Zustand übergehen und von dort aus wieder in einen balancierten konvergieren usw.
  Dem Selektionskriterium des Algorithmus entsprechend wird nun in einem Schritt $l-1$ der kleinste Wert c_{1l-1} ausgewählt; da dieser das kleinste Δ liefert, geht man von gleichen Startniveaus aus. Im darauffolgenden Schritt l wird nun der optimale Wert bzgl. des minimalen Δ-Wertes selektiert, was unserem Ziel enspricht. Würde jedoch immer ein schlechterer Wert ausgewählt, könnte ein balancierter Zustand nicht mehr erreicht werden, denn die Δ-Werte würden sich stets weiter vom optimalen Wert entfernen, statt sich ihm anzunähern. Dies widerspricht unserem Ziel.
  Eine schematische Darstellung dieser Überlegungen findet sich in Abbildung 3.4, welche die korrekte Selektion darstellt: Im Schritt $l-1$ wird das Optimum für diesen Schritt ausgewählt, nämlich c_1, darauf aufbauend dann erneut c_1 in Schritt l. Die vom Algorithmus markierten Elemente sind mit * gekennzeichnet. In Abbildung 3.5 ist der Fall dargestellt, dass ein nicht-optimales Element ausgewählt wird, welches nicht mehr zu einem optimalen Zustand führen kann: Wird im Schritt $l-1$ ein nicht-optimaler Wert ausgewählt, so kann im Allgmeinen in Schritt l der Wert nicht mehr optimal werden. Die in diesem Fall markierten Elemente sind mit * gekennzeichnet.
  Demnach muss der Algorithmus grundsätzlich mit dieser Selektion vorgehen, von einigen Sonderfällen, die im Folgenden behandelt werden, abgesehen.
- **Es gibt *mehrere* minimale Δ-Werte und *genau einen* minimalen Summenwert**:
  Würde hier unspezifiziert sein, welches Element ausgewählt wird, kann es zu einer Ungleichverteilung kommen, die sich auf spätere Schritte auswirkt. Dies hat folgenden Grund: Gibt es mehrere gleich kleine Werte und einen Großen, so werden Shares

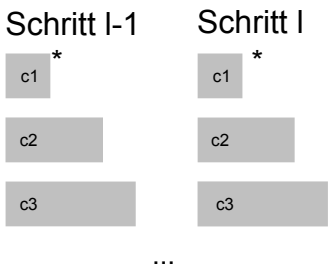

Abbildung 3.4: Rechtfertigung der minΔ-Auswahlstrategie. Korrekter Fall.

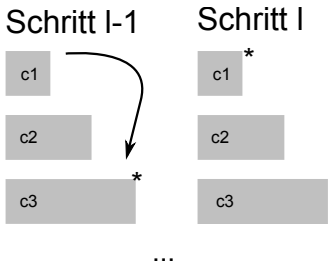

Abbildung 3.5: Rechtfertigung der minΔ-Auswahlstrategie. Widersprüchlicher Fall.

solange auf die Provider mit kleinen Risiken zugewiesen, bis diese „aufgefüllt sind". Gibt es aber keinen Rundlauf zwischen den Providern, so wird das Risiko nicht ausbalanciert, was zu einem schlechteren Δ in gewissen Schritten führen kann. Solch eine Situation ist in Abbildung 3.6 dargestellt. Diese Behandlung garantiert also eine Ausbalancierung, die unserer Problemstellung entspricht.

- **Es gibt *mehrere* minimale Δ-Werte und *mehrere* minimale Summenwerte**:
  Dann wird die Zuweisung durchgeführt, die die größten Kosten verursacht. Dies ist

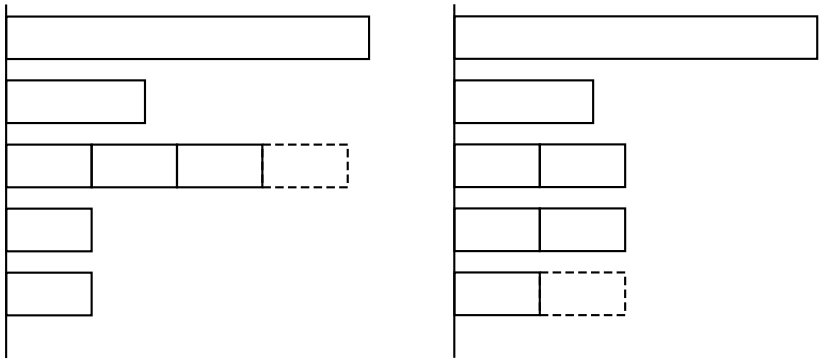

Abbildung 3.6: Motivation für das zweite Kriterium „minimales Maximum"für den Algorithmus von Szenario 2-b: Links im Bild ohne dieses Kriterium, rechts mit diesem.

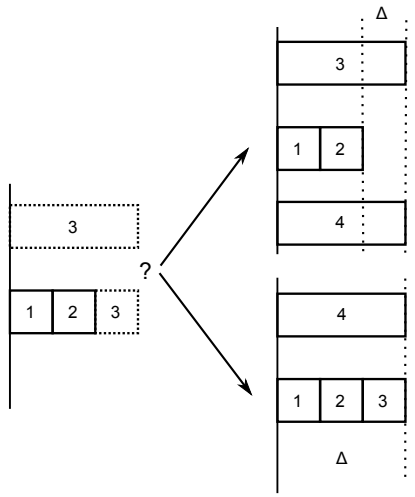

Abbildung 3.7: Rechtfertigung der Selektionsstrategie des Algorithmus für Szenario 2-b, die bei gleichen Δ- und Summenwerten das größere Element auswählt.

zu rechtfertigen, da trotzdem ein minimales Δ ausgewählt (und somit obige Selektionsstrategie nicht unterlaufen) wird. Im Hinblick auf weitere Selektionen ist es sogar nötig, wenn es zu der in Abbildung 3.7 dargestellten Situation kommt: Sind bei einer Selektionsentscheidung sowohl minimales Δ wie auch minimale Summe gleich, so gibt es zwei Möglichkeiten, wie eine gerade betrachtete Share zugewiesen wird.

Hier muss aber die größere Zuweisung getätigt werden, denn falls ein gleich großes Element folgt, war die Selektion des größeren geschickter, was in 3.7 dargestellt wird. Dieser Fehler würde sich ohne diese Strategie erst in einem weiteren Schritt wieder amortisieren. Dies stellt sicher, dass bei gleich großen Elementen die „kritischen“ großen Elemente früh zugewiesen werden.

Im Folgenden muss noch auf einen Sonderfall des Algorithmus eingegangen werden, denn es kann sein, dass durch die oben dargelegte Selektionsstrategie der Optimalwert nicht erreicht werden kann, sondern eine begrenzte Anzahl von Backtracking-Schritten notwendig ist.
Wann genau tritt nun eine Fehlstellung auf? Sie tritt dann auf, wenn die unteren Grenzen alle ungefähr den gleichen Wert haben, also von einem (annähernd) balancierten Zustand aus weitere Selektionsentscheidungen getroffen werden und dabei mehrere relativ große und relativ kleine Elemente vorkommen. Hauptsächlich tritt dies auf, wenn mehrere Elemente gleich groß sind, da dann in einigen Schritten gleiche Ober- und Untergrenzen bestehen.
Da immer nur eine Zuweisung für sich betrachtet wird, kann es zu dem Fall kommen, dass lange Zeit ein relativ kleines Element selektiert wird; nach einer gewissen Anzahl von Schritten kommt der Algorithmus zu dem Punkt, an dem retrospektiv eine andere Selektion

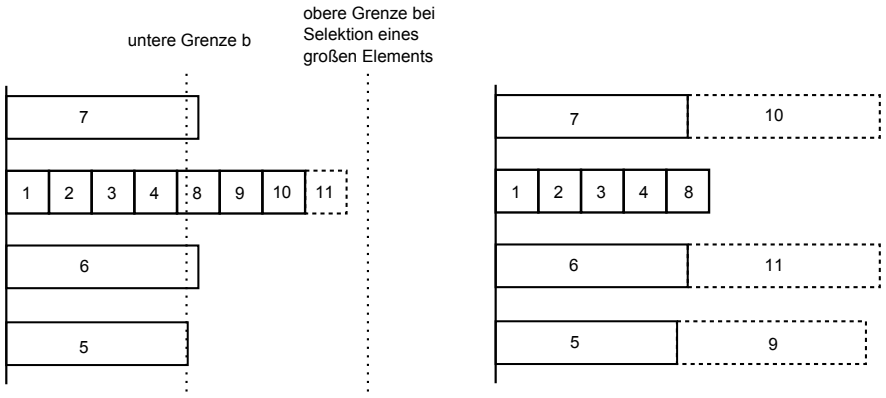

Abbildung 3.8: Problematischer Fall des Algorithmus für Szenario 2-b, schematische Darstellung.

in einem vorhergehenden Schritt besser gewesen wäre.
Solch eine Situation ist in Abbildung 3.8 dargestellt: Für Schritt 11 wäre es besser gewesen, ab Schritt 8 schon die Share auf den Provider mit höherem Risiko zuzuweisen, also ein lokales Non-optimum zu erreichen. Da der Algorithmus nicht vorausschauen kann, wird für diese konkrete Konstellation das Optimierungsziel nicht erreicht. Die großen Elemente „sperren“ sich gegenseitig: Steht man vor einer Selektionsentscheidung l und würde eine Share auf einen Provider mit hohem Risiko c_g zugewiesen werden und die bisherige Untergrenze ist b, so ergibt sich $\Delta = c_g - b$. Dies gilt ebenso für weitere Provider mit hohem Risiko c_g. Somit wird ein Provider mit kleinerem Risiko $c_k < c_g$ selektiert, was für sich betrachtet bzgl. dieses Schritts korrekt ist, im Hinblick auf weitere aber evtl. nicht. Hätten jedoch schon alle Provider, vor allem jene mit höherem Risiko c_g, eine Share zugewiesen bekommen, hätte sich hier ein kleineres Δ einstellen können.
Von der anderen Seite her betrachtet heißt dies für den problematischen Fall, dass wenn stattdessen alle anderen Provider mit hohen Risiken eine Share zugewiesen bekommen hätten, die Fehlstellung nicht möglich wäre. Dies bedeutet, dass die anderen $m - 1$ Provider jeweils eine Zuweisung bekommen sollten, um diesen Zustand zu erreichen.
Nun ist zu erkennen, dass $m - 1$ Schritte korrigiert werden müssen, indem bis zu $m - 1$ vorhergehende Zuweisungen auf die Provider verteilt werden müssen. Daraus folgt, dass nur $m - 1$ Backackting Schritte notwendig sind.

Der Fehler amortisiert sich über weitere Schritte wieder, nämlich genau dann, wenn der Algorithmus die geschilderte problematische Ausgangslage wieder zu einem balancierten Zustand führt. Diese Amortisierung findet spätestens dann statt, wenn alle m Provider befüllt wurden.

Eine geschlossene Formel?

Im Laufe des Erstellungsprozesses dieser Arbeit wurde die Frage aufgeworfen, ob sich dieser Fall nicht auch mittels einer geschlossenen Formel lösen ließe. Wird der Balancezustand als Ziel gesetzt, so würde sich daraus zunächst $r_1 \cdot n_1 = r_2 \cdot n_2 = \cdots = r_m \cdot n_m$ ergeben. Ähnlich wie in Szenario 1-b würde sich dann n_i als der Wert ergeben, der das spezifische Providerrisiko r_i im Hinblick auf das Gesamtsystem ausgleicht, wobei n fest vorgegebene Shares aufgeteilt werden müssen. Diesen Sachverhalt reflektiert $n_i = \frac{n}{T}$, wobei T hier für die Aufteilung steht, die noch näher spezifiziert werden muss. Indem $T = \sum_{k=1}^{m} \frac{1}{r_k}$ (Gesamtvertrauen, was die Kehrwerte widerspiegeln) gesetzt wird, folgt $n_i = \frac{n}{\sum_{k=1}^{m} \frac{1}{r_k}}$. Dann würden aber die Shares so aufgeteilt werden, als würde das Gesamtvertrauen zu gleichen Teilen zerlegt werden. Dieses muss noch mit dem spezifischen Risikowert r_i in Verbindung gesetzt werden, was den relativen Anteil ausmacht. Dies führt zu einer Formel:

$$n_i = \frac{n}{r_i \cdot \sum_{k=1}^{m} \frac{1}{r_k}} \tag{3.29}$$

Bei genauer Betrachtung der Formel zeigt sich jedoch, dass diese im Allgemeinen nicht korrekt ist. Als Ergebnis für n_i entstehen reelle Zahlen, also geteilte Shares. Die aufzuteilenden n Shares lassen sich aber nicht zerlegen.
Eine einheitliche Regel zur Rundung auf ganze Zahlen liefert in einigen Fällen eine Differenz zwischen vorgegebenem Wert n und $\sum_{i=1}^{m} n_i$, im Folgenden als n_{diff} bezeichnet. Es gilt nun, entweder n_{diff} Shares auf die Provider hinzuzufügen oder sie von ihnen wegzunehmen. Dabei muss die Optimalität der Zuweisung gewährleistet sein. Dies ist wieder ein Zuweisungsproblem mit einer entsprechenden exponentiellen Komplexität. In Experimenten konnte gezeigt werden, dass n_{diff} mit der Zahl der Provider m steigt, so dass es asymptotisch eine gleiche Laufzeitkomplexität wie oben vorgestellter Algorithmus hat und damit diesbzgl. keine Vorteile bietet.

Provider ungewichtet / Shares gewichtet (Szenario 2-c)

Szenario 2: vordefinierte Anzahl von n Shares		
Fall	**Providerrisiken r_i**	**Sharerisiken s_j**
2-a	gleich	gleich
2-b	gewichtet	gleich
2-c	gleich	gewichtet
2-d	gewichtet	gewichtet

In diesem Fall sind die Provider ungewichtet, wohingegen die Shares eine gewisse, fest vorgegebene Gewichtung besitzen.
Auch hier degeneriert – wie auch in Szenario 2-b in Abschnitt 3.3.2 – die Kostenmatrix zu einem Vektor, der nur noch aus Sharegewichtungen besteht, denn es gilt $c_{ij} = s_j$.

$$C = \begin{pmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ c_{m1} & c_{m2} & \cdots & c_{mn} \end{pmatrix} = \begin{pmatrix} c_1 & c_2 & \cdots & c_n \\ c_1 & c_2 & \cdots & c_n \\ \vdots & \vdots & \vdots & \vdots \\ c_1 & c_2 & \cdots & c_n \end{pmatrix} = \begin{pmatrix} c_1 & c_2 & \cdots & c_n \end{pmatrix} = \begin{pmatrix} s_1 & s_2 & \cdots & s_n \end{pmatrix} \tag{3.30}$$

Beispiel Seien $n = 6$ Shares mit den Gewichtungen $s_1 = 7$, $s_2 = 5$, $s_3 = 4$, $s_4 = 6$, $s_5 = 1$, $s_6 = 2$ gegeben und sollen diese auf $m = 3$ gleich große Provider verteilt werden, so würde die Kostenmatrix wie folgt aussehen:

$$C = \begin{pmatrix} 7 & 5 & 4 & 6 & 1 & 2 \\ 7 & 5 & 4 & 6 & 1 & 2 \\ 7 & 5 & 4 & 6 & 1 & 2 \end{pmatrix} = \begin{pmatrix} 7 & 5 & 4 & 6 & 1 & 2 \end{pmatrix} \tag{3.31}$$

Wie kann nun eine angemessene Verschlechterung der Privatsphäre erreicht werden? Diese soll über einen Balancezustand $R_1 = R_2 = \cdots = R_m$ zwischen den Providern hergestellt werden, $R_i = r_i \cdot S_i$ mit $S_i = \sum_{\text{j auf i zugeordnet}} s_j$ und $r = 1$. Dann ergibt sich für den Balancezustand Formel (3.32):

$$S_1 = S_2 = \cdots = S_m \tag{3.32}$$

Dieses Problem lässt sich wie ein Behälter- bzw. Rucksack-Problem interpretieren: Man setze die Shares als n Gewichte im Behälterproblem und gehe von einer festen Anzahl von m

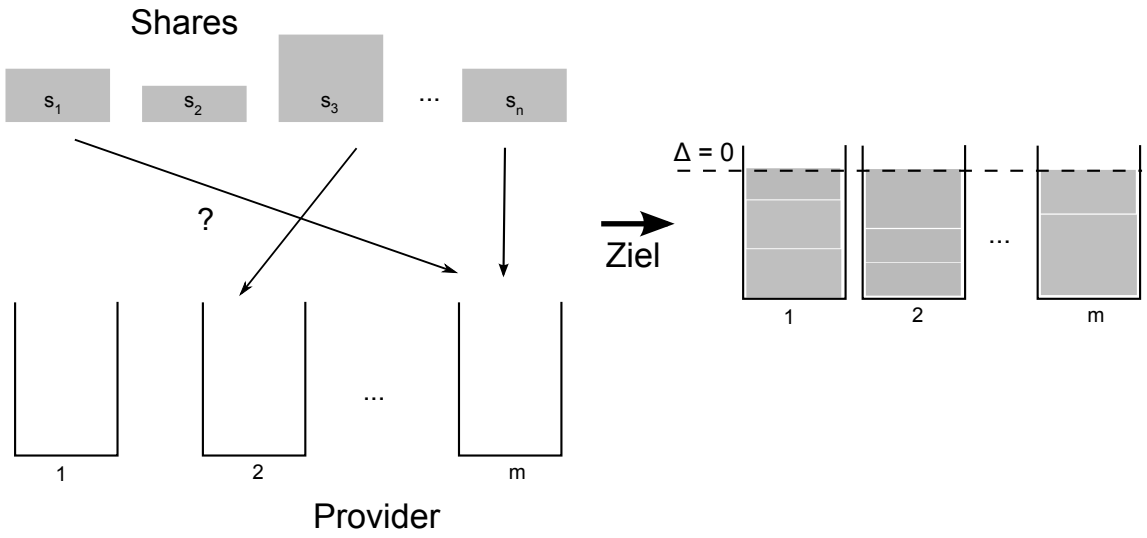

Abbildung 3.9: Ähnlichkeit von Szenario 2-c zum Behälterproblem.

Behältern aus, wobei jeder Behälter eine unbegrenzte Kapazität hat. Diese Interpretation ist in Abbildung 3.9 dargestellt.

Die hier dargestellte Ähnlichkeit zum Behälterproblem soll einerseits zeigen, dass es sich auch hier, obwohl die Form der Matrix degeneriert ist, um ein schweres kombinatorisches Optimierungsproblem handelt.
Andererseits könnten sich dafür Hinweise auf Lösungsmöglichkeiten ergeben, die an Algorithmen für Rucksackprobleme angelehnt sind. Jedoch wird es sich immer um Heuristiken handeln, wenn sie in polynomieller Zeit ablaufen. Darum wird, um auf den nächsten Abschnitt vorzugreifen, auch hier eine Anwendung von evolutionären Algorithmen angedacht. In diesem Kapitel wird allerdings eine Heuristik vorgeschlagen, um später zuverlässig einige Individuen in die Startpopulation einbringen zu können. Die Heuristik ist in Algorithmus 3.2 dargestellt und funktioniert folgendermaßen:
Da die Behälter zunächst keine Kapazitätsbeschränkung vorweisen, muss zunächst mit der Heuristik für Szenario 2-b eine obere Grenze *border* bestimmt werden (Zeile 1). Jetzt verfügen die Behälter nur noch über begrenzte Aufnahmefähigkeit, wie im eigentlichen Behälterproblem. Die Sortierung in Zeile 2 ist nicht unbedingt nötig, zeigte aber bessere Ergebnisse in der Praxis. Dann wird mittels einer Schleife (Zeilen 3 - 6) die Grenze sukzessive abgesenkt und die Shares neu auf die Provider verteilt mittels der bekannten First-Fit-Heuristik für das Behälterproblem. Indem die Grenze nach unten gedrückt wird, soll erreicht werden, dass sich die Gewichte besser austarieren und dass somit der Δ-Wert weiter sinkt. Die Schleife endet, wenn die Grenze zu niedrig gewählt wurde, so dass die Shares keinen Platz mehr haben oder wenn der Optimalwert von $\Delta = 0$ erreicht wurde.

Die Laufzeit richtet sich nach der Laufzeit des First-Fit-Algorithmus und der Sortierung. Die While-Schleife wird selten ausgeführt, wie sich in der Praxis zeigte. Die Heuristik hat

Algorithmus 3.2 Heuristik für Szenario 2-c

```
1: border ← HeuristicAlgorithm2b   // obere Grenze mit dem Algorithmus 2b bestimmen
2: sortDecreasing(C)                // Kosten absteigend sortieren
3: while Behälter groß genug do   // Veränderte First-Fit-Decreasing-Strategie anwenden
4:     border ← border − 1
5:     FF(C)                        // First-Fit-Algorithmus 2.1 anwenden
6: end while
```

somit eine polynomielle Zeitkomplexität und liegt aufgrund der Laufzeitkomplexität des FF-Algorithmus und der Sortierung in $\mathcal{O}(n \cdot \log(n))$ [VK08].

Der Algorithmus hat allerdings nur den Status einer Heuristik und ist somit fehlerbehaftet. Auch liegt seine Fehlerrate über der eines evolutionären Algorithmus, wie sich in der Evaluationssektion zeigen wird. Daher kann diese Heuristik für einen evolutionären Algorithmus genutzt werden, um Individuen in die Startpopulation einzubringen, was schon erwähnt wurde. Das Konzept des evolutionären Algorithmus wird für das nächste Teilszenario erörtert.

Provider gewichtet / Shares gewichtet (Szenario 2-d)

Szenario 2: vordefinierte Anzahl von n Shares		
Fall	**Providerrisiken r_i**	**Sharesrisiken s_j**
2-a	gleich	gleich
2-b	gewichtet	gleich
2-c	gleich	gewichtet
2-d	gewichtet	gewichtet

Sind sowohl die Providerrisiken unterschiedlich wie auch die Shares gewichtet, so hat diese Unterklasse im Gegensatz zu den anderen Problemausprägungen eine volle Kostenmatrix, wie in Matrix (3.33) dargestellt.

$$C = (c_{ij}) = \begin{pmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ c_{m1} & c_{m2} & \cdots & c_{mn} \end{pmatrix} \tag{3.33}$$

Beispiel Seien $m = 3$ Provider mit den Gewichtungen $r_1 = 2$, $r_2 = 5$ und $r_3 = 1$ gegeben. Es sollen $n = 7$ Shares auf die Provider verteilt werden, wobei jede Share wie in Tabelle 3.6 gewichtet ist, dann ergibt sich die Kostenmatrix wie in Matrix (3.35).

Share Nr.	1	2	3	4	5	6	7
Gewichtung	4	3	1	1	2	4	2

Tabelle 3.6: Beispielhafte Gewichtungen für Fall 2-d

Die Kostenmatrix wird konstruiert, indem die Risikowerte r_i des Providers i mit der Gewichtung s_j der entsprechenden Share multipliziert werden, also

$$c_{ij} = r_i \cdot s_j \tag{3.34}$$

$$C = \begin{pmatrix} 8 & 6 & 2 & 2 & 4 & 8 & 4 \\ 20 & 15 & 5 & 5 & 10 & 20 & 10 \\ 4 & 3 & 1 & 1 & 2 & 4 & 2 \end{pmatrix} \tag{3.35}$$

Wie erkennbar ist, handelt es sich bei den Spalten um Vielfache voneinander. Dies führt jedoch zu keiner Vereinfachung der Matrix oder des Verfahren, denn die Matrix degeneriert

dadurch nicht in ihrer Form. Wie in Abschnitt 3.3.2 formal bewiesen wurde, handelt es sich durch diese volle Kostenmatrix definitv um ein NP-schweres Problem.
Mit dieser Erkenntnis steht fest, dass für dieses Szenario abgewogen werden muss zwischen einem exakten Algorithmus, der eine exponentielle und damit ineffiziente Laufzeit besitzt und einem Näherungsalgorithmus, der dafür in annehmbarer Zeit arbeitet. Die vollständige Enumeration scheidet von vornherein aus. Branch-and-Bound kommt für dieses Szenario nur eingeschränkt in Betracht, was in der Diskussion in Abschnitt 3.3.2 erörtert wurde. Daher wurde ein evolutionärer Algorithmus entworfen, der sich an [Wei02] orientiert und an das Problem angepasst wurde. Das Konzept der evolutionären Algorithmen erschien aus folgenden Gründen erfolgsversprechend:

- Beim vorliegenden Optimierungsproblem handelt es sich – um mit Weicker [Wei02] zu sprechen – nicht um ein „gutmütiges“ Problem in Bezug auf die Wertelandschaft: Die Zielfunktion (Gleichung (3.17)) ist eine nicht-lineare Funktion, was bedeutet, dass lokale Optima nicht zu globalen führen müssen und das wirkliche globale Optimum evtl. in einem Bereich liegt, wo es in keinster Weise zu erwarten war. Des Weiteren zirkulieren die Werte mit variabler Zahl der Zuweisungen um den theoretischen Optimalwert 0, es lässt sich nicht erkennen, dass bei steigender Zahl von zugewiesenen Shares sich der Optimalwert irgendwie stabilisiert.
  Der Mutationsoperator der evolutionären Algorithmen stellt jedoch sicher, dass ein großes Einzugsgebiet abgesucht wird und dass wieder aus lokalen Schein-Optima entkommen werden kann.

- Entscheidend für Geschwindigkeit wie Güte eines evolutionären Algorithmus ist die Fitnessfunktion [Wei02]: Ist die Bewertung schnell, so ist es auch der Gesamtalgorithmus. Der Selektionsdruck wird durch die Strenge der Funktion festgelegt, hat also Einfluss darauf, wie schnell oder wie gut ein Optimum erreicht werden kann. Unter Selektionsdruck wird verstanden, wie schnell ein Algorithmus gegen ein Optimum konvergiert, wobei gilt, dass je höher der Selektionsdruck, desto höher ist die Wahrscheinlichkeit, dass es sich nur um ein lokales Optimum handelt.
  Unserem Problem ist eine einfache Bewertung zu eigen: Je näher der Wert bei 0 ist, desto näher ist man am gesuchten globalen Optimum. Die Bewertung ist daher schnell und der Selektionsdruck anpassbar.

- Die Kombinierbarkeit mit anderen, problemspezifischen Heuristiken ist möglich [Wei02], was zu wesentlichen Verbesserungen der Güte der Ergebnisse führt. So kann die für Szenario 3.3.2 erarbeitete Heuristik in Betracht gezogen werden.

Nach diesen allgemeinen Argumenten und der Bezugnahme auf unsere Problemstellung soll jetzt zur Erläuterung der konkreten konzeptuellen Entscheidungen gekommen werden und schließlich der Algorithmus vorgestellt werden. Die natürlichsprachliche Erklärung des Algorithmus findet sich im folgenden Abschnitt, wobei der Pseudocode in Algorithmus 3.3 dargestellt ist.

Zur Erstellung eines Algorithmus für unser Problem muss der Grundzyklus der evolutionären Algorithmen implementiert und angepasst werden.
Zunächst wird für das Problem eine gewisste Codierung verwendet, die sich von der oben benutzten Matrizendarstellung unterscheidet, aber leicht transformiert werden kann.

$$X_{GA} = (x_{i1}, x_{i2}, \cdots, x_{in}) \tag{3.36}$$

Die Platzierung von Share j auf Provider i kann mit Formel (3.36) wie folgt repräsentiert werden: Der Wert der j-ten Komponente x_{ij} des Vektors X_{GA} entspricht dem Provider i, auf dem die Share lokalisiert ist.
Diese Darstellung ist vorteilhaft, da sie genau einem Genom entspricht, an dem einzelne Änderungen wie Mutationen leicht durchgeführt werden können. Nun folgt der evolutionäre Grundzyklus, der auf diese Codierung zurückgreift:

- **Initialisierung** Die erste Generation umfasst 10 Individuen und wird teilweise randomisiert, teilweise durch eine Heuristik erzeugt. Dafür wurde die Heuristik aus Abschnitt 3.3.2 folgendermaßen angepasst: Es wird nicht mehr die Differenz, sondern das Maximum minimiert. Dies erwies sich experimentell als besser im Hinblick auf das Optimierungsziel.
  3 Individuen der 10 initialen Individuen wurden auf diese Weise, der Rest komplett randomisiert erzeugt.

- **Rekombination** Hier wird die potentielle neue Generation gebildet. Aus den 10 Individuen der vorhergehenden (oder im ersten Zyklus der Startpopulation) werden 40 neue durch Rekombination erzeugt.
  Die Rekombinationsoperation wird mit einer Wahrscheinlichkeit von 0.5 als uniformer Crossover auf zwei zufällig gewählten Individuen ausgeführt. Durch den uniformen Crossover sollen Individuen zu gleichen Teilen gemischt werden. Wird kein Crossover durchgeführt, so wird ein Indivdiduum zufällig selektiert und unverändert in die neue Generation übernommen.
  Sollte durch den Crossover eine ungültige Belegung in Bezug auf die dem Problem inhärenten Nebenbedingungen (siehe (3.18) und (3.19)) erzeugt werden, wird diese verworfen und ein zufälliges Indivduum in die neue Generation übernommen. Auf diese Weise wird von Anfang an vermieden, dass ungültige Platzierungen erzeugt werden – sie werden frühzeitig eliminiert, so dass sie sich im Evolutionsprozess nicht durchsetzen.

- **Mutation** Da der Mutationsoperator von hoher Bedeutung für die Abdeckung des ganzen Suchraumes ist, wird dieser in jedem Fall nach der Rekombination angewandt. Realisiert ist der Mutationsoperator als Vertauschung zweier Platzierungen: Der Ort zweier zufällig bestimmer Shares wird vertauscht. Diese Vertauschungsoperation bietet den Vorteil, dass die Platzierung gültig bleibt, da auf den entsprechenden Providern

nur die jeweils andere Share platziert wird. Somit hat nach wie vor jeder Provider eine Share.

- **Bewertung** Die 40 temporär erzeugten Individuen werden bewertet. Die Bewertung lässt sich berechnet, indem der Maximalwert und der Minimalwert der Zuweisungskosten auf einem Provider bestimmt wird. Dazu sind lediglich ein Scan über dem Genom X_{GA} nötig, der die Platzierungen notiert und die Werte speichert. Dann wird die Differenz gebildet. Je näher dieser Wert bei 0 liegt, desto besser ist ein Individium im Hinblick auf unser Optimierungsziel *minimiere* Δ.
- **Selektion** In diesem Schritt werden die besten 10 der 40 erzeugten Individuen ausgewählt. Dann beginnt der Zyklus entweder von Neuem, oder die Abbruchbedigung ist erfüllt und der Algorithmus endet.
- **Abbruchkriterium** Der Algorithmus endet, wenn entweder eine vorgegebene Zahl von Generationen durchlaufen wurde oder $\Delta = 0$ gilt.
  Die Zahl der Generationen wurde auf 2000 gesetzt. Zwar findet der Algorithmus meist in weitaus weniger[5] Schritten ein Optimum – der Wert ergibt sich daraus, dass dem Algorithmus Zeit gelassen werden soll, um sich auf ein Optimum „einzupendeln“. Vor allem bei stark unterschiedlichen Gewichten zeigte sich dies in der Praxis als notwendig.

Der hier natürlichsprachlich dargelegte Algorithmus wird nun als Pseudocode in Algorithmus 3.3 angegeben.

Die Laufzeit von Algorithmus 3.3 ist maßgeblich bestimmt durch die Operationen, die bei evolutionären Algorithmen auftreten. Diese operieren auf dem Genom X_{GA}, welches als Vektor bzw. Array der Länge n (Anzahl der Shares) codiert ist. Da 2000 Generationen erzeugt werden und die Kombinationsoperation, die selbst n Schritte benötigt, 40 mal ausgeführt wird, entspricht dies also $80000 \cdot n$ Schritten. Weiterhin kommen für die Evaluation der nochmal $2000 * 40 \cdot n$ Schritte hinzu, was in einer polynomiellen Komplexitätsklasse liegt. Jedoch werden mittels der Heuristik für Szenario 2-b einige Individuen erzeugt, was die Laufzeit auf die Komplexität dieser Heuristik anhebt. Somit gilt, dass die Komplexität des evolutionären Algorithmus in $\mathcal{O}(n \cdot m)$ liegt.

[5] Wie sich in der Evaluation in Kapitel 4 zeigen wird, genügen schon 200 Schritte, um das erste Mal ein Optimum zu finden.

Algorithmus 3.3 Evolutionärer Algorithmus für Szenario 2-d

```
1: t ← 0
2: population[1,2,3,4,6,7,8] ← RandomPlacement
3: population[0,5,9] ← HeuristicPlacement                          // Initialisierung
4:
5: while t < 2000 AND Δ > 0 do                                     // Evolutionärer Grundzyklus
6:     for p = 1 to 40 do                                          // Vervielfältigung der Generation auf 40 Ind.
7:
8:         ind1 ← RandomInd
9:         ind2 ← RandomInd
10:        u ← RandomBoolean
11:        if u = true then
12:            populationTemp[p] ← UniformCrossover(ind1, ind2)    // Uniformer Crossover
13:            if Infeasible(populationTemp[p]) then               // wenn ungültig
14:                populationTemp[p] ← ind2
15:            end if
16:        else
17:            populationTemp[p] ← ind1
18:        end if
19:        populationTemp[p] ← Mutation(populationTemp[p])         // Mutation
20:    end for
21:
22:    Eval(populationTemp[])                                      // Evaluation
23:    population[] ← Select10Best(populationTemp[])               // Selektion
24:    Δ ← BestΔ(population[])
25:
26:    t ← t + 1
27: end while
```

3.3.3 Tabellarische Zusammenfassung der Lösungskonzepte

Da es sich um insgesamt 8 verschiedene Teilszenarien handelt und jeweils mindestens ein Lösungskonzept vorgeschlagen wurde, findet sich in Tabelle 3.7 ein Überblick, in welchem Fall welches Szenario eingesetzt wurde. In Szenario 1 wurde von einer wählbaren Größe von Shares und somit auch von einer freien Wählbarkeit ihrer Gewichtungen ausgegangen. Bei Szenario 2 wurde die Zahl der Shares und somit auch deren Gewichtungen als fest vorgegeben angenommen, mit denen eine noch möglichst optimale Zuweisung durchgeführt werden sollte.

	Szenario 1	**Szenario 2**		
	Formel	exakt	Heuristik	EA
a	✓	✓	-	-
b	✓	✓	✓	-
c	✓	-	✓	✓
d	✓	-	-	✓

Tabelle 3.7: Übersicht über die behandelten Lösungsansätze.

In Tabelle 3.8 findet sich ein Überblick über die Laufzeitkomplexitäten, welche für die jeweiligen Konzepte abgeschätzt wurden. Zum Vergleich ist jeweils noch die naive Methode eingetragen.

	Szenario 1	**Szenario 2**			
	Formel	naiv	exakt	Heuristik	EA
a	$\mathcal{O}(1)$	$\mathcal{O}(m^n)$	$\mathcal{O}(1)$	-	-
b	$\mathcal{O}(m)$	$\mathcal{O}(m^n)$	$\mathcal{O}(m^{m-1})$	$\mathcal{O}(n \cdot m)$	-
c	$\mathcal{O}(1)$	$\mathcal{O}(m^n)$	-	$\mathcal{O}(n \cdot \log(n))$	$\mathcal{O}(n \cdot m)$
d	$\mathcal{O}(m)$	$\mathcal{O}(m^n)$	-	-	$\mathcal{O}(n \cdot m)$

Tabelle 3.8: Übersicht über die Laufzeitkomplexitäten, wobei m die Zahl der Provider und n die Zahl der Shares darstellt.

3.4 Implementierung

Im Hinblick auf die Evaluation in Kapitel 4 wurden die vorgeschlagenen Konzepte für Szenario 2 in der Programmiersprache *Java* implementiert.
Dazu wurde eine Basisklasse erstellt, die allgemeine Methoden zum formalisierten Zuweisungsproblem BaGAP bereitstellt, wie z.B. die Überprüfung von Zuweisungsmatrizen auf Gültigkeit bzgl. der Nebenbedingungen, die Ausgabe von Δ-Werten, Providerrisken usw. auf der Standardausgabe sowie Umwandlungsmethoden zwischen verschiedenen Darstellungsformen (Zuweisungsmatrix und Vektor).
Dann wurden die konkreten Verfahren 2-b, 2-c und 2-d in eigenen Klassen implementiert, die jeweils von der Basisklasse abgeleitet wurden und so alle Methoden usw. erbten.
Für Szenario 2-d zeigte sich ein enger Verzahnungsprozess zwischen Implementierung und Evaluation, da für evolutionäre Algorithmen Erkenntnisse aus der Empirie von hoher Bedeutung sind. Für dieses Szenario wurden auch unterschiedliche Startheuristiken, die von der Heuristik für Szenario 2-b abgeleitet wurden, implementiert, getestet und abgewogen.

4 Evaluation

In diesem Kapitel sollen die in Kapitel 3 erarbeiteten Konzepte experimentell evaluiert werden um zu überprüfen, ob sie den spezifizierten Anforderungen entsprechen.
Für Szenario 1, bei dem die Anzahl der Shares frei gewählt werden kann, war keine automatisierte Evaluation nötig. Die Optimalität der Platzierung ergibt sich hierbei direkt durch die Justierung der Anzahl und Gewichtung der Shares, die möglichst geschickt zu wählen sind, so dass die Sicherheitsrisiken der Server ausbalanciert werden. Dies ist im entsprechenden Abschnitt 3.3.1 für jedes Teilszenraio hergeleitet und erörtert. Die Berechnungszeiten hierfür liegen höchstens in einer linearen Komplexitätsklasse mit der Zahl der Provider.
In Szenario 2 ist die Anzahl der Shares als fest anzunehmen und ein Zuweisungsproblem zu lösen. Dazu wurden die in Abschnitt 3.3.2 vorgestellten Konzepte evaluiert. Es sind jeweils noch einmal die Übersichtstabellen eingeblendet, die darstellen, welcher Unterfall gerade betrachtet wird.

4.1 Evaluationsbedingungen

Zunächst folgen einige Ausführungen zu den Evaluationsbedignungen. Es wurden Korrektheit und Geschwindigket getesten, außerdem für die evolutionären Algorithmen auch deren Konvergenzverhalten.

Korrektheit

Da die Hauptanforderung aus Kapitel 3.2 die Erreichung eines Gleichgewichts war, muss die Korrektheit der vorgestellten Algorithmen evaluiert werden. Eine Platzierung ist optimal, wenn sie den minimalen Wert von Δ liefert.
Dazu wurden jeweils 1000 zufällige Probleminstanzen generiert, mit einem der erarbeiteten Konzepte gelöst und dann mit der naiven Lösung verglichen.
Fehlerrate wird hier als Rate definiert, in wie vielen Fällen der Algorithmus nicht den optimalen Wert findet.
Die Abweichung vom Optimalwert konnte nicht sinnvoll gemessen werden, da es sich hierbei immer um eine relative Rate handelt. Aufgrund des relativen Charakters der Risiko- und Ergebniswerte wäre eine Abweichungsmessung wenig aussagekräftig.

Geschwindigkeit

Darüber hinaus wurden Messungen über das Zeitverhalten der Algorithmen durchgeführt, da eine gute Laufzeitkomplexität eine weitere Anforderung darstellt. Die Messungen wurden dabei auf einem *Intel Core2 Quad*-System mit *2,4GHz* innerhalb von *Java* durchgeführt.
Dazu wurden exakter Algorithmus, Heuristik und evolutionärer Algorithmus evaluiert. Für jeden wurde die Zahl der Shares n variiert und für jede Variation eine Probleminstanz zufällig erzeugt und berechnet. Diese Berechnung wurde 1000 mal durchgeführt, damit ein zuverlässiger Mittelwert gebildet werden konnte, um simulationsbedingte Schwankungen auszugleichen. Da sich die Berechnungsdauern teilweise im Bereich von Nanosekunden aufhielten, war diese Wiederholung notwendig.

Konvergenz

Die Untersuchung des Konvergenzverhaltens lieferte auch wichtige Aufschlüsse über das Verhalten der evolutionären Algorithmen, welche zur Feinabstimmung dieser Verfahren unabdingbar sind. Hierbei wurde die Fitness, die dem Δ-Wert entspricht, gegenüber der Zahl der Generationen aufgetragen.
Dafür wurde zunächt eine spezielle Probleminstanz exakt berechnet. Dann wurde diese Probleminstanz mit dem evolutionären Algorithmus berechnet.
Dies wurde für zwei Fälle durchgeführt, die sich durch eine unterschiedliche Streuung der Risikowerte unterschieden. In einem Fall lagen die Risikowerte nah beieinander, es wurde also von einer relativen Balanciertheit der vorgegebenen Risikowerte ausgegangen. Im anderen Fall waren diese Risikowerte weit voneinander entfernt.

4.2 Szenario 2-b

Szenario 2: vordefinierte Anzahl von n Shares		
Fall	**Providerrisiken r_i**	**Sharerisiken s_j**
2-a	gleich	gleich
2-b	gewichtet	gleich
2-c	gleich	gewichtet
2-d	gewichtet	gewichtet

4.2.1 Korrektheit

Exakter Algorithmus Da für den exakten Algorithmus 3.1 seine Korrektheit in Teil 3.3.2 dieser Arbeit bewiesen wurde, ist eine Evaluation der Korrektheit nicht mehr notwendig.

Heuristik Allerdings ist es möglich, die dem Algorithmus zugrunde liegenden Backtracking-Schritte abzuschalten, so dass dieser nur noch den Status einer Heuristik hat. Dies ist einerseits von Bedeutung, weil diese Heuristik in einer Abwandlung später (im Fall 2-d) für die evolutionären Algorithmen als Startheuristik genutzt wird, andererseits aber auch, weil die Backtracking-Schritte für eine große Zahl von Providern den Algorithmus ineffizent werden lassen, so dass in diesen Fällen auf die Heuristik zurückgegriffen werden muss.
Die Fehlerrate ist nun in Schaubild dargestellt 4.1.

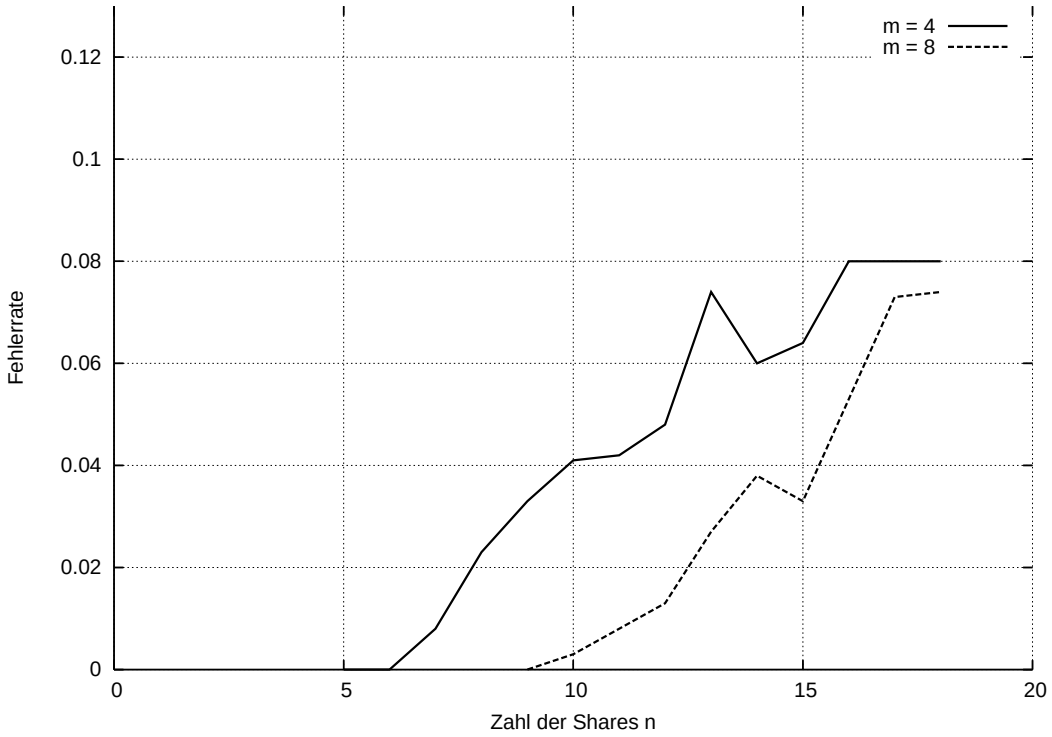

Abbildung 4.1: Szenario 2-b: Fehlerrate der Heuristik (Deaktiviertes Backtracking) für $m = 4$ und $m = 8$ Provider.

Die Fehlerrate wurde für $m = 4$ und $m = 8$ Provider bei variabler Anzahl der Shares gemessen. Dabei ist festzuhalten, dass die Fehlerrate zunächst steigt und sich dann wahrscheinlich auf einem Niveau von 8% einpendelt. Dies konnte jedoch nicht weiter überprüft werden, weil die Vergleiche mit der naiven Methode zu lange dauerten.

4.2.2 Geschwindigkeit

Exakter Algorithmus Für den Algorithmus 2-b wurden auch Messungen zur Laufzeit durchgeführt. Das Laufzeitverhalten für den exakten Algorithmus mit aktiviertem Backtracking ist für $m = 4$ und für $m = 8$ Provider in Schaubild 4.2 dargestellt, wobei die Zahl der Shares variabel ist. Im Vergleich der Graphen ist erkennbar, dass die Geschwindigkeit

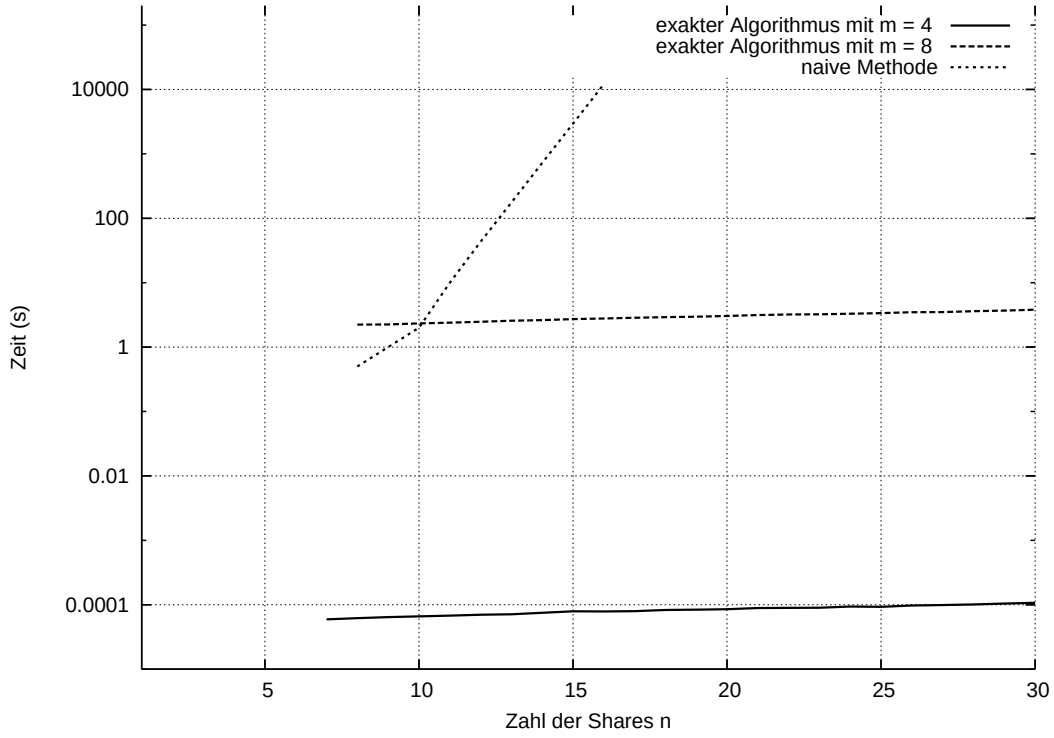

Abbildung 4.2: Szenario 2-b: Geschwindigkeit des exakten Algorithmus mit $m = 4$ und $m = 8$ Providern. Naive Methode zum Vergleich. Logarithmische Skalierung.

stark von der Zahl der Provider bestimmt ist. Dies entspricht der theoretischen Überlegung, wonach Backtracking für die letzten $m - 1$ Schritte durchgeführt werden muss. Für $m = 4$ Provider liegt die Laufzeit noch im Bereich von Mikrosekunden, für $m = 8$ schon im Bereich von Sekunden.
Dies legt den Schluss nahe, dass dieses Verfahren nur für eine Zahl von Providern geeignet ist, die unter 10 liegt, da man bereits mit 8 Providern im Sekundenbereich ist. Jedoch ist das Verhalten des Algorithmus relativ stabil gegenüber zunehmender Zahl von Shares, was

wiederum den theoretischen Überlegungen entspricht. Im Schaubild 4.2 wurde auch noch die Laufzeit eines naiven Aufzählungsverfahrens dargestellt, die exponentiell mit der Zahl der Shares steigt. Dieser Aufzählungsvariante ist der exakte Algorithmus in jedem Fall vorzuziehen.

Heuristik Schließlich wurde noch die Laufzeit der Heuristik evaluiert, was für verschiedene Anzahlen von Providern Provider in Schaubild 4.3 dargestellt ist. Das Schaubild zeigt, dass die Laufzeit mit zunehmender Zahl der Shares linear ansteigt. Den Faktor des linearen Anstiegs liefert dabei die Zahl der Provider: Je größer die Zahl der Provider, desto stärker (aber immer noch linear) steigt die Laufzeit mit der Zahl der Shares an. Allerdings liegt die Laufzeit in jedem Fall im Bereich von Mikrosekunden. Dies entspricht der theoretischen Abschätzung, wonach die Laufzeit der Heuristik in $\mathcal{O}(n \cdot m)$ liegt.

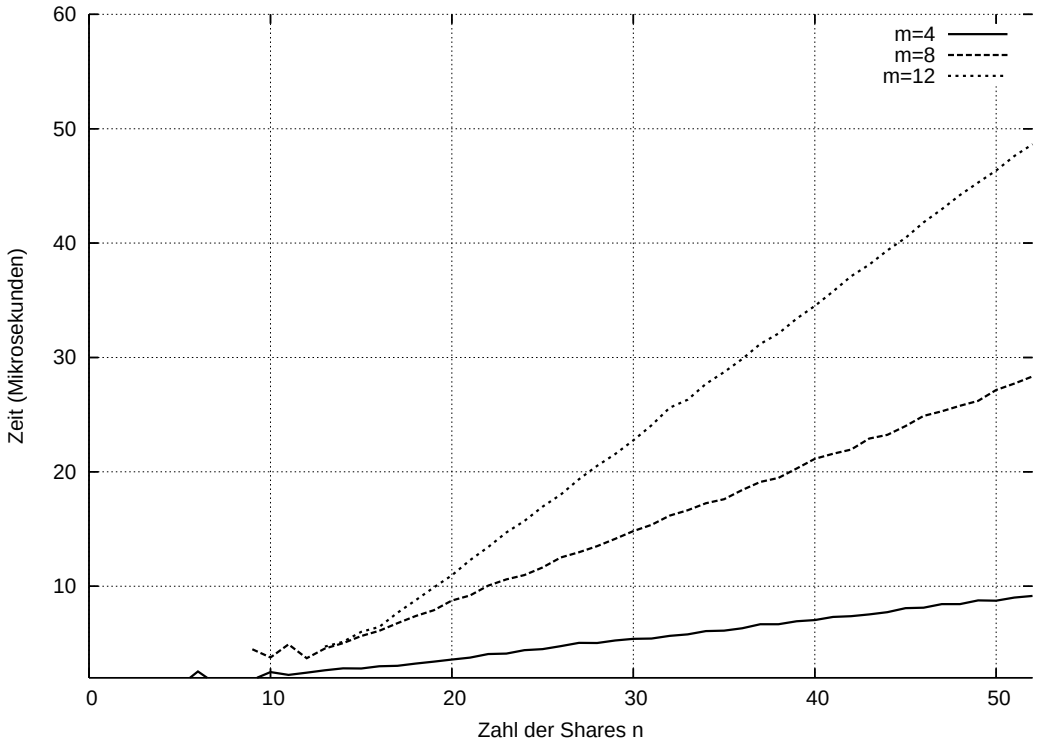

Abbildung 4.3: Szenario 2-b: Geschwindigkeit der Heuristik für verschiedene Anzahlen von Providern m.

4.3 Szenario 2-c

Szenario 2: vordefinierte Anzahl von n Shares		
Fall	**Providerrisiken r_i**	**Sharesrisiken s_j**
2-a	gleich	gleich
2-b	gewichtet	gleich
2-c	gleich	gewichtet
2-d	gewichtet	gewichtet

Heuristik Die in Szenario 2-c entwickelte Heuristik wurde implementiert und einer Untersuchung unterzogen, wobei die Messreihen ergaben, dass die Fehlerrate der Heuristik unter 10% lagen.

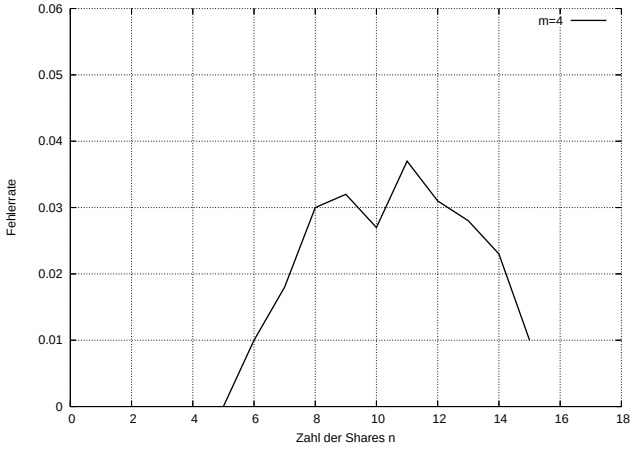

Abbildung 4.4: Szenario 2-c: Fehlerrate bei Anwendung eines evolutionären Algorithmus bei $m = 4$ Providern.

Evolutionärer Algorithmus Wurde ein evolutionärer Algorithmus für dieses Unterszenario verwendet, so zeigt sich die Fehlerrate wie in Schaubild 4.4 dargestellt. Sie liegt für die testbaren Problemgrößen immer unterhalb von 4% und ist damit der reinen Heuristik vorzuziehen.
Die Ergebnisse der Laufzeitmessung entsprechen den Ergebnissen aus dem nächsten Abschnitt, da der evolutionäre Algorithmus in gleicher Weise auch auf Szenario 2-c angewendet wird.

4.4 Szenario 2-d

Szenario 2: vordefinierte Anzahl von n Shares		
Fall	**Providerrisiken r_i**	**Sharerisiken s_j**
2-a	gleich	gleich
2-b	gewichtet	gleich
2-c	gleich	gewichtet
2-d	gewichtet	gewichtet

Für den Lösungsantz von Szenario 2-d, bei dem sowohl Shares als auch Provider gewichtet sind, wurde die Methode der evolutionären Algorithmen gewählt. Hier lassen sich neben der Korrektheit und Geschwindigkeit noch die Konvergenz des Algorithmus sowie ein Vergleich verschiedener Startheuristiken evaluieren.

4.4.1 Korrektheit

Schaubild 4.5 zeigt die Fehlerrate bei laufender Zahl von Shares und bei unterschiedlicher, aber fester Zahl von Providern. Dabei lässt sich feststellen, dass es immer bei $n_{hoch} = 2 \cdot m$ einen Hochpunkt der Fehlerrate gibt und dann für steigende Zahl von n die Fehlerrate gegen 0 strebt. Dies konnte leider nur für kleine Werte von m getestet werden, da die naive Methode den verwendeten PC wegen der Komplexität der Enumeration (exponentiell mit Eingabelänge!) schnell an seine Leistungsgrenze trieb. Die Tendenz sollte jedoch klar erkennbar sein.

4.4.2 Geschwindigkeit

Schaubild 4.6 zeigt das Zeitverhalten des evolutionären Algorithmus bei verschiedenen Anzahlen von Providern und laufender Zahl von Shares. Dabei zeigt sich, dass die Geschwindigkeit im Bereich von Millisekunden und die Laufzeit mit der Zahl der Provider schwach linear ansteigt. Dies entspricht den theoretischen Überlegungen aus Abschnitt 3.3.2, wonach der evolutionäre Algorithmus in der linearen Komplexitätsklasse $\mathcal{O}(n)$ liegt. Die Zahl der Provider verschiebt lediglich die Zeit nach oben – ansonsten ist das Verfahren, wie erwartet stabil gegenüber der Zahl der Provider.
Das Verfahren eignet sich somit für die Aufteilung der Shares auf die Provider, da ein Bereich von Millisekunden zur Erzeugung eines Platzierungsplans einen vertretbaren Aufwand darstellt.

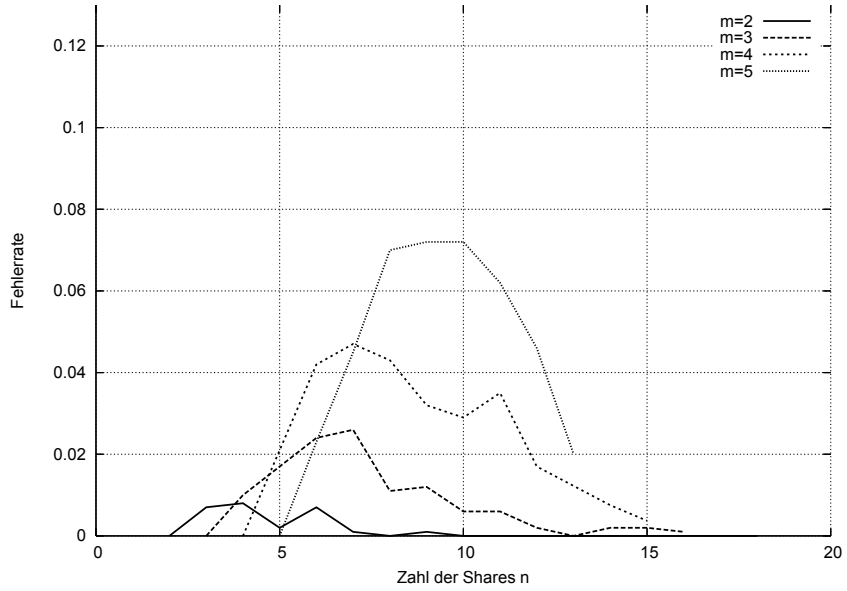

Abbildung 4.5: Szenario 2-d: Fehlerrate des evolutionären Algorithmus für verschiedene Anzahlen von Providern m.

4.4.3 Konvergenz

Für einen evolutionären Algorithmus ist das Konvergenzverhalten über die Zahl der Generationen von hoher Bedeutung: Wie schnell konvergiert ein evolutionärer Algorithmus zum Optimalwert hin? Für dieses Experiment wurden zwei verschiedene Risikoverteilungen getestet, wobei jeweils von $m = 5$ Providern und $n = 12$ Shares ausgegangen wurde:
Im ersten Fall ist die Streuung des Risikos der Provider gering, im zweiten Fall ist sie groß. Dann wurde die Fitness der bewerteten Generationen gegen die Generationen dargestellt. Der Fall mit geringer Streuung der Risikorate wird in Schaubild 4.7 dargestellt. Hier wird der Optimalwert bereits ca. nach der 25. Generation erreicht und pendelt sich dann ein, wobei die durch die Mutationen bedingten schlechteren Generationen nahe beim Optimalwert liegen, welcher in diesem Fall 50 beträgt und im Schaubild eingezeichnet ist.
Der zweite Fall, bei dem die Streuung der Risikowerte der Probleminstanz groß ist, wird in Schaubild 4.8 dargestellt. Hierbei ist zu erkennen, dass es fast 100 Generationen benötigt, bis der Optimalwert zum ersten Mal erreicht wird. Der Optimalwert liegt für diese Probleminstanz bei 40. Auch weichen die schlechteren, durch Mutation erzeugten späteren Populationen stärker vom Optimalwert ab.
Diese Experimente führten dazu, dass die Zahl der Generationen bzw. Iterationen es evolu-

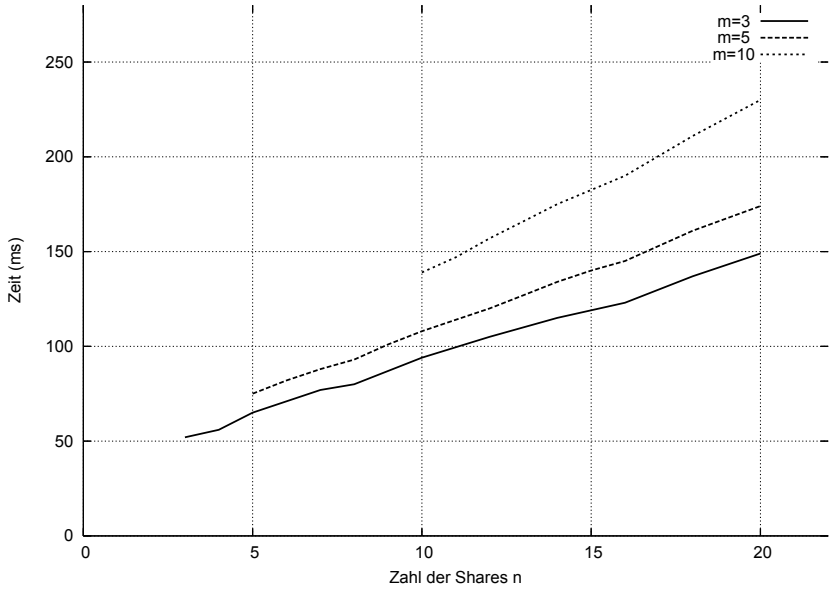

Abbildung 4.6: Szenario 2-d: Laufzeitverhalten des evolutionären Algorithmus für verschiedene Anzahlen von Providern m.

tionären Algorithmus auf 2000 festgelegt wurde, damit bei stark unterschiedlichen Größen das Optimum gefunden wird.

4.4.4 Initiale Heuristik

Ein weiterer Aspekt, der nicht unerheblich für die Güte eines evolutionären Algorithmus ist, ist das Problemwissen, mit dem dieser Algorithmus ausgestattet werden kann. Wie in Abschnitt 3.3.2 besprochen, werden von Startpopulation, welche aus 10 Individuen besteht, 3 Individuen mittels einer Heuristik erzeugt, die an das Konzept aus Abschnitt angelehnt ist. Jedoch wird hier nicht die Heuristik der minimalen Δ („minDelta-Strategie“)-, sondern diejenige mit der minimalen Maximum („minMax-Strategie“)- Auswahlstrategie benutzt. Experimentell zeigte sich dabei, wie in Abbildung 4.9 dargestellt, dass die minMax-Strategie einen kleineren Hochpunkt liefert und somit für den evolutionären Algorithmus besser geeignet ist, da sie von vorne herein „stärkere“ Individuen in die Startpopulation einbringen kann.

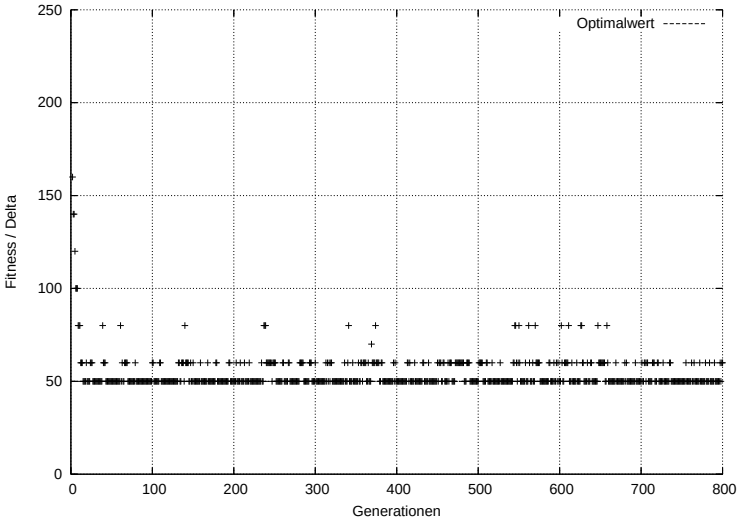

Abbildung 4.7: Szenario 2-d: Konvergenz des evolutionären Algorithmus zum Optimalwert bei geringer Streuung des Risikos. $m = 5$ Provider und $n = 12$ Shares.

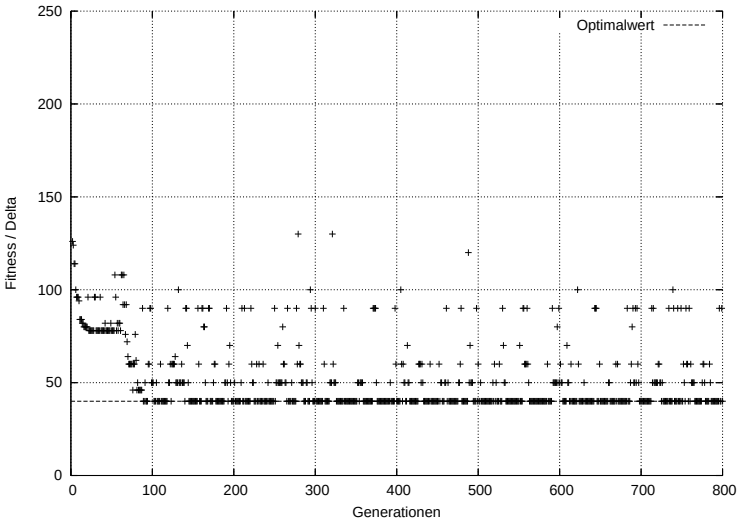

Abbildung 4.8: Szenario 2-d: Konvergenz des evolutionären Algorithmus zum Optimalwert bei starker Streuung des Risikos. $m = 5$ Provider und $n = 12$ Shares.

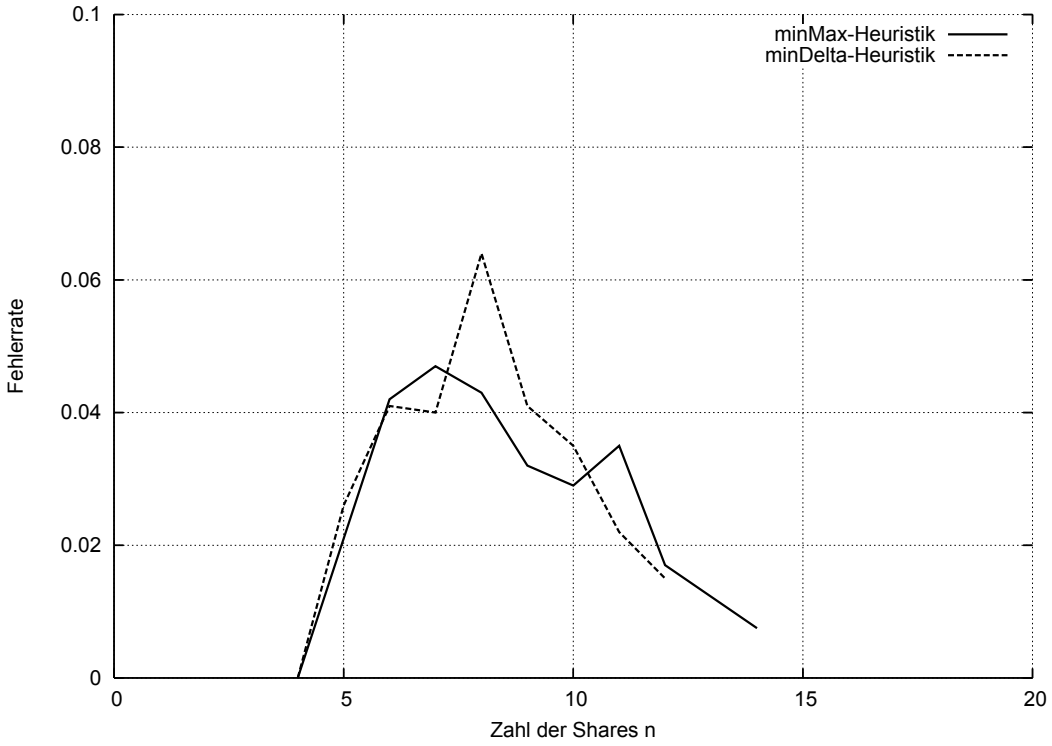

Abbildung 4.9: Szenario 2-d: Vergleich zweier Heuristiken für die initiale Population beim evolutionären Algorithmus mit $m = 4$ Providern

5 Fazit

5.1 Zusammenfassung und Diskussion

Zu Beginn der Arbeit wurden im zweiten Kapitel einige Argumente dafür vorgetragen, warum Sicherheit der *Privatsphäre* gerade bei *Location-based Services* von hoher Bedeutung ist: LBS sind eine äußerst sensible Technik in Bezug auf die Privatsphäre, da sie mit der Positionsinformation der Individuen arbeiten. In dieser Diplomarbeit wurde versucht, die Sicherheit der Privatsphäre in technischer Hinsicht zu erhöhen.
Dazu wurde von einem Ansatz der verteilten Positionsinformationen ausgegangen, wonach die exakte Positionsinformation in Teile zerlegt wird. Diese sog. *Shares* werden dann auf unabhängigen Location Servern, genannt *Provider*, abgelegt. Die exakte Position lässt sich nur mit allen diesen Shares rekonstruieren. Ein bestehendes Systemmodell [DSR11], welches von ungewichteten Providern ausgeht, wurde um eine *Trust Database* erweitert, in der verschiedene Vertrauens- oder Risikowerte zu jedem Server gespeichert sind. Zusätzlich können auch noch die Shares spezifisch gewichtet sein.
Die Aufgabe ist es schließlich, die Teile der Positionsinformation den Providern so zuzuweisen, dass die *Fragilitäten* der Provider ausbalanciert werden, so dass risikoreiche Provider relativ weniger Shares zugewiesen bekommen als vertrauenswürdige. Eine Balance zwischen den Fragilitäten der Providern führt im Fall, dass die Shares gewichtet sind, zu einer wenigstens angemessenen Verschlechterung der Privatsphäre bei fortlaufender Anzahl kompromittierter Server, was bedeutet, dass bei jedem weiteren gehackten Provider genauso viele Informationen bekannt werden wie bei jedem anderen auch. Somit gibt es keine Schwachstelle im System, die die Privatsphäre überproportional beeinträchtigen könnte. Sind die Shares nicht gewichtet, so ergibt sich dadurch eine Austarierung der risikobehafteten Provider, wobei risikoreiche Provider relativ weniger Shares bekommen.
In der vorliegenden Arbeit wurden zwei Szenarien betrachtet, die noch weiter in Unterfälle differenziert wurden, wonach entweder Provider, Shares oder beide gewichtet sind. Die Anzahl der Provider sowie deren Risiken war immer als gegeben vorausgesetzt. Im *ersten Szenario* wurde davon ausgegangen, dass die Zahl der Shares und somit auch deren Gewichtungen frei wählbar sind. Im *zweiten Szenario* wurde von einer fest definierten Anzahl von Shares ausgegangen (und auch von festgelegten Gewichtungen der Shares). Nun muss diskutiert werden, inwiefern die Szenarien und die Lösungsansätze für welche Situation in der Praxis tauglich sind. Hierzu wurden die algorithmischen Konzepte für Szenario 2 implementiert und bezüglich Korrekt und Laufzeitverhalten evauliert.

In Szenario 1, in dem sowohl die Zahl der Shares wie auch deren Gewichtungen nicht vorgegeben, sondern vom Nutzer zu bestimmen sind, lässt sich die Optimalität der Platzierung über geschlossene Formeln in linearer oder sogar konstanter Zeit berechnen. Szenario 1 zeigt seine Berechtigung, weil die Generierung der Shares im Systemmodell [DSR11] auf der Seite der mobilen Geräte erfolgt. Also werden in diesem Gerät genau die Shares erzeugt, die zur Ausbalancierung der Sicherheitsrisiken der ausgewählten Location Server notwendig sind. Selbst wenn mobile Geräte auch heute noch stationären PCs leistungsmäßig hinterherhinken, sind diese Berechnungen in maximal linearer Zeit kein Problem. Des weiteren muss noch betont werden, dass sich nur mit den Teilszenarien 1-c und 1-d eine angemessene Verschlechterung erreichen lässt. Die Shares müssen gewichtet sein, da der Genauigkeitsgewinn bei Zusammenfügung der Shares nur über die Sharegewichte bestimmt wird.

Szenario 2 geht von einer festen Anzahl von Providern und Shares aus. Nun müssen die Shares auf die Provider verteilt werden, damit eine Balance der Fragilitäten erreicht wird. Mit den Ergebnissen aus Abschnitt 2.3 konnte dieses Problem als Zuweisungsproblem eingeordnet und formalisiert werden. Dieses Zuweisungsproblem wurde als *Balanced Generalized Assignment Problem* (*BaGAP*) benannt. Dieses Problem ist nach dem Wissensstand des Autors dieser Arbeit bisher noch nirgendwo formalisiert und bearbeitet worden. Es konnte durch eine Reduktion eines verwandten Problems gezeigt werden, dass das BaGAP selbst NP-schwer ist.
Für den Teilszenario 2-b, wonach die Provider gewichtet sind und die Shares nicht, wurde ein Algorithmus vorgeschlagen, der eine begrenzte Zahl von Backtracking-Schritten ausführt. Im Verlauf dieser Arbeit wurde sehr viel Zeit auf die Suche nach einer Heuristik verwendet, die dem Algorithmus ohne Backtracking-Schritte entspricht. In der Evaluation konnte die theoretische Überlegung, dass bei einer exakten Lösung die Komplexität exponentiell mit der Zahl der Provider steigt, bestätigt werden – für 4 Provider befindet sich die Laufzeit noch im Bereich von Mikrosekunden, während für 8 Provider die Berechnung schon mehrere Sekunden benötigt.
Sind sowohl Provider wie auch Shares gewichtet (Szenrio 2-d), so wurde ein metaheuristisches Verfahren implementiert, nämlich die evolutionären Algorithmen, die sich als vielversprechend erwiesen, was im Abschnitt 3.3.2 erörtert wurde. Dieser zeigt ein gutes Laufzeitverhalten im linearen Bereich in der Größenordnung von Millisekunden. Jedoch wird nicht in allen Fällen der Optimalwert erreicht, was in der Natur der Sache eines metaheuristischen Verfahrens lieg. Die Fehlerrate ist jedoch akzeptierbar, wobei Fehlerrate die Fälle bezeichnet, in denen der Algorithmus nicht das Optimum findet. Je größer die Zahl der Shares, desto stärker nähert sich die Fehlerrate 0% an. Szenario 2-c wurde schließlich ebenfalls mit evolutionären Algorithmen gelöst. Zwar existiert auch hierfür eine Heuristik, jedoch liegt ihre Fehlerrate zu hoch.
Szenario 2 ist praktisch relevant vor allem für den Fall, dass mit schon erzeugten Shares umgegangen werden muss. Dies tritt beispielsweise dann auf, wenn die Shares auf einem mobilen Gerät erzeugt werden, die Erstellung des Platzierungsplans dann aber auf einem

anderen Gerät oder Server erfolgt. Vor allem für ein leistungsschwächeres mobiles Gerät könnte eine exakte Berechnung durch die exponentielle Komplexität mittels des für 2-b vorgestellten Backtracking-Verfahrens schon für kleine Anzahlen von Providern problematisch sein. Treten viele Provider auf, bleibt nichts anderes übrig, als die Heuristik zur Platzierungsberechnung zu verwenden und eine gewisse Fehlerrate zu akzeptieren. Für Szenario 2-d wird von vorne herein zu einem metaheuristischen Verfahren geraten. Es liegt im Wesen NP-schwerer Probleme, diese Herangehensweise notwendig zu machen.

5.2 Ausblick

Da aus der praktischen Anwendung heraus auf eine sehr theoretische Fragestellung, nämlich auf ein vermutlich neues Zuweisungsproblem gestoßen wurde, finden sich hier einige Anknüpfungspunkte. Auch weil viel Zeit für die Suche nach einer guten Heuristik investiert wurde, sind einige Aspekte in diesem Rahmen nicht bearbeitet worden. Im Folgenden findet sich ein Überblick.

Da es sich beim BaGAP um ein Zuweisungsproblem handelt, das nach bestem Wissen noch nicht in der Literatur behandelt wurde, steht eine algorithmentheoretische Bearbeitung dieses Problems noch aus. Darunter verstehe ich vor allem die genaue theoretische Bearbeitung des Problems, das Beleuchten verschiedener Lösungsmöglichkeiten in einem theoretischen Sinne und die Abschätzung von Approximationsalgorithmen durch Methoden der theoretischen Informatik.

Darüber hinaus wäre es fruchtbar, neben den hier vorgestellten evolutionären Algorithmen zur Lösung noch weitere Metaheuristiken, wie z.B. die simulierte Abkühlung, zu implementieren und zu vergleichen. Speziell die simulierte Abkühlung, welche allgemein auch zuverlässige Näherungsergebnisse liefert, könnte etwas schneller sein als die hier vorgestellten evolutionären Algorithmen. Zur Abwägung wäre ein genauer Vergleich von Korrektheit und Geschwindigkeit der Verfahren nötig.

Des Weiteren wäre eine Implementierung und Evaluation unter realen Bedingungen nötig, da die vorgestellten Konzepte auf mobiler Hardware wie Smartphones laufen sollten, weil die Erzeugung und somit auch die Erstellung des Platzierungsplans dort stattfindet. Dies wäre wichtig, weil dadurch weitere Messwerte über das Zeitverhalten der Algorithmen vorliegen würden und sich so Rückschlüsse für die Praxis liefern ließen.

Literaturverzeichnis

[ACD+07] C. A. Ardagna, M. Cremonini, E. Damiani, S. D. C. di Vimercati, P. Samarati. Location privacy protection through obfuscation-based techniques. In *Proceedings of the 21st annual IFIP WG 11.3 working conference on Data and applications security*, pp. 47–60. Springer-Verlag, Berlin, Heidelberg, 2007. (Zitiert auf den Seiten 21 und 26)

[AP97] S. Arora, M. C. Puri. A variant of time minimizing assignment problem. *European Journal of Operational Research*, 110(2):314–325, 1997. (Zitiert auf den Seiten 43 und 60)

[BDM09] R. Burkhard, M. Dell'Amico, S. Martello. *Assignment Problems*. siam, 2009. (Zitiert auf den Seiten 36, 39, 40 und 41)

[CLRS04] T. H. Cormen, C. E. Leiserson, R. Rivest, C. Stein. *Algorithmen - Eine Einführung*. Oldenbourg Wissenschaftsverlag GmbH, München, 2004. (Zitiert auf den Seiten 32, 39 und 67)

[CVW92] D. G. Cattrysse, L. N. Van Wassenhove. A survey of algorithms for the generalized assignment problem. *European Journal of Operational Research*, 60(3):260–272, 1992. (Zitiert auf Seite 42)

[DF03] J. E. Dobson, P. F. Fisher. Geoslavery. *Technology and Society Magazine, IEEE*, 22(1):47–52, 2003. (Zitiert auf den Seiten 23 und 24)

[DK06] M. Duckham, L. Kulik. Location privacy and location-aware computing. In J. Drummond, R. Billen, D. Forrest, E. Joao, editors, *Dynamic & Mobile GIS: Investigating Change in Space and Time*, chapter 3, pp. 34–51. CRC Press, Boca Rator, FL, 2006. (Zitiert auf den Seiten 15, 23, 25 und 26)

[DS03] M. Dorigo, T. Stützle. The Ant Colony Optimization Metaheuristic: Algorithms, Applications, and Advances. *Handbook of Metaheuristicse*, 57, 2003. (Zitiert auf Seite 36)

[DSR11] F. Dürr, P. Skvortsov, K. Rothermel. Position Sharing for Location Privacy in Non-trusted Systems. In *2011 IEEE International Conference on Pervasive Computing and Communications (PerCom), Seattle (March 21-25, 2011)*. 2011. (Zitiert auf den Seiten 9, 16, 17, 19, 21, 22, 25, 26, 28, 45, 46, 47, 52, 97 und 98)

[Egl90] R. W. Eglese. Simulated annealing: A tool for operational research. *European Journal of Operational Research*, 46(3):271 – 281, 1990. (Zitiert auf den Seiten 34 und 35)

[Gut06] A. Gutscher. Coordinate transformation - a solution for the privacy problem of location based services? In *Proceedings of the 20th international conference on Parallel and distributed processing*, IPDPS'06, pp. 354–354. IEEE Computer Society, Washington, DC, USA, 2006. (Zitiert auf den Seiten 9, 26 und 27)

[Gut07] A. Gutscher. A Trust Model for an Open, Decentralized Reputation System. In *Proceedings of the Joint iTrust and PST Conferences on Privacy Trust Management and Security (IFIPTM 2007)*. 2007. (Zitiert auf Seite 48)

[HMU03] J. E. Hopcroft, R. Motwani, J. D. Ullman. *Introduction to automata theory, languages, and computation - international edition (2. ed)*. Addison-Wesley, 2003. (Zitiert auf den Seiten 32 und 42)

[Hu82] T. C. Hu. *Combinatorial Algorithms*. Addison-Weslex, 1982. (Zitiert auf den Seiten 30 und 33)

[LW66] E. L. Lawler, D. E. Wood. Branch-And-Bound Methods: A Survey. *Operations Research*, 14(4):699–719, 1966. (Zitiert auf Seite 33)

[MCA01] F. M. Müller, M. M. Camozzato, O. C. B. de Araujo. Exact Algorithms for the Imbalanced Time Minimizing Assignment Problem. *Electronic Notes in Discrete Mathematics*, 7:122–125, 2001. (Zitiert auf Seite 43)

[MN88] J. B. Mazzola, A. W. Neebe. Bottleneck generalized assignment problems. *Engineering Costs and Production Economics*, 14(1):61–65, 1988. (Zitiert auf den Seiten 42, 43 und 60)

[MN92] J. B. Mazzola, A. W. Neebe. An algorithm for the bottleneck generalized assignment problem. *Comput. Oper. Res.*, 20:355–362, 1992. (Zitiert auf den Seiten 42 und 43)

[MPTW84] S. Martello, W. R. Pulleyblank, P. Toth, D. de Werra. Balanced optimization problems. *Operations Research Lett.*, 3:275–278, 1984. (Zitiert auf den Seiten 40 und 41)

[MRF+03] S. K. Mostéfaoui, O. Rana, N. Foukia, S. Hassas, G. D. Marzo, C. V. Aart, A. Karageorgos. Self-Organising Applications: A Survey. *Engineering Self-Organising Applications, First International Workshop, ESOA 2003. Melbourne, Victoria, July 15th, 2003. Workshop Notes*, pp. 62–69, 2003. (Zitiert auf Seite 36)

[MT90] S. Martello, P. Toth. *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley & Sons, Chichester, NY, revised edition, 1990. (Zitiert auf den Seiten 30, 33, 41 und 42)

[MT95] S. Martello, P. Toth. The bottleneck generalized assignment problem. *European Journal of Operational Research*, 83(3):621 – 638, 1995. (Zitiert auf den Seiten 42 und 43)

[Nis97] V. Nissen. *Einführung in evolutionäre Algorithmen: Optimierung nach dem Vorbild der Evolution*. Braunschweig: Vieweg, 1997. (Zitiert auf Seite 35)

[Pen05] D. W. Pentico. Assignment problems: A golden anniversary survey. *European Journal of Operational Research*, 176(2):774–793, 2005. (Zitiert auf den Seiten 40, 41 und 42)

[RDD+03] K. Rothermel, D. Dudkowski, F. Dürr, M. Bauer, C. Becker. Ubiquitous Computing - More than Computing Anytime Anyplace? In D. Fritsch, editor, *Photogrammetric Week*, pp. 3–11. Stuttgart, Germany, 2003. (Zitiert auf Seite 15)

[REF+06] K. Rothermel, T. Ertl, D. Fritsch, P. Kühn, B. Mitschang, E. Westkämper, C. Becker, D. Dudkowski, A. Gutscher, C. Hauser, L. Jendoubi, D. Nicklas, S. Volz, M. Wieland. SFB 627 Umgebungsmodelle für mobile kontextbezogene Systeme. *Informatik - Forschung und Entwicklung*, pp. 105–113, 2006. (Zitiert auf den Seiten 15 und 16)

[RM98] A. Regenbogen, U. Meyer. *Wörterbuch der philosophischen Begriffe*, chapter Ubiquität, p. 682. Wissenschaftliche Buchgesellschaft: Darmstadt, 1998. (Zitiert auf Seite 15)

[RM03] B. Rao, L. Minakakis. Evolution of mobile location-based services. *Commun. ACM*, 46:61–65, 2003. (Zitiert auf Seite 22)

[RPB09] D. Riboni, L. Pareschi, C. Bettini. Privacy in Georeferenced Context-Aware Services: A Survey. In C. Bettini, S. Jajodia, P. Samarati, X. Wang, editors, *Privacy in Location-Based Applications*, volume 5599 of *Lecture Notes in Computer Science*, pp. 151–172. Springer Berlin / Heidelberg, 2009. (Zitiert auf Seite 25)

[SDfMb09] A. Solanas, J. Domingo-ferrer, A. Martínez-ballesté. Location Privacy in Location-Based Services: Beyond TTP-based Schemes. In C. Bettini, S. Jajodia, P. Samarati, X. Wang, editors, *Privacy in Location-Based Applications*, volume 5599 of *Lecture Notes in Computer Science*, pp. 151–172. Springer Berlin / Heidelberg, 2009. (Zitiert auf Seite 25)

[Tan03] A. S. Tanenbaum. *Computernetzwerke*. Pearson Studium, München, 2003. (Zitiert auf Seite 23)

[VK08] J. Vygen, B. Korte. *Kombinatorische Optimierung: Theorie und Algorithmen*. Springer, Berlin, 2008. (Zitiert auf den Seiten 11, 30, 31, 32, 33, 34 und 75)

[VTV+01] K. Virrantaus, H. Tirri, J. Veijalainen, J. Markkula, A. Katanosov, A. Garmash, V. Terziyan. Developing GIS-Supported Location-Based Services. In *Proc. of WGIS 2001*, volume 2, p. 66. IEEE Computer Society, 2001. (Zitiert auf den Seiten 21 und 22)

[Wei91] M. Weiser. The Computer for the Twenty-First Century. *Scientific American*, 265(3):94–104, 1991. (Zitiert auf Seite 15)

[Wei02] K. Weicker. *Evolutionäre Algorithmen*. Teubner, Stuttgart, 2002. (Zitiert auf den Seiten 9, 34, 35, 36, 62 und 77)

[Wes67] A. F. Westin. *Privacy and Freedom*. Atheneum, New York, 1967. (Zitiert auf Seite 23)

Alle URLs wurden zuletzt am 28.04.2011 geprüft.

Erklärung

Hiermit versichere ich, diese Arbeit selbständig verfasst und nur die angegebenen Quellen benutzt zu haben.

(Björn Schembera)