FAKULTÄT FÜR !INFORMATIK

Exakte und heuristische Optimierungsmethoden zur Lösung von Video Server Load Re-Balancing

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering & Internet Computing

eingereicht von

Jakob Walla
Matrikelnummer 0126068

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung:
Betreuer: ao.Univ.-Prof. Dipl.-Ing. Dr. Günther Raidl
Mitwirkung: Univ.-Ass. Dipl.-Ing. Mario Ruthmair

Wien, 06.04. 2009

(Unterschrift Verfasser) (Unterschrift Betreuer)

Erklärung zur Verfassung der Arbeit

Jakob Walla
Budinskygasse 11/1/11
1190 Wien

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

Wien, 6. April 2009,

Zusammenfassung

Ein Video-on-Demand (VoD) System besteht häufig aus einer großen Anzahl unabhängiger Video-Server. Um mit einer gegebenen Anzahl an Video-Servern eine möglichst große Anzahl gleichzeitiger Zugriffe bedienen zu können, soll ein Ausgleich der Netzwerklast zwischen den vorhandenen Servern erzielt werden. Das Lastverteilungsproblem in einem VoD-System besteht darin, ausgehend von einer Schätzung der pro Video-Clip maximal gleichzeitig zu erwartenden Zugriffe eine Anzahl von Replikaten jedes Video-Clips und deren Platzierung auf den vorhandenen Servern zu ermitteln. Gleichzeitig erfolgt eine Zuordnung der geschätzten Zugriffe zu diesen Replikaten, sodass für jeden Server des Systems entsprechend seiner Übertragungskapazität eine gerechte Auslastung während der Phase höchster Nachfrage erreicht wird. Diese Diplomarbeit beschreibt eine Formulierung dieses Lastverteilungsproblems als kombinatorisches Optimierungsproblem, genannt *Video-Server Load Re-Balancing* (VSLRB). Es berücksichtigt im Gegensatz zu vielen Arbeiten aus der Literatur auch die Minimierung des Reorganisationsaufwands zur Herstellung der neu ermittelten Replikatszuordnung aus der bereits bestehenden. Zur exakten Lösung dieses Problems wird eine Formulierung als gemischt-ganzzahliges lineares Programm (MIP) entwickelt. Um auch Lösungen für größere Instanzen dieses Problems ermitteln zu können, wird weiters eine Anwendung der Metaheuristik Variable Neighbourhood Search (VNS) beschrieben. Diese verwendet unter anderem eine Nachbarschaftsstruktur basierend auf zyklischen Vertauschungen (Cyclic Exchange Neighbourhood) und eine Nachbarschaftsstruktur, die unter Verwendung des MIP-Ansatzes durchsucht wird. Tests mit insgesamt zehn Testinstanzen von unterschiedlicher Größe zeigen, dass das beschriebene Verfahren in der Lage ist, in jedem dieser Fälle Lösungen mit praktisch zu vernachlässigenden Abweichungen der Serverlasten von zuvor berechneten zu erzielenden Lasten zu ermitteln.

Abstract

A Video-on-Demand (VoD) system usually consists of a large number of independent video servers. In order to serve a maximal number of concurrent requests with a given number of servers the overall network load should be equally balanced among the available servers. A load balancing procedure for a VoD system relies on the prediction of the expected maximal number of parallel accesses to each video file. Based on this estimation a required number of replicas per video file and their placement on the available servers as well as an assignment of the predicted requests to these replicas should be determined. This assignment should ensure a fair load for each server during the period of highest user interest, taking into account its share of the overall upload capacity of the VoD system. This master's thesis gives a formalisation of the VoD load balancing problem in terms of a combinatorial optimization problem, called *Video-Server Load Re-Balancing* (VSLRB). In contrast to many works in literature this formulation incorporates minimisation of the reorganisation costs which arise from the transformation of the current replica assignment into the newly obtained one. An equivalent formulation in terms of a mixed integer linear program (MIP) is given as an exact approach to solving this problem. Furthermore this thesis describes a heuristic approach in the form of an application of Variable Neighbourhood Search (VNS) to VSLRB. This VNS features a neighbourhood structure based on cyclic exchanges of requests and a neighbourhood structure based on the MIP approach. Tests where conducted on ten test instances of varying size. Results show that in each case the described approach is able to identify solutions with practically negligible deviations of server loads from pre-calculated target server loads.

Danksagung

Nachdem die letzten Worte dieser Diplomarbeit geschrieben sind und damit das Ende meiner Studienzeit unweigerlich in greifbare Nähe rückt, gebührt in der Rückschau Dank vor allem meinen Eltern, ohne deren finanzielle Unterstützung ich mich, vor allem während des Masterstudiums, nicht in der Weise auf meinen Studienerfolg hätte konzentrieren können, wie ich es während dieser Zeit (zumeist) getan habe. Die Möglichkeit gehabt zu haben, mich im vergangenen Jahr ohne größere materielle Sorgen intensiv mit der Problemstellung in dieser Arbeit beschäftigen zu können und mich schrittweise von unausgegorenen anfänglichen Ideen zum jetzigen Endprodukt zu bewegen, bedeutet mir sehr viel.

Dank gilt weiters meinen Betreuern Prof. Dr. Günther Raidl und Dipl.-Ing. Mario Ruthmair dafür, dass ich meinen Themenvorschlag als Diplomarbeit am Institut für Computergraphik und Algorithmen umsetzen konnte. Weiters bedanke ich mich für eine sehr angenehme und konstruktive Arbeitsatmosphäre, vor allem während der schwierigen anfänglichen Orientierungs- und Ideenfindungsphase und für spätere sehr engagierte Korrekturen und Verbesserungsvorschläge. Mario Ruthmair gebührt dabei spezieller Dank für jede nachgerechnete Formel und jedes einzelne gefundene “VSRLB”.

Ganz besonderer Dank geht nicht zuletzt an meine Freundin Stefanie für unverzichtbare moralische Unterstützung, für sehr hilfreiche Kritik zur optischen Gestaltung und vor allem für die schöne gemeinsame Zeit.

> “Time is the school in which we learn, Time is the fire in which we burn”
>
> Delmore Schwartz, aus “For Rhoda” (1938)

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Liste der Algorithmen

Kapitel 1

Einleitung

Der Begriff Video-on-Demand (VoD) beschreibt die Auslieferung von Videomaterial in Echtzeit und auf Anfrage des Betrachters, im Gegensatz zu herkömmlichem Fernsehen, das dem Konsumenten eine rein passive Rolle zugesteht. Anwendungen dieser Art erfreuen sich zur Zeit vor allem dank Internet-Video-Plattformen wie YouTube [1] großer Beliebtheit. Der Betrieb eines solchen Systems ist allerdings im Allgemeinen mit einer hohen Investition in die technische Infrastruktur verbunden. Dies ergibt sich sowohl aus der sehr großen Anzahl potentieller Nutzer als auch aus der Zeitabhängigkeit des Mediums Video: Im Unterschied zur Auslieferung von statischen Inhalten muss bei der Auslieferung eines Video-Clips ein bestimmter, durch die Bitrate des Video-Clips vorgegebener Anteil der Bandbreite der Netzwerkschnittstelle eines Video-Servers über die gesamte Spieldauer garantiert zur Verfügung stehen. Daraus ergeben sich enorme Anforderungen an die Übertragungskapazität eines einzelnen Video-Servers. Zur Verteilung dieser Netzwerklast werden häufig geographisch verteilte Cluster von Video-Servern eingesetzt.

Um die gesamte in einem Video-on-Demand-System zur Verfügung stehende Übertragungskapazität möglichst effizient zu nutzen und so eine möglichst große Zahl an Video-Clips mit einer gegebenen Zahl von Video-Servern parallel ausliefern zu können, ist es notwendig, einen Lastenausgleich zwischen den Video-Servern des Video-on-Demand-Systems herzustellen. Ausgangspunkt eines solchen Lastverteilungsverfahrens ist die Schätzung der pro Video-Clip maximal zu erwartenden gleichzeitigen Zugriffe. Dieses sogenannte *Access Profile* basiert unter anderem auf einem Modell der zeitlichen Verteilung der Zugriffe und der Modellierung der Auswahlwahrscheinlichkeit jedes angebotenen Video-Clips. Ausgehend von dieser Schätzung soll

- pro Video-Clip v eine Menge von Replikaten und deren Platzierung auf den vorhandenen Video-Servern ermittelt, sowie gleichzeitig
- jedem der Replikate von v eine Teilmenge der erwarteten gleichzeitigen Zugriffe auf v zugeordnet werden,

sodass für jeden Video-Server entsprechend der Bandbreite seiner Netzwerkschnittstelle eine gerechte Auslastung während der Phase höchster Nachfrage erzielt wird. Diese Lastverteilungsberechnung zielt auf eine Optimierung der Auslastung im Worst Case ab, d.h. wenn alle vorhergesagten Zugriffe des Access Profiles gleichzeitig aktiv sind.

Die vorliegende Masterarbeit formuliert ein solches Lastverteilungsproblem für ein konkretes Video-on-Demand-System als kombinatorisches Optimierungsproblem, genannt *Video-Server Load Re-Balancing* (VSLRB). Neben der Herstellung einer gerechten Auslastung

berücksichtigt die verwendete Formulierung einen weiteren Aspekt des Problems, der in der Literatur häufig vernachlässigt wird: Sobald eine neue Zuordnung von Replikaten zu Video-Servern ermittelt wurde, muss diese Zuordnung durch Kopier- und Löschaktionen aus der bereits bestehenden Zuordnung physisch hergestellt werden. Dies macht die Übertragung von potentiell sehr großen Datenmengen zwischen den einzelnen Servern des VoD-System notwendig, wodurch für eine gewisse Zeitspanne weniger Übertragungskapazität für die Auslieferung von Video-Clips zur Verfügung steht. Es ist daher wünschenwert, die Dauer dieser an die Lastverteilungsberechnung anschließenden Reorganisationsphase zu minimieren, sodass eine gerechte Auslastung bei gleichzeitig möglichst geringem Reorganisationsaufwand erzielt wird[1].

Abbildung 1.1 illustriert das Vorgehen zur Herstellung einer gerechten Auslastung anhand eines konkreten Beispiels: Nach der Erstellung eines neuen Access Profiles weisen die drei Server A, B und C des dargestellten Systems eine gravierende Abweichung von ihrer jeweiligen Soll-Auslastung auf (siehe Abbildung 1.1a). Dies könnte in diesem Beispiel durch einen Anstieg der Beliebtheit der Video-Clips 1 und 2 bei einem gleichzeitigen Absinken der Beliebtheit der Video-Clips 3 und 4 geschehen sein. Um eine gerechte Auslastung zu erzielen, werden die Zugriffe auf Video-Clip 1, die bisher allein durch Server A abgewickelt wurden, auf die Server A und B verteilt. Weiters wird ein Teil der Zugriffe auf Video-Clip 2 von Server B auf Server C verschoben. Damit Server B und C diese neuen Zugriffe abwickeln können müssen diese ein Replikat von Video-Clip 1 bzw. Video-Clip 2 erhalten. (siehe Abbildung 1.1b). Nach Durchführung dieser Reorganisationsschritte befindet sich das System wieder in einem Zustand gleichmäßiger Auslastung (siehe Abbildung 1.1c).

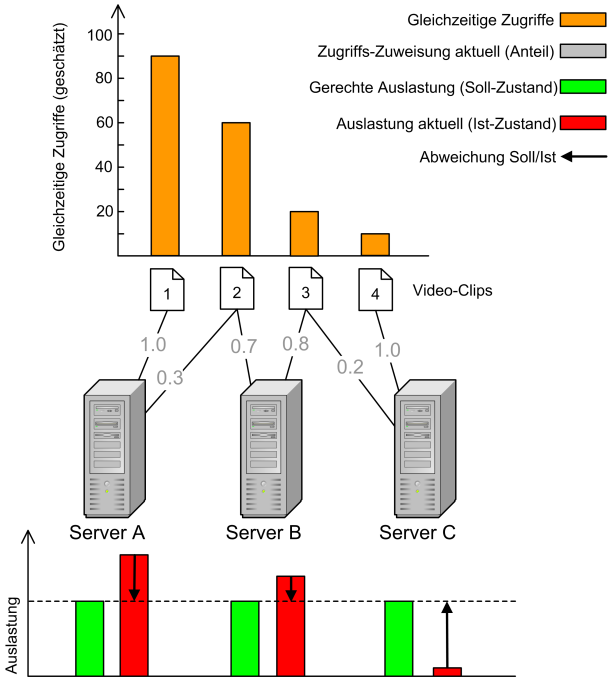
Zur exakten Lösung dieses Problems wird im Verlauf dieser Diplomarbeit eine Formulierung als gemischt-ganzzahliges lineares Programm (MIP) entwickelt. Diese kann zur beweisbar optimalen Lösung von Instanzen mit einer geringen Anzahl von Servern eingesetzt werden. Um auch große, praxisrelevante Instanzen von VSLRB in angemessener Zeit lösen zu können, wird in dieser Arbeit weiters ein heuristischer Ansatz auf Basis der Metaheuristik *Variable Neighbourhood Search* (VNS) beschrieben. Dieser Ansatz verwendet neben einfachen Verschiebungs- bzw. Austauschnachbarschaften

- eine Nachbarschaftsstruktur basierend auf zyklischen Vertauschungen von Zugriffen (Cyclic Exchange Neighbourhood) sowie
- eine Nachbarschaftsstruktur, die mit Hilfe des MIP-Ansatzes durchsucht wird.

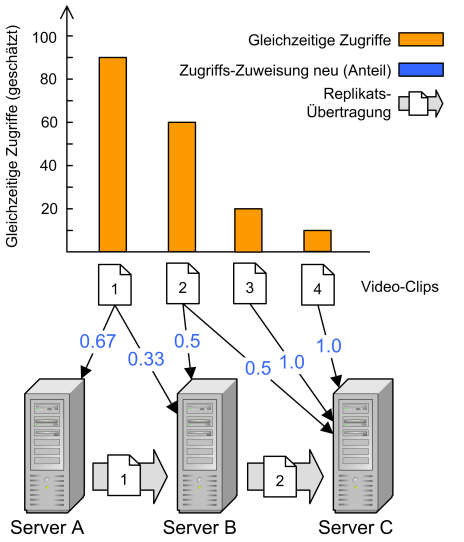
Beide Verfahren gehören zu den Methoden der *Very Large Scale Neighbourhood* (VLSN) *Search*, welche sich durch die Verwendung von Nachbarschaften mit einer sehr großen Anzahl von Nachbarlösungen auszeichnen. Diese können dennoch aufgrund geeigneter Konstruktionsweisen trotz ihrer Größe effizient nach verbesserten bzw. besten Nachbarlösungen durchsucht werden [4, 41].

Die Verwendung einer Cyclic Exchange Neighbourhood wurde erstmals von Thompson und Orlin [44] zur Lösung von allgemeinen Partitionierungsproblemen beschrieben: Eine gegebene Menge von Elementen soll in eine vorgegebene Anzahl von Teilmengen mit minimalen Gesamtkosten zerlegt werden. Als logische Erweiterung des Austauschs von zwei

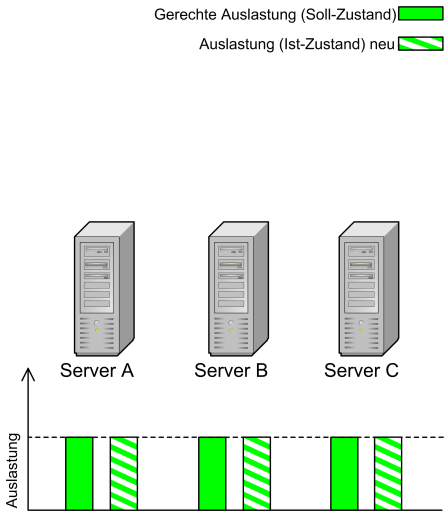
[1] Zwar findet eine solche Reorganisation üblicherweise während der sogenannten *Off-Peak Hours* [34] statt, während derer Reorganisationsaufgaben durchgeführt werden können, ohne den Systembetrieb gravierend zu beeinträchtigen, allerdings muss nicht jedes konkrete VoD-System eine solche ausgeprägte tägliche Phase niedriger Auslastung aufweisen.



(a) Nach Ermittlung eines neuen Access Profiles liegt bei Verwendung der bisherigen Verteilung von Zugriffen eine Abweichung von den gerechten Server-Auslastungen vor.



(b) Durch neue Zuordnung der zu erwartenden Zugriffe und eventueller Übertragung von Replikaten ...



(c) ... wird eine gerechte Server-Auslastung wieder hergestellt.

Abbildung 1.1: Ablauf der in dieser Arbeit behandelten Lastverteilung

Elementen zwischen zwei Teilmengen sind die Nachbarlösungen in einer solchen Nachbarschaft durch alle möglichen zyklischen Verschiebungen von k Elementen über k Teilmengen gegeben. Die Suche nach solchen zyklischen Verschiebungen geschieht durch die Suche nach speziellen Zyklen mit minimalen Kosten in einem geeignet konstruierten Graphen, dem sogenannten *Improvement Graph*. Dieser Graph enthält eine Kante für jede gültige Verschiebung eines Elements von einer Teilmenge in eine andere. Die Kantenkosten entsprechen dabei der Differenz des Zielfunktionswerts, welche durch die Verschiebung des jeweiligen Elements entsteht. Ein Zyklus in diesem Graphen, der einem gültigen zyklischen Austausch entspricht, muss die Eigenschaft besitzen, dass jeder der Knoten in diesem Zyklus einer unterschiedlichen Teilmenge angehört. Die Suche nach solchen Zyklen stellt zwar ihrerseits wiederum ein schwieriges Problem dar, kann jedoch mittels schneller Heuristiken erfolgen [4, 41].

Da VSLRB als ein spezielles Partitionierungsproblem aufgefasst werden kann, wurde eine angepasste Variante der Cyclic Exchange Neighbourhood eingesetzt, welche einen Improvement-Graph verwendet, der mit Hilfe des in [5] beschriebenen *Modified label-correcting Algorithm* durchsucht und bei Änderungen an der zugrundeliegenden Lösung inkrementell aktualisiert wird.

Die zweite verwendete VLSN-Nachbarschaftsstruktur basiert auf der Hybridisierung von VNS mit Methoden der mathematischen Programmierung: Das Verfahren bestimmt auf heuristische Weise ein Unterproblem mit möglichst großem Verbesserungspotential, das unter Verwendung des MIP-Ansatzes exakt gelöst wird. Durch die Einbringung einer auf diese Weise erzielten Lösung des Unterproblems in die originale Lösung kann für diese eine Verbesserung des Zielfunktionswerts erreicht werden. Diese Vorgangsweise stellt in der Terminologie von [42] einen *integrativen* und *schwach gekoppelten* Ansatz zur Bildung eines hybriden Verfahrens dar, da der MIP-Ansatz als eingebettetes Verfahren zur Lösung eines klar abgegrenzten Unterproblems eingesetzt wird, ohne die Eigenschaften der verwendeten Verfahren selbst zu verändern.

Struktur der Diplomarbeit

Kapitel 2 gibt einen Überblick über das Forschungsgebiet der Video-on-Demand-Systeme sowie über verwandte Arbeiten und Problemstellungen aus der Literatur. Kapitel 3 beschreibt die Formalisierung der Problemstellung als kombinatorisches Optimierungsproblem *Video-Server Load Re-Balancing* (VSLRB) und liefert einen Beweis der $\mathcal{NP}$-Vollständigkeit dieses Problems. Kapitel 4 enhält eine kurze Einführung in die Theorie der linearen und ganzzahligen linearen Programmierung. Darauf aufbauend beschreibt Kapitel 5 eine Umsetzung der Formalisierung von VSLRB als gemischt-ganzzahliges lineares Programm. Kapitel 6 und 7 widmen sich der Metaheuristik Variable Neighbourhood Search sowie der Anwendung dieser Metaheuristik auf VSLRB, in deren Rahmen wie beschrieben Unterprobleme mittels der Formulierung als gemischt-ganzzahliges lineares Programm aus Kapitel 5 gelöst werden. Kapitel 8 enhält eine kurze Übersicht über die Implementierung der Anwendung von Variable Neighbourhood Search auf VSLRB. Schließlich beschreiben die Kapitel 9 und 10 Testergebnisse und Schlussfolgerungen sowie mögliche weiterführende Arbeiten.

Kapitel 2

Video-on-Demand Systeme

> "Television? The word is half Greek and half Latin. No good will come of this device."
>
> *C.P. Scott, 1936*

Ein Video-on-Demand System erlaubt seinen Nutzern, einen Katalog angebotenen Videomaterials zu durchsuchen und jederzeit ein bestimmtes Video aus diesem Angebot auszuwählen und abzuspielen, ohne dieses Video vorher auf dem verwendeten Endgerät abzuspeichern. Die Übertragung der Inhalte zum Endgerät des Nutzers erfolgt in Echtzeit mittels eines Video-Streams.

Obwohl zu Beginn als Ergänzung bzw. als Ersatz des herkömmlichen Fernsehens erdacht, ist Video-on-Demand heute vor allem aufgrund von Internetplattformen wie YouTube jedem ein Begriff. So verzeichnet die Streaming-Video-Plattform YouTube täglich 20 Millionen Zugriffe bei einer bisherigen kumulierten Spieldauer von 10.000 Jahren[1] [29]. Video-Server unterscheiden sich von herkömmlichen File- und Web-Servern einerseits durch die Zeitabhängigkeit des Mediums Video, woraus sich hohe Anforderungen an die Übertragungskapazität ergeben, da eine gewisse Bandbreite für die gesamte Dauer des Abspielvorgangs garantiert werden muss. Andererseits besitzen Videodateien in Spielfilmlänge auch bei Einsatz moderner Kompressionsverfahren eine beträchtliche Größe, was bedeutende Anforderungen an die Speicherkapazität eines Video-Servers stellt. Mit fortschreitender Durchsetzung von Breitbandinternetzugängen in den Haushalten und stetig fallenden Preisen für Massenspeicher stellt allerdings heutzutage hauptsächlich die Bandbreite der Netzwerkanbindung von Video-Servern die begrenzende Ressource bei der Auslieferung von Streaming Video dar. Es ist daher die Aufgabe jedes Video-on-Demand Systems, diese knappe Ressource so effizient wie möglich zu nutzen [23].

Die Forschung im Bereich der Video-on-Demand Systeme erstreckt sich über ein weites Feld, beginnend bei Fragen der Systemarchitektur, sowohl für einzelne Video-Server als auch für Verbundsysteme, über Speicherorganisation und Streaming-Protokolle bis hin zur Erforschung des Verhaltens von Nutzern derartiger Systeme, um Vorhersagen über temporale und geographische Zugriffsmuster treffen zu können.

Video-on-Demand Systeme werden in der Literatur anhand des Grades der den Nutzern des Systems ermöglichten Interaktivität kategorisiert [34, 35] (siehe Tabelle 2.1).

[1] Zahlen aus dem Jahr 2007

Bezeichnung	Beschreibung
No-VoD	Ähnlich herkömmlichem Fernsehen; Der Nutzer hat keinerlei Interaktionsmöglichkeiten.
PPV	*Pay-per-View*: Freischaltung von Inhalten nach vorheriger Bezahlung.
QVoD	*Quasi Video-on-Demand*: Freischaltung von Sendungen nach Gruppenzugehörigkeit des Nutzers. Geringer Grad an Interaktion durch Wechseln der Gruppe.
NVoD	*Near Video-on-Demand*: Das Vor- und Zurückspulen innerhalb einer Sendung ist in diskreten Zeitintervallen möglich. Dies kann durch zeitversetzte parallele Ausstrahlung derselben Inhalte erreicht werden.
TVoD	*True Video-on-Demand*: Der Nutzer hat volle Kontrolle über den Abspielvorgang, u.a. durch Vor- und Zurückspulen, Pausieren und Positionieren.

Tabelle 2.1: Klassifikation von VoD-Systemen laut [35]

Während *Pay-Per-View* (PPV), *Quasi Video-on-Demand* (QVoD) und *Near Video-on-Demand* (NVoD) noch durch mehrere parallel genutzte konventionelle Fernsehkanäle unter Verwendung eines speziellen Empfangsgeräts umgesetzt werden können [34], ist für die Umsetzung von *True Video-on-Demand* (TVoD) auch ein Rückkanal nötig, um Steuerbefehle des Nutzers an das VoD-System übermitteln zu können. Mit fortschreitendem Grad an Interaktivität ist der Nutzer zunehmend in der Lage, mit dem VoD-System wie mit einem herkömmlichen Videorecorder bzw. DVD-Player zu interagieren: Neben dem Pausieren des Abspielvorgangs ist auch das Vor- und Zurückspringen innerhalb eines Video-Streams möglich. Dies wird in der Literatur unter dem Stichwort *VCR Controls* bzw. *VCR Functionality* zusammengefasst [10, 20, 35].

Ein Teilbereich der Forschung zu VoD-Systemen konzentriert sich auf die Speicherarchitektur eines isoliert betrachteten Video-Servers [48, 49]. Forschungsgegenstand ist dabei die Verteilung der durch den Server verwalteten Video-Objekte auf eine Reihe von Speichermedien (sogenannte *Direct Access Storage Devices*) zur bestmöglichen Ausnutzung der vorhandenen I/O-Bandbreiten sowie zu Zwecken der Lastverteilung zwischen den eingesetzten Speichermedien, um einen größtmöglichen Durchsatz zu erreichen. Die dafür eingesetzten Techniken umfassen die Replikation von Video-Objekten auf verschiedene Speichermedien sowie das Striping von Video-Objekten über mehrere Speichermedien. Die Beschränkungen, die bei der tatsächlichen Auslieferung der Inhalte an die Nutzer durch die Bandbreite der Netzwerkschnittstelle entstehen, werden dabei teilweise außer Acht gelassen. Wang et al. beschreiben in [48] eine Heuristik zur Ermittlung der minimalen Anzahl von Speichermedien für eine gegebene Menge von Video-Objekten sowie eine gegebene Anzahl gleichzeitig auf diese Video-Objekte zu unterstützenden Zugriffe. Wolf et al. beschreiben in [49] ein Verfahren zur Lastverteilung zwischen Speichermedien durch Kombination von Striping und Replikation. Durch Einsatz einer Heuristik wird eine möglichst geringe Anzahl von Replikaten pro Video-Objekt ermittelt, die anschließend auf sogenannte *Disk-Striping Groups* verteilt werden.

Die integrierte Betrachtung der I/O-Bandbreiten des Speichersystems sowie der Bandbreite der Netzwerkschnittstelle eines Video-Servers und weiterer Server-Ressourcen führt

zum Begriff des sogenannten *Channels*. Ein Channel umfasst alle Ressourcen eines Servers, die notwendig sind, um das unterbrechungsfreie Abspielen eines Videos durch einen Nutzer sicherzustellen [10,20,35]. Die Anzahl der durch einen Video-Server zur Verfügung gestellten Channels stellt ein abstraktes Maß seiner Leistungsfähigkeit dar. In der einfachsten Variante zur Umsetzung von TVoD wird pro Nutzer und abgespieltem Video ein solcher Channel reserviert. Um Ressourcen zu sparen und mit einem einzigen Server möglichst viele Anfragen abdecken zu können, wird versucht, mehrere Anfragen unter Verwendung eines einzigen Channels zu bearbeiten. Dies geschieht durch Einreihung eintreffender Anfragen in eine Warteschlange sowie durch Einführung einer Wartezeit, des sogenannten *Batching Intervals*. Innerhalb dieses Wartezeitraums in die Warteschlange eingereihte Anfragen für das selbe Video-Objekt können durch einen einzigen Channel unter Verwendung von Multicast-Techniken bearbeitet werden [35]. Das Ziel dabei ist, eine Ausgewogenheit zwischen der dem Nutzer zugemuteten Wartezeit und der eingesparten Übertragungskapazität zu erreichen. Es existieren mehrere Varianten dieser sogenannten *Batching Policies*, zum Beispiel die von Dan et al. in [14] untersuchten Strategien *First Come First Served* und *Maximum Queue Length* sowie das von Aggerwal et. al. in [3] beschriebene *Maximum Factor Queue Length Batching*.

Ein ähnliches Problem ergibt sich bei der Verwendung von VCR Controls. Wird der Abspielvorgang gestoppt und später wieder aufgenommen bzw. vor- oder zurückgespult, muss die Übertragung aus dem aktuellen Batch ausgegliedert und mit möglichst geringer Wartezeit in einen neuen Batch eingegliedert werden. Li et al. beschreiben zu diesem Zweck in [33] das sogenannte *Split-and-Merge* Protokoll.

Jüngere Publikationen im Bereich Video-on-Demand widmen sich vor allem verteilten VoD-Server-Architekturen, sodass jeder beteiligte Server mehrere Video-Objekte zur Verfügung stellt und jedes Video-Objekt seinerseits durch einen- oder mehrere Server zur Verfügung gestellt wird [19,52]. Die dabei verwendete Technik der Replikation von Video-Objekten wird von vielen Autoren als dem serverübergreifenden Striping überlegen bezeichnet, da es sowohl zur Reduzierung der Komplexität als auch zu verbesserter Skalierbarkeit und Ausfallsicherheit durch Isolation der Server voneinander führt [12]. Die Anzahl der pro Video-Objekt notwendigen Replikate basiert auf einem sogenannten *Access Profile* [10,48], das pro Video-Objekt die maximal zu erwartende gleichzeitige Anzahl von Zugriffen beschreibt [52]. Je beliebter ein bestimmtes Video-Objekt ist und je größer dementsprechend die Anzahl parallel abzudeckender Zugriffe ist, umso größer muss auch die Anzahl auf verschiedenen Servern des Systems vorhandener Replikate sein, um alle zu erwartenden Zugriffe abdecken zu können. Gleichzeitig muss auf eine ausgewogene Platzierung der Replikate beliebter Videos geachtet werden, mit dem Ziel, einen Lastenausgleich zwischen den Servern des Systems zu erreichen. Entsprechende Verfahren finden sich in der Literatur z.B. unter den Stichworten *Predictive Placement* [46] und *Popularity-based Assignment/Placement* [12, 19, 35, 52].

Ein verwandtes Problem ist die strategische Platzierung von Video-Objekten auf Servern in geographischer Nähe der erwarteten Nutzer. Dieses Problem tritt vor allem bei hierarchisch organisierten und räumlich weit verteilten VoD-Systemen auf, in denen Server, die ein bestimmtes Video nicht besitzen, Anfragen an übergeordnete Server weiterleiten [22, 50].

Existieren von einem Video-Objekt mehrere Replikate auf verschiedenen Servern, ist es

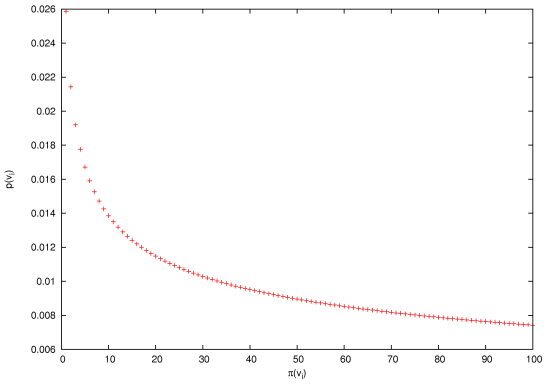


Abbildung 2.1: Zipf-artige Verteilung für 100 Videos mit $\theta = 0.271$

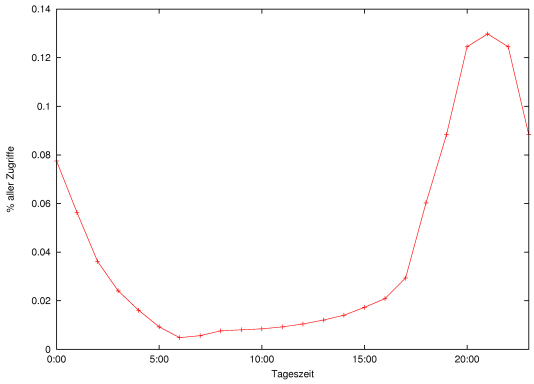


Abbildung 2.2: Daily Access Pattern [34]

weiters notwendig, bei Eintreffen einer Anfrage einen Server mit vorhandenen freien Ressourcen bzw. Channels auszuwählen [46]. Diese Aufgabe verkompliziert sich bei Berücksichtigung von Batching Policies, da nicht nur auf die vorhandenen freien Channels Rücksicht genommen werden muss, sondern auch auf die vorhandenen Warteschlangen und die Parameter der eingesetzten Batching Policy [24].

Weitere Arbeiten in diesem Forschungsfeld beschäftigen sich hauptsächlich mit dem Studium des Verhaltens von Nutzern von VoD-Systemen, um bessere Vorhersagen über die zu erwartenden Zugriffe machen zu können und daraus bessere Access Profiles abzuleiten [11, 22, 51]. Die Wahrscheinlichkeit, mit der ein bestimmtes Video durch einen Nutzer ausgewählt wird (Zugriffswahrscheinlichkeit), hängt in hohem Maß von seiner aktuellen Beliebtheit ab. In der Literatur wird zur Annäherung dieser Wahrscheinlichkeiten durchgängig eine sogenannte Zipf-artige Verteilung verwendet [14, 19, 20, 24, 35]:

$$p(v_i) = \frac{\pi(v_i)^{-\theta}}{\sum_{k=1}^{n} \pi(v_k)^{-\theta}} \qquad 0 \leq \theta \leq 1$$

wobei $\pi(v_i)$ den Rang von Video v_i in der aktuellen Beliebtheitsreihenfolge und θ den *Skew Factor* bezeichnet. Für VoD-Systeme, die Spielfilme anbieten, wird gemeinhin ein Skew Factor von $\theta = 0.271$ angenommen, da hier ähnlich einer Videothek einige wenige Videos für den Großteil aller Zugriffe verantwortlich sind [14,35]. Abbildung 2.1 zeigt eine solche Zipf-artige Verteilung für 100 Videos mit $\theta = 0.271$. Diese Annäherung der Zugriffswahrscheinlichkeiten eignet sich für eine kurzfristige Modellierung, ist aber nicht geeignet, um den gesamten Lebenszyklus eines Videos abzubilden, da die langfristige Zugriffswahrscheinlichkeit auch von zahlreichen externen Faktoren abhängt und dadurch regelmäßig wieder ansteigen kann [22].

Weiters wird versucht, Vorhersagen über den Einsatz der VCR Controls und die durchschnittliche Betrachungsdauer eines Videos (*Session Length*) zu treffen. Je länger eine solche Sitzung bereits dauert, um so wahrscheinlicher ist es, dass der Nutzer das jeweilige Video zu Ende sehen wird. Gleichzeitig steigt mit größerer Sitzungsdauer aber auch die Wahrscheinlichkeit einer Unterbrechung durch Verwendung von VCR Controls [11]. Durch Berücksichtigung dieser Daten kann gesteuert werden, welche Systemressourcen wie lange reserviert bleiben bzw. freigegeben werden, um eine größere Anzahl von Zugriffen ab-

wickeln zu können [51].

Um aufgrund von geschätzten Zugriffswahrscheinlichkeiten die Anzahl der voraussichtlich abzuwickelnden Zugriffe eines Video-Objekts ermitteln zu können, ist es weiters notwendig, die zeitliche Verteilung der eintreffenden Zugriffe zu modellieren (sog. "Daily Access Patterns" [51]). Diese schwankt üblicherweise im Laufe eines Tages stark: Für ein System, das Spielfilme anbietet, könnte diese Verteilung ähnlich wie in Abbildung 2.2 aussehen: Die Anzahl der gleichzeitig aktiven Zugriffe ist unter Tags moderat, steigt abends stark an und erreicht ihren Höhepunkt um rund 21:00 Uhr. Eine solche Verteilung muss nicht global gelten, sondern kann sich auch je nach Benutzergruppe und Genre unterscheiden.

Soweit dem Autor bekannt ist, existieren in der Literatur erst relativ wenige Ansätze, um das Problem der Bestimmung einer optimalen Menge von Replikaten und deren Zuordnung zu Servern mit Hilfe von Methoden der kombinatorischen Optimierung zu lösen. So beschreiben zum Beispiel Wang et al. in [48] eine Greedy-Heuristik sowie einen Branch-and-Bound-Alorithmus für die Ermittlung einer Menge von Replikaten und deren Platzierung, sodass ein gegebenenes Access Profile erfüllt werden kann. Wolf et al. beschreiben in [49] eine Heuristik zur Ermittlung von Replikaten und deren Platzierung auf sogenannten *Disk Striping Groups* auf der Grundlage eines gegebenen Access Profiles, sodass die den Disk Striping Groups zugeordneten *Forecast Loads* möglichst wenig von ihren Übertragungskapazitäten abweichen und die Speicherkapazitäten der Disk Striping Groups nicht überschritten werden. Zhou et al. beschreiben schließlich in [52] einen exakten sowie einen auf Simulated Annealing basierenden Algorithmus zur Bestimmung einer Menge von Replikaten und deren Bitraten sowie zur Zuordnung der Replikate zu Servern auf Basis eines gegebenen Access Profiles, sodass die durchschnittliche Anzahl von Replikaten sowie die durchschnittliche Bitrate maximiert und der sogenannte *Load Imbalance Degree* minimiert wird.

Die vorliegende Arbeit beschreibt einen Ansatz zur Lösung eines eng verwandten Problems, das neben der Bestimmung und Zuordnung von Replikaten auch die Zuordnung von Zugriffen eines vorgegebenen Access Profile zu den Replikaten umfasst, sodass die vorhandenen Server eines VoD-System entsprechend ihres Anteils an der gesamten Übertragungskapazität des Systems gerecht belastet werden. Der Ansatz besteht in der Kombination der Metaheuristik *Variable Neighbourhood Search* (siehe Kapitel 6) mit einer Formulierung als *Mixed Integer Program* (siehe Kapitel 4).

Kapitel 3

Problemstellung

3.1 Problembeschreibung

Das im Rahmen dieser Arbeit untersuchte Video-on-Demand System dient zur Auslieferung von Nachrichtenbeiträgen und Presse-Videos an registrierte Kunden. Diese können im Webbrowser eine Vorschau der Videoclips betrachten, diese in einem Online-Shop erwerben und danach in einer hochaufgelösten Version betrachten oder herunterladen.

Die Architektur des Systems folgt größtenteils der in [52] beschriebenen: Das System besteht aus einer zentralen Komponente, dem *Dispatcher*, der eintreffende Anfragen an eine Reihe von nachgeschalteten *Content-Servern* weiterleitet, welche die Anfragen direkt beantworten, sodass eine Überlastung des Dispatchers vermieden wird. Die Content-Server verfügen im Sinne einer *Shared-Nothing Architektur* [38] über jeweils eigene, voneinander getrennte Speichersysteme. Diese auch als *Distributed Storage* bezeichnete Speicherarchitektur vermeidet Engpässe, die durch den Zugriff auf ein zentrales Speichersystem enstehen könnten, isoliert die Server voneinander und führt zu höherer Skalierbarkeit und Ausfallsicherheit [52]. Abbildung 3.1 zeigt eine Übersicht über die verwendete Architektur. Der dem Dispatcher vorgeschaltete Web-Server dient zum Betrieb des Online-Shops und reicht Anfragen für Video-Clips an den Dispatcher weiter.

Die durch das System angebotenen Video-Clips stammen aus unterschiedlichen Quellen mit teilweise geringer Aufnahmequalität, wie zum Beispiel von Mini-Camcordern und Mo-

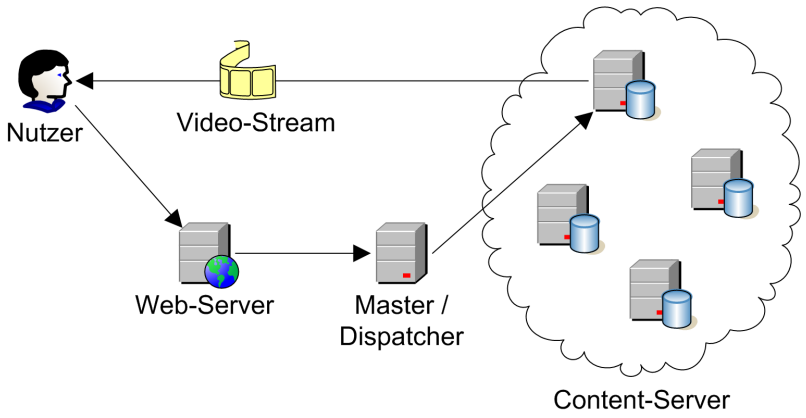


Abbildung 3.1: Überblick über die Architektur des betrachteten VoD-Systems

biltelefonen. Die Video-Clips liegen jeweils in drei verschiedenen Qualitäts-Versionen, im Folgenden *Datei-Typen* genannt, vor: *Thumbnails* dienen zur Vorschau in Suchergebnissen, *Previews* dienen zur Voransicht und als Entscheidungsgrundlage für den Kauf. Schließlich werden nur die *HiRes*-Versionen nach dem Kauf an die Kunden ausgeliefert. Die Thumbnail- und Preview-Versionen der Video-Clips werden mit einer einheitlichen Auflösung bzw. Bitrate angeboten, während die HiRes-Versionen in der originalen Auflösung bzw. Bitrate vorliegen. Aus praktischen Gründen kann keine einheitliche Länge der Video-Clips angenommen werden, wie dies teilweise in der Literatur getan wird [3, 46, 49, 52]. Weiters wird auch keine Einheitlichkeit der Speicher- und Übertragungskapazitäten der verwendeten Content-Server gefordert.

Um mit möglichst geringen Hardware- und Bandbreitenressourcen eine möglichst große Anzahl an gleichzeitigen Zugriffen abdecken zu können, soll für dieses System ein Verfahren zur Lastverteilung im Sinne von Kapitel 2 entwickelt werden: Ausgehend von einem gegebenen, täglich neu erstellten, auf Zugriffsstatistiken sowie einer Zipf-basierenden Schätzung der Zugriffswahrscheinlichkeiten beruhenden Access Profile (siehe Kapitel 2) soll pro Video-Objekt eine Menge von Replikaten und deren Platzierung auf den vorhandenen Content-Servern ermittelt werden. Weiters ist eine Zuordnung der laut Access Profile zu erwarteten Zugriffe zu den platzierten Replikaten gesucht. Jede Zuordnung eines Zugriffs erhöht die *Last* des jeweiligen Content-Servers um die Bitrate des gewünschten Video-Clips[1]. Die Zuordnung der Zugriffe soll so geschehen, dass jeder der beteiligten Content-Server entsprechend seinem Anteil an der gesamten Uploadkapazität des Systems fair belastet wird.

Die Zuordnung der Zugriffe zu den vorhandenen Content-Servern bildet für die Dispatcher-Komponente die Grundlage für die Entscheidung, welcher Content-Server bei Eintreffen eines Zugriffs auf ein bestimmtes Video-Objekt zur Bearbeitung des Zugriffs ausgewählt wird: Verarbeitet Server j $Q(i,j)$ Zugriffe von insgesamt q_i Zugriffen auf Video-Objekt i, so wird Server j mit einer Wahrscheinlichkeit von $\frac{Q(i,j)}{q_i}$ zur Bearbeitung eines Zugriffs auf i gewählt. Die Entscheidung für einen Content-Server hängt weiters auch von der verwendeten Batching-Strategie und der aktuellen Server-Last ab. Alternativ ist auch die Verwendung einer Round-Robin-Strategie denkbar, sodass bei Eintreffen von Zugriffen auf Video-Objekt i der Reihe nach jeder Server gewählt wird, der ein Replikat von i besitzt. In diesem Fall muss auch jedem dieser Server der selbe Anteil an den gesamten Zugriffen q_i zugewiesen werden.

Die Lastverteilungsberechnung muss einerseits auf die Speicherkapazitäten der Content-Server achten: Kein Server darf eine Menge von Replikaten erhalten, deren Gesamtgröße seine Speicherkapazität übersteigt. Weiters ist zu beachten, dass die durch das Verfahren ermittelte Replikatszuordnung aus der aktuell bestehenden Replikatszuordnung hergestellt werden muss, indem nicht mehr benötigte Replikate gelöscht und bisher noch nicht vorhandene neue Replikate von Servern, die sie bereits besitzen, übertragen werden müssen. Die Dauer dieser an die Zuordnungsberechnung anschließende *Reorganisationsphase* hängt von der Anzahl und der Größe der zu übertragenden Replikate ab. Für die Berechnung der Dauer der Reorganisationsphase wird immer von einer Worst-Case-Situation ausgegangen, in der keinerlei Parallelisierung stattfinden kann und alle Übertragungen streng sequentiell durchgeführt werden müssen.

[1] Es wird davon ausgegegangen, dass im Worst Case alle Zugriffe parallel aktiv sind und keine Batching-Strategie verwendet wird bzw. verwendet werden kann

Um Situationen zu vermeiden, in denen zuerst nicht mehr benötigte Replikate gelöscht oder verschoben werden müssten, um Platz für eingehende Replikate zu schaffen, darf die Gesamtgröße aller eingehenden Replikate eines Content-Servers seine freie Speicherkapazität vor Beginn der Zuordnungsberechnung nicht übersteigen. Nur so können potentielle Deadlock-Situationen während der Reorganisationsphase vermieden werden.

Als zusätzliche Einschränkung kommt hinzu, dass pro Content-Server Einschränkungen bezüglich der akzeptierten Menge von Datei-Typen definiert werden können: Replikate von Video-Clips dürfen nur auf einem Content-Server platziert werden, wenn der Typ des Video-Clips in der Menge der vom Content-Server akzeptierten Datei-Typen enthalten ist.

Das Ziel der Lastverteilungsberechnung ist die Herstellung einer möglichst guten Last-Balancierung, ausgedrückt durch möglichst geringe summierte Abweichungen der Server-Lasten von den sogenannten Last-Zielwerten, bei einer gleichzeitig möglichst kurzen Dauer der Reorganisationsphase.

3.2 Formalisierung

Dieser Abschnitt beschreibt die mathematische Formalisierung der informellen Problembeschreibung des vorhergehenden Abschnitts, auf die sich die restlichen Ausführungen in dieser Arbeit beziehen. Wie beschrieben erfolgt die Lastverteilungsberechnung auf Basis einer bestehenden Zuordnung von Replikaten und Zugriffen. Zur Vereinfachung der Notation werden alle Symbole, die sich auf die bisherige Zuordnung beziehen, mit einem Oberstrich versehen. Beschreibt also x eine Größe, die sich auf die neu zu ermittelnde Zuordnung bezieht, beschreibt $\overline{x}$ die entsprechende Größe in der bisherigen Zuordnung.

3.2.1 Entwicklung der Problemformalisierung

Das im weiteren Verlauf *Video-Server Load Re-Balancing* (VSLRB) genannte Problem der Ermittlung von Replikaten und der Verteilung von Zugriffen ist wie folgt definiert:

Gegeben sei eine Menge F von n Video-Objekten sowie eine Menge C von m Servern. Jedes Video-Objekt $i \in F$ besitzt die folgenden grundlegenden Eigenschaften:

w_i	Größe
t_i	Datei-Typ, $t_i \in \{Thumbnail, Preview, HiRes\}$
b_i	Bitrate
q_i	Anzahl der maximal gleichzeitig abzuwickelnden Zugriffe laut aktuellem Access Profile
$\overline{q}_i$	Anzahl der maximal gleichzeitig abzuwickelnden Zugriffe laut bisherigem Access Profile

Jeder Server $j \in C$ besitzt die folgenden grundlegenden Eigenschaften:

W_j	Speicherkapazität
D_j	Bandbreite Download
U_j	Bandbreite Upload
T_j	Menge akzeptierter Dateitypen $\subseteq \{Thumbnail, Preview, HiRes\}$
η_j	Last-Zielwert

Aus den Eigenschaften U_j und D_j ergibt sich die Matrix c der maximal möglichen Übertragungsgeschwindigkeiten von einem Server k zu jedem anderen Server l:

$$c_{kl} = \begin{cases} \min(U_k, D_l) & \text{wenn } k \neq l \\ \infty & \text{sonst} \end{cases}$$

Die weiteren Formalisierungen beruhen auf den Begriffen *Replikat* und *Zuweisung*:

Definition 3.2.1. Als *Replikat* wird in dieser Arbeit ein Video-Objekt $i \in F$ bezeichnet, das auf einem Server $j \in C$ abgelegt wird. Die Erzeugung eines Replikats ist nur dann möglich, wenn $t_i \in T_j$.

Die Menge aller auf einem Server j abgelegten Replikate wird als F_j bezeichnet. Die Aufnahme eines Replikats eines Video-Objekts i in F_j verringert die verfügbare Speicherkapazität von Server j um w_i. Für die F_j muss gelten, dass $\bigcup_{j \in C} F_j = F$, d.h. es darf kein Video-Objekt unberücksichtigt bleiben. Analog zu den F_j kann als $C_i,\ i \in F$ die Menge aller Server definiert werden, auf denen ein Replikat von Video-Objekt i vorhanden ist:

$$C_i = \{j \in C \mid i \in F_j\} \quad \forall i \in F$$

Weiters bezeichnet $A_i\ i \in F$ die Menge der Server, auf denen Datei i abgelegt werden darf:

$$A_i = \{j \in C \mid t_i \in T_j\} \quad \forall i \in F$$

Definition 3.2.2. Als *Zuweisung* wird in dieser Arbeit eine Zuordnung einer Anzahl von Zugriffen > 0 des Access Profiles auf eine Datei $i \in F$ zu einem Content-Server $j \in C$ bezeichnet, die durch j abgewickelt werden. Eine solche Zuweisung ist nur dann möglich, wenn $i \in F_j$.

Die Gesamtheit aller Zuweisungen kann als eine Funktion $Q : F \times C \to \mathbb{N}_0$, aufgefasst werden, die jedem Paar von Video-Objekten und Servern eine Anzahl abgedeckter Zugriffe zuordnet, wobei

$$Q(i,j) = \begin{cases} > 0 & \text{wenn } i \in F_j \\ 0 & \text{sonst} \end{cases}$$

Eine Zuweisung von Zugriffen auf Video-Objekt i zu Server j darf also nur dann vorgenommen werden, wenn dieser ein Replikat von i besitzt. Umgekehrt darf ein solches Replikat nur dann auf j existieren, falls tatsächlich eine Zuweisung von Zugriffen auf i erfolgt. Für Q muss weiters gelten dass $\sum_{j \in C_i} Q(i,j) = q_i\ \forall i \in F$, d.h. es müssen alle Zugriffe des Access Profiles zugewiesen werden. Aufgrund der vorhandenen Zuweisungen kann die Last eines Servers definiert werden:

Definition 3.2.3 (Last)**.** Die aktuelle *Last* eines Servers j aufgrund der ihm zugewiesenen Zugriffe ist definiert als $\mathcal{L}(j) = \sum_{i \in F_j} b_i Q(i,j)$, da alle Zugriffe im Worst Case gleichzeitig aktiv sein können.

Weiters macht die Aufnahme eines Replikats von Video-Objekt i in F_j die Übertragung von i nach j notwendig, falls $i \notin \overline{F}_j$. Die Dauer dieser Übertragung kann abgeschätzt werden mit

$$T(i,j) = \sum_{k \in C} \mathcal{T}(i,k,j), \qquad \mathcal{T}(i,k,j) = \tfrac{1}{\overline{q}_i} \overline{Q}(i,k) w_i \tfrac{1}{c_{kj}}$$

In dieser idealisierten Berechnung der Übertragungsdauer übernimmt jeder Content-Server k, der bereits ein Replikat von Video-Objekt i besitzt, jenen Teil an der zu übertragenden Datenmenge, der seinem bisherigen Lastanteil an i, $\frac{1}{\overline{q}_i}\overline{Q}(i,k)$, entspricht. $\mathcal{T}(i,k,j)$ bezeichnet dabei die Dauer der Übertragung dieses Anteils.

Um eine Lösung einer Instanz von VSLRB zu ermitteln, müssen sowohl die Mengen $F_j,\ j \in C$ als auch die Zuweisungs-Funktion Q ermittelt werden. Dabei ist die folgende Zielfunktion zu minimieren:

$$\begin{aligned} Z &= \alpha Z_1 + \beta Z_2 \\ Z_1 &= \sum_{j \in C} \left| \eta_j - \mathcal{L}(j) \right| \\ Z_2 &= \sum_{k \in C} \sum_{\substack{l \in C \\ l \neq k}} \sum_{\substack{i \in (F_l \setminus \overline{F}_l) \\ \cap \overline{F}_k}} \mathcal{T}(i,k,l) \end{aligned}$$

Der erste Teil der Zielfunktion, Z_1, beschreibt den Grad der Balanciertheit der Lösung durch Summation der Abweichungen der Serverlasten von den jeweiligen Last-Zielwerten η_j. Die Formalisierung als Summe von absoluten Abweichungen anstatt als Summe quadrierter Lasten wurde gewählt, um die Formalisierung als lineares Programm umsetzen zu können (siehe Abschnitte 5.1 und 5.2). Durch geeignete Wahl der η_j können aber auch bei linearer Bestrafung von Abweichungen vergleichbare Resultate erzielt werden. Für Details zur Berechnung der η_j siehe Abschnitt 3.2.3.

Der zweite Teil der Zielfunktion, Z_2, beschreibt die Dauer der Reorganisationsphase im Worst Case, d.h. wenn keine Parallelisierung der Übertragungen stattfindet. Für jedes Paar aus Quell- und Zielservern $(k,l) \in C \times C$ wird für jedes Replikat, das sich bereits auf k befindet und auf l neu hinzukommt, die idealisierte Übertragungsdauer der Datenmenge aufsummiert, die von k nach l übertragen werden muss.

Die Parameter $\alpha, \beta \in \mathbb{R}$ können zur unterschiedlichen Gewichtung der beiden Zielfunktionsteile verwendet werden. Im weiteren Verlauf dieser Arbeit wird sowohl $\alpha = 1$ als auch $\beta = 1$ angenommen.

Zusätzlich muss jede zulässige Lösung einer Instanz von VSLRB die folgenden Nebenbedingungen erfüllen: Einerseits muss die Speicherkapazität jedes Content-Servers eingehalten werden:

$$\sum_{i \in F_j} w_i \leq W_j \qquad \forall j \in C$$

Weiters muss die im vorhergehenden Abschnitt beschriebene Beschränkung der eingehenden Datenmenge eingehalten werden:

$$\sum_{i \in F_j \setminus \overline{F}_j} w_i \leq W_j - \sum_{i \in \overline{F}_j} w_i \qquad \forall j \in C$$

Eine weitere Nebenbedingung ergibt sich, falls die Dispatcher-Komponente des Systems im Round-Robin-Betrieb verwendet wird. In diesem Fall muss jeder Content-Server, der einen Anteil an den Zugriffen eines Video-Objekts i übernimmt, den selben Anteil übernehmen wie alle anderen Content-Server, die Zugriffe auf i übernehmen. Diese Bedingung kann auf folgende Weise ausgedrückt werden:

$$|Q(i,j) - Q(i,k)| \leq 1 \qquad \forall i \in F,\ \forall j,k \in C_i,\ j \neq k$$

Bei ganzzahliger Aufteilung von q_i Zugriffen auf l Server kann immer eine Zuordnung gefunden werden, sodass die maximale Abweichung zwischen zwei Zuweisungen genau eins beträgt: Der größtmögliche Rest $q_i \mod l$ der Division q_i durch l beträgt $l-1$. Diese $l-1$ Zugriffe können wiederum gleichmäßig auf $l-1$ der l Server verteilt werden, sodass die maximale Abweichung zwischen zwei Zuweisungen genau eins beträgt.

3.2.2 Zusammenfassung der Formalisierung von VSLRB

Gegeben sei eine Menge F von n Video-Objekten sowie eine Menge C von m Servern. Ermittle die m Mengen F_j sowie eine Zuweisungsfunktion $Q : F \times C \to \mathbb{N}_0$ durch Lösung des folgenden Minimierungsproblems:

$$\min Z = \alpha Z_1 + \beta Z_2$$

$$Z_1 = \sum_{j \in C} \left| \eta_j - \mathcal{L}(j) \right|$$

$$Z_2 = \sum_{k \in C} \sum_{\substack{l \in C \\ l \neq k}} \sum_{\substack{i \in (F_l \setminus \overline{F}_l) \\ \cap \overline{F}_k}} \mathcal{T}(i,k,l)$$

$$\mathcal{L}(j) = \sum_{i \in F_j} b_i Q(i,j) \tag{3.1}$$

$$\mathcal{T}(i,k,j) = \tfrac{1}{\overline{q}_i} \overline{Q}(i,k) w_i \tfrac{1}{c_{kj}} \tag{3.2}$$

unter den Nebenbedingungen

$$\bigcup_{j \in C} F_j = F \tag{3.3}$$

$$F_j \subseteq \{ i \in F \mid t_i \in T_j \} \qquad \forall j \in C \tag{3.4}$$

$$\sum_{j \in C_i} Q(i,j) = q_i \qquad \forall i \in F \tag{3.5}$$

$$i \in F_j \Leftrightarrow Q(i,j) > 0 \qquad \forall (i,j) \in F \times C \tag{3.6}$$

$$i \notin F_j \Leftrightarrow Q(i,j) = 0 \qquad \forall (i,j) \in F \times C \tag{3.7}$$

$$\sum_{i \in F_j} w_i \leq W_j \qquad \forall j \in C \tag{3.8}$$

$$\sum_{i \in F_j \setminus \overline{F}_j} w_i \leq W_j - \sum_{i \in \overline{F}_j} w_i \qquad \forall j \in C \tag{3.9}$$

und der optionalen Nebenbedingung

$$|Q(i,j) - Q(i,k)| \leq 1 \qquad \forall i \in F,\, \forall j,k \in C_i, j \neq k \tag{3.10}$$

3.2.3 Berechnung der Last-Zielwerte

Die Berechnung der Last-Zielwerte η_j beruht unter anderem auf den folgenden drei Begriffen:

Definition 3.2.4 (Gesamtlast)**.** Als *Gesamtlast* L wird jene Last bezeichnet, die entsteht, wenn alle Zugriffe des Access Profiles parallel aktiv sind:

$$L = \sum_{i \in F} b_i q_i$$

Definition 3.2.5 (Dateityplast)**.** Als *Dateityplast* L_t wird jene Last bezeichnet, die entsteht, wenn alle Zugriffe auf Video-Objekte mit Typ $t \in \{Thumbnail, Preview, HiRes\}$ gleichzeitig aktiv sind:

$$L_t = \sum_{\{i \in F \,|\, t_i = t\}} b_i q_i$$

Definition 3.2.6 (Faire Last)**.** Als *Faire Last* Λ_j eines Server $j \in C$ wird der Anteil an der durch das Access Profile vorgegebenen Gesamtlast L proportional zum Anteil der Uploadkapazität U_j an der gesamten Upload-Kapazität des Systems bezeichnet.

$$\Lambda_j = \frac{U_j}{\sum_{k=1}^{m} U_k} L$$

Definition 3.2.7 (Optimal balancierte Lösung)**.** Eine Lösung einer Instanz von VSLRB wird als *optimal balanciert* bezeichnet, wenn $\sum_{j \in C} (\mathcal{L}(j) - \Lambda_j)^2$ minimal ist, d.h. wenn die Serverlasten möglichst wenig von den fairen Lasten abweichen.

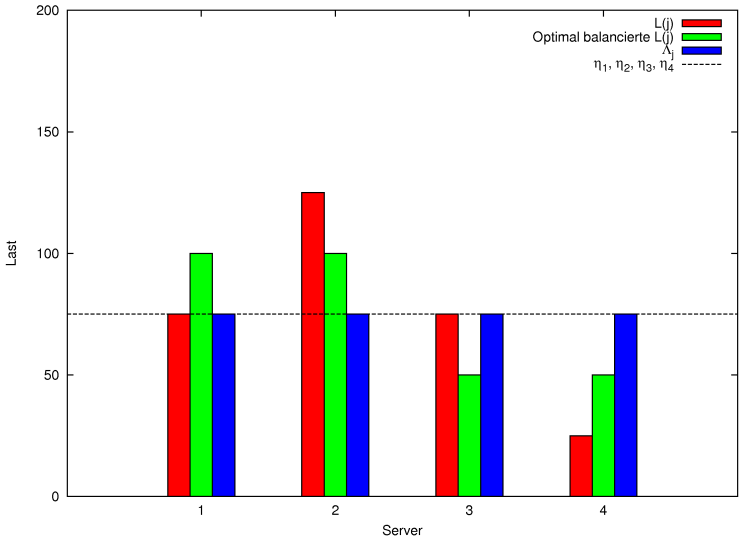
Da Abweichungen von den Last-Zielwerten η_j in der Zielfunktion linear bestraft werden, kommt der Berechnung der η_j besondere Bedeutung für die Lösungsqualität zu. Diese müssen die Eigenschaft besitzen, die Server-Lasten der unbekannten Optimallösung möglichst gut anzunähern. Zur Illustration dieser Problematik soll das in Abbildung 3.2 dargestellte Beispiel dienen. Gegeben sei ein System mit vier Content-Servern, den beiden Dateitypen A und B mit Dateityplast $L_A = 200$ und $L_B = 100$ und damit einer Gesamtlast von $L = L_A + L_B = 300$.

Würde $\eta_j = \Lambda_j = \frac{L}{4}$ gewählt, könnte eine mögliche Lösung wie die in Abbildung 3.2b dargestellt aussehen: Aufgrund der linearen Bestrafung der Abweichungen von den Last-Zielwerten in der Zielfunktion besitzt diese Lösung den selben Zielfunktionswert wie die optimal balancierte Lösung (in Abbildung 3.2 durch grüne Balken dargestellt), die dadurch nicht notwendigerweise erreicht werden muss. Werden die η_j hingegen wie in Abbildung 3.2c unter Berücksichtigung der akzeptierten Datei-Typen gewählt, wird jede Abweichung von der optimalen Balancierung in der Zielfunktion bestraft.

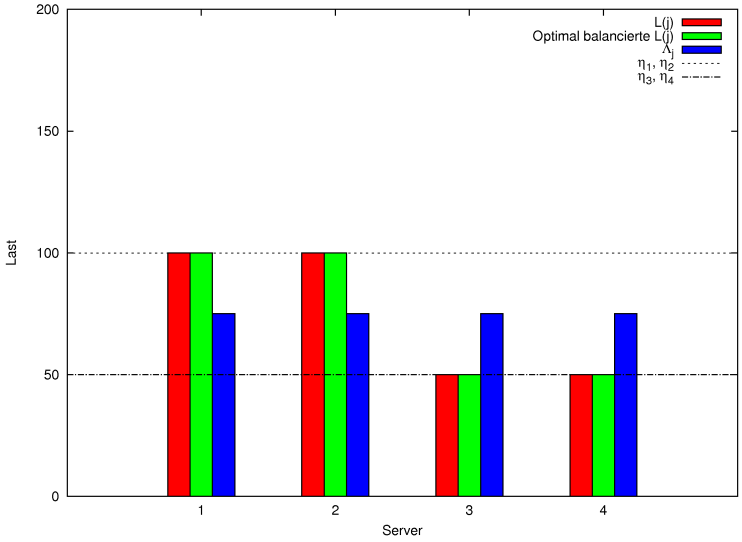
Die Berechnung der η_j verkompliziert sich bei Berücksichtigung nicht-uniformer Upload-Kapazitäten sowie bei Überschneidungen der Mengen der akzeptierten Datei-Typen. Um

Server j	1	2	3	4
T_j	$\{A\}$	$\{A\}$	$\{B\}$	$\{B\}$
Erlaubte Last aufgrund T_j	200	200	100	100
U_j	300	300	300	300
Λ_j	75	75	75	75

(a) Vereinfachte Beispielinstanz von VSLRB, $T_A = 200$, $T_B = 100$



(b) Lösung bei ungeeigneter Wahl der η_j



(c) Lösung bei individueller Berechnung der η_j basierend auf der jeweils akzeptierten Datei-Typen

Abbildung 3.2: Beispielinstanz von VSLRB mit zwei Lösungen bei unterschiedlicher Berechnungsweise der η_j

eine gerechte Belastung zu erreichen, sollen die zu bestimmenden η_j einerseits möglichst wenig von den fairen Lasten Λ_j abweichen. Andererseits darf, wie im obigen Beispiel illustriert, nur Last von Video-Objekten berücksichtigt werden, die auch auf Server j platziert werden können. Um beide Ziele zu erfüllen, werden die η_j durch Lösung des folgenden Minimierungsproblems bestimmt:

Bestimme η_j und x_j^t durch Minimierung des folgenden Ausdrucks:

$$\min \sum_{j \in C} (\Lambda_j - \eta_j)^2$$

wobei

$$\eta_j = \sum_{t \in T_j} x_j^t L_t \quad \forall j \in C$$

unter den Nebenbedingungen

$$\sum_{\{j | t \in T_j\}} x_j^t = 1 \quad \forall t \in T = \bigcup_{j \in C} T_j$$

und

$$\eta_j, x_j^t \geq 0 \quad \forall j \in C, t \in T$$

Die x_j^t bezeichnen dabei den Anteil der Dateityplast L_t, den Server j übernehmen soll. Diese Minimierungsaufgabe ist ein konvexes Quadratisches Programm mit $\leq 4m$ Variablen und $\leq 4m + 3$ Nebenbedingungen und wird bei der Erzeugung einer Probleminstanz von VSLRB mit Hilfe der kommerziellen Solver-Software CPLEX bzw. des freien Pakets CvxOpt [13] zur Lösung konvexer linearer und quadratischer Optimierungsprobleme gelöst.

Der Wert $L_t x_j^t$, der den durch die Berechnung der η_j implizit vorgegebenen Last-Zielwert für Zugriffe auf Video-Objekte vom Typ t beschreibt, wird im Folgenden verkürzend als η_j^t bezeichnet. Analog bezeichnet im weiteren Verlauf $\mathcal{L}_t(j)$ die gesamte Last eines Servers j von Zugriffen auf Video-Objekte vom Typ t.

3.2.4 Zusammenfassung der Notation

Im weiteren Verlauf dieser Arbeit werden die folgenden, in den vorangegangen Unterabschnitten eingeführten Symbole verwendet:

Symbole, die sich auf Video-Objekte beziehen:

w_i	Größe
t_i	Datei-Typ, $t_i \in \{Thumbnail, Preview, HiRes\}$
b_i	Bitrate
q_i	Anzahl der maximal gleichzeitig abzuwickelnden Zugriffe während laut aktuellem Lastprofil
C_i	Menge der Server $\subseteq C$, die ein Replikat von i besitzen
A_i	Menge der Server $\subseteq C$, auf denen i abgelegt werden darf

Symbole, die sich auf Server beziehen:

W_j	Speicherkapazität
D_j	Bandbreite Download
U_j	Bandbreite Upload
T_j	Menge der akzeptierten Dateitypen $\subseteq \{Thumbnail, Preview, HiRes\}$
Λ_j	Faire Last von Server j proportional zum Anteil seiner Upload-Kapazität an der gesamten Uploadkapazität des Systems
η_j	Last-Zielwert
η_j^t	Impliziter Last-Zielwert für Video-Objekte vom Typ t
F_j	Menge der Replikate $\subseteq F$, die Server j besitzt
$\mathcal{L}(j)$	Last von Server j
$\mathcal{L}_t(j)$	Last von Server j, eingeschränkt auf Video-Objekte vom Typ t

Weitere Symbole:

T	Menge aller Datei-Typen $= \{HiRes, Preview, Thumbnail\}$
L	Gesamt-Last aller Video-Objekte
L_t	Gesamt-Last aller Dateien vom Typ t (Dateityplast)
$Q(i,j)$	Zuweisung von Zugriffen auf Video-Objekt i zu Server j
$\mathcal{T}(i,k,j)$	Idealisierte Übertragungszeit der Datenmenge, die bei der Übertragung eines Replikats von Video-Objekt i zu Server j von Server k übertragen werden muss
$T(i,j)$	Idealisierte Übertragungszeit eines Replikats von Video-Objekt i auf Server j

3.3 $\mathcal{NP}$-Vollständigkeit

Das in Abschnitt 3.2 beschriebene Problem VSLRB ist $\mathcal{NP}$-vollständig, d.h. sofern $\mathcal{P} \neq \mathcal{NP}$ existiert kein Algorithmus zur Ermittlung einer beweisbar optimalen Lösung in polynomieller Zeit. Der Beweis der $\mathcal{NP}$-Vollständigkeit von VSLRB geschieht durch Reduktion des SUBSET SUM-Problems:

Definition 3.3.1. SUBSET SUM-Problem

Gegeben:	Endliche Menge A, Größe $s(a) \in \mathbb{Z}^+$ für alle $a \in A$ sowie $B \in \mathbb{Z}^+$
Frage:	Existiert Teilmenge $A' \subseteq A$, sodass $\sum_{a \in A'} s(a) = B$?

Satz 3.3.2. *SUBSET SUM ist $\mathcal{NP}$-vollständig*

Beweis. Siehe [18] □

Definition 3.3.3. Entscheidungsvariante von VSLRB

Gegeben:	Instanz von VSLRB, $K \in \mathbb{Z}$
Frage:	Existiert eine Lösung mit Zielfunktionswert $\leq K$?

Satz 3.3.4. *Die Entscheidungsvariante von VSLRB ist $\mathcal{NP}$-vollständig*

Beweis. Gegeben sei eine beliebige Instanz von SUBSET SUM. Konstruiere daraus auf folgende Weise eine Instanz der Entscheidungsvariante von VSLRB:

Setze $\alpha = 1$, $\beta = 0$. Für jedes Element $a \in A$ erzeuge Datei i mit $q_i = 1$, $b_i = s(a)$, $w_i = 1$, $t_i = HiRes$.

Erzeuge zwei Server: $\eta_1 = B$, $W_1 = |A|$, U_1, D_1 beliebig, $T_1 = \{HiRes\}$ und $\eta_2 = \sum_{a \in A} s(a) - B$, $W_2 = |A|$, U_2, D_2 beliebig, $T_2 = \{HiRes\}$. Erzeuge beliebige bestehende Datei-Zuweisungen $\overline{F}_1$ und $\overline{F}_2$ sowie eine entsprechende Zuordnung von Zugriffen $\overline{Q}(i,j)$ und setze $K = 0$.

Aufgrund von $q_i = 1 \ \forall i \in F$ ist sichergestellt, dass es in einer zulässigen Lösung dieser Instanz zu keiner Zuweisung einer Datei zu mehr als einem Server kommt. Dadurch und aufgrund von $W_1 = W_2 = |A|$ und $w_i = 1 \ \forall i \in F$ ist weiters sichergestellt, dass die Nebenbedingungen 3.8 und 3.9 immer erfüllt sind.

Aufgrund der Konstruktion der VSLRB-Instanz lautet die Zielfunktion:

$$
\begin{aligned}
Z &= \left|\eta_1 - \mathcal{L}(1)\right| + \left|\eta_2 - \mathcal{L}(2)\right| \\
&= \left|B - \sum_{i \in F_1} Q(i,1) b_i\right| + \left|\sum_{a \in A} s(a) - B - \sum_{i \in F_2} Q(i,2) b_i\right| \\
&= \left|B - \sum_{i \in F_1} b_i\right| + \left|\sum_{a \in A} s(a) - B - \sum_{i \in F_2} b_i\right| \\
&= \left|B - \sum_{i \in F_1} s(a)\right| + \left|\sum_{a \in A} s(a) - B - \sum_{i \in F_2} s(a)\right| \\
&= 2\left|B - \sum_{i \in F_1} s(a)\right|
\end{aligned}
$$

Wenn ein Algorithmus die Fragestellung der auf diese Weise konstruierten Instanz der Entscheidungsvariante von VSLRB positiv beantwortet, dann gilt dies auch für die Instanz von SUBSET SUM: wegen $Z \geq 0$ und $K = 0$ existiert eine Lösung mit $Z = 0$ für die gilt: $B = \sum_{i \in F_1} s(a)$. Daher existiert mit $F_1 = A'$ auch eine Lösung, welche die Fragestellung von SUBSET SUM positiv beantwortet. □

Kapitel 4

Lineare Programmierung

Die lineare Programmierung ist ein wichtiger Teilbereich des weiten Felds des Operations Research, das sich mit der Entwicklung mathematischer Methoden zur Entscheidungsfindung in technischen und betriebswirtschaftlichen Problemstellungen befasst [32]. Der Begriff "Programmierung" ist in diesem Zusammenhang im Sinne von "Planung" zu verstehen.

Mathematisch gesehen befasst sich die lineare Programmierung mit Verfahren zur Lösung linearer Extremwertprobleme mit Nebenbedingungen, die in diesem Kontext als *lineare Programme* bzw. *linear programs* (LPs) bezeichnet werden. Große Bedeutung besitzt der Mitte des 20. Jahrhunderts von G.B. Dantzig entwickelte *Simplex-Algorithmus* [32], der in der Praxis sehr erfolgreich zur Lösung linearer Programme eingesetzt wird, obwohl er theoretisch eine exponentielle Worst-Case Laufzeit besitzt. Weiters existieren auch Algorithmen zur Lösung linearer Programme mit polynomieller Worst-Case Laufzeit, wie die *Ellipsoid-Methode* und *Innere-Punkte-Verfahren* [9].

Einen wichtigen Spezialfall stellen lineare Optimierungsprobleme dar, deren Lösungen auf ganzzahlige Werte beschränkt sind. Diese eignen sich zur Modellierung einer Vielzahl von kombinatorischen Optimierungsproblemen. Man spricht in diesem Fall von *ganzzahliger linearer Programmierung* bzw. *ganzzahligen linearen Programmen* (engl.: *integer linear programs* - ILPs). Während die Lösung von linearen Programmen noch in effizienter Weise möglich ist, stellt die Lösung von ganzzahligen linearen Programmen zumeist ein $\mathcal{NP}$-schweres Problem dar.

Die folgenden beiden Abschnitte geben einen kurzen Überblick über einige Aspekte der Theorie der linearen Programmierung sowie über Lösungsverfahren für ganzzahlige lineare Programme. Die Ausführungen orientieren sich größtenteils an dem Werk *Introduction to Linear Optimization* von Bertsimas und Tsitsiklis [9].

4.1 Grundzüge der Linearen Programmierung

Das allgemeine Problem der linearen Programmierung besteht in der Minimierung bzw. Maximierung einer linearen Zielfunktion unter linearen Nebenbedingungen. Es kann gemäß [9] auf folgende Weise angeschrieben werden:

$$
\begin{array}{rll}
\min/\max & \mathbf{c}'\mathbf{x} & \\
\text{unter} & \mathbf{a}_i'\mathbf{x} \geq b_i & i \in M_1, \\
& \mathbf{a}_i'\mathbf{x} \leq b_i & i \in M_2, \\
& \mathbf{a}_i'\mathbf{x} = b_i & i \in M_3, \\
& x_j \geq 0 & j \in N_1, \\
& x_j \leq 0 & j \in N_2.
\end{array}
\tag{4.1}
$$

Gesucht ist ein Vektor $\mathbf{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n$, der die Zielfunktion $\mathbf{c}'\mathbf{x} = \sum_{i=1}^{n} c_i x_i$ minimiert. Die Komponenten $x_1, \ldots, x_n$ dieses Vektors werden als *Entscheidungsvariablen* bezeichnet. Der Vektor $\mathbf{c} \in \mathbb{R}^n$, welcher die Zielfunktion vorgibt, wird auch als *Kostenvektor* bezeichnet.

Die Definition der Nebenbedingungen erfolgt mit Hilfe dreier paarweise disjunkter Indexmengen M_1, M_2 und M_3. Für jedes $i \in M_1, M_2, M_3$ ist ein Skalar b_i sowie ein Vektor $\mathbf{a}_i \in \mathbb{R}^n$ vorgegeben, welche die i-te Nebenbedingung $\mathbf{a}_i'\mathbf{x} \circ_i b_i$ angeben, wobei

$$
\circ_i = \begin{cases} \geq & \text{falls } i \in M_1 \\ \leq & \text{falls } i \in M_2 \\ = & \text{falls } i \in M_3 \end{cases}
$$

Die Mengen $N_1, N_2 \subseteq \{1 \ldots n\},\ N_1 \cap N_2 = \emptyset$ geben weiters die Indizes jener Variablen an, die auf positive bzw. negative Werte beschränkt sind. Ist ein Index j weder in N_1 noch in N_2 enthalten, wird x_j als *freie Variable* bezeichnet.

Die vorhandenen Nebenbedingungen definieren die Menge aller *zulässigen* Vektoren. Die Vereinigungsmenge aller zulässigen Vektoren wird auch als *zulässiger Bereich* bezeichnet [32]. Weiters wird ein zulässiger Vektor $\mathbf{x}^*$ als *optimal* bezeichnet, wenn kein anderer zulässiger Vektor $\mathbf{x} \neq \mathbf{x}^*$ mit $\mathbf{c}'\mathbf{x} < \mathbf{c}'\mathbf{x}^*$ (bei Minimierung) bzw. $\mathbf{c}'\mathbf{x} > \mathbf{c}'\mathbf{x}^*$ (bei Maximierung) existiert.

Jedes Problem der linearen Programmierung kann in ein äquivalentes Minimierungsproblem unter Nebenbedingungen der Form $\mathbf{a}_i'\mathbf{x} \geq b_i$ umgewandelt werden, indem die folgenden Transformationsschritte angewendet werden:

1. Falls ein Maximierungsproblem vorliegt, multipliziere $\mathbf{c}$ mit -1, um ein Minimierungsproblem zu erhalten
2. Ersetze Nebenbedingungen der Form $\mathbf{a}_i'\mathbf{x} = b_i$ durch die beiden Nebenbedingungen $\mathbf{a}_i'\mathbf{x} \leq b_i$ und $\mathbf{a}_i'\mathbf{x} \geq b_i$
3. Multipliziere Nebenbedingungen der Form $\mathbf{a}_i'\mathbf{x} \leq b_i$ mit -1
4. Multipliziere jede Nebenbedingung der Form $x_j \leq 0$ mit -1
5. Ersetze jedes Auftreten einer freien Variable x_j durch $x_j^+ - x_j^-$, wobei $x_j^+ \geq 0$ und $x_j^- \geq 0$
6. Ersetze jede Bedingung der Form $x_j \geq 0$ durch $\mathbf{e}_j'\mathbf{x} \geq 0$, wobei $\mathbf{e}_j$ den j-ten Einheitsvektor bezeichnet

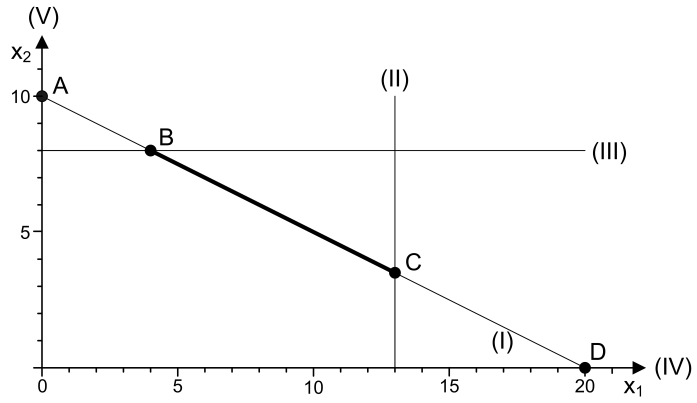


Abbildung 4.1: Die Punkte A, B, C und D sind Basislösungen, B und C sind darüber hinaus benachbarte zulässige Basislösungen

Das resultierende Problem kann nun in kompakter Weise angeschrieben werden:

$$z_{\text{LP}} = \min\{\mathbf{c}'\mathbf{x} \mid \mathbf{A}\mathbf{x} \geq \mathbf{b}, \mathbf{x} \in \mathbb{R}^n\} \tag{4.2}$$

Für die Charakterisierung der Lösungsvektoren eines linearen Programms sind die Begriffe *Basislösung* und *zulässige Basislösung* von großer Bedeutung:

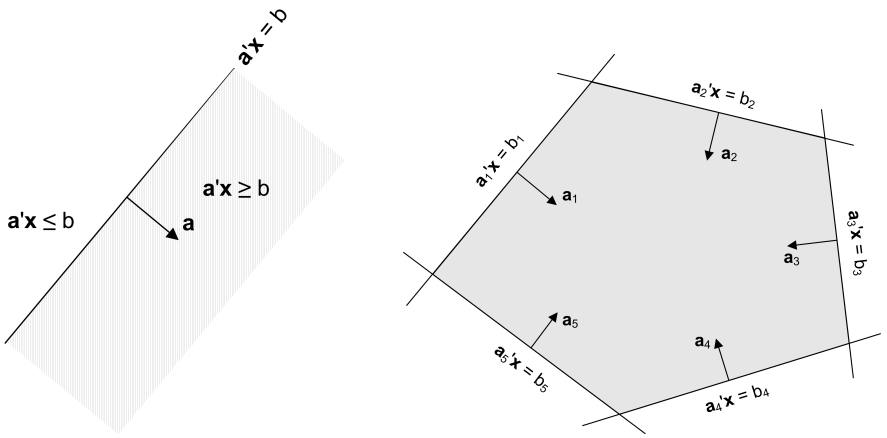
Definition 4.1.1 (Basislösung und zulässige Basislösung)**.** Gegeben sei ein lineares Programm mit Gleichheits- und Ungleichheitsnebenbedingungen sowie eine zulässige Lösung $\mathbf{x} \in \mathbb{R}^n$. Der Vektor $\mathbf{x}$ wird *Basislösung* genannt wenn insgesamt n linear unabhängige Nebenbedingungen mit Gleichheit erfüllt sind, wobei alle Gleichheitsnebenbedingungen des linearen Programms erfüllt sein müssen. Weiters wird $\mathbf{x}$ als *zulässige Basislösung* bezeichnet, wenn darüber hinaus alle weiteren Nebenbedingungen erfüllt sind.

Definition 4.1.2 (Benachbarte Basislösung)**.** Gegeben sei ein lineares Programm sowie zwei unterschiedliche Basislösungen $\mathbf{x}$ und $\mathbf{y}$. Diese werden als *benachbart* bezeichnet, wenn $n-1$ linear unabhängige Nebenbedingungen in beiden Basislösungen mit Gleichheit erfüllt sind.

Zur Illustration dieser Begriffe soll das folgende Beispiel dienen:

$$\begin{aligned}
\text{minimiere} \quad & -x_1 - x_2 & \\
\text{unter} \quad & x_1 + 2x_2 = 20 & \text{(I)} \\
& x_1 \leq 13 & \text{(II)} \\
& x_2 \leq 8 & \text{(III)} \\
& x_1, x_2 \geq 0 & \text{(IV,V)}
\end{aligned}$$

Der zulässige Bereich dieses linearen Programms ist in Abbildung 4.1 graphisch dargestellt. Dieser befindet sich aufgrund von Nebenbedingung I auf der Geraden $x_2 = -\frac{1}{2}x_1 + 10$. Die Punkte A, B, C und D sind Basislösungen, da in ihnen jeweils die Gleichheits-Nebenbedingung I sowie eine weitere Nebenbedingung mit Gleichheit erfüllt ist. Die Punkte B und C sind darüber hinaus zulässige Basislösungen, da in ihnen alle Nebenbedingungen des linearen Programms erfüllt sind. Weiters sind die Basislösungen A, B, C und D jeweils untereinander benachbart, da in jeder von ihnen Nebenbedingung I, und



(a) (b)

Abbildung 4.2: (a) Eine Hyperebene im $\mathbb{R}^2$ und die beiden von ihr begrenzten Halbräume (b) Zulässiger Bereich eines linearen Programms als Durchschnitt von fünf Halbräumen. (Nach [9])

damit $n-1$ Nebenbedingungen mit Gleichheit erfüllt sind.

Der folgende Abschnitt verallgemeinert die geometrische Interpretation des zulässigen Bereichs eines linearen Programms in beliebigen Dimensionen.

4.1.1 Geometrische Interpretation

Die geometrische Interpretation des zulässigen Bereichs eines linearen Programms beruht auf dem Begriff der konvexen Menge:

Definition 4.1.3 (Konvexe Menge)**.** Eine Menge $M \subset \mathbb{R}^n$ heißt *konvex*, wenn für je zwei Punkte $\mathbf{x} \in M$ und $\mathbf{y} \in M$ für jedes $\lambda \in \mathbb{R}$ mit $0 \leq \lambda \leq 1$ auch der Punkt $\mathbf{z} = (1-\lambda)\mathbf{x}+\lambda\mathbf{y}$ zu M gehört.

Eine Menge M ist also dann konvex, wenn die Verbindungslinie zwischen zwei beliebigen Punkten in dieser Menge vollständig in M liegt.

Definition 4.1.4 (Beschränkte Menge)**.** Eine Menge $M \subset \mathbb{R}^n$ heißt *beschränkt*, falls eine Konstante $K \in \mathbb{R}$ existiert, sodass $|x_i| < K, i = 1 \ldots n \; \forall \mathbf{x} = (x_1, \ldots, x_n) \in M$.

Definition 4.1.5 (Hyperebene und Halbraum)**.** Gegeben seien $\mathbf{a} \in \mathbb{R}^n$ und $b \in \mathbb{R}$. Dann nennt man die Menge $H = \{\mathbf{x} \in \mathbb{R}^n \mid \mathbf{a}'\mathbf{x} = b\}$ *Hyperebene*. Mit $H^+ = \{\mathbf{x} \in \mathbb{R}^n \mid \mathbf{a}'\mathbf{x} \geq b\}$ bezeichnet man den von H begrenzten *positiven abgeschlossenen Halbraum* und mit $H^- = \{\mathbf{x} \in \mathbb{R}^n \mid \mathbf{a}'\mathbf{x} \leq b\}$ den von H begrenzten *negativen abgeschlossenen Halbraum*.

Bei Hyperebenen und Halbräumen handelt es sich um konvexe Mengen. Abbildung 4.2a zeigt ein Beispiel für eine Hyperebene und die von ihr begrenzten Halbräume im $\mathbb{R}^2$.

Die hinsichtlich jeder der Nebenbedingungen eines linearen Programms (4.2) zulässigen Punkte entsprechen jeweils einem Halbraum im $\mathbb{R}^n$. Der zulässige Bereich P eines linearen Programms entspricht dem Durchschnitt all dieser Halbräume [32] und ist wiederum eine konvexe Menge. P wird als *konvexes Polytop* bzw. *konvexes Polyeder* bezeichnet:

Definition 4.1.6 (Konvexes Polytop und Konvexes Polyeder)**.** Eine Menge $P \subset \mathbb{R}^n$ heißt *konvexes Polytop*, wenn sich P als Durchschnitt endlich vieler Halbräume darstellen lässt. Ein beschränktes Polytop wird als *konvexes Polyeder* bezeichnet.

Abbildung 4.2b zeigt ein Beispiel für einen zulässigen Bereich im $\mathbb{R}^2$. Aus den Eigenschaften von P können die folgenden Eigenschaften eines linearen Programms abgeleitet werden [28]:

- $P = \emptyset$: Das Problem ist unzulässig, d.h. es existiert keine zulässige Lösung.
- $P \neq \emptyset$, aber P ist unbeschränkt: Das Problem ist ebenfalls unbeschränkt, d.h. die Zielfunktion kann einen beliebig kleinen Wert annehmen.
- $P \neq \emptyset$ und P ist beschränkt: Das Problem ist zulässig und es existiert zumindest eine optimale Lösung.

Ein weiterer wichtiger Begriff der geometrischen Interpretation eines linearen Programms ist der des extremen Punkts:

Definition 4.1.7 (Extremer Punkt)**.** Ist $M \subset \mathbb{R}^n$ konvex, dann heißt $\mathbf{z} \in M$ *extremer Punkt* bzw. *Eckpunkt*, falls keine zwei verschiedenen $\mathbf{x}, \mathbf{y} \in M$ existieren, sodass $\mathbf{z} = (1-\lambda)\mathbf{x} + \lambda\mathbf{y}$ für $\lambda \in \mathbb{R}$, $0 \leq \lambda \leq 1$.

Den extremen Punkten des zulässigen Bereichs P kommt besondere Bedeutung zu, da unter den optimalen Lösungen eines linearen Programms immer ein Eckpunkt zu finden ist:

Satz 4.1.8. *Gegeben sei ein lineares Programm* $\min\{\mathbf{c}'\mathbf{x} \,|\, \mathbf{A}\mathbf{x} \geq \mathbf{b}, \mathbf{x} \in \mathbb{R}^n\}$ *mit zulässigem Bereich P. Falls P zumindest einen extremen Punkt besitzt und eine optimale Lösung existiert, dann existiert eine optimale Lösung, die auch ein extremer Punkt von P ist.*

Beweis. Siehe [9] □

Der folgende Satz verknüpft den Begriff des extremen Punkts mit dem algebraischen Begriff der zulässigen Basislösung:

Satz 4.1.9. *Gegeben sei ein lineares Programm* $\min\{\mathbf{c}'\mathbf{x} \,|\, \mathbf{A}\mathbf{x} \geq \mathbf{b}, \mathbf{x} \in \mathbb{R}^n\}$ *mit zulässigem Bereich P. Die folgenden Begriffe sind äquivalent:*

1. *Extremer Punkt von P*
2. *Zulässige Basislösung des linearen Programms*

Beweis. Siehe [9] □

Da die optimale Lösung eines linearen Programms unter den extremen Punkten des zulässigen Bereichs P zu finden ist, muss sie auch eine zulässige Basislösung sein. Diese Entsprechung macht sich der Simplex-Algorithmus zur Lösung linearer Programme zu Nutze, indem ausgehend von einer Startlösung systematisch von einer zulässigen Basislösung zu einer benachbarten zulässigen Basislösung mit verbessertem Zielfunktionswert gesprungen wird, bis keine Verbesserung des Zielfunktionswerts mehr möglich ist.

4.2 Ganzzahlige lineare Programmierung

Wird ein lineares Programm um die Forderung der Ganzzahligkeit der Lösungen ergänzt, ergibt sich ein *Ganzzahliges lineares Programm* bzw. *Integer Linear Program* (ILP):

$$z_{\text{ILP}} = \min\{\mathbf{c}'\mathbf{x} \mid \mathbf{A}\mathbf{x} \geq \mathbf{b}, \mathbf{x} \in \mathbb{Z}^n\} \tag{4.3}$$

Wird Ganzzahligkeit nur für bestimmte Entscheidungsvariablen gefordert, spricht man von einem *Gemischt-ganzzahligen linearen Programm* bzw. einem *Mixed-Integer Program* (MIP). Die folgenden Ausführungen beschränken sich auf rein ganzzahlige Probleme.

Die exakte Lösung eines ILPs stellt im Normalfall ein $\mathcal{NP}$-schweres Problem dar. Die zu diesem Zweck eingesetzten Methoden sind unter anderem *Schnittebenenverfahren*, *Branch-and-Bound* sowie ihre Kombination, *Branch-and-Cut*. Für eine detaillierte Beschreibung dieser und weiterer Verfahren siehe [37].

Sowohl Schnittebenenverfahren als auch Branch-and-Bound beruhen auf der sogenannten *LP-Relaxation* eines ILP:

Definition 4.2.1 (LP-Relaxation)**.** Gegeben sei ein ILP $z_{\text{ILP}} = \min\{\mathbf{c}'\mathbf{x} \mid \mathbf{A}\mathbf{x} \geq \mathbf{b}, \mathbf{x} \in \mathbb{Z}^n\}$. Das zugehörige LP $z_{\text{LP}} = \min\{\mathbf{c}'\mathbf{x} \mid \mathbf{A}\mathbf{x} \geq \mathbf{b}, \mathbf{x} \in \mathbb{R}^n\}$ wird *LP-Relaxation* des ILP genannt.

Für die optimalen Lösungen der LP-Relaxation und des originalen Problems gilt, dass $z_{\text{LP}} \leq z_{\text{ILP}}$, da der zulässige Bereich des originalen Problems in jenem der LP-Relaxation enthalten ist.

4.2.1 Schnittebenenverfahren

Schnittebenenverfahren zur Lösung von ILPs beruhen auf der Lösung einer Folge von LP-Relaxationen. Algorithmus 4.1 zeigt das allgemeine Vorgehen im Rahmen eines Schnittebenenverfahrens. Der Algorithmus löst die LP-Relaxation des ILP und ermittelt so eine Lösung $\mathbf{x}^*$. Ist $\mathbf{x}^*$ ganzzahlig, so ist die optimale Lösung des ILP gefunden, und das Verfahren wird abgebrochen. Andernfalls wird das ILP um eine Nebenbedingung erweitert, die von allen ganzzahligen Lösungen, aber nicht von $\mathbf{x}^*$ erfüllt wird. Dieses Verfahren wird fortgesetzt, bis eine ganzzahlige Lösung gefunden wurde.

Algorithmus 4.1 : Schnittebenenverfahren

```
1 repeat
2 |  x* ← ermittle die optimale Lösung der LP-Relaxation
3 |  if x* ist nicht ganzzahlig then
4 |  |  Ergänze das ILP um eine Nebenbedingung, die von allen ganzzahligen
  |  |  Lösungen, aber nicht von x* erfüllt wird
5 until x* ist ganzzahlig
```

Die Ermittlung einer durch $\mathbf{x}^*$ verletzten Nebenbedingung wird als *Separation Problem* [41] bezeichnet. Die Qualität eines Schnittebenenverfahrens hinsichtlich der Anzahl der notwendigen Schritte und damit der Anzahl der zu lösenden LP-Relaxationen hängt von der Methode zur Lösung des Separation Problems ab. Einerseits können hierfür generische

Verfahren wie der *Gomory Cutting-Plane Algorithmus* [9] herangezogen werden, welche die Lösung jedes beliebigen ILPs garantieren. Anderseits können auch problemangepasste Verfahren verwendet werden, die problemspezifisches Wissen ausnutzen, um Schnittebenen zu generieren, die in jedem Schritt einen größeren Teil des Suchraums eliminieren können und damit schneller zu einer Lösung des ILPs führen.

4.2.2 Branch-and-Bound

Algorithmus 4.2 : Generisches Branch-and-Bound

```
1  U ← ∞
2  L ← {P}
3  repeat
4      F_i ← Wähle aktives Unterproblem aus L
5      if F_i ist unzulässig then
6          L ← L \ F_i
7      else
8          Berechne b(F_i)
9          if b(F_i) ≥ U then
10             L ← L \ F_i
11         else if F_i optimal lösbar then
12             Löse F_i optimal und aktualisiere gegebenenfalls U
13         else
14             Teile Unterproblem F_i in weitere Unterprobleme und füge diese zu L
               hinzu
15 until L = ∅
```

Der Branch-and-Bound-Ansatz zur Lösung ganzzahliger linearer Programme beruht auf der geschickten Enumeration des Suchraums aller zulässigen ganzzahligen Lösungen. Dies geschieht durch Zerteilung des Problems in unabhängige Unterprobleme, die in Form eines Baums dargestellt werden können. Kann ein Unterproblem nicht optimal gelöst werden, wird es wiederum in Unterprobleme zerteilt, bis schließlich eine ganzzahlige Lösung ermittelt werden kann. Um Teile des Berechnungs-Baums und damit auch Teile des Suchraums nicht untersuchen zu müssen, stützt sich der Algorithmus auf zwei Schranken:

- Die globale obere Schranke des Zielfunktionswerts U. Diese entspricht dem Zielfunktionswert der besten bisher gefundenen ganzzahligen Lösung.
- Eine untere Schranke des Zielfunktionswerts eines Teilproblems F_i, bezeichnet mit $b(F_i)$.

Falls für ein Unterproblem $b(F_i) \geq U$ gilt, d.h. falls die untere Schranke des Unterproblems F_i die globale obere Schranke nicht unterschreitet, muss es nicht weiter behandelt werden, da die optimale Lösung von F_i nicht besser ist als die bisher beste ganzzahlige Lösung. Algorithmus 4.2 zeigt das allgemeine Vorgehen des Branch-and-Bound-Verfahrens. In diesem Algorithmus sind Variationsmöglichkeiten hinsichtlich des Verfahrens zur Auswahl des aktuellen aktiven Unterproblems, der Methode zur Berechnung von $b(F_i)$ sowie des Verfahrens zur Ermittlung neuer Unterprobleme vorhanden.

Die naheliegendste Möglichkeit zur Berechnung von $b(F_i)$ ist die Lösung der LP-Relaxation von F_i, da für die Zielfunktionswerte der optimalen Lösungen z_i^{LP} und z_i^{ILP} der LP-Relaxation bzw. des originalen Problems $z_i^{\mathrm{LP}} \leq z_i^{\mathrm{ILP}}$ gilt.

Algorithmus 4.3 zeigt eine in dieser Weise angepasste Version des generischen Branch-and-Bound-Algorithmus. Immer wenn die durch Lösung der LP-Relaxation gewonnene untere Schranke $b(F_i)$ kleiner als die globale obere Schranke U ist und die Lösung der LP-Relaxation nur ganzzahlige Werte enthält, wurde eine verbesserte zulässige Lösung des ILP gefunden und es erfolgt eine Aktualisierung von U. Ist die Lösung der LP-Relaxation nicht ganzzahlig, muss das aktuelle Unterproblem in weitere Unterprobleme zerlegt werden.

Algorithmus 4.3 : Branch-and-Bound mit LP-Relaxation

```
1  U ← ∞
2  L ← {P}
3  repeat
4  |  F_i ← Wähle aktives Unterproblem aus L
5  |  x_i^* ← Löse die LP-Relaxation von F_i
6  |  if F_i ist unzulässig then
7  |  |  L ← L \ F_i
8  |  else
9  |  |  v_i^* ← c'x^*
10 |  |  if v_i^* ≥ U then
11 |  |  |  L ← L \ F_i
12 |  |  else if x_i^* ist ganzzahlig then
13 |  |  |  /* Verbesserte ganzzahlige Lösung gefunden */
14 |  |  |  U ← v_i^*
15 |  |  |  x^* ← x_i^*
16 |  |  |  L ← L \ F_i
17 |  |  else
18 |  |  |  Teile Unterproblem F_i in weiteres Unterprobleme und füge diese zu L
   |  |  |  hinzu
19 until L = ∅
```

Die Zerlegung von F_i in Unterprobleme geschieht durch Hinzufügen neuer Nebenbedingungen. Dazu wird eine Variable x_j der Lösung der LP-Relaxation mit nicht-ganzzahligem Wert x_j^* ausgewählt und jeweils eine der beiden Nebenbedingungen

$$x_j \leq \lfloor x_j^* \rfloor$$

bzw.

$$x_j \geq \lceil x_j^* \rceil$$

zu F_i hinzugefügt, wodurch zwei neue Unterprobleme entstehen, die in L aufgenommen werden.

Die Kombination von Branch-and-Bound und Schnittebenenverfahren wird als *Branch-and-Cut* bezeichnet. Ein solches Verfahren wendet ein Schnittebenenverfahren auf neu erzeugte Unterprobleme an, um den Suchraum weiter einzuschränken.

Kapitel 5

MIP-Formulierung von VSLRB

Dieses Kapitel beschreibt die Ableitung einer Formulierung von VSLRB als gemischtganzzahliges lineares Programm (MIP) aus der in Abschnitt 3.2 beschriebenen Formalisierung.

5.1 Entwicklung der MIP-Formulierung

Eine Lösung für eine Instanz von VSLRB besteht aus m Mengen $F_j \subseteq F,\ j \in C$ sowie einer Zuweisungsfunktion $Q : F \times C \to \mathbb{N}_0$. Dies wird in der MIP-Formulierung des Problems durch die Entscheidungsvariablen p_j^i und x_j^i ausgedrückt: Die binären Entscheidungsvariablen p_j^i repräsentieren, ob ein Replikat von Video-Objekt $i \in F$ auf Server $j \in C$ platziert werden soll und stellen damit eine direkte Entsprechung der Mengen F_j dar. Die ganzzahligen Entscheidungsvariablen x_j^i repräsentieren analog zu $Q(i, j)$ eine Zuweisung von Zugriffen auf Video-Objekt $i \in F$ zu Server $j \in C$.

Um Bedingung 3.4 von VSLRB zu erfüllen und nur gültige Platzierungen von Replikaten vorzunehmen, werden nur Entscheidungsvariablen in die MIP-Formulierung von VSLRB aufgenommen, die zulässigen Replikatsplatzierungen bzw. Zuweisungen entsprechen. Dazu wird die Indexmenge der zulässigen Platzierungen bzw. Zuweisungen definiert:

$$P = \{(i, j) \in F \times C \mid t_i \in T_j\}$$

Die Entscheidungsvariablen des Modells sind daher

$$x_j^i,\ p_j^i \quad \forall (i, j) \in P$$

Da gemäß Bedingung 3.6 von VSLRB eine Zuweisung von Zugriffen auf ein Video-Objekt i zu einem Server j nur erfolgen kann, wenn j auch ein Replikat von i besitzt, muss gelten:

$$p_j^i - \frac{x_j^i}{q_i} \geq 0 \quad \forall (i, j) \in P$$

Weiters muss gemäß Bedingung 3.6 eine Zuweisung von Zugriffen erfolgen, wenn ein Replikat auf einem Server platziert wird:

$$p_j^i - \frac{x_j^i}{q_i} \leq 1 - \frac{1}{q_i} \quad \forall (i, j) \in P$$

Für die Umsetzung der Zielfunktion in der MIP-Formulierung muss einerseits berücksichtigt werden, dass Z_1 eine Summe von Absolutbeträgen darstellt, die in der MIP-Formulierung gesondert behandelt werden muss. Weiters muss eine Entsprechung der

Mengenoperationen zur Ermittlung der hinzukommenden Replikate in Z_2 gefunden werden. Die Summe von Absolutbeträgen im ersten Teil der Zielfunktion wird durch Einführung zusätzlicher Entscheidungsvariablen $y_j,\ j \in C$ und je zwei zusätzlichen Nebenbedingungen in die MIP-Formulierung übertragen [9], sodass der erste Teil der Zielfunktion in der MIP-Formulierung

$$Z_1^{\text{MIP}} = \sum_{j \in C} y_j$$

lautet. Pro neuer Entscheidungsvariable y_j werden die folgenden beiden Nebenbedingungen hinzugefügt:

$$\eta_j - \sum_{\{i \in F | t_i \in T_j\}} b_i x_j^i \leq y_j \quad \forall j \in C$$

$$-\eta_j + \sum_{\{i \in F | t_i \in T_j\}} b_i x_j^i \leq y_j \quad \forall j \in C$$

Die Mengenoperationen im zweiten Teil der Zielfunktion können durch Multiplikation mit geeigneten Konstanten umgesetzt werden:

Satz 5.1.1. *Gegeben seien zwei Server $l, k \in C, l \neq k$ sowie ein Video-Objekt $i \in F$. Es gilt: $i \in F_l \setminus \overline{F}_l \cap \overline{F}_k \Leftrightarrow \overline{p}_k^i p_l^i (1 - \overline{p}_l^i) = 1$ und $i \notin F_l \setminus \overline{F}_l \cap \overline{F}_k \Leftrightarrow \overline{p}_k^i p_l^i (1 - \overline{p}_l^i) = 0$.*

Beweis. 1) $i \in F_l \setminus \overline{F}_l \cap \overline{F}_k \Leftrightarrow i \in F_l \wedge i \notin \overline{F}_l \wedge i \in \overline{F}_k \Leftrightarrow p_l^i = 1 \wedge \overline{p}_l^i = 0 \wedge \overline{p}_k^i = 1 \Leftrightarrow \overline{p}_k^i p_l^i (1 - \overline{p}_l^i) = 1$.
2) $i \notin F_l \setminus \overline{F}_l \cap \overline{F}_k \Leftrightarrow i \notin F_l \vee i \in \overline{F}_l \vee i \notin \overline{F}_k \Leftrightarrow p_l^i = 0 \vee \overline{p}_l^i = 1 \vee \overline{p}_k^i = 0 \Leftrightarrow \overline{p}_k^i p_l^i (1 - \overline{p}_l^i) = 0$. □

Da $\frac{\overline{x}_k^i}{\overline{q}_i} = 0 \Leftrightarrow \overline{p}_k^i = 0$ und $\frac{\overline{x}_k^i}{\overline{q}_i} > 0 \Leftrightarrow \overline{p}_k^i = 1$, kann der zweite Teil der Zielfunktion in der MIP-Formulierung auf folgende Weise angeschrieben werden:

$$Z_2^{\text{MIP}} = \sum_{k \in C} \left[\sum_{\substack{l \in C \\ l \neq k}} \left[\sum_{\{i \in F | t_i \in T_l\}} \frac{1}{\overline{q}_i} \overline{x}_k^i \, p_l^i (1 - \overline{p}_l^i) w_i \right] \frac{1}{c_{kl}} \right]$$

Immer dann, wenn in der originalen Formulierung $i \notin F_l \setminus \overline{F}_l \cap \overline{F}_k$ gilt und das jeweilige Replikat damit nicht berücksichtigt wird, nimmt der entsprechende Summand in dieser Formulierung den Wert null an.

Bedingung 3.5 von VSLRB, welche die Berücksichtigung aller Zugriffe des Access Profile fordert, wird in naheliegender Weise folgendermaßen umgesetzt:

$$\sum_{\{j \in C | t_i \in T_j\}} x_j^i = q_i \quad \forall i \in F \tag{5.1}$$

Durch diese Bedingung ist auch die (redundante) Bedingung 3.3 von VSLRB erfüllt:

Satz 5.1.2. *Wenn Bedingung 5.1 der MIP-Formulierung von VSLRB erfüllt ist, dann ist auch Bedingung 3.3 von VSLRB erfüllt.*

Beweis. Gegeben sei ein Video-Objekt $i \in F$. Da $q_i > 0$ und $\sum_{\{j \in C | t_i \in T_j\}} x_j^i = q_i$ existiert zumindest ein $k \in C$ sodass $x_k^i > 0$. $x_k^i > 0 \Leftrightarrow p_k^i = 1 \Leftrightarrow i \in F_k$. Für jedes Video-Objekt $i \in F$ existiert daher zumindest ein $F_k,\ k \in C$ sodass $i \in F_k$. Daher gilt $\bigcup_{j \in C} F_j = F$. □

Weiters müssen noch die Nebenbedingungen für die Einhaltung der Speicherkapazitäten (3.8) und der maximalen Transfervolumina (3.9) umgesetzt werden:

$$\sum_{\{i \in F | t_i \in T_j\}} p_j^i w_i \leq W_j \quad \forall j \in C$$

$$\sum_{\{i \in F | t_i \in T_j\}} p_j^i (1 - \overline{p}_j^i) w_i \leq W_j - \left(\sum_{i \in F} \overline{p}_j^i w_i \right) \quad \forall j \in C$$

Die Mengenoperation auf der linken Seite von Bedingung 3.9 wird analog zum Vorgehen beim zweiten Teil der Zielfunktion durch Multiplikation der Entscheidungsvariablen p_j^i mit $(1 - \overline{p}_j^i)$ umgesetzt, sodass der jeweilige Summand den Wert null annimmt, wenn ein Replikat von i bereits auf j platziert ist.

Die optionale Bedingung 3.10 von VSLRB zur Realisierung einer Round-Robin-Zuweisungsstrategie, gemäß derer sich Zuweisungen des selben Video-Objekts zu verschiedenen Servern um maximal einen Zugriff unterscheiden dürfen, kann durch Einführung neuer Nebenbedingungen für jedes Paar von Zuweisungen $x_k^i,\ x_j^i,\ k \neq j$ in die MIP-Formulierung übertragen werden. Die Herleitung dieser Nebenbedingungen geschieht über den Umweg neuer Entscheidungsvariablen $\widetilde{x}_{jk}^i$. Diese sollen die Eigenschaft besitzen, dass $\widetilde{x}_{jk}^i \geq x_k^i - x_j^i$ falls $x_k^i > 0,\ x_j^i > 0$ und $x_k^i > x_j^i$ wobei $i \in F,\ k, j \in C,\ k \neq j$. Andernfalls soll $\widetilde{x}_{jk}^i \geq 0$ gelten.

Zu diesem Zweck wird in der folgenden Bedingung jede mögliche Zuweisung x_j^i mit jeder anderen möglichen Zuweisung $x_k^i,\ j \neq k$ in Beziehung gesetzt, wobei eine positive Differenz $x_k^i - x_j^i$ in den Überschussvariablen $\widetilde{x}_{jk}^i$ aufgefangen wird:

$$p_j^i - \frac{1}{q_i} x_j^i + \frac{1}{q_i} x_k^i \leq 1 + \frac{1}{q_i} \widetilde{x}_{jk}^i \quad \forall (i, j, k) \in F \times C \times C \,|\, t_i \in T_j \wedge t_i \in T_k,\ j \neq k \quad (5.2)$$

Die folgende Aufstellung zeigt alle möglichen Fälle:

p_j^i	x_j^i	x_k^i	Erg.	$\widetilde{x}_{jk}^i$
1	> 0	0	< 1	≥ 0
1	> 0	$< x_j^i$	< 1	≥ 0
1	> 0	$= x_j^i$	1	≥ 0
1	> 0	$> x_j^i$	> 1	$\geq x_k^i - x_j^i > 0$
0	0	0	0	≥ 0
0	0	> 0	< 0	≥ 0

Die linke Seite der Ungleichung nimmt also nur dann einen Wert größer als eins an, wenn $x_j^i = Q(i, j) > 0$ (und damit $p_j^i = 1$), $x_k^i = Q(i, k) > 0$ und $x_k^i > x_j^i$. Nur in diesem Fall muss auch auf der rechten Seite der Ungleichung $\widetilde{x}_{jk}^i > 0$ gelten. Durch Einschränkung des höchsten Werts der $\widetilde{x}_{jk}^i$ kann die größtmögliche Abweichung zwischen zwei Zuweisungen festgelegt werden. In diesem Fall darf diese höchstens eins betragen:

$$\widetilde{x}_{jk}^i \in \{0, 1\} \quad \forall (i, j, k) \in F \times C \times C \,|\, t_i \in T_j \wedge t_i \in T_k,\ j \neq k$$

Diese obere Schranke für die Abweichung zwischen zwei Zuweisungen kann auf der rechten Seite von Bedingung 5.2 eingesetzt werden:

$$p_j^i - \frac{1}{q_i}x_j^i + \frac{1}{q_i}x_k^i \leq 1 + \frac{1}{q_i} \quad \forall (i,j,k) \in F \times C \times C \,|\, t_i \in T_j \wedge t_i \in T_k,\, j \neq k$$

Somit darf die Abweichung zwischen zwei $x_j^i > 0$, $x_k^i > 0$ höchstens eins betragen und es ist eine äquivalente Formulierung zu Bedingung 3.10 von VSLRB gefunden, sodass Zuweisungen desselben Video-Objekts zu verschiedenen Servern um maximal einen Zugriff voneinander abweichen dürfen.

Schließlich wird zur Vereinfachung des Modells die Platzierung und Zuweisung der Thumbnails fixiert, da weder ihre Größe noch die durch sie verursachte Last ins Gewicht fallen:

$$x_j^i = \bar{x}_j^i \quad \forall (i,j) \in P \,|\, t_i = Thumbnail$$

$$p_j^i = \bar{p}_j^i \quad \forall (i,j) \in P \,|\, t_i = Thumbnail$$

5.2 Zusammenfassung der MIP-Formulierung

Ermittle x_j^i, p_j^i, $(i,j) \in P$, y_j, $j \in C$ und ggf. $\widetilde{x}_{jk}^i$, $(i,j,k) \in F \times C \times C \,|\, t_i \in T_j \wedge t_i \in T_k$, $j \neq k$, sodass der folgende Ausdruck unter Berücksichtigung der Nebenbedingungen 5.3 bis 5.14 und ggf. 5.15 minimal wird:

$$\min\, \alpha \sum_{j \in C} y_j + \beta \sum_{k \in C} \left[\sum_{\substack{l \in C \\ l \neq k \\ \{i \in F \mid t_i \in T_l\}}} \left[\sum \tfrac{1}{q_i} \overline{x}_k^i \, p_l^i (1 - \overline{p}_l^i) w_i \right] \frac{1}{c_{kl}} \right]$$

Unter den folgenden Nebenbedingungen:

$$\eta_j - \sum_{\{i \in F \mid t_i \in T_j\}} b_i x_j^i \leq y_j \quad \forall j \in C \tag{5.3}$$

$$-\eta_j + \sum_{\{i \in F \mid t_i \in T_j\}} b_i x_j^i \leq y_j \quad \forall j \in C \tag{5.4}$$

$$\sum_{\{j \in C \mid t_i \in T_j\}} x_j^i = q_i \quad \forall i \in F \tag{5.5}$$

$$p_j^i - \frac{x_j^i}{q_i} \geq 0 \quad \forall (i,j) \in P \tag{5.6}$$

$$p_j^i - \frac{x_j^i}{q_i} \leq 1 - \frac{1}{q_i} \quad \forall (i,j) \in P \tag{5.7}$$

$$\sum_{\{i \in F \mid t_i \in T_j\}} p_j^i w_i \leq W_j \quad \forall j \in C \tag{5.8}$$

$$\sum_{\{i \in F \mid t_i \in T_j\}} p_j^i (1 - \bar{p}_j^i) w_i \leq W_j - \left(\sum_{i \in F} \bar{p}_j^i w_i \right) \quad \forall j \in C \tag{5.9}$$

$$x_j^i = \bar{x}_j^i \quad \forall (i,j) \in P \,|\, t_i = Thumbnail \tag{5.10}$$

$$p_j^i = \bar{p}_j^i \quad \forall (i,j) \in P \,|\, t_i = Thumbnail \tag{5.11}$$

$$x_j^i \in \{0 \dots q_i\} \quad \forall (i,j) \in P \tag{5.12}$$

$$p_j^i \in \{0,1\} \quad \forall (i,j) \in P \tag{5.13}$$

$$y_j \geq 0 \quad \forall j \in C \tag{5.14}$$

Unter den optionalen Nebenbedingungen:

$$p_j^i - \frac{1}{q_i} x_j^i + \frac{1}{q_i} x_k^i \leq 1 + \frac{1}{q_i} \quad \forall (i,j,k) \in F \times C \times C \,|\, t_i \in T_j \wedge t_i \in T_k,\ j \neq k \tag{5.15}$$

Kapitel 6

Lokale Suchverfahren

Viele in der Praxis auftretende Optimierungsprobleme sind kombinatorischer Natur, d.h. sie bestehen darin, Werte für diskrete Variablen zu ermitteln, sodass eine vorgegebene Zielfunktion unter Einhaltung vorgegebener Nebenbedingungen einen optimalen Wert annimmt [41]. Beispiele für solche Probleme umfassen Reihenfolgeprobleme, die Erstellung von Zeitplänen, die Planung von Touren und das Design von Kommunikationsnetzwerken. Die meisten dieser Probleme sind $\mathcal{NP}$-schwer, sodass vermutlich keine Algorithmen mit polynomieller Laufzeit existieren, um sie beweisbar optimal zu lösen, es sei denn es gälte $\mathcal{P} = \mathcal{NP}$ [18].

Während kleine Instanzen dieser Probleme noch mit Hilfe exakter Verfahren (z.B. Branch-and-Bound, Dynamic Programming, Ganzzahlige Lineare Programmierung) beweisbar optimal gelöst werden können [41], werden die dafür notwendigen Laufzeiten für große Instanzen häufig schnell unpraktikabel. In vielen Fällen können Heuristiken schnell gute Lösungen lieferen, auch wenn deren Güte theoretisch beliebig weit von jener der unbekannten Optimallösung entfernt sein kann. Eine der einfachsten Ausprägungen einer solchen Heuristik die Lokale Suche, die nach verbesserten Lösungen einer Probleminstanz in der näheren Umgebung der bisher besten gefundenen Lösung sucht, bis keine verbesserte Lösung mehr gefunden werden kann.

Metaheuristiken sind Frameworks zur Erstellung von Heuristiken für kombinatorische Optimierungsprobleme [26]. Sie geben eine problemunabhängige Folge von Schritten vor, die bei der Anwendung auf ein konkretes Problem durch die Implementierung von problemspezifischen Schritten ergänzt werden muss. Beispiele für Metaheuristiken sind unter anderem *Simulated Annealing*, *Tabu-Suche*, *Variable Neighbourhood Search*, populationsbasierende Verfahren wie *Evolutionäre Algorithmen* sowie auf Schwarmintelligenz basierende Verfahren wie *Ant Colony Optimization*. Für eine Übersicht über dieses Gebiet siehe z.B. [21].

Das folgende Kapitel gibt einen kurzen Überblick über lokale Suche für kombinatorische Optimierungsprobleme sowie über die darauf aufbauende Metaheuristik Variable Neighbourhood Search (VNS).

6.1 Einfache lokale Suche

Gegeben sei ein kombinatorisches Optimierungsproblem:

$$\min f(x)$$

unter der Nebenbedingung, dass

$$x \in X,$$

wobei f die Zielfunktion und X die Menge der zulässigen Lösungen des kombinatorischen Optimierungsproblems bezeichnet. Die Zielfunktion f beschreibt die Güte einer Lösung $x \in X$. Eine Lösung $x^* \in X$ wird als *optimal* bzw *global optimal* bezeichnet, wenn kein anderes $x \in X$ mit besserem Zielfunktionswert existiert:

$$f(x^*) \leq f(x) \quad \forall x \in X$$

Eine Lösung x_L^* wird als *lokal optimal* (bezüglich der Nachbarschaftsstruktur $\mathcal{N}$) bezeichnet, wenn keine Lösung mit besserem Zielfunktionswert in ihrer Umgebung existiert:

$$f(x_L^*) \leq f(x) \quad \forall x \in \mathcal{N}(x)$$

Die folgende Definition präzisiert den Begriff der Umgebung einer Lösung:

Definition 6.1.1 (Nachbarschaftsstruktur)**.** Eine Nachbarschaftsstruktur ist eine Funktion $\mathcal{N} : X \to \mathcal{P}(X)$, die jeder Lösung $x \in X$ eine Menge von Nachbarlösungen $\mathcal{N}(x) \subseteq X$ zuweist, wobei $\mathcal{P}(X)$ die Potenzmenge von X bezeichnet.

Die konkrete Menge von Nachbarlösungen $\mathcal{N}(x)$ einer Lösung $x \in X$ wird verkürzend auch als *Nachbarschaft* von x bezeichnet. Eine solche Nachbarschaft enthält im Allgemeinen solche Lösungen, die sich nur in einigen wenigen Eigenschaften von x unterscheiden. Nachbarschaftsstrukturen können in vielen Fällen implizit durch elementare Operationen definiert sein, die eine zulässige Lösung eines Problems in eine andere zulässige Lösung des Problems transformieren, z.B. das Vertauschen von Elementen in einer Anordnung oder das Verschieben von Elementen zwischen Mengen. Je größer die Anzahl der möglichen Transformationschritte ausgehend von einer vorgegebenen Lösung ist, um so größer ist die entsprechende Nachbarschaft und um so größer ist auch der Aufwand, der für die Durchsuchung der Nachbarschaft notwendig ist.

Basierend auf dem Begriff der Nachbarschaftsstruktur kann eine einfache lokale Suchprozedur entwickelt werden (siehe Algorithmus 6.1). Ausgehend von einer Startlösung x_s wird die Nachbarschaft $\mathcal{N}(x)$ solange nach einer Lösung mit besserem Zielfunktionswert durchsucht, bis eine Abbruchbedingung erfüllt ist.

Algorithmus 6.1 : Lokale Suche

Eingabe : Startlösung x_s

```
1 x ← x_s
2 repeat
3 |   x' ← step(N(x))
4 |   if f(x') ≤ f(x) then
5 |   |   x ← x'
6 until Abbruchbedingung erfüllt
```

Besondere Bedeutung kommt dabei der Funktion *step* zu: diese dient dazu, die Lösung x' aus der Nachbarschaft $\mathcal{N}(x)$ auszuwählen. Mögliche Strategien dafür sind u.a. [2]:

Algorithmus 6.2 : Descent Heuristic

Eingabe : Startlösung x_s

```
1 x ← x_s
2 repeat
3 |   x' ← improvement-step(N(x))
4 |   if f(x') ≤ f(x) then
5 |   |   x ← x'
6 until f(x') > f(x)
```

- *Next Improvement*: Auswahl der ersten Nachbarlösung mit verbessertem Zielfunktionswert
- *Best Improvement*: Auswahl der Nachbarlösung mit dem besten Zielfunktionswert

Denkbar ist unter anderem auch die zufällige Auswahl einer Nachbarlösung oder der Einsatz einer der beiden genannten Strategien mit gewissen Einschränkungen (siehe Abschnitte 7.2.3 und 7.2.4). Die Wahl der Schrittfunktion beeinflusst, welche lokale Optima durch die lokale Suche gefunden werden können. Welche dieser Strategien für ein konkretes Problem am besten geeignet ist, hängt sowohl von der Beschaffenheit der Zielfunktion, als auch von Zeitbeschränkungen ab, die für die Durchsuchung der Nachbarschaft bestehen.

Falls in jedem Schritt der lokalen Suche eine verbesserte Lösung aus der Nachbarschaft gewählt wird und die Suche abgebrochen wird, wenn keine solche Lösung mehr gefunden werden konnte, spricht man von einer *Descent Heuristic* [26] (siehe Algorithmus 6.2). Eine auf diese Weise erzielte Lösung ist lokal optimal bezüglich der Nachbarschaftsstruktur $\mathcal{N}$.

Im Allgemeinen liefert eine lokale Suche ausgehend von einer beliebigen Startlösung nicht das globale Optimum. Eine Möglichkeit, ein durch lokale Suche erreichtes Optimum wieder zu verlassen, ist die Verwendung von mehr als nur einer einzigen Nachbarschaftsstruktur. Diese Idee kommt in der im folgenden Abschnitt beschriebenen Metaheuristik *Variable Neighbourhood Search* zum Einsatz.

6.2 Variable Neighbourhood Search

Variable Neighbourhood Search ist eine von Hansen und Mladenovic [25, 26, 36] vorgeschlagene Metaheuristik, die sich auf den systematischen Wechsel zwischen verschiedenen Nachbarschaftsstrukturen stützt. Dies wird sowohl zur Intensivierung (siehe Abschnitt 6.2.1) als auch zur Diversifizierung (siehe Abschnitt 6.2.2) der Suche eingesetzt.

Einerseits nutzt das Verfahren die Tatsache, dass ein lokales Optimum bezüglich einer bestimmten Nachbarschaftsstruktur nicht notwendigerweise ein lokales Optimum bezüglich einer anderen Nachbarschaftsstruktur sein muss. Wurde das lokale Optimum bezüglich einer bestimmten Nachbarschaftsstruktur bereits gefunden, kann die Suche nach verbesserten Lösungen in einer anderen Nachbarschaft fortgesetzt werden.

Weiters nutzt das Verfahren die für viele Optimierungsprobleme geltende Beobachtung, dass lokale Optima nahe beieinander liegen, d.h. sich in nur wenigen Variablen unterscheiden. Eine zufällige kleine Veränderung eines bekannten lokalen Optimums erscheint daher

vielversprechender, um den Suchvorgang in vielversprechende Bereiche des Suchraums zu lenken, als der Neustart der Suche von einem zufälligen Punkt wie im Fall der Multi-Start Local Search.

6.2.1 Variable Neighbourhood Descent

Variable Neighbourhood Descent dient im Rahmen der VNS zur *Intensivierung* des Suchvorgangs. Ausgehend von einer Startlösung sucht das Verfahren eine neue Lösung, die ein lokales Optimum bezüglich aller zuvor definierten Nachbarschaftsstrukturen $\mathcal{N}_1, \ldots, \mathcal{N}_{l_{max}}$ darstellt. Die in Algorithmus 6.3 dargestellte VND kann somit als eine Verallgemeinerung der Descent Heuristic aus Abschnitt 6.1 angesehen werden. Sobald in der Nachbarschaft $\mathcal{N}_l(x)$ eine verbesserte Lösung gefunden werden konnte, wechselt das Verfahren wieder zur ersten Nachbarschaftsstruktur $\mathcal{N}_1$, andernfalls wird mit $\mathcal{N}_{l+1}$ fortgesetzt. Das Verfahren endet, wenn auch unter Verwendung der letzten Nachbarschaftsstruktur $\mathcal{N}_{l_{max}}$ keine verbesserte Lösung gefunden werden konnte.

Algorithmus 6.3 : Variable Neighbourhood Descent

```
   Eingabe : Startlösung x_s
 1 x ← x_s
 2 l ← 1
 3 repeat
 4 |   x' ← improvement-step(N_l(x))
 5 |   if f(x') ≤ f(x) then
 6 |   |   x ← x'
 7 |   |   l ← 1
 8 |   else
 9 |   |   l ← l + 1
10 until l > l_max
```

Große Bedeutung kommt dabei der Auswahl und der Reihenfolge der verwendeten Nachbarschaftsstrukturen $\mathcal{N}_1, \ldots, \mathcal{N}_{l_{max}}$ zu. Einerseits sollen die gewählten Nachbarschaftsstrukturen einen großen Bereich des Suchraums erschließen, andererseits aber auch in angemessener Zeit durchsucht werden können. Die konkrete Auswahl der Nachbarschaftsstrukturen hängt sowohl von der Beschaffenheit des jeweiligen Problems als auch von der gewünschten Lösungsqualität ab. Die Reihenfolge der Nachbarschaftsstrukturen wird dabei oft durch ihre Größe bzw. ihre Komplexität vorgegeben, sodass kleinere Nachbarschaften öfter durchsucht werden als größere.

6.2.2 Reduced Variable Neighbourhood Search

Während *Variable Neighbourhood Descent* versucht, ausgehend von einer Startlösung eine bestmögliche Lösung zu erreichen, fokussiert *Reduced Variable Neighbourhood Search* (RVNS) (siehe Algorithmus 6.4) auf die Erschließung neuer Bereiche des Suchraums und das Verlassen von bereits erreichten lokalen Optima. Dies geschieht ebenfalls unter der Verwendung einer Reihe von Nachbarschaftsstrukturen $\mathcal{N}_1, \ldots, \mathcal{N}_{k_{max}}$, die allerdings in anderer Weise eingesetzt werden. Anstatt wie im Fall der VND die Nachbarschaft $\mathcal{N}_k(x)$ nach verbesserten Lösungen zu durchsuchen, wird eine zufällige Lösung aus $\mathcal{N}_k(x)$ gewählt. Dies wird

in [26] als *Shaking* bezeichnet.

Algorithmus 6.4 : Reduced Variable Neighbourhood Search

Eingabe : Startlösung x_s

$x \leftarrow x_s$
$k \leftarrow 1$
repeat
  repeat
    $x' \leftarrow$ *random-neighbour*$(\mathcal{N}_k(x))$
    if $f(x') \leq f(x)$ **then**
      $x \leftarrow x'$
      $k \leftarrow 1$
    else
      $k \leftarrow k + 1$
  until $k > k_{max}$
until Abbruchbedingung erfüllt

Die in der RVNS verwendeten Nachbarschaftstrukturen sind im Allgemeinen so sortiert, dass $\mathcal{N}_1(x) \subseteq \mathcal{N}_2(x) \subseteq \ldots \subseteq \mathcal{N}_{k_{max}}(x)$, d.h. die nächstgrößere Nachbarschaft von x enthält die vorhergehende Nachbarschaft. Dies kann mit wachsendem k zur Erzeugung von Lösungen mit immer größerer Entfernung von x führen.

6.2.3 General Variable Neighbourhood Search

Das Verfahren *General Variable Neighbourhood Search* kombiniert das Vorgehen von VND und RVNS. Während RVNS verwendet wird, um lokale Optima zu verlassen, wird VND eingesetzt, um ausgehend von der durch das Shaking ermittelten Lösung x' ein lokales Optimum x'' zu finden. Ist diese Lösung besser als die aktuell beste Lösung, wird sie als neue beste Lösung übernommen. Dabei ist anzumerken, dass sich die im Rahmen der RVNS verwendeten Nachbarschaften $\mathcal{N}_1, \ldots, \mathcal{N}_{k_{max}}$ von den im Rahmen der VND eingesetzten Nachbarschaften $N_1, \ldots, N_{l_{max}}$ unterscheiden.

Algorithmus 6.5 : General Variable Neighbourhood Search

Eingabe : Startlösung x_s

$x \leftarrow x_s$
$k \leftarrow 1$
repeat
  repeat
    $x' \leftarrow$ *random-neighbour*$(\mathcal{N}_k)$
    $l \leftarrow 1$
    repeat
      $x'' \leftarrow$ *improvement-step*$(N_l(x'))$
      if $f(x'') \leq f(x')$ **then**
        $x' \leftarrow x''$
        $l \leftarrow 1$
      else
        $l \leftarrow l + 1$
    until $l > l_{max}$
    if $f(x') \leq f(x)$ **then**
      $x \leftarrow x'$
      $l \leftarrow 1$
    else
      $l \leftarrow l + 1$
  until $k > k_{max}$
until Abbruchbedingung erfüllt

Kapitel 7

Anwendung von Variable Neighbourhood Search auf VSLRB

Das folgende Kapitel beschreibt die Nachbarschaftsstrukturen, die in der GVNS für VSLRB zum Einsatz kommen. Zwei dieser Nachbarschaftsstrukturen verwenden Techniken der *Very Large-Scale Neighbourhood* (VLSN) *Search*, um speziell definierte Nachbarschaften mit einer sehr großen Anzahl von Nachbarlösungen effizient durchsuchen zu können.

Die in Abschnitt 7.2.3 beschriebene k-Server MIP Neighbourhood verwendet die in Kapitel 5 entwickelte MIP-Formulierung von VSLRB, um ein Unterproblem, bestehend aus einer Auswahl von k Servern, exakt zu lösen. Diese k Server werden mit einer einfachen Heuristik so gewählt, dass in der Auswahl sowohl Server enthalten sind, die ihre Last-Zielwerte überschreiten, als auch solche, die ihre Last-Zielwerte unterschreiten. Sofern die gewählten Server eine Überschneidung der von ihnen akzeptierten Datei-Typen aufweisen, ist zu erwarten, dass durch die optimale Lösung eines auf diese Weise erzeugten Unterproblems auch eine verbesserte Lösung für das originale Problem erzielt werden kann.

Die Verwendung von auf Methoden der mathematischen Programmierung basierenden Nachbarschaftsstrukturen stellt einen Ansatz zur Hybridisierung von exakten und metaheuristischen Optimierungsverfahren dar. Durch Ausnutzung der komplementären Eigenschaften von Metaheuristiken und exakten Verfahren werden Synergie-Effekte erzielt, die oftmals zu besseren Lösungen als bei isolierter Verwendung dieser beiden Ansätze führen [41, 42]. In der Terminologie von [41] handelt es sich bei dem in dieser Arbeit vorgestellten Ansatz um einen *integrativen* und *schwach gekoppelten* hybriden Ansatz, da der MIP-Ansatz als Unterprozedur der VNS eingesetzt wird, ohne die Charakteristika eines der beiden Verfahren zu verändern.

Die in Abschnitt 7.2.4 beschriebene Cyclic Exchange Neighbourhood [5,44,45] ist eine weitere Technik zur effizienten Durchsuchung einer sehr großen Nachbarschaft, die zur Lösung von Partitionierungsproblemen entwickelt wurde. Ein solches Problem besteht in der Teilung einer vorgegebenen Menge von Elementen in eine gegebene Anzahl von Teilmengen mit minimalen Gesamtkosten. Als logische Erweiterung einer Swap-Nachbarschaften bestehen Züge in einer Cyclic Exchange Neighbourhood aus der zyklischen Verschiebung von k Elementen über k Teilmengen. Die effiziente Ermittlung solcher zyklischer Verschiebungen geschieht durch Suche von speziellen Zyklen mit minimalen Kosten in einem sogenannten *Improvement-Graph*. Dieser enhält einen Knoten für jedes Element und eine Kante für jede mögliche Verschiebung eines Elements von einer Teilmenge in eine andere.

die Kantenkosten entsprechen der Zielfunktionswertdifferenz, welche durch die Verschiebung des jeweiligen Elements enstünde. Gültige Zyklen müssen die Eigenschaften der *Teilmengendisjunktheit* aufweisen, d.h. jedes der Elemente eines gültigen Zyklus muss einer unterschiedlichen Teilmenge angehören. Die Suche nach solchen Zyklen stellt zwar ihrerseits wiederum ein $\mathcal{NP}$-schweres Problem, dieses kann allerdings durch schnelle Heuristiken zufriedenstellend gelöst werden.

7.1 Operationen auf Lösungen von VSLRB

Die beiden grundlegenden Operationen, die in den in diesem Abschnitt beschriebenen Nachbarschaftsstrukturen verwendet werden, um eine gegebene Lösung einer Instanz von VSLRB in eine neue Lösung zu transformieren, sind in Algorithmus 7.1 und Algorithmus 7.2 dargestellt.

Algorithmus 7.1 : assign

Eingabe : Lösung einer VSLRB-Instanz S, Video-Objekt $i \in F$, Server $j \in C$ und eine Anzahl von Zugriffen $a \in \mathbb{Z}^+$

1 **if** $Q(i,j) = 0$ **then**
2 $\quad F_j \leftarrow F_j \cup i$
3 $\quad updateReorgObjective(i,j)$
4 $Q(i,j) \leftarrow Q(i,j) + a$
5 $\mathcal{L}'(j) \leftarrow \mathcal{L}(j)$
6 $\mathcal{L}(j) \leftarrow \mathcal{L}(j) + a * q_i$
7 $updateLoadObjective(j, \mathcal{L}'(j))$

Algorithmus 7.2 : unassign

Eingabe : Lösung einer VSLRB-Instanz S, Video-Objekt $i \in F$, Server $j \in C$ und eine Anzahl von Zugriffen $a \in \mathbb{Z}^+$

1 **if** $Q(i,j) = a$ **then**
2 $\quad F_j \leftarrow F_j \setminus i$
3 $\quad updateReorgObjective(i,j)$
4 $Q(i,j) \leftarrow Q(i,j) - a$
5 $\mathcal{L}'(j) \leftarrow \mathcal{L}(j)$
6 $\mathcal{L}(j) \leftarrow \mathcal{L}(j) - a * q_i$
7 $updateLoadObjective(j, \mathcal{L}'(j))$

Die Prozedur `assign()` erhöht die Zuweisung eines Video-Objekts $i \in F$ zu einem Server $j \in C$ um a Zugriffe. Ist bisher noch keine solche Zuweisung vorhanden, muss ein Replikat von i in F_j aufgenommen werden. Analog verringert die Prozedur `unassign()` die Zuweisung von i zu j um a Zugriffe. Wurden damit alle Zugriffe der Zuweisung entfernt, muss auch das entsprechende Replikat von i aus F_j entfernt werden.

In jedem Fall muss nach Durchführung einer der beiden Operationen der Wert des ersten Teils der Zielfunktion aktualisiert werden. Dies ist, wie in Algorithmus 7.3 dargestellt, unter der Vorraussetzung der inkrementellen Berechnung von $\mathcal{L}(j)$ in `assign()` und `unassign()` in konstanter Zeit möglich. Weiters muss bei Änderungen an der Menge der

Algorithmus 7.3 : updateLoadObjective

Eingabe : Lösung einer VSLRB-Instanz S, Server $j \in C$, bisherige Last $\mathcal{L}'(j)$

1 $Z_1 \leftarrow Z_1 - |\eta_j - \mathcal{L}'(j)| + |\eta_j - \mathcal{L}(j)|$

Algorithmus 7.4 : updateReorgObjective

Eingabe : Lösung einer VSLRB-Instanz S, Video-Objekt $i \in F$, Server $j \in C$

1 **if** $i \in F_j$ **then**
2 $\quad Z_2 \leftarrow Z_2 + \mathcal{R}(i,j)$
3 **else**
4 $\quad Z_2 \leftarrow Z_2 - \mathcal{R}(i,j)$

Replikate F_j eine Aktualisierung des Werts des zweiten Teils der Zielfunktion vorgenommen werden (siehe Algorithmus 7.4). Auch dies ist in konstanter Zeit möglich, falls die idealisierte Dauer der Übertragung von Video-Objekt i zu Server j, in Algorithmus 7.4 als $\mathcal{R}(i,j)$ bezeichnet, im Voraus berechnet wird:

$$\mathcal{R}(i,j) = \begin{cases} 0 & \text{falls } i \in F_j \\ \sum_{\{k \in C | i \in F_k\}} \mathcal{T}(i,k,j) & \text{sonst} \end{cases}$$

Damit besitzen sowohl `assign()` als auch `unassign()` eine Laufzeit von $O(1)$. Darauf aufbauend werden in den Nachbarschaftsstrukturen noch die Operationen `move()` (siehe Algorithmus 7.5) und `swap()` (siehe Algorithmus 7.6) zur Verschiebung von Zugriffen von einem Server zu einem anderen Server bzw. zum Austausch von Zugriffen zwischen zwei Servern verwendet. Diese beiden Operationen besitzen ebenfalls eine Laufzeit von $O(1)$.

Algorithmus 7.5 : move

Eingabe : Lösung einer VSLRB-Instanz S, Video-Objekt $i \in F$, Server $j, k \in C$, Anzahl von Zugriffen $a \in \mathbb{Z}^+$

1 $unassign(i,j,a)$
2 $assign(i,k,a)$

7.2 Nachbarschaftsstrukturen der VND

Der folgende Abschnitt beschreibt die Nachbarschaftsstrukturen, die in der Variable Neighbourhood Descent-Strategie für VSLRB zum Einsatz kommen. Jeder Unterabschnitt widmet sich einer dieser Nachbarschaftstrukturen, wobei jeweils eine formale Definition der Nachbarschaft, ihre Größe, die Strategie zur Durchsuchung der Nachbarschaft sowie gegebenenfalls ein theoretischer Hintergrund der verwendeten Verfahren angegeben ist.

7.2.1 Access-Move Neighbourhood

Diese Nachbarschaft enthält alle Nachbarlösungen, die durch die Verschiebung eines Zugriffs von einem Server zu einem anderen Server erzeugt werden können. Sie kann formal angeschrieben werden als:

Algorithmus 7.6 : swap

Eingabe : Lösung einer VSLRB-Instanz S, Video-Objekte $i, f \in F$, Server $j, c \in C$, Anzahl von Zugriffen $a, b \in \mathbb{Z}^+$

1 $unassign(i, j, a)$
2 $unassign(f, c, b)$
3 $assign(i, c, a)$
4 $assign(f, j, b)$

$$\mathcal{N}_{\text{Move}}(S) = \{S' \mid S' \text{ kann aus } S \text{ durch die Operation } \texttt{move}(i, j, k, 1), \\ (i, j) \in F \times C \mid Q(i, j) > 0, j \neq k, t_i \in T_k \text{ erzeugt} \\ \text{werden}\}$$

Diese Nachbarschaft enthält $O(m^2 n)$ Nachbarlösungen, welche leicht durch Ausführung aller von S aus möglichen `move()`-Operationen, die keine Nebenbedingungen verletzen, aufgezählt werden können. Da aufgrund der inkrementellen Berechnung der Zielfunktion jede `move()`-Operation in konstanter Zeit durchgeführt werden kann, ist die vollständige Durchsuchung der Nachbarschaft in einer Zeit von $O(m^2 n)$ möglich.

Die Durchsuchung dieser Nachbarschaft erfolgt im Rahmen der VND für VSLRB mittels einer *Next Improvement*-Strategie.

7.2.2 Access-Swap Neighbourhood

Diese Nachbarschaft enthält alle Nachbarlösungen, die durch Vertauschung eines Zugriffs mit einem Zugriff auf eine andere, einem anderen Server zugeordnete Datei erzeugt werden können. Sie kann formal angeschrieben werden als:

$$\mathcal{N}_{\text{Swap}}(S) = \{S' \mid S' \text{ kann aus } S \text{ durch die Operation } \texttt{swap}(i, j, 1, f, c, 1), \\ (i, j) \in F \times C \mid Q(i, j) > 0, (f, c) \in F \times C \mid Q(f, c) > 0, \\ j \neq c, i \neq f, t_i \in T_c, t_f \in T_j \text{ erzeugt werden}\}$$

Diese Nachbarschaft enthält $\leq \frac{(n+1)n}{2} \frac{(m+1)m}{2} = O(n^2 m^2)$ Nachbarlösungen, da jene Server bzw. Dateien, die bereits als "Quelle" eines Austauschs verwendet wurden, nicht mehr als "Ziel" eines anderen Austauschs herangezogen werden müssen. Die Nachbarlösungen können auf einfache Weise durch Ausführung aller von S aus zulässigen `swap()`-Operationen aufgezählt werden. Da auch jede `swap()`-Operation in konstanter Zeit ausgeführt werden kann, ist die vollständige Durchsuchung der Nachbarschaft ebenfalls in einer Zeit proportional zu ihrer Größenordnung möglich.

Ebenso wie die im vorhergehenden Abschnitt beschriebene Move Neighbourhood wird diese Nachbarschaft unter Verwendung einer *Next Improvement*-Strategie durchsucht.

7.2.3 k-Server MIP Neighbourhood

Diese Nachbarschaft der VND für VSLRB beruht auf der exakten Lösung eines in geeigneter Weise erzeugten Unterproblems unter Verwendung der MIP-Formulierung für VSLRB

(siehe Abschnitte 5.1 und 5.2).

Nachbarschaftsstrukturen auf Basis einer Formulierung als mathematisches Programm sind eine mögliche Form der Hybridisierung von exakten und heuristischen Methoden zur Lösung von kombinatorischen Optimierungsproblemen (siehe [41, 42]). Durch Formulierung der Suche nach der besten Nachbarlösung als Optimierungsproblem und durch Verwendung von spezialisierter Solver-Software für ganzzahlige bzw. gemischt-ganzzahlige lineare Programme können im Sinne einer *Very Large-Scale Neighbourhood Search* auch große Nachbarschaften in angemessener Zeit nach der bestmöglichen Nachbarlösung durchsucht werden.

Solche Nachbarschaftsstrukturen bzw. in entsprechender Weise definierte Unterprobleme finden sich in der Literatur unter anderem in Palpant et al. [39] für das *Resource-Constrained Project Scheduling Problem*, in Duarte et al. [15] für das *Referee Assignment Problem* und in Prandtstetter und Raidl [40] sowie Estellon et al. [16] für das *Car Sequencing Problem*.

Zur Bildung eines geeigneten Unterproblems müssen l der insgesamt z Variablen der ILP- bzw. MIP-Formulierung des jeweiligen Problems ausgewählt werden, aus denen sich das Unterproblem zusammensetzt. Alle anderen $z - l$ Variablen werden auf ihren aktuellen Wert fixiert. Die Auswahl der l "freien" Variablen kann entweder auf zufällige Weise oder, wie im folgenden Abschnitt beschrieben, mittels einer problemangepassten Heuristik erfolgen.

7.2.3.1 Erzeugung des Unterproblems

Die im Folgenden beschriebene Strategie zur Auswahl der l freien Variablen beruht auf den guten Resultaten des reinen MIP-Ansatzes für Instanzen mit einer kleinen Anzahl von Servern (siehe Abschnitt 9.2). Da der reine MIP-Ansatz dabei auch für größere Anzahlen von Dateien zuverlässig und schnell sehr gute Lösungen liefert, erscheint es naheliegend, für das Unterproblem jene Variablen auszuwählen, welche die gesamten Zuweisungen einiger weniger ausgewählter Server beschreiben. Um durch optimale Lösung dieses Unterproblems eine Zielfunktionsverbesserung für das ursprüngliche Problem zu erreichen, werden sowohl Server in das Unterproblem aufgenommen, die ihren Last-Zielwert η_j überschreiten, als auch Server, die ihren Last-Zielwert unterschreiten. Es ist zu erwarten, dass durch optimale Lösung des so erzeugten Unterproblems eine geringere Abweichung von den jeweiligen η_j und damit auch ein besserer Zielfunktionswert für das ursprüngliche Problem erreicht werden kann.

Algorithmus 7.7 zeigt das Vorgehen, um die k Server des zur Durchsuchung der Nachbarschaftsstruktur eingesetzten Unterproblems auszuwählen: Die Server werden absteigend nach der Differenz ihrer aktuellen Last und ihrer Ziellast η_j sortiert. Danach werden $\frac{k}{2}$ Server vom Anfang der so erzeugten Liste ausgewählt. Die fehlenden $\frac{k}{2}$ Server werden beginnend vom Ende der Serverliste so gewählt, dass Überschneidungen zwischen den von ihnen akzeptierten Dateitypen und den akzeptierten Dateitypen der ersten $\frac{k}{2}$ Server bestehen. Ohne diese Einschränkung könnten Unterprobleme mit disjunkten Mengen von akzeptierten Dateitypen entstehen, die keine Verbesserung des Zielfunktionswertes erlauben. Ergebnis dieser Auswahl ist eine Menge von Servern C'.

Ausgehend von der so gewählten Menge von Servern C' wird eine vollständige neue Instanz

Algorithmus 7.7 : SelectServers

Eingabe : Lösung einer VSLRB-Instanz S

```
1  sorted ← Sortiere die Server j = 1...m absteigend nach L(j) − η_j
2  C' ← ∅
3  coveredTypes ← ∅
4  for l ← 1 to k/2 do
5  |  C' ← C' ∪ sorted[l]
6  |  coveredTypes ← T_sorted[l] \ {Thumbnail}
7  l ← m
8  while |C'| < k ∧ l > k/2 do
9  |  if coveredTypes ∩ T_sorted[l] ≠ ∅ then
10 |  |  C' ← C' ∪ sorted[l]
11 |  l ← l − 1
```

von VSLRB erzeugt, die mit Hilfe ihrer Repräsentation als MIP exakt gelöst wird. Um die Symbole, welche die Instanz des Unterproblems beschreiben, von jenen der Beschreibung der ursprünglichen Instanz unterscheiden zu können, werden alle Symbole, welche die Instanz des Unterproblems betreffen, mit einem Strich "′" versehen: Bezeichnet x eine Größe der ursprünglichen Instanz, bezeichnet x' die gleiche Größe in der Instanz des Unterproblems. Die folgende Aufstellung fasst die Berechnung der weiteren Größen der neuen Instanz ausgehend von der Menge der gewählten Server C' zusammen, sofern sie von der jeweiligen ursprünglichen Definition (siehe Abschnitte 3.2.1 und 3.2.2) abweichen:

$$F' = \bigcup_{j \in C'} F_j$$

$$A'_i = A_i \cap C'$$

$$q'_i = \sum_{j \in C'} Q(i,j) \quad \forall i \in F'$$

$$\overline{q}'_i = \sum_{j \in C'} \overline{Q}(i,j) \quad \forall i \in F'$$

$$\overline{Q}'(i,j) = \overline{Q}(i,j) \quad \forall i \in F', j \in C'$$

Weiters müssen die fairen Lasten Λ'_j sowie die Last-Zielwerte η'_j der Server des Unterproblems ausgehend von den tatsächlich vorhandenen Dateien bzw. Zugriffen neu berechnet werden, um die Instanz zu vervollständigen. Formal kann die so definierte Nachbarschaftsstruktur angeschrieben werden als

$$\mathcal{N}_{k-\mathrm{MIP}}(S) = \{T \mid T = fixed(S) \cup S'\}$$

wobei $fixed(S)$ die Menge aller fixierten Variablenbelegungen von S sowie S' eine zulässige Lösung des Unterproblems bezeichnet. Insgesamt existieren

$$O\left(\prod_{i \in F'} \binom{|A'_i| + q'_i - 1}{q'_i}\right)$$

viele Lösungen des Unterproblems und damit ebenso viele Nachbarlösungen in der Nachbarschaft $\mathcal{N}_{k-MIP}$, da für jede Datei $i \in F'$ des Unterproblems q'_i viele Zugriffe existieren, die auf $|A'_i|$ viele Server aufgeteilt werden müssen. Dafür existieren jeweils

$$\binom{|A'_i| + q'_i - 1}{q'_i}$$

viele Möglichkeiten.

Die Durchsuchung dieser Nachbarschaft geschieht unter Verwendung des kommerziellen ILP-Solvers CPLEX, wobei eine zeitbeschränkte *Best Neighbour*-Strategie verfolgt wird, d.h. es wird die beste Lösung des Unterproblems verwendet, die innerhalb eines vorgegebenen Zeitlimits durch den Solver gefunden werden kann.

7.2.4 Cyclic Exchange Neighbourhood

Der zyklische Austausch von Elementen zwischen Teilmengen stellt eine natürliche Erweiterung der einfachen 2-Exchange bzw. Swap-Nachbarschaft dar. Diese Nachbarschaft enthält im Normalfall eine wesentlich größere Anzahl von Nachbarlösungen als eine Swap-Nachbarschaft. Um die Nachbarschaft trotzdem in angemessener Zeit durchsuchen zu können, geschieht die Suche nach besseren Nachbarlösungen im Gegensatz zur Swap-Nachbarschaft nicht durch naive vollständige Enumeration, sondern durch die Suche nach speziellen Kreisen mit negativen Kosten in einer Graphen-Repräsentation der Nachbarschaft.

Diese Methode wurde schon mehrmals erfolgreich auf spezielle Partitionierungsprobleme angewandt, zum Beispiel von Ahuja et al. [5] auf das *Capacitated Minimum Spanning Tree Problem*, von Thompson und Psaraftis [45], Fahrion und Wrede [17] und Ibaraki et al. [30] auf Varianten des *Vehicle Routing Problem* sowie Ösoy und Pinar [43] auf das *Capacitated Vertex P-Center Problem*.

Abschnitt 7.2.4.1 gibt einen kurzen Überblick über die Grundlagen des zyklischen Austauschs, während Abschnitt 7.2.4.2 die Anwendung dieser Methode auf das VSLRB-Problem beschreibt.

7.2.4.1 Theorie des zyklischen Austauschs von Elementen

Der folgende Abschnitt beschreibt den theoretischen Hintergrund der Nachbarschaftssuche in einer "Cyclic Exchange Neighbourhood". Die weiteren Ausführungen beziehen sich auf das wie folgt definierte generische Partitionierungsproblem [7, 44]:

Definition 7.2.1 (Generic Partitioning)**.** Gegeben sei eine endliche Menge $A = \{a_1, a_2, \ldots, a_n\}$ von n Elementen, eine Kostenfunktion $c : \mathcal{P}(A) \to \mathbb{R}$, wobei $\mathcal{P}(A)$ die Potenzmenge von A bezeichnet, und $K \in \mathbb{Z}^+$. Gesucht ist eine Menge von paarweise disjunkten Teilmengen $S = \{S_1, S_2, \ldots, S_K\}$, für die gilt $\bigcup_{i=1}^{K} S_i = A$, sodass die Gesamtkosten $C(S) = \sum_{i=1}^{K} c(S_i)$ minimiert werden. Es wird im Folgenden immer davon ausgegangen, dass die Kosten jeder Teilmenge S_i unabhängig von der Teilmengenzuordnung der Elemente sind, die sich nicht in der Teilmenge S_i befinden ("separable over subsets" [6]).

Thompson und Orlin [44] beschreiben neben diesem dort als *Unrooted Partitioning Problem* bezeichneten Problem auch den allgemeineren Fall des *Rooted Partitioning Problem*,

das als eine Abstraktion für eine Vielzahl von speziellen Partitionierungsproblemen dienen kann. Dieser Abschnitt beschränkt sich aber auf das einfachere, als Abstraktion für VSLRB ausreichende *Unrooted Partitioning Problem*.

Weiters ist anzumerken, dass die Berechnung der Kosten einer Teilmenge nicht notwendigerweise eine triviale Aufgabe darstellt, sondern auch das Lösen schwieriger Unterprobleme umfassen kann. Als Beispiel sei das *Vehicle Routing Problem* genannt, in dem zur Auswertung von c jeweils ein Travelling Salesman Problem (TSP) pro Teilmenge gelöst werden muss.

Definition 7.2.2 (Zyklischer Austausch)**.** Gegeben sei eine Lösung des generischen Partitionierungsproblems $S = \{S_1, S_2, \ldots, S_K\}$ sowie eine Folge von Elementen $i_1 - i_2 - \ldots - i_r - i_1$ aus A, mit der Eigenschaft, dass $S[i_l] \neq S[i_k]\ \forall l, k \in \{1 \ldots r\},\ l \neq k$, wobei $S[i_s], s = 1 \ldots r$ jene Menge in S bezeichnet, die i_s enthält ("Teilmengendisjunktheit"). Ein *zyklischer Austausch* bzw. ein *zyklischer Transfer* ist die simultane Verschiebung der Elemente i_l von $S[i_l]$ nach $S[i_{l+1}]$, wobei $l = 1 \ldots r$ und $i_{r+1} = i_1$.

Ein zyklischer Austausch wird als *zulässig* bezeichnet, wenn die durch den Austausch erzeugten Mengen $\{i_{p-1}\} \cup S[i_p] \setminus \{i_p\}$, $p = 1, \ldots, r$, $i_0 = i_r$ alle über die Einschränkungen des generischen Partitionierungsproblems hinausgehenden problemspezifischen Einschränkungen erfüllen. Unter der Voraussetzung der unabhängigen Berechenbarkeit der Kosten der Teilmengen sowie der Teilmengendisjunktheit kann die Veränderung der Gesamtkosten bei Durchführung des zyklischen Austauschs angeschrieben werden als [5]:

$$C(S') - C(S) = \sum_{p=1}^{r} c(\{i_{p-1}\} \cup S[i_p] \setminus \{i_p\}) - c(S[i_p])$$

Ein solcher zyklischer Austausch wird als *profitabel* bezeichnet, falls $C(S') - C(S) < 0$.

Basierend auf dem so definierten zyklischen Austausch von Elementen kann die folgende Nachbarschaftsstruktur für das generische Partitionierungsproblem definiert werden:

Definition 7.2.3 (Cyclic Exchange Neighbourhood)**.** Sei $S = \{S_1, S_2, \ldots, S_K\}$ eine Lösung des generischen Partitionierungsproblems. Dann bezeichnet $\mathcal{N}_{CX}(S) = \{S' \mid S'$ kann aus S durch einen zyklischen Austausch $i_1 - \ldots - i_r - i_1$, $2 \leq r \leq K$ erzeugt werden$\}$ die Nachbarschaft von S bezüglich des zyklischen Austauschs von Elementen.

Für einen fixierten Wert von K existieren in einer solchen zyklischen Nachbarschaft $O(n^K)$ Nachbarlösungen von S. Falls K von n abhängt, kann diese Nachbarschaft auch eine exponentielle Größe annehmen [4]. Wie kann nun eine solch große Nachbarschaftsstruktur sinnvoll durchsucht werden? In der Literatur werden dazu heuristische Verfahren auf Basis eines sogenannten *Improvement Graph* [4,5,7] bzw. *Auxilliary Graph* [44] eingesetzt. Das Auffinden von profitablen zyklischen Vertauschungen wird dabei auf das Auffinden von speziellen Kreisen mit negativen Kosten im Improvement-Graph zurückgeführt.

Definition 7.2.4 (Improvement-Graph)**.** Gegeben sei eine beliebige Instanz sowie eine Lösung S des generischen Partitionierungsproblems. Der Improvement-Graph $G = \langle V, E, \alpha \rangle$ ist ein gerichteter, gewichteter Graph, der auf die folgende Weise erzeugt wird:

- Für jedes Element $a \in A$ erzeuge einen Knoten v_a in V

- Für jedes Paar $(i, j) \in A \times A$, $i \neq j$, $S[i] \neq S[j]$ erzeuge eine Kante (i, j) in E falls die Verschiebung von i nach $S[j]$ sowie die gleichzeitige Entfernung von j aus $S[j]$ einen zulässigen Teil-Austausch darstellt
- Die Bewertungsfunktion α weist jeder Kante die Kostenveränderung bei Durchführung des durch die Kante repräsentierten Teil-Austauschs zu:

$$\alpha_{ij} = c(\{i\} \cup S[j] \setminus \{j\}) - c(S[j])$$

Der folgende Satz verknüpft die Suche nach speziellen Zyklen in diesem Graphen mit der Suche nach profitablen zyklischen Transfers:

Satz 7.2.5. *Jeder Zyklus mit negativen Kosten in einem laut Definition 7.2.4 erzeugten Graphen, der nur Knoten enthält, deren korrespondierende Elemente $a \in A$ verschiedenen Teilmengen in S zugeordnet sind, entspricht einem profitablen zyklischen Austausch.*

Beweis. Siehe [7]. □

Die Durchsuchung der Nachbarschaft $\mathcal{N}_{CX}$ kann also mit Hilfe eines Verfahrens zur Ermittlung von teilmengendisjunkten Zyklen mit negativen Kosten im Improvement-Graph erfolgen. Zwar ist dieses Problem $\mathcal{NP}$-schwer [5], es existieren allerdings schnelle Heuristiken, wie zum Beispiel der in [7] beschriebene *Modified label-correcting Algorithm* (Siehe Algorithmus 7.8).

Algorithmus 7.8 : Modified Label-Correcting Algorithm

Eingabe : Improvement-Graph $G = (V, E, \alpha)$, Startknoten $s \in V$

```
1  foreach v ∈ V \ s do
2      d(v) ← ∞
3      pred(v) ← null
4  d(s) ← 0
5  LIST ← ⟨s⟩
6  while LIST ≠ ∅ do
7      u ← take(LIST)
8      if P[u] ist teilmengendisjunkt then
9          foreach (u, v) ∈ E do
10             if d(v) > d(u) + α_uv then
11                 if P[u] enthält v then
12                     Teilmengendisjunkten Zyklus merken oder abbrechen
13                 else if P[u] ∪ v ist teilmengendisjunkt then
14                     d(v) ← d(u) + α_uv
15                     pred(v) ← u
16                     LIST ← LIST ∪ v
```

Dieser Algorithmus ist eine Variante des Label-Correcting Algorithmus zur Ermittlung der kürzesten Pfade in einem gerichteten Graphen von einem Startknoten s zu allen anderen Knoten. Der Algorithmus speichert für jeden Knoten v eines Graphen die Eigenschaften $d(v)$, das "distance label" von v sowie $pred(v)$, den "predecessor index". $d(v)$ ist entweder ∞, in diesem Fall wurde noch kein Pfad von s nach v gefunden, ansonsten gibt $d(v)$ die

Länge des aktuell besten Pfades von s nach v an. $pred(v)$ speichert einen Verweis auf den Vorgänger von v auf dem aktuell besten Pfad von s nach v. Durch Zurückverfolgung der $pred(v)$ ist jederzeit die Ermittlung des aktuell kürzesten Pfades von s nach u möglich. Dieser Pfad wird im Folgenden als $P[u]$ bezeichnet. Der Algorithmus initialisiert zu Beginn $d(v)$ mit ∞ und $pred(v)$ mit *null* für alle $v \in V$, sowie $d(s)$ mit 0. Der Algorithmus identifiziert Kanten (u, v), welche die Optimalitätsbedingung $d(v) \leq d(u) + \alpha_{uv}$ verletzen und führt in einem solchen Fall einen "distance update step" durch und aktualisiert $d(v)$ auf $d(u) + \alpha_{uv}$ sowie $pred(v)$ auf u. Dies geschieht solange, bis keine Kante mehr die Optimalitätsbedingung verletzt. Um solche Kanten effizient aufzufinden, verwaltet der Algorithmus weiters eine Liste $LIST$, welche die Eigenschaft besitzt, dass Knoten, die ausgehende Kanten besitzen, welche die Optimalitätsbedingung verletzen, in ihr enthalten sein müssen.

Um diesen Algorithmus zu einer Heuristik zum Auffinden teilmengendisjunkter negativer Kreise zu erweitern, wird immer dann, wenn ein Knoten u aus $LIST$ entfernt wird, die Teilmengendisjunktheit von $P[u]$ überprüft. Ist $P[u]$ nicht teilmengendisjunkt, wird dieser Pfad nicht weiterverfolgt. Andernfalls werden alle von u ausgehenden Kanten (u, v) auf die Verletzung der Optimalitätsbedingung überprüft. Ist dies der Fall, sind drei Fälle zu unterscheiden:

1. $P[u]$ enthält bereits v: in diesem Fall wurde ein teilmengendisjunkter Kreis mit negativen Kosten gefunden.
2. $P[u] \cup v$ ist teilmengendisjunkt: der Pfad wird weiterverfolgt, indem $d(v)$ sowie $pred(v)$ korrigiert werden und v in $LIST$ aufgenommen wird.
3. Andernfalls ist jede Fortsetzung von $P[u]$ kein teilmengendisjunkter Kreis und der Pfad wird nicht weiterverfolgt.

Die Worst-Case Laufzeit dieses Algorithmus hängt von der Strategie ab, mit der Knoten aus $LIST$ entfernt bzw. zu $LIST$ hinzugefügt werden. In [27] werden dafür unter anderem die folgenden Strategien beschrieben:

- Verwendung einer FIFO-Strategie: Knoten werden am Ende von $LIST$ hinzugefügt und vom Anfang von $LIST$ entfernt. Der Algorithmus ist bei Verwendung dieser Strategie äquivalent zum Bellmann-Ford-Moore-Algorithmus mit einer Worst-Case Laufzeit von $O(|V||E|)$.
- Verwendung einer Double-Ended Queue ("Deque"): Knoten werden am Anfang von $LIST$ hinzugefügt, wenn sie sich bereits einmal in $LIST$ befunden haben, ansonsten am Ende, und immer vom Anfang von $LIST$ entfernt. Diese Listenorganisation führt zu einer Worst-Case Laufzeit von $O(|V|2^{|V|})$ bei einem in der Praxis besseren Laufzeitverhalten bei spärlichen Graphen.
- Verwendung von zwei Queues, wobei bereits angetroffene Knoten in die erste Queue aufgenommen wird, und solche, die noch nicht angetroffen wurden, in die zweite. Knoten werden zuerst aus der ersten Queue entfernt. Diese Implementierung besitzt eine Worst-Case Laufzeit von $O(|V||E|)$.

Die jeweilige Worst-Case Laufzeit erhöht sich durch die Modifikation um den Faktor K, da die Zurückverfolgung des aktuellen Pfades zu u immer nach maximal K Schritten abgebrochen wird. Da nur K verschiedene Teilmengen auftreten können, muss bei Zurückverfolgung von $P[u]$ bei nicht gegebener Teilmengendisjunktheit nach spätestens K Schritten

eine bereits aufgetretene Teilmenge erneut auftreten. Ist andererseits der aktuelle Pfad zu u teilmengendisjunkt, kann er nicht mehr als K Knoten enthalten.

7.2.4.2 Anwendung von zyklischen Vertauschungen auf VSLRB

Der folgende Abschnitt beschreibt die Implementierung der Cyclic Exchange Neighbourhood, die in der VND für VSLRB zum Einsatz kommt.

Erzeugung des Improvement-Graph

VSLRB kann als ein Partitionierungsproblem im Sinne des vorigen Abschnitts betrachtet werden, in dem die m Server den K zu ermittelnden Teilmengen und die Gesamtheit aller abzuwickelnden Zugriffe der Menge A der zu partitionierenden Elemente des generischen Partitionierungsproblems entspricht. Ein Problem stellt dabei die im Vergleich zur Anzahl von Servern oder Dateien oftmals wesentlich größere Anzahl an Zugriffen dar, die bei direkter Verwendung als Knoten im Improvement-Graph zu für den praktischen Einsatz zu großen Graphen führt. Da allerdings alle einem Server zugeordneten Zugriffe auf die selbe Datei gleichartig sind und während eines zyklischen Austauschs immer nur ein einzelner Zugriff verschoben wird, kann ein äquivalenter Improvement-Graph auf Basis der Datei-Zuweisungen definiert werden:

Definition 7.2.6 (Improvement-Graph für VSLRB)**.** Gegeben sei eine Instanz von VSLRB sowie eine entsprechende Lösung S. Der Improvement-Graph wird auf die folgende Weise konstruiert:

- Für jede in S existierende Zuweisung d.h. für alle $(f, c) \in F \times C$ für die gilt $Q(f, c) > 0$, erzeuge den Knoten v_{fc} in V
- Für jedes Paar $(v_{fc}, v_{gd}) \in V \times V$ erzeuge eine Kante in E falls $c \neq d$, $f \neq g$ und $t_f \in T_d$
- Die Bewertungsfunktion weist jeder Kante $(v_{fc}, v_{gd}) \in E$ die Kostenveränderung zu, die entsteht, wenn ein Zugriff auf f Server d zugewiesen wird und gleichzeitig ein Zugriff auf g von Server d entfernt wird.

Satz 7.2.7. *Gegeben sei eine Instanz von VSLRB sowie eine entsprechende Lösung S. Sei $G_1 = \langle V_1, E_1, \alpha_1 \rangle$ ein laut Definition 7.2.4 erzeugter Improvement-Graph sowie $G_2 = \langle V_2, E_2, \alpha_2 \rangle$ ein laut Definition 7.2.6 erzeugter Improvement-Graph. G_1 und G_2 beschreiben die selbe Menge von Nachbarlösungen.*

Beweis. Sei v_{a_1} - v_{a_2} - ... - v_{a_r} ein teilmengendisjunkter Kreis in G_1. Jedem dieser Knoten aus V_1 kann in eindeutiger Weise ein Knoten $v_{F[a_i]C[a_i]} \in V_2$ zugeordnet werden, wobei $F[a_i]$ die Datei, auf die Zugriff a_i erfolgt und $C[a_i]$ den Server, dem a_i zugewiesen ist, bezeichnet. Dieser Knoten muss existieren, da a_i Server $C[a_i]$ zugewiesen ist und daher $Q(F[a_i], C[a_i]) > 0$ gilt. Wenn die Kante (a_i, a_{i+1}) in G_1 existiert, gilt $t_{F[a_i]} \in T_{C[a_{i+1}]}$, daher muss in G_2 die Kante $(v_{F[a_i]C[a_i]}, v_{F[a_{i+1}]C[a_{i+1}]})$ existieren. Da diese beiden Kanten den selben Teil-Austausch repräsentieren, müssen auch die zugeordneten Kantengewichte gleich sein. Im umgekehrten Fall ist ein teilmengendisjunkter Kreis $v_{f_1c_1}$ - $v_{f_2c_2}$ - ... - $v_{f_rc_r}$ gegeben. Für jeden Knoten v_{f_i,c_i} existieren $Q(f_i, c_i)$ viele Knoten in V_1. Sei $v_{A[f_i,c_i]}$ ein beliebiger solcher Knoten, wobei $A[f_i, c_i]$ einen beliebigen c_i zugeordneten Zugriff auf f_i bezeichnet. Da die Kante $(v_{f_ic_i}, v_{f_{i+1}c_{i+1}})$ in G_2 existiert, gilt $t_{f_i} \in T_{c_{i+1}}$, daher muss auch für beliebig gewählte $v_{A[f_i,c_i]}$, $v_{A[f_{i+1},c_{i+1}]} \in V_1$ eine Kante in E_1 existieren. All diese

Kanten besitzen dasselbe Kantengewicht. Für einen teilmengendisjunkten Kreis in G_2 existieren daher mehrere entsprechende Kreise in G_1, die denselben zyklischen Austausch repräsentieren. Die Vereinigungsmenge aller teilmengendisjunkten Kreise in G_1 entspricht also der Vereinigungsmenge aller teilmengendisjunkten Kreise in G_2. □

Dieses Vorgehen führt im Normalfall zu einem Improvement-Graph mit einer reduzierten Knotenanzahl:

Lemma 7.2.1. *Für den Improvement-Graph laut Definition 7.2.6 gilt:*

1. *Im Worst Case enthält der Improvement-Graph die selbe Anzahl Knoten wie der über der Menge aller Zugriffe definierte Improvement-Graph.*
2. *Sobald eine einzige Datei eine Anzahl an Zugriffen besitzt, die größer als die Anzahl der Server ist, gilt $|V_2| < |V_1|$.*

Beweis. 1. Dies ist der Fall, falls für alle Dateien $i \in F$ gilt: $q_i = 1$. 2. Falls eine Datei i existiert, für die $q_i > m$ gilt, dann muss ein Server j existieren, für den $Q(i, j) > 1$ gilt. Daher ist die Anzahl der Zuweisungen kleiner als die Anzahl der Zugriffe und damit $|V_2| < |V_1|$. □

Im Regelfall ist die Anzahl der Zugriffe für die meisten Dateien weit größer als die Anzahl der Server, woraus eine Reduzierung der Knotenanzahl im Improvement-Graph resultiert. Im Worst Case enthält ein solcher Improvement-Graph $O(mn)$ Knoten, da die Größe der Menge der Zuweisungen, auf der er definiert ist, von der Ordnung $O(mn)$ ist. Um den Improvement-Graph zu erzeugen, müssen daher $O(n^2m^2)$ Kantengewichte berechnet werden. Die dafür höchstens notwendigen $2n^2m^2$ `assign()`- und $2n^2m^2$ `unassign()`-Operationen können auf einfache Weise auf jeweils höchstens $nm + n^2m^2$ Operationen reduziert werden, indem die folgenden beiden Tatsachen ausgenutzt werden:

1. Die Kosten einer Kante (i, j) hängen nur von ihrem Zielknoten j ab.
2. Für die Berechnung der Gewichte aller eingehenden Kanten (i, j) eines Knoten j muss die Entfernung von j aus $S[j]$ nur einmal durchgeführt werden.

Die Ausnutzung dieser Beobachtung führt zum Vorgehen in Algorithmus 7.9.

Aktualisierung des Improvement-Graph

Wird eine Änderung an einer Lösung vorgenommen, muss entweder der Improvement-Graph neu erzeugt oder eine Aktualisierung der Kantengewichte des Improvement-Graph vorgenommen werden. Da die Knotenmenge des Improvement-Graph für VSLRB auf einer Menge mit einer variablen Anzahl von Elementen beruht, kann sich die Struktur des Improvement-Graph im Gegensatz zum generischen Improvement-Graph aus Abschnitt 7.2.4.1 ebenfalls ändern. Dies geschieht in den folgenden beiden Fällen:

1. Wenn ein Server einen Zugriff erhält, deren korrespondierende Datei noch nicht auf diesem Server abgelegt ist. In diesem Fall kommt ein Knoten im Improvement-Graph hinzu.
2. Wenn der letzte Zugriff auf eine Datei von einem Server entfernt wird. In diesem Fall wird ein Knoten aus dem Improvement-Graph entfernt.

Algorithmus 7.9 : Erzeugung des Improvement-Graph für VSLRB

Eingabe : Lösung einer VSLRB-Instanz S

```
1  foreach d ∈ C do
2      foreach g ∈ F_d | t_g ≠ Thumbnail do
3          V ← V ∪ v_gd
4          objOld ← Zielfunktionswert
5          unassign(d, g, 1)
6          foreach s ∈ C | d ≠ s do
7              foreach f ∈ F_s | t_f ≠ Thumbnail ∧ t_f ∈ T_d do
8                  V ← V ∪ v_fs
9                  assign(d, f, 1)
10                 objNew ← Zielfunktionswert
11                 E ← E ∪ (v_fs, v_gd)
12                 α_{v_fs v_gd} ← objNew − objOld
13                 unassign(d, f, 1)
14         assign(d, g, 1)
```

Dies muss bei der Aktualisierung des Improvement-Graph berücksichtigt werden. Die Aktualisierung des Improvement-Graph kann transparent nach Ausführung der Operationen `assign()` bzw. `unassign()` (Siehe Abschnitt 7.1) erfolgen:

Algorithmus 7.10 : UpdateAfterAssign

Eingabe : Datei $i \in F$, Server $j \in C$, Anzahl hinzugefügter Zugriffe $a \in \mathbb{Z}^+$

```
1  if Q(i, j) = a then
2      V ← V ∪ v_ij
3      foreach v_fc ∈ V \ v_ij | t_i ∈ T_c do
4          E ← E ∪ (v_ij, v_fc)
5          UpdateArcCosts(v_ij, v_fc)
6      foreach v_fc ∈ V \ v_ij | t_c ∈ T_j do
7          E ← E ∪ (v_fc, v_ij)
8  foreach i' ∈ F_j do
9      foreach (v_fc, v_i'j) ∈ E do
10         UpdateArcCosts(v_fc, v_i'j)
```

Das Vorgehen nach dem Hinzufügen von Zugriffen zu einer Zuweisung ist in Algorithmus 7.10 dargestellt. Falls die Anzahl hinzugefügter Zugriffe a auf Datei i genau $Q(i,j)$ beträgt, liegt eine neue Zuweisung vor. In diesem Fall muss ein neuer Knoten für diese Zuweisung erzeugt und für jeden anderen Knoten eine ein- bzw. ausgehende Kante aufgenommen werden, falls der durch die Kante repräsentierte Teil-Austausch zulässig ist. Weiters ist es in jedem Fall notwendig, die Kosten aller eingehenden Kanten aller Knoten zu aktualisieren, die Zuweisungen zu Server j repräsentieren. Die Worst-Case Laufzeit dieses Algorithmus beträgt $O(n^2m)$, da höchstens $2mn$ Kanten neu erzeugt, sowie die Kantengewichte von maximal mn Kanten zu maximal n Knoten aktualisiert werden müssen.

Algorithmus 7.11 : UpdateAfterUnassign

Eingabe : Datei $i \in F$, Server $j \in C$

if $Q(i,j) = 0$ **then**
  foreach $(v_{fc}, v_{ij}) \in E$ **do**
    $E \leftarrow E \setminus (v_{fc}, v_{ij})$
  foreach $(v_{ij}, v_{fc}) \in E$ **do**
    $E \leftarrow E \setminus (v_{ij}, v_{fc})$
  $V \leftarrow V \setminus v_{ij}$
foreach $i' \in F_j$ **do**
  foreach $(v_{fc}, v_{i'j}) \in E$ **do**
    $UpdateArcCosts(v_{fc}, v_{i'j})$

Das Vorgehen nach der Entfernung einer Zuweisung ist in Algorithmus 7.11 dargestellt. Falls nach einer Ausführung von `unassign()` $Q(i,j)$ gleich 0 ist, wurde die Zuweisung entfernt, und der entsprechende Knoten und all seine aus- und eingehenden Kanten müssen aus dem Improvement-Graph entfernt werden. Weiters müssen wiederum die Kosten der eingehenden Kanten aller Knoten, die Zuweisungen zu j repräsentieren, aktualisiert werden. Die Worst-Case Laufzeit dieses Algorithmus beträgt ebenfalls $O(n^2m)$.

Um unnötige Aktualisierungen des Improvement-Graph zu vermeiden, wird dieser in der VND für VSLRB erst bei der ersten Durchsuchung der Cyclic-Exchange Neighbourhood erzeugt und danach bei Änderungen an der Lösung aktualisiert.

Durchsuchung der Nachbarschaft

Gemäß Definition 7.2.3 kann auch die in der VND für VSLRB eingesetzte Variante der Cyclic Exchange Neighbourhood angeschrieben werden als

$$\mathcal{N}_{\text{Cyclic}}(S) = \{S' \mid S' \text{ kann aus } S \text{ durch einen zulässigen zyklischen Austausch erzeugt werden}\}$$

Die Suche nach teilmengendisjunkten Zyklen im Improvement-Graph und damit die Durchsuchung der Nachbarschaft $\mathcal{N}_{Cyclic}$ geschieht mittels des in Abschnitt 7.2.4.1 beschriebenen Modified Label-Correcting Algorithm. Die in [5] für diese Aufgabe eingesetzte Deque-Implementierung wurde hier bewusst nicht verwendet, da sich ihr praktischer Laufzeitvorteil nur bei spärlichen Graphen auswirkt [8]. Bei sehr dichten Graphen, wie dem Improvement-Graph für VSLRB, erweist sich dagegen das polynomielle Worst-Case Laufzeitverhalten der FIFO-Implementierung als Vorteil.

Als Schrittfunktion wird eine eingeschränkte *Best Improvement*-Strategie verwendet, indem der Algorithmus mit einem beliebigen Startknoten s gestartet und der Zyklus mit den höchsten negativen Kosten ermittelt wird, der von s aus auffindbar ist.

7.3 Nachbarschaften der VNS für VSLRB

Der folgende kurze Abschnitt beschreibt die Nachbarschaftsstruktur, die im Rahmen der Variable Neighbourhood Search für VSLRB für die Shaking-Prozedur zum Einsatz kommt.

$$\mathcal{N}_{k-\text{Shaking}}(S) = \{S' \mid S' \text{ kann aus } S \text{ durch } k \text{ aufeinanderfolgende } \texttt{swap()}\text{-Operationen erzeugt werden}\}$$

Eine zufällige Lösung aus dieser Nachbarschaft wird gewählt, indem k mal ein zufälliger Nachbar aus einer (jeweils neu erzeugten) Access-Swap Neighbourhood (siehe Abschnitt 7.2.2). ausgewählt wird. Da jede Nachbarlösung der jeweiligen Access-Swap Neighbourhood mit der selben Wahrscheinlichkeit gewählt wird, diese Nachbarschaft aber zu viele Nachbarlösungen enthält, um sie auf einmal im Speicher zu halten, wird eine Variante der *Reservoir Sampling Method* [31, 47] verwendet, um eine zufällige Nachbarlösung auszuwählen, ohne ihre Gesamtanzahl zu kennen oder alle Nachbarlösungen auf einmal im Speicher halten zu müssen (siehe Algorithmus 7.12).

Algorithmus 7.12 : One-Element Reservoir Sampling

Eingabe : Lösung einer VSLRB-Instanz S

Initialisiere $\mathcal{N}_{Swap}$ mit S
selected $\leftarrow$ *leer*
$t \leftarrow 1$
while *$\mathcal{N}_{Swap}$ enthält eine weitere Nachbarlösung* **do**
  current $\leftarrow$ *erzeuge nächste Nachbarlösung in $\mathcal{N}_{Swap}$*
  if *selected ist leer* **then**
    selected $\leftarrow$ *current*
  else
    Setze selected auf current mit $P = \frac{1}{t+1}$
  $t \leftarrow t + 1$

7.4 Sortierung der Nachbarschaften

Die Nachbarschaften der VND für VSLRB sind nach ihrer Komplexität geordnet. Obwohl von einem theoretischen Standpunkt aus gesehen die Komplexität der Cyclic Exchange Neighbourhood geringer ist als jene der k-Server-MIP-Neighbourhood (polynomiell mit sehr hohem Grad gegenüber einer exponentiellen Größe), besitzt die Cyclic Exchange Neighbourhood einen beträchtlichen Overhead, der sich in der Erzeugung des Improvement-Graph und der danach notwendigen Aktualisierung desselben niederschlägt. Aus diesem Grund ist die 2-Server-MIP-Neighbourhood trotz ihrer exponentiellen Größe in der VND für VSLRB vor der Cyclic Exchange Neighbourhood gereiht:

$$\mathcal{N}_1 = \mathcal{N}_{\text{Move}} \qquad \mathcal{N}_2 = \mathcal{N}_{\mathcal{S}\sqsupseteq\dashv_{\surd}} \qquad \mathcal{N}_3 = \mathcal{N}_{2-\text{MIP}} \qquad \mathcal{N}_4 = \mathcal{N}_{\text{Cyclic}}$$

Kapitel 8

Implementierung

Die Umsetzung der Variable Neighbourhood Search für VSLRB erfolgte aus Gründen der Plattformunabhängigkeit in Java 6. Die Implementierung besteht aus insgesamt drei spezialisierten Paketen:

- **MathProg**: Dieses Paket dient zur abstrakten Modellierung linearer und quadratischer Programme. Weiters stellt es Möglichkeiten zur Serialisierung dieser Modelle in das Eingabeformat einiger Solver-Pakete, Wrapper-Klassen für verschiedene Solver sowie Parser für die Ausgaben verschiedener Solver zur Verfügung.
- **VNS**: Dieses Paket stellt ein Framework für Variable Neighbourhood Search und Variable Neighbourhood Descent zur Verfügung.
- **VSLB**: Dieses Paket enthält die eigentliche Implementierung der in Kapitel 7 beschriebenen Anwendung von VNS auf VSLRB. Es enthält Repräsentationen von Instanzen und Lösungen von VSLRB sowie Implementierungen der im Rahmen der VNS verwendeten Nachbarschaften.

8.1 MathProg

Da abzusehen war, dass einerseits die kommerzielle Solver-Software CPLEX, die während der Entstehung dieser Arbeit verwendet wurde, aus Kostengründen im praktischen Einsatz höchstwahrscheinlich nicht zur Verfügung stehen würde und anderseits die Berechnung der Last-Zielwerte mit Hilfe eines quadratischen Programms (siehe Abschnitt 3.2.3) auf verschiedenen Rechnern mit jeweils unterschiedlichen Solver-Paketen stattfinden musste, wurde schon früh im Entwicklungsprozess die Entscheidung getroffen, Möglichkeiten zur Spezifikation und Lösung von mathematischen Programmen vorzusehen, die unabhängig von einem konkreten Solver-Paket verwendet werden können.

Zu diesem Zweck stellt das Paket `MathProg` eine Klassenbibliothek zur Modellierung linearer und quadratischer Programme zur Verfügung. Die wichtigsten Klassen dieser Bibliothek sind in Abbildung 8.1 dargestellt. Die Erstellung eines Modells beginnt mit der Erzeugung einer Instanz der Klasse `Program`. Mittels der Methoden `obtainBinary()`, `obtainInteger()` und `obtainFractional()` können Variablen, repräsentiert durch die Klasse `Variable`, des jeweiligen Typs erstellt und dem Modell hinzugefügt werden. Der Wertebereich einer Variablen kann mit Hilfe der Methode `setBounds()` festgelegt werden.

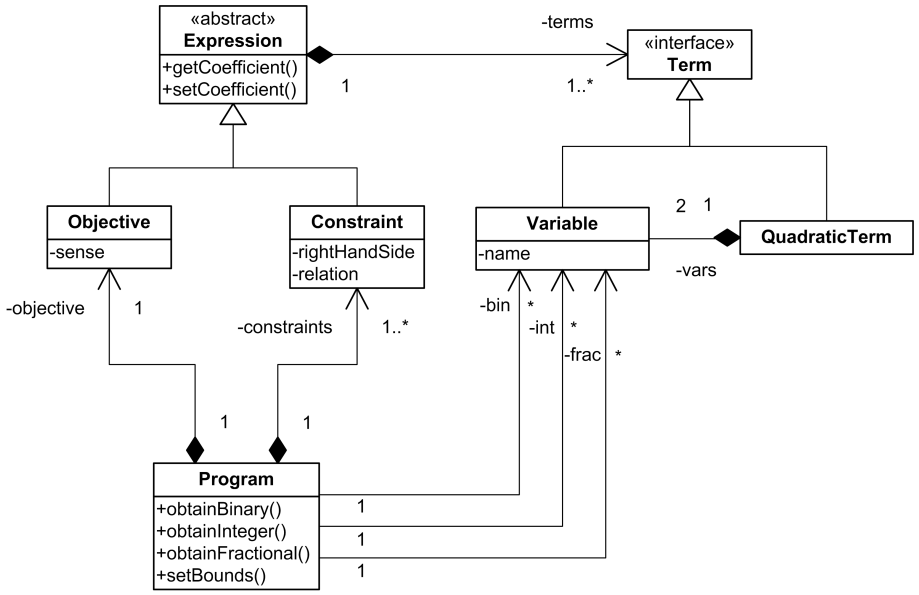


Abbildung 8.1: Klassenbibliothek zur Modellierung linearer und quadratischer Programme

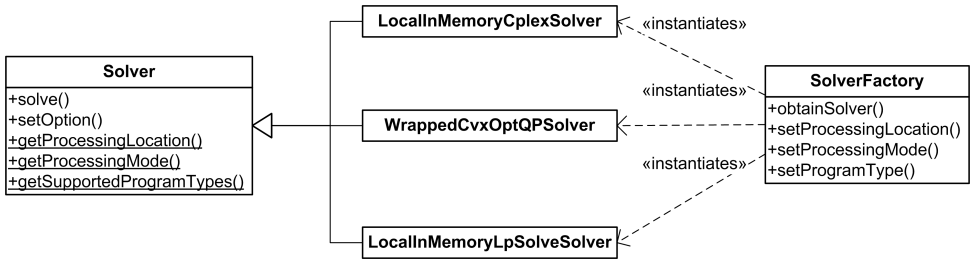


Abbildung 8.2: Solver-Modell

Soll das Modell in der Zielfunktion oder in einer Nebenbedingung quadratische Ausdrücke enthalten, müssen dafür Instanzen von `QuadraticTerm` erstellt werden, die wiederum aus je zwei Instanzen von `Variable` bestehen.

Sowohl `Variable` als auch `QuadraticTerm` implementieren das Interface `Term` und können dementsprechend sowohl in der Zielfunktion (Klasse `Objective`) als auch in Nebenbedingungen (Klasse `Constraint`) verwendet werden.

Solver für in dieser Weise erzeugte Modelle werden durch Klassen repräsentiert, die das Interface `Solver` implementieren. Die Methode `solve()` erwartet als Parameter eine Instanz der Klasse `Program`. Dieses Modell wird in das von der jeweiligen Solver-Software verwendete Format übersetzt. Danach wird die Solver-Komponente des jeweiligen Pakets zur Lösung des Modells aufgerufen und das Ergebnis in einem einheitlichen Format zurückgeliefert. Die Solver-Klassen bieten über statische Methoden Meta-Informationen über die Eigenschaften des jeweiligen Solvers an. Die wichtigste dieser Eigenschaften stellt dabei die Menge der unterstützten Modell-Typen (z.B. ILP, MIQP) dar. Die Klasse `SolverFactory` dient schließlich dazu, basierend auf den Eigenschaften der verfügbaren Solver und den Eigenschaften des jeweiligen Modells eine Instanz einer passenden Solver-Klasse zu erzeugen. Abbildung 8.2 zeigt einen Überblick über die verwendeten Klassen.

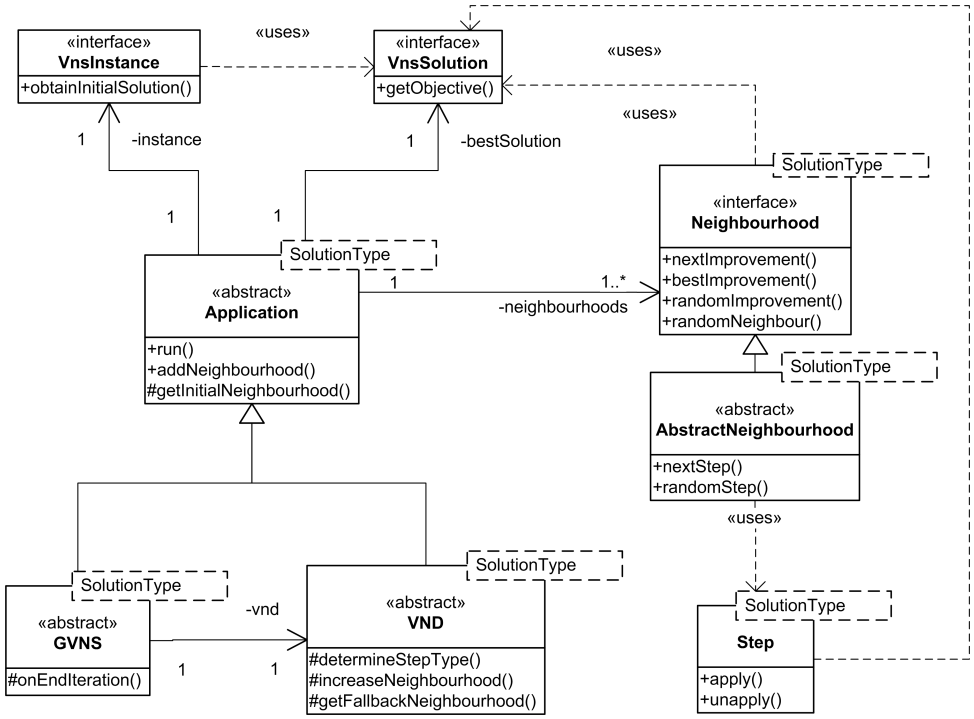


Abbildung 8.3: VNS-Framework

Weiters stellt das Paket Möglichkeiten zur Konvertierung eines mathematischen Programms in das (textuelle) Eingabeformat einiger Solver-Pakete sowie Parser zum Einlesen der Lösungsdateien einiger Solver-Pakete zur Verfügung.

8.2 VNS

Das Paket `VNS` definiert ein Framework für die Erstellung von Anwendungen von Variable Neighbourhood Search (siehe Kapitel 6). Klassen, die eine Instanz bzw. eine Lösung des zu behandelnden Optimierungsproblems beschreiben, müssen die Interfaces `VnsInstance` bzw. `VnsSolution` implementieren. Diese geben jeweils nur eine wichtige Methode vor: `obtainInitialSolution()` dient zur Erzeugung einer Startlösung und `getObjective()` liefert den Zielfunktionswert einer Lösung. Konkrete Anwendungen von VNS müssen von den abstrakten Basisklassen `GVNS` (für General Variable Neighbourhood Search) bzw. `VND` (für Variable Neighbourhood Descent) abgeleitet und mit der jeweiligen Lösungsklasse parametrisiert werden.

Nachbarschaftsstrukturen werden innerhalb des Frameworks durch Klassen repräsentiert, die das Interface `Neighbourhood` implementieren. Für die Schrittfunktionen *Next Improvement*, *Best Improvement* und *Random Improvement* definiert dieses Interface die Methoden `nextImprovement()`, `bestImprovement()` und `randomImprovement()`. Für die Realisierung von Nachbarschaften einer General Variable Neighbourhood Search wird weiters die Methode `randomNeighbour()` angeboten. Weiters steht auch die abstrakte Basisklasse `AbstractNeighbourhood` zur Verfügung, die Implementierungen der Methoden von `Neighbourhood` basierend auf der Repräsentation von zulässigen Änderungsschritten durch die Klasse `Step` anbietet. Diese Abstraktion eignet sich vor allem für Nachbarschaf-

ten, die durch vollständige Aufzählung durchsucht werden.

8.3 VSLB

Das Paket `VSLB` enthält die eigentliche Implementierung der Variable Neighbourhood Search für VSLRB. Das Paket enthält Klassen zur Repräsentation von Instanzen und Lösungen von VSLRB (`VslbInstance`, `VslbSolution`), Klassen zur Repräsentation von Domänenobjekten (`VslbFile`, `VslbServer`) und Implementierungen der durch das Paket `VNS` vorgegebenen Interfaces zur Umsetzung der Variable Neighbourhood Search (`VslbVND`, `VslbGVNS`, `*Neighbourhood`). Abbildung 8.4 zeigt eine Übersicht über die wichtigsten verwendeten Klassen.

Die Klasse `VslbInstance` dient zur Repräsentation einer Probleminstanz von VSLRB. Diese wird beim Programmstart aus den drei zu einer Instanz gehörenden Dateien servers.xml, files.xml und instance.xml erzeugt. Die folgenden Tabellen 8.1 bis 8.3 zeigen die in der jeweiligen Datei enthaltenen Datenelemente.

Datenelement	Symbol	Beschreibung
`id`		Eindeutige Kennung
`capacity`	W_j	Speicherkapazität
`upload`	U_j	Bandbreite Upload
`download`	D_j	Bandbreite Download
`accepted`	T_j	Akzeptierte Dateitypen

Tabelle 8.1: Datenelemente von servers.xml

Datenelement	Symbol	Beschreibung
`id`		Eindeutige Kennung
`size`	w_j	Größe
`bitrate`	b_i	Bitrate
`type`	t_i	Datei-Typ
`popularity`		Rang in der Beliebtheitsreihenfolge

Tabelle 8.2: Datenelemente von files.xml

Die Klasse `VslbInstance` bietet Methoden zur Abfrage aller Eigenschaften der vorhandenen Server und Video-Objekte sowie eine Methode zur Erzeugung des MIP-Modells der Instanz an. Weiters erlaubt die Klasse die Erzeugung einer neuen Probleminstanz basierend auf vorgegebenen Servern und Video-Objekten, wobei unter anderem die fairen Lasten Λ_j und die Last-Zielwerte η_j gemäß Abschnitt 3.2.3 berechnet werden.

Die Klasse `VslbSolution` repräsentiert eine Lösung einer Probleminstanz von VSLRB. Sie verwaltet die Replikate und Zuweisungen jedes Servers, bietet Methoden zur Transformation der Lösung an (`assign()`, `unassign()`, `move()`, `swap()`), kümmert sich um die inkrementelle Berechnung des Zielfunktionswerts und erstellt bzw. verwaltet ggf. den Improvement Graph. Weiters bietet die Klasse Methoden zur initialen Erzeugung einer

Datenelement	Symbol	Beschreibung
`serversLocation`		Verweis auf die Position der servers.xml
`filesLocation`		Verweis auf die Position der files.xml
`alpha`	α	Wert des Faktors α
`beta`	β	Wert des Faktors β
`accesses`	q_i	Anzahl Zugriffe auf i laut Access Profile, $i \in F$
`targetLoad`	η_j	Last-Zielwert für Server j, $j \in C$
`currentSolutionLocation`		Verweis auf die bestehende Lösung

Tabelle 8.3: Datenelemente von instance.xml

Lösung (`createGreedy()`, `createRandom()`)

Die Klasse `VslbInstance` enthält weiters einen Verweis auf eine `VslbSolution`, welche die gegenwärtige Zuordnung beschreibt.

Die Implementierung der Variable Neighbourhood Search für VSLRB beruht auf dem im Paket `VNS` definierten Framework. Die von `GVNS` abgeleitete Klasse `VslbGVNS` implementiert eine General Variable Neighbourhood Search, während die von `VND` abgeleitete Klasse `VslbVND` die eingebettete Variable Neighbourhood Descent implementiert. Die von der VND bzw. der GVNS verwendeteten Nachbarschaftsstrukturen werden durch die Klassen `AccessMoveNeighbourhood`, `AccessSwapNeighbourhood`, `DeepMipNeighbourhood` und `CyclicExchangeNeighbourhood` bzw. `ShakingNeighbourhood` realisiert.

Das Verhalten einiger Komponenten kann durch Konfigurationsoptionen in den drei Konfigurationsdateien VNS.properties, VSLB.properties und SolverFactory.properties gesteuert werden. Die verfügbaren Konfigurationsoptionen sind in Tabelle 8.4 dargestellt.

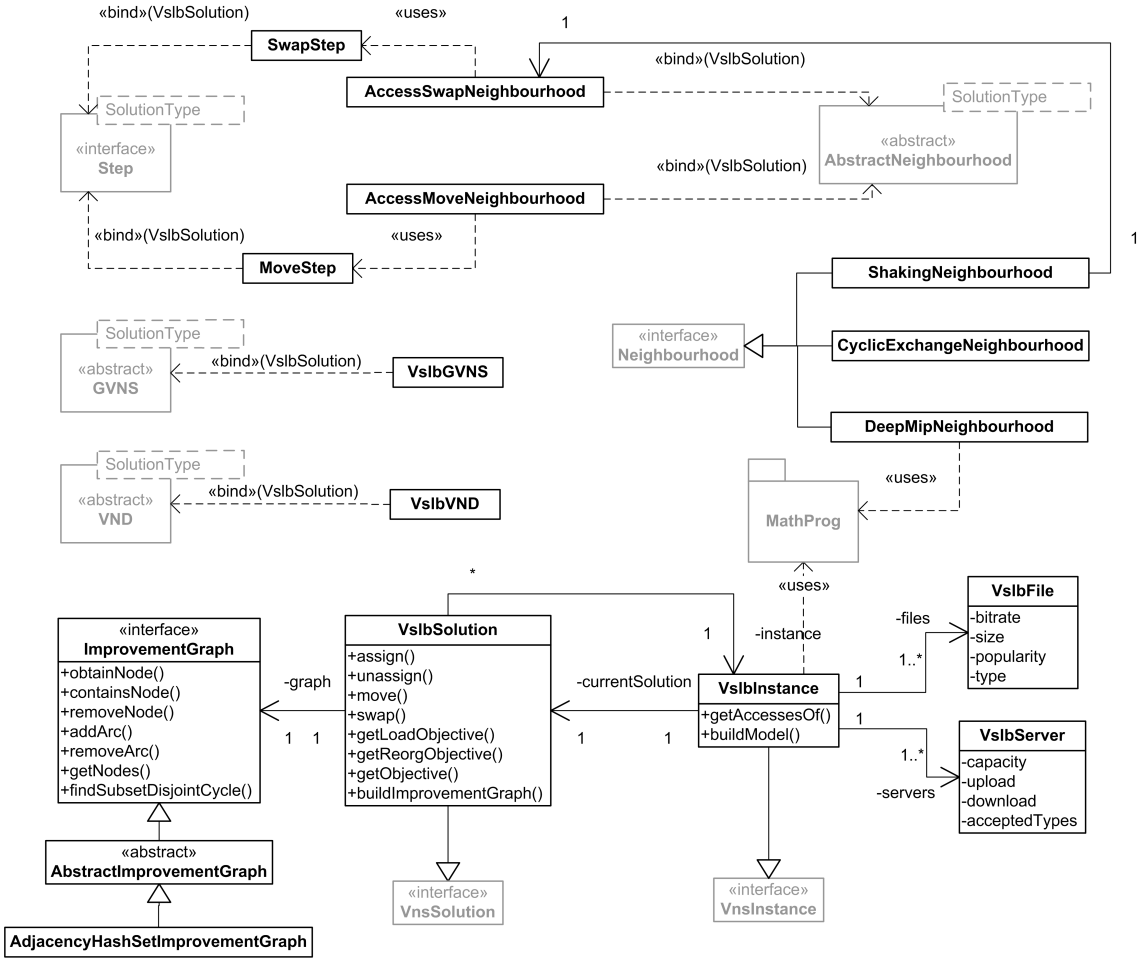


Abbildung 8.4: Wichtigste Klassen des Pakets `VSLB`. Alle Elemente, die sich auf die Pakete `MathProg` und `VNS` beziehen, sind ausgegraut dargestellt

Konfigurationsoption	Typ	Beschreibung
`SolverFactory.AvailableSolvers`	Liste	Klassen der auf der aktuellen Plattform verfügbaren Solver
`VND.MinRelativeImprovement`	float	Geringste noch akzeptierte relative Verbesserung des Zielfunktionswerts in der VND
`GVNS.MinRelativeImprovement`	float	Geringste noch akzeptierte relative Verbesserung des Zielfunktionswerts in der GVNS
`VslbGVNS.NumberOfNeighbourhoods`	int	Anzahl zu verwendender Instanzen von `ShakingNeighbourhood`
`VslbVND.Neighbourhoods`	Liste	Klassen der in der VND zu verwendenden Nachbarschaftsstrukturen
`DeepMipNeighbourhood.ConsideredServers`	int	Anzahl Server k in der k-Server MIP Neighbourhood
`DeepMipNeighbourhood.LoadOptimalityTolerance`	float	Unterschreitet $\lvert\mathcal{L}(j)-\eta_j\rvert$ diesen Wert wird Server j in der k-Server MIP Neighbourhood nicht mehr berücksichtigt
`DeepMipNeighbourhood.MipTimelimit`	float	Steuerungsparameter für den MIP-Solver: Maximale erlaubte Laufzeit
`DeepMipNeighbourhood.MipRelativeGapTolerance`	float	Steuerungsparameter für den MIP-Solver: Geringste akzeptierte relative Abweichung zwischen dem Zielfunktionswert der aktuellen LP-Relaxation und der besten oberen Schranke
`DeepMipNeighbourhood.MipAbsoluteGapTolerance`	float	Steuerungsparameter für den MIP-Solver: Geringste akzeptierte absolute Abweichung zwischen dem Zielfunktionswert der aktuellen LP-Relaxation und der besten oberen Schranke
`CyclicExchangeNeighbourhood.UseRandomStartNode`	bool	Steuert, ob ein zufälliger oder der erste Knoten des Improvement Graph als Startknoten für die Suche nach teilmengendisjunkten Kreisen herangezogen werden soll
`CyclicExchangeNeighbourhood.UseIncrementalUpdates`	bool	Steuert, ob der Improvement Graph transparent aktualisiert wird, oder ob der Improvement Graph vor Durchsuchung der CyclicExchangeNeighbourhood neu erstellt werden soll
`AbstractImprovementGraph.UseDequeLCA`	bool	Steuert, ob die Deque- oder die FIFO-Implementierung des Label Correcting Algorithm verwendet werden soll

Tabelle 8.4: Konfigurationsoptionen

Kapitel 9

Testergebnisse

Das folgende Kapitel beschreibt die Testresultate des MIP- und des VNS-Ansatzes zur Lösung von VSLRB. Alle Tests wurden auf einem Rechner mit zwei AMD Dual-Core Opteron Prozessoren, getaktet mit je 2 GHz, und 8 GB Hauptspeicher durchgeführt.

9.1 Testinstanzen

Die wichtigsten Eigenschaften der Instanzen von VSLRB, die zum Test des MIP- und des VNS-Ansatzes herangezogen wurden, sind in Tabelle 9.1 dargestellt.

Instanz	$\|C\|$	$\|F\|$	$\sum_{i \in F} q_i$	Z	T_j
1	4	60	489	30710.20	Zwei Server *HiRes*, zwei Server *Thumbnail* und *Preview*
2	4	300	637	13152.30	
3	5	1200	1328	32844.49	Zwei Server *HiRes*, zwei Server *Preview* und *HiRes*, ein Server alle
4	7	3000	3064	14492.57	
5	12	4500	4547	24711.2	
6	3	15000	15238	192513.20	Ein Server *HiRes*, ein Server *Preview* und *HiRes*, ein Server alle
7	20	9000	9027	58700.34	Ein Server *HiRes*, ein Server *Preview* und *HiRes*, Rest alle
8	20	3000	3064	31709.60	
9	25	3000	3406	36424.82	
10	25	12000	12680	68269.14	
11	2	15000	19352	6833.00	

Tabelle 9.1: Verwendetete Testinstanzen

In dieser Übersicht bezeichnet Z den Zielfunktionswert der bestehenden Zuordnung, die jeweils zufällig erzeugt wurde.

9.2 Testresultate des MIP-Ansatzes

Die Testresultate dieses Abschnitts wurden unter Verwendung des kommerziellen ILP-Solvers ILOG CPLEX 11.1 ermittelt. Tabelle 9.2 zeigt die Testergebnisse des MIP-Ansatzes, wobei bei jeder Instanz die durchschnittliche Laufzeit der VNS als Zeitlimit verwendet wurde. Der MIP-Ansatz konnte innerhalb dieses Zeitlimits für die Instanzen 1, 2, 3 und 6, die eine geringe Anzahl von Servern aufweisen, eine bessere Lösung als die

Instanz	Z	t [s]	*RelGap* [%]	*AbsGap*
1	1.16	3.01	44.19	0.51
2	2.07	15.01	77.53	1.60
3	1.67	100.04	90.10	1.51
4	3.07	141.04	99.19	3.05
5	29.74	434.11	99.77	29.67
6	56.10	219.04	2.20	1.23
7	639.63	92.11	97.09	621.01
8	133.44	407.06	92.65	123.62
9	214.09	731.25	97.97	209.75
10	592.96	175.34	98.83	586.04

Tabelle 9.2: Ergebnisse des MIP-Ansatzes bei Verwendung eines durch die Ergebnisse des VNS-Ansatzes vorgegebenen Zeitlimits

Instanz	Z_{MIP}	$\bar{Z}_{\text{VNS}}$	$\sigma_{Z_{\text{VNS}}}$	t [s]
1	1.16	1.60	0.86	2.38
2	2.07	3.43	1.06	14.67
3	1.67	103.59	388.82	99.87
4	3.07	0.45	0.34	140.68
5	29.74	0.94	0.67	434.32
6	56.10	67.47	1.53	219.24
7	639.63	73.29	5.41	92.10
8	133.44	41.93	3.99	406.47
9	214.09	58.15	5.11	731.12
10	592.96	83.85	5.36	174.49

Tabelle 9.3: Vergleich der durch den MIP-Ansatz und der durchschnittlichen durch den VNS-Ansatz, unter Verwendung aller Nachbarschaftsstrukturen, erreichten Zielfunktionswerte

VNS ermitteln (siehe Tabelle 9.3).

Die in Tabelle 9.2 neben dem erzielten Zielfunktionswert Z und der Laufzeit t angegebenen Werte *RelGap* und *AbsGap* setzen die beste ermittelte untere Schranke L^* und die durch die erzielte ganzzahlige Lösung vorgegebene obere Schranke $U = Z$ in Beziehung: $\text{RelGap}(L^*, U)$ bezeichnet die relative

$$RelGap(L^*, U) = \frac{|L^* - U|}{|U|} \tag{9.1}$$

bzw. $\text{AbsGap}(L^*, U)$ die absolute Abweichung dieser beiden Schranken:

$$AbsGap(L^*, U) = |L^* - U| \tag{9.2}$$

Für eine optimale Lösung gilt sowohl $\text{RelGap}(L^*, U) = 0$ als auch $\text{AbsGap}(L^*, U) = 0$.

Weiters können durch den MIP-Ansatz für Instanzen mit einer kleinen Anzahl von Servern innerhalb kurzer Zeit Lösungen gefunden werden, deren Qualität als für die Praxis zufriedenstellend angesehen werden kann. Tabelle 9.4 zeigt die Ergebnisse eines Tests, bei dem für jede Instanz ein Zeitlimit von 30 Sekunden, eine *Relative Gap Tolerance* von 0.01 sowie eine *Absolute Gap Tolerance* von 1 verwendet wurde. Für die Instanzen 1, 2, 3, 6 und 11, die eine geringe Anzahl von Servern aufweisen, werden innerhalb dieses Zeitlimits

Instanz	Z	t [s]	*RelGap* [%]	*AbsGap*
1	1.61	0.11	61.42	0.99
2	1.46	22.26	68.01	0.99
3	1.67	30.00	90.12	1.51
4	21.40	30.00	99.88	21.37
5	104.94	30.00	99.94	104.87
6	57.75	30.00	4.99	2.88
7	1426.97	30.00	98.70	1408.35
8	286.05	30.00	96.57	276.24
9	844.98	30.00	99.49	840.68
10	905.62	30.00	99.24	898.70
11	16.60	30.00	100.00	16.60

Tabelle 9.4: Ergebnisse des MIP-Ansatzes bei Verwendung eines einheitlichen Zeitlimits von 30 Sekunden sowie einer einheitlichen Gap Tolerance von $RelGap = 0.01$ und $AbsGap = 1$, analog zum Einsatz in der k-Server MIP Neighbourhood

Instanz	Z	t [s]	*RelGap* [%]	*AbsGap*
1	0.89	3600.00	15.10	0.13
2	0.86	3600.00	41.68	0.36
3	1.67	3600.00	90.02	1.50
4	1.08	3600.00	97.68	1.05
5	16.95	3600.00	99.61	16.89
6	55.69	3600.00	1.47	0.82
7	117.99	3600.00	84.22	99.36
8	91.58	3600.00	89.19	81.68
9	136.95	3600.00	96.83	132.61
10	137.47	3600.00	94.94	130.51
11	0.20	3600.00	100.00	0.20

Tabelle 9.5: Ergebnisse des MIP-Ansatzes bei Verwendung eines einheitlichen Zeitlimits von 3600 Sekunden

durchwegs gute Lösungen erzielt. Dieses Verhalten wird in der k-Server MIP Neighbourhood der VNS ausgenützt, indem Unterprobleme mit einer geringen Anzahl von Servern mit Hilfe des MIP-Ansatzes gelöst werden (siehe Abschnitt 7.2.3).

Schließlich sind in Tabelle 9.5 die Ergebnisse des MIP-Ansatzes mit einem Zeitlimit von einer Stunde dargestellt. Auch bei Verwendung dieses hohen Zeitlimits liefert der VNS-Ansatz für die Instanzen 4, 5, 7, 8, 9 und 10 bessere Ergebnisse als der reine MIP-Ansatz.

9.3 Testresultate des VNS-Ansatzes

Die Resultate der Testläufe der VNS für VSLRB (siehe Kapitel 6) sind in den Tabellen 9.7, 9.8 und 9.9 dargestellt. Jede dieser Tabelle zeigt für die Testinstanzen 1-10 die folgenden Durchschnittswerte, die aus jeweils 30 Testläufen resultieren:

Symbol	Beschreibung
$\bar{Z}_1$	Durchschnittlicher erzielter Wert für den ersten Teil der Zielfunktion
σ_{Z_1}	Standardabweichung des erzielten Werts für den ersten Teil der Zielfunktion
$\bar{Z}_2$	Durchschnittlicher erzielter Wert für den zweiten Teil der Zielfunktion
σ_{Z_2}	Standardabweichung des erzielten Werts für den zweiten Teil der Zielfunktion
$\bar{Z}$	Durchschnittlicher Zielfunktionswert
σ_Z	Standardabweichung des Zielfunktionswerts
$\bar{t}$	Durchschnittliche Laufzeit
$\bar{\sigma_t}$	Standardabweichung der Laufzeit
$\mathcal{N}_x$ / $\bar{f}$	Durchschnittliche Anzahl in Nachbarschaftsstruktur $\mathcal{N}_x$ erzielter Verbesserungen
$\mathcal{N}_x$ / $\bar{\Delta}$	Durchschnittliche gesamte durch Verwendung von Nachbarschaftsstruktur $\mathcal{N}_x$ erzielte Verbesserung des Zielfunktionswerts
$\mathcal{N}_x$ / $\bar{t}$	Durchschnittliche von Nachbarschaftsstruktur $\mathcal{N}_x$ benötigte Laufzeit

Tabelle 9.6: Beschreibung der in den Tabellen 9.7 bis 9.9 verwendeten Symbole

Um die Eigenschaften der verschiedenen verwendeten Nachbarschaftsstrukturen zu untersuchen, wurde jeweils eine unterschiedliche Auswahl der zur Verfügung stehenden Nachbarschaftsstrukturen eingesetzt:

1. $\mathcal{N}_{\text{Move}}$, $\mathcal{N}_{\text{Swap}}$, $\mathcal{N}_{\text{2-Mip}}$ und $\mathcal{N}_{\text{Cyclic}}$
2. $\mathcal{N}_{\text{Move}}$, $\mathcal{N}_{\text{Swap}}$ und $\mathcal{N}_{\text{2-Mip}}$
3. $\mathcal{N}_{\text{Move}}$, $\mathcal{N}_{\text{Swap}}$ und $\mathcal{N}_{\text{Cyclic}}$

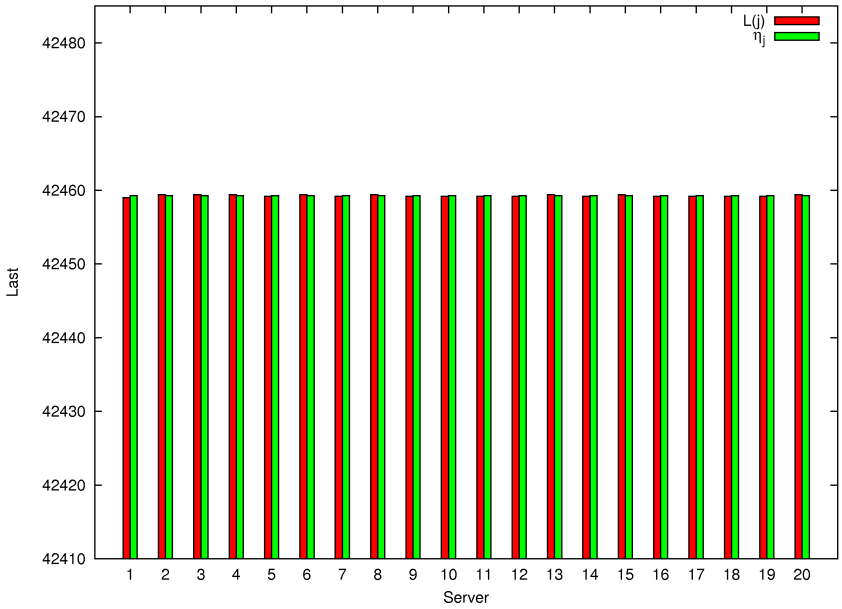
Tabelle 9.7 zeigt die Ergebnisse der Tests bei Verwendung aller zur Verfügung stehenden Nachbarschaftsstrukturen, wo dies möglich war. Die Nachbarschaft $\mathcal{N}_{\text{Cyclic}}$ konnte bei den Instanzen 6, 7 und 10 nicht eingesetzt werden, da in diesen Fällen der Improvement-Graph zu groß war, um im durch die Java Virtual Machine verwalteten Heap-Speicher gehalten zu werden.

Die Abbildungen 9.1 und 9.2 zeigen exemplarisch die Last-Zielwerte und die erreichten Server-Lasten für die besten durch den VNS-Ansatz gefundenen Lösungen der Testinstanzen 7, 8, 9 und 10.

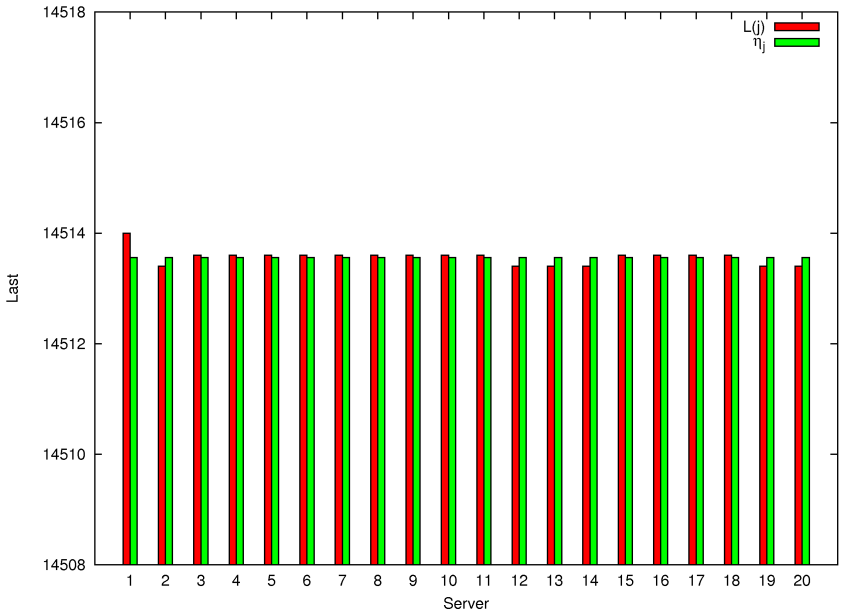
Weiters illustriert Abbildung 9.3 den Zusammenhang zwischen den berechneten Last-Zielwerten und der Struktur der ermittelten Lösungen. Im Beispiel in Abbildung 9.3a, die eine Lösung für Instanz 3 zeigt, wurden, obwohl insgesamt drei Server den Datei-Typ *Preview* akzeptieren, die η_j durch Lösung des quadratischen Programms aus Abschnitt 3.2.3 in einer Weise berechnet, dass Server 5 alleine für alle Zugriffe auf Video-Objekte mit Datei-Typ *Preview* verantwortlich ist, d.h. $\eta_5^{Preview} = L_{Preview}$. Da die η_j^t nicht in die Zielfunktion von VSLRB einfließen, ist durch die auf diese Weise berechneten η_j nur implizit vorgegeben, dass in einer Optimallösung für diese Instanz ausschließlich Server 5 *Preview*-Zugriffe erhalten darf. Trotzdem wurde in der in Abbildung 9.3a dargestellten Lösung für Instanz 3 diese Struktur eingehalten, sodass die $\mathcal{L}_t(j)$ die η_j^t annähern. Im Beispiel in Abbildung 9.3b ist die implizite Notwendigkeit, die η_j^t anzunähern nicht gegeben, da jeder der Server alle Datei-Typen akzeptiert.

Wird auf die Verwendung von $\mathcal{N}_{\text{Cyclic}}$ verzichtet (siehe Tabelle 9.8), werden durchwegs geringfügig schlechtere Zielfunktionswerte bei gleichzeitig stark verringerter durchschnittlicher Laufzeit erzielt, was aufgrund des hohen Laufzeit-Overheads von $\mathcal{N}_{\text{Cyclic}}$ zu erwarten war. Auffällig ist das wesentlich schlechtere Abschneiden bei Testinstanz 3. Dies ist durch die spezielle Struktur dieser Testinstanz zu erklären: In der Optimallösung dieser Instanz müsste einer der Server alle Video-Objekte mit $t_i = Preview$ erhalten, obwohl auch andere Server diesen Datei-Typ akzeptieren. Gleichzeitig ist dies auch der einzige Server mit einer negativen Abweichung von η_j. Aufgrund der Vorgehensweise bei der Auswahl der Server für das Unterproblem in der k-Server MIP Neighbourhood (siehe Abschnitt 7.2.3) kann in einem solchen Fall oftmals keine gültige Serverauswahl ermittelt werden. Abschnitt 10.2.1 beschreibt eine mögliche Verbesserung des Auswahlverfahrens, um dieses Problem zu vermeiden.

Wird auf die Verwendung von $\mathcal{N}_{\text{2-Mip}}$ verzichtet (siehe Tabelle 9.9), erweist sich wiederum die Struktur von Testinstanz 3 als Problem: Da hier häufig nur ein einziger der Server eine negative Abweichung von η_j aufweist, müsste dieser ausschließlich Zugriffe erhalten, ohne selbst Zugriffe abzugeben, was durch die Anwendung eines zyklischen Austauschs nicht realisiert werden kann. Abhilfe würde hier die Erweiterung von $\mathcal{N}_{\text{Cyclic}}$ um *Path Exchanges* [5] schaffen (siehe Abschnitt 10.2.2). Im Fall der Testinstanzen 8 und 9, die eine gleichförmige Struktur bezüglich der akzeptierten Dateitypen aufweisen, konnten hingegen bessere Zielfunktionswerte als bei Verwendung aller Nachbarschaftsstrukturen erzielt werden.

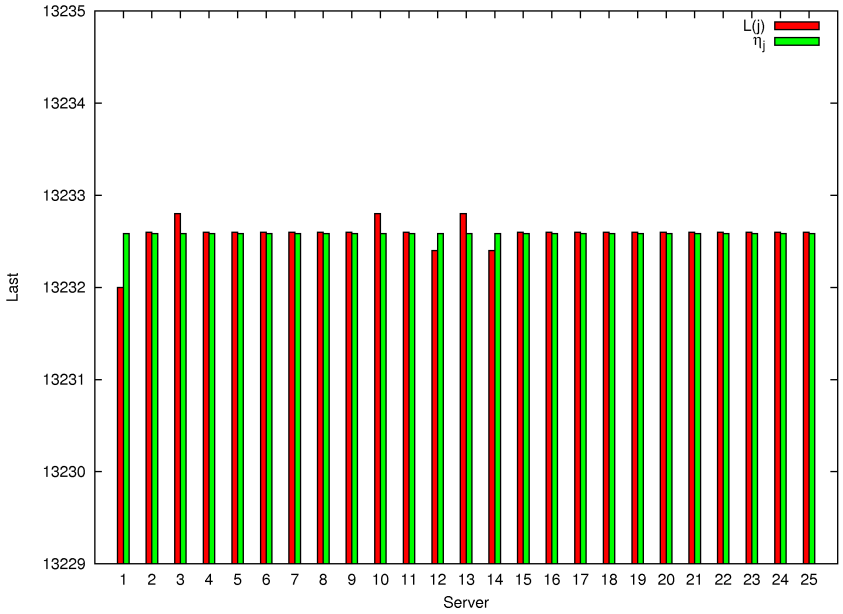


(a)

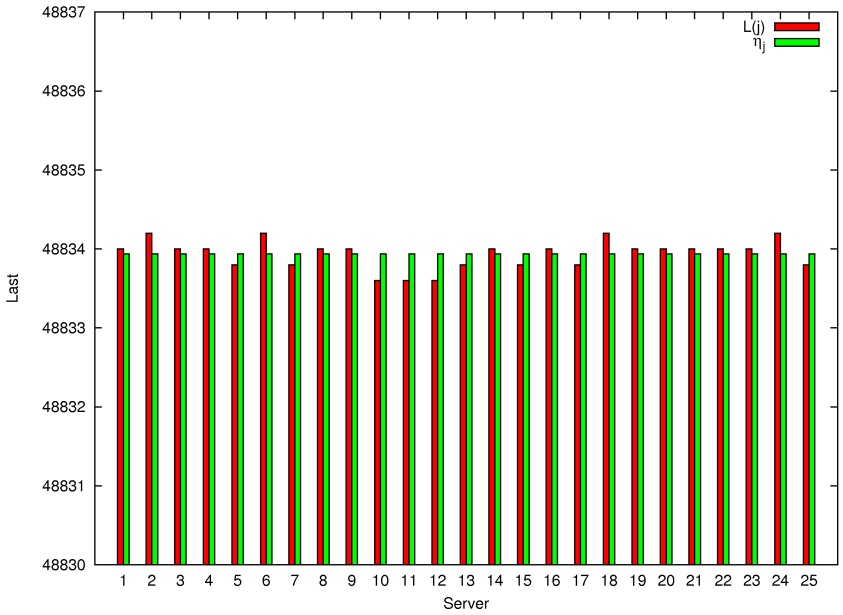


(b)

Abbildung 9.1: Gegenüberstellung der Last-Zielwerte und der erreichten Lasten der besten erzielten Lösungen für (a) Instanz 7 und (b) Instanz 8

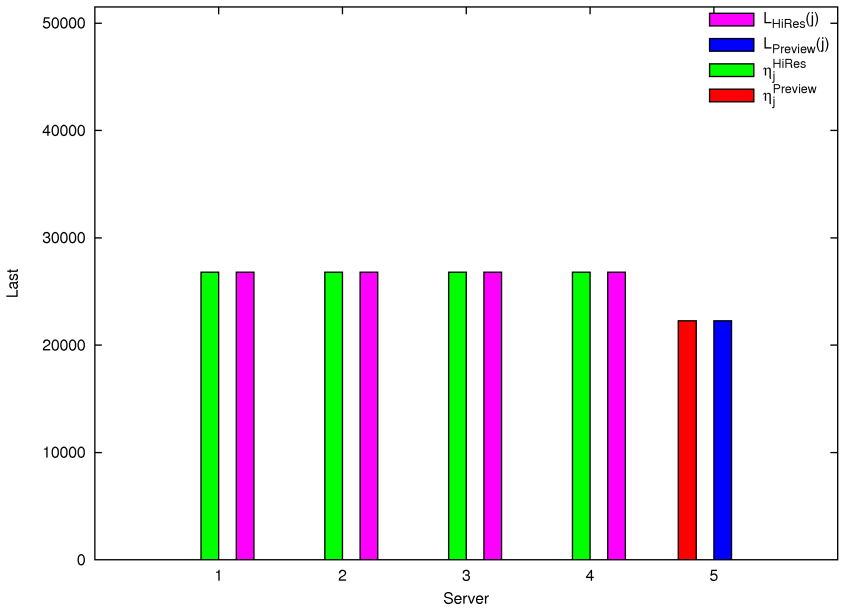


(a)

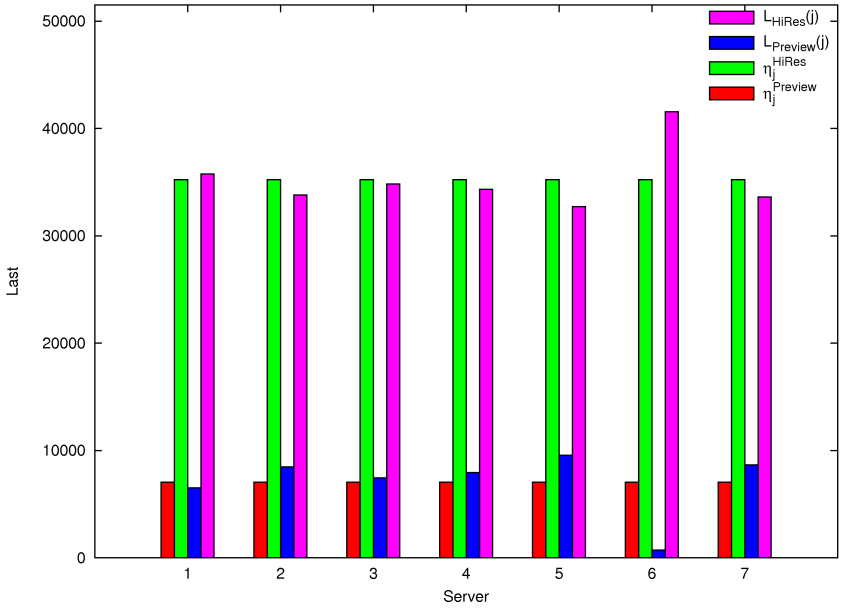


(b)

Abbildung 9.2: Gegenüberstellung der Last-Zielwerte und der erreichten Lasten der besten erzielten Lösungen für (a) Instanz 9 und (b) Instanz 10



(a)



(b)

Abbildung 9.3: Vergleich des Verhaltens bei nicht-uniformen (a) und uniformen (b) akzeptierten Datei-Typen am Beispiel der besten erzielten Lösungen für die Test-Instanzen 3 und 4. Im ersten Fall müssen sich zur Erreichung einer möglichst guten Lösung die $\mathcal{L}_t(j)$ and die η_j^t annähern, im zweiten Fall nicht.

Instanz	$\bar{Z}_1$	σ_{Z_1}	$\bar{Z}_2$	σ_{Z_2}	$\bar{Z}$	σ_Z	$\bar{t}$ [s]	σ_t
1	0.20	0.00	1.40	0.84	1.60	0.86	2.38	0.92
2	0.30	0.00	3.13	1.01	3.43	1.06	14.67	12.49
3	103.36	374.85	0.23	0.07	103.59	388.82	99.87	40.85
4	0.34	0.00	0.10	0.01	0.45	0.34	140.68	142.53
5	0.67	0.00	0.27	0.02	0.94	0.67	434.32	444.73
6	0.91	0.18	66.56	1.28	67.47	1.53	219.24	153.82
7	3.21	1.03	70.08	4.02	73.29	5.41	92.10	22.51
8	2.03	0.26	39.89	3.43	41.93	3.99	406.47	372.49
9	3.17	1.29	54.99	3.58	58.15	5.11	731.12	702.65
10	4.32	1.06	79.53	2.69	83.85	5.36	174.49	94.31

(a) Zielfunktionswerte und Laufzeiten

Instanz	$\mathcal{N}_{\text{Move}}$			$\mathcal{N}_{\text{Swap}}$			$\mathcal{N}_{\text{2-Mip}}$			$\mathcal{N}_{\text{Cyclic}}$		
	$\bar{f}$	$\bar{\Delta}$	$\bar{t}$ [s]	$\bar{f}$	$\bar{\Delta}$	$\bar{t}$ [s]	$\bar{f}$	$\bar{\Delta}$	$\bar{t}$ [s]	$\bar{f}$	$\bar{\Delta}$	$\bar{t}$ [s]
1	9.62	639.79	0.04	46.52	564.07	0.27	5.38	7.82	1.30	0.00	0.00	0.13
2	80.52	2668.80	0.22	135.62	317.81	3.25	10.90	28.23	8.61	0.83	0.05	1.43
3	10.52	741.00	0.14	46.90	765.12	6.88	17.41	3242.91	74.44	12.62	449.58	15.86
4	2.76	206.33	0.19	61.59	508.07	3.59	25.59	1848.41	14.06	1.66	0.72	116.72
5	1.69	83.50	0.27	75.52	371.33	7.60	34.66	2353.52	17.04	2.28	1.42	393.89
6	12.86	3884.09	0.99	73.55	3122.17	5.82	10.90	186175.67	142.12			
7	1.76	15.76	0.60	114.76	1606.29	13.48	59.79	12462.12	35.82			
8	5.14	99.18	0.55	168.52	584.42	79.76	27.79	1974.55	18.04	7.90	2.68	298.83
9	2.21	128.46	0.26	204.41	1892.14	31.18	62.86	8774.38	45.86	23.76	7.66	643.53
10	6.28	576.94	0.94	92.24	1179.68	43.31	60.45	5408.70	53.50			

(b) Statistiken der Nachbarschaftsstrukturen

Tabelle 9.7: Ergebnisse von 30 Testläufen unter Verwendung aller Nachbarschaftsstrukturen

Instanz	$\bar{Z}_1$	σ_{Z_1}	$\bar{Z}_2$	σ_{Z_2}	$\bar{Z}$	σ_Z	$\bar{t}$ [s]	σ_t
1	0.20	0.00	1.62	0.80	1.82	0.83	1.25	0.60
2	0.30	0.00	3.15	0.85	3.45	0.90	10.43	7.77
3	1277.30	1275.79	0.21	0.04	1277.50	1805.29	48.12	1230.32
4	0.37	0.09	0.11	0.02	0.48	0.38	22.37	22.24
5	0.78	0.27	0.29	0.04	1.07	0.83	41.65	41.64
6	0.96	0.20	66.71	0.64	67.67	1.09	207.75	143.93
7	3.41	0.97	71.26	3.80	74.67	5.44	88.21	26.18
8	4.09	0.73	39.68	2.46	43.76	4.93	40.28	15.57
9	5.74	0.96	55.84	3.12	61.58	6.69	26.49	35.39
10	4.19	0.84	78.03	2.73	82.23	5.07	171.64	93.26

(a) Zielfunktionswerte und Laufzeiten

Instanz	$\mathcal{N}_{\text{Move}}$			$\mathcal{N}_{\text{Swap}}$			$\mathcal{N}_{\text{2-Mip}}$		
	$\bar{f}$	$\bar{\Delta}$	$\bar{t}$ [s]	$\bar{f}$	$\bar{\Delta}$	$\bar{t}$ [s]	$\bar{f}$	$\bar{\Delta}$	$\bar{t}$ [s]
1	10.60	495.87	0.03	45.43	621.93	0.21	5.20	10.38	0.96
2	70.57	2654.48	0.19	131.67	243.48	3.02	10.83	15.11	7.02
3	4.87	354.23	0.08	22.53	335.06	0.67	9.50	2628.67	46.55
4	2.50	154.81	0.17	70.40	570.30	2.98	27.33	2017.47	13.38
5	2.83	133.47	0.32	94.27	445.09	7.33	43.17	2960.41	18.10
6	12.10	3837.61	0.82	75.30	3215.84	5.86	11.17	186123.66	134.79
7	1.07	16.56	0.57	116.40	1675.84	12.48	56.53	12338.16	33.80
8	5.73	96.70	0.46	152.27	408.46	22.50	27.30	2077.52	11.53
9	3.67	126.51	0.35	208.83	1894.82	6.28	55.10	8407.27	13.45
10	6.83	605.22	0.89	90.27	1211.90	51.78	60.03	5300.13	48.67

(b) Statistiken der Nachbarschaftsstrukturen

Tabelle 9.8: Ergebnisse von 30 Testläufen ohne Verwendung von $\mathcal{N}_{\text{Cyclic}}$

Instanz	$\bar{Z}_1$	σ_{Z_1}	$\bar{Z}_2$	σ_{Z_2}	$\bar{Z}$	σ_Z	$\bar{t}$ [s]	σ_t
1	0.20	0.00	2.85	0.38	3.05	0.43	0.19	2.87
2	0.66	0.50	6.16	0.71	6.83	0.96	3.72	3.23
3	2397.54	69.58	0.17	0.00	2397.71	2398.55	40.40	2357.31
4	0.34	0.00	0.15	0.02	0.49	0.34	330.93	341.66
5	0.73	0.13	0.34	0.06	1.08	0.75	784.59	809.31
8	2.62	1.33	38.27	3.97	40.89	4.65	471.86	444.98
9	2.42	0.73	53.50	2.92	55.92	3.84	1371.99	1329.67

(a) Zielfunktionswerte und Laufzeiten

Instanz	$\mathcal{N}_{\text{Move}}$			$\mathcal{N}_{\text{Swap}}$			$\mathcal{N}_{\text{Cyclic}}$		
	$\bar{f}$	$\bar{\Delta}$	$\bar{t}$ [s]	$\bar{f}$	$\bar{\Delta}$	$\bar{t}$ [s]	$\bar{f}$	$\bar{\Delta}$	$\bar{t}$ [s]
1	6.87	439.45	0.01	33.10	445.18	0.04	0.43	0.87	0.04
2	10.10	621.21	0.02	81.03	651.75	2.53	6.27	927.95	0.95
3	1.60	225.39	0.02	0.13	0.46	7.56	46.67	2151.57	32.25
4	4.43	182.37	0.13	53.97	407.56	117.42	16.60	2061.27	207.27
5	4.03	103.96	0.32	66.77	287.89	198.30	17.90	2517.74	571.37
8	10.50	131.87	0.56	116.13	235.40	112.72	21.90	2134.83	352.82
9	3.47	160.49	0.20	105.27	1266.98	71.30	52.07	9191.55	1294.40

(b) Statistiken der Nachbarschaftsstrukturen

Tabelle 9.9: Ergebnisse von 30 Testläufen ohne Verwendung von $\mathcal{N}_{\text{2-Mip}}$

Kapitel 10

Zusammenfassung und zukünftige Verbesserungen

10.1 Zusammenfassung

In dieser Arbeit wurde ein im Bereich der Video-on-Demand-Systeme angesiedeltes Lastverteilungsproblem untersucht. Dieses in dieser Arbeit als VSLRB bezeichnete Problem, das in der Ermittlung einer Zuordnung von Replikaten von Video-Objekten und von Zugriffen zu einer gegebenen Menge von Video-Servern besteht, sodass pro Video-Server eine gerechte Auslastung erzielt wird, wurde als kombinatorisches Optimierungsproblem formuliert. Ausgehend von dieser Formulierung wurde eine äquivalente Formulierung als gemischt-ganzzahliges lineares Programm entwickelt, welche ihrerseits in einer Anwendung der Metaheuristik Variable Neighbourhood Search (VNS) auf VSLRB eingesetzt wurde. Da VSLRB als ein spezielles Partitionierungsproblem aufgefasst werden kann, wurde weiters die Anwendbarkeit einer Nachbarschaftsstruktur basierend auf zyklischen Vertauschungen von Zugriffen untersucht.

Aus den Ergebnissen der Tests geht hervor, dass der MIP-Ansatz zur Lösung von VSLRB sehr gut geeignet ist, um innerhalb kurzer Zeit zufriedenstellende Lösungen für Instanzen mit einer geringen Zahl von Video-Servern ($m < 10$) zu ermitteln. Die Anzahl von Video-Objekten bzw. Zugriffen einer Instanz scheint dabei in wesentlich geringerem Ausmaß in die benötigte Laufzeit einzufließen als die Anzahl von Video-Servern. Diese Beobachtung wurde zur Hybridisierung von VNS und MIP-Ansatz genutzt, indem in der k-Server MIP Neighbourhood Unterprobleme mit einer geringen Anzahl von Video-Servern mit Hilfe des MIP-Ansatzes gelöst werden.

Die Testergebnisse der VNS für VSLRB zeigen, dass diese in der Lage ist, für fast jede der verwendeten Testinstanzen innerhalb praktikabler Laufzeiten Lösungen mit für die Praxis sehr zufriedenstellender Güte zu ermitteln, wobei die Zielfunktionswerte für Instanzen mit einer größeren Anzahl von Servern wesentlich besser ausfallen als jene des reinen MIP-Ansatzes. Die besten VNS-Resultate werden unter Verwendung aller beschriebenen Nachbarschaftsstrukturen erzielt. Wird auf die Verwendung von $\mathcal{N}_{\text{Cyclic}}$ verzichtet, werden etwas schlechtere Ergebnisse erzielt, bei allerdings deutlich verbesserter Laufzeit.

Weiters geht aus den Testresultaten die grundsätzliche Anwendbarkeit einer Nachbarschaftsstruktur basierend auf zyklischen Vertauschungen auf VSLRB hervor. Wurde auf die Verwendung von $\mathcal{N}_{\text{2-Mip}}$ verzichtet, konnten für zwei große Testinstanzen mit ein-

heitlichen akzeptierten Datei-Typen bessere Resultate als unter Verwendung aller Nachbarschaftsstrukturen erzielt werden. Trotzdem scheint angesichts des hohen Speicherverbrauchs und des großen Laufzeit-Overheads, der durch die Erstellung und Verwaltung des Improvement-Graph entsteht, ein verstärkter Einsatz MIP-basierender Nachbarschaftsstrukturen vielversprechender zu sein. Es ist zu erwarten, dass k-Server MIP Neighbourhoods mit $k = 3$ oder $k = 4$ in ähnlichem oder stärkerem Ausmaß zur Lösungsqualität betragen könnten wie $\mathcal{N}_{\text{Cyclic}}$, bei gleichzeitig wesentlich geringerer Laufzeit.

10.2 Mögliche Verbesserungen

10.2.1 Verbesserung der Serverauswahl der k-Server MIP Neighbourhood

Die schlechten Ergebnisse der VNS für Testinstanz 3 erklären sich zu einem Teil aus dem Verfahren zur Auswahl der berücksichtigten Server in der k-Server MIP Neighbourhood: Sobald eine Situation eintritt, in der die Server mit negativen Abweichungen von η_j keine Überschneidungen bezüglich der akzeptierten Datei-Typen mit den $\frac{k}{2}$ Servern mit positiver Abweichung von η_j aufweisen, kann keine gültige Server-Auswahl ermittelt werden.

Um alle möglichen Fälle abzudecken, könnte die Auswahl der k Server mittels des folgenden ILP-Modells erfolgen, das k Server mit maximalem Verbesserungspotential wählt, wobei die gewählten Server eine Überschneidung der von ihnen akzeptierten Datei-Typen in zumindest einem Datei-Typ aufweisen müssen.

Die binären Entscheidungsvariablen $x_j,\ j \in C$ dieses Modells beschreiben, ob Server j in die Server-Auswahl aufgenommen wird. Das Verbesserungspotential einer Server-Auswahl wird durch die folgende Zielfunktion beschrieben:

$$\max \sum_{j \in C^+} x_j \Delta(j) - \sum_{j \in C^-} x_j \Delta(j) - \left| \sum_{j \in C^+} x_j \Delta(j) + \sum_{j \in C^-} x_j \Delta(j) \right|$$

wobei

$$\Delta(j) = \mathcal{L}(j) - \eta_j \quad \forall j \in C$$

die Differenz zwischen der Last $\mathcal{L}(j)$ eines Servers und seinem Last-Zielwert η_j bezeichnet. Die weiters in dieser Formulierung verwendeten Mengen von Servern C^+ und C^- bezeichnen all jene Video-Server der aktuellen Lösung, die ihren Last-Zielwert über- bzw. unterschreiten:

$$C^+ = \{j \in C \mid \Delta(j) > 0\}$$

$$C^- = \{j \in C \mid \Delta(j) < 0\}$$

Eine Server-Auswahl besitzt maximales Verbesserungspotential, wenn sowohl die Summe aller positiven als auch aller negativen $\Delta(j)$ möglichst groß ist und gleichzeitig diese Summen möglichst wenig voneinander abweichen. Dieser Zusammenhang ist in der Visualisierung der Zielfunktion in Abbildung 10.1 dargestellt.

Zusätzlich muss gelten, dass die auf diese Weise ermittelte Server-Auswahl aus genau k Servern besteht:

$$\sum_{j \in C} x_j = k$$

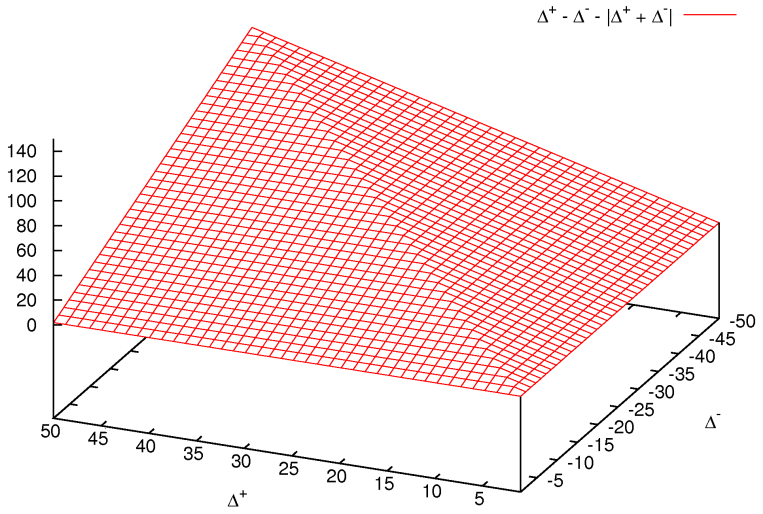


Abbildung 10.1: Visualisierung der Zielfunktion der verbesserten Serverauswahl. $\Delta^+ = \sum_{j \in C^+} x_j \Delta(j)$, $\Delta^- = \sum_{j \in C^-} x_j \Delta(j)$

Eine Überschneidung in einem der von der Server-Auswahl akzeptierten Datei-Typ $\tau \in T$ liegt vor, wenn zumindest zwei Server der Auswahl τ akzeptieren. Dies kann durch die folgende Bedingung ausgedrückt werden:

$$\sum_{j \in C} x_j t_j^\tau \geq 2 \quad \forall \tau \in T \tag{10.1}$$

$$t_j^\tau = \begin{cases} 1 & \text{falls } \tau \in T_j \\ 0 & \text{sonst} \end{cases} \quad \forall j \in C,\ \tau \in T$$

Durch Multiplikation der rechten Seite von Bedingung 10.1 mit neuen binären Entscheidungsvariablen $y_\tau, \tau \in T$ und Hinzufügen einer weiteren Nebenbedingung kann modelliert werden, dass diese Einschränkung nicht für alle, sondern zumindest einen der Datei-Typen gelten muss:

$$\sum_{j \in C} x_j t_j^\tau \geq 2 y_\tau \quad \forall \tau \in T$$

$$\sum_{\tau \in T} y_\tau \geq 1$$

10.2.1.1 Zusammenfassung der Formulierung

Ermittle y, $x_j,\ j \in C$ und $y_\tau,\ \tau \in T$, sodass der folgende Ausdruck unter Berücksichtigung der Nebenbedingungen 10.2 bis 10.9 maximal wird:

$$\max \sum_{j \in C^+} x_j \Delta(j) - \sum_{j \in C^-} x_j \Delta(j) - y$$

Unter den folgenden Nebenbedingungen:

$$\sum_{j \in C^+} x_j \Delta(j) + \sum_{j \in C^-} x_j \Delta(j) \leq y \tag{10.2}$$

$$-\sum_{j\in C^+} x_j\Delta(j) - \sum_{j\in C^-} x_j\Delta(j) \leq y \tag{10.3}$$

$$\sum_{j\in C} x_j = k \tag{10.4}$$

$$\sum_{j\in C} x_j t_j^\tau \geq 2y_\tau \quad \forall \tau \in T \tag{10.5}$$

$$\sum_{\tau\in T} y_\tau \geq 1 \tag{10.6}$$

$$x_j \in \{0,1\} \quad \forall j \in C \tag{10.7}$$

$$y_\tau \in \{0,1\} \quad \forall \tau \in T \tag{10.8}$$

$$y \geq 0 \tag{10.9}$$

wobei

$$\Delta(j) = \mathcal{L}(j) - \eta_j \quad \forall j \in C$$

$$C^+ = \{j \in C \mid \Delta(j) > 0\}$$

$$C^- = \{j \in C \mid \Delta(j) < 0\}$$

$$t_j^\tau = \begin{cases} 1 & \text{falls } \tau \in T_j \\ 0 & \text{sonst} \end{cases} \quad \forall j \in C,\ \tau \in T$$

10.2.2 Verwendung von Path Exchanges

Testinstanz 3 zeigt auch eine Schwäche der Cyclic Exchange Neighbourhood auf: Kann eine Verbesserung des Zielfunktionswerts nur erreicht werden, indem ein Server Zugriffe erhält, ohne gleichzeitig Zugriffe abzugeben, kann dies nicht durch einen zyklischen Austausch geschehen. Ahuja et al. beschreiben in [5] eine Verallgemeinerung von Cyclic Exchanges auf Path Exchanges, sodass Elemente nicht mehr in einem teilmengendisjunkten Zyklus sondern entlang eines teilmengendisjunkten Pfades verschoben werden. Das Auffinden solcher Pfade kann durch das Hinzufügen von Dummy-Knoten im Improvement-Graph realisiert werden, deren ein- und ausgehende Kanten Kosten von null besitzen. Die Suche nach Pfaden von Verschiebungen im Improvement-Graph kann nun auf die selbe Weise erfolgen wie die Suche nach zyklischen Vertauschungen. Sobald ein teilmengendisjunkter Zyklus mit negativen Kosten im Improvement-Graph einen Dummy-Knoten enthält, wurde ein Pfad von Verschiebungen gefunden.

Anhang A

Lebenslauf

Zur Person

Name	Jakob Walla
Geboren	24.5.1982
Nationalität	Österreich
Sprachen	Deutsch, Englisch

Ausbildung

10/2006 – heute	Masterstudium Software Engineering & Internet Computing an der TU Wien
10/2001 – 03/2006	Bakkalaureatsstudium Medieninformatik / Computergrafik & Digitale Bildverarbeitung an der TU Wien Abschluss mit Auszeichnung
09/1999 – 06/2001	HTBLuVA Wr. Neustadt, Abteilung für EDV & Organisation Abschluss mit Auszeichnung

Berufserfahrung

10/2005 – 09/2006	Ableistung des Zivildiensts bei der Caritas d. ED Wien Tätigkeitsbereich: Systemadministration, Helpdesk, Scripting Windows & Linux
01/2003 – heute	Selbständige Tätigkeit im Bereich Softwareentwicklung und IT-Dienstleistungen
06/2002 – 10/2002	Siemens Business Services Tätigkeitsbereich: Softwareentwicklung Content Management Systeme

Literaturverzeichnis

[1] YouTube.com. `http://www.youtube.com`. [Online; zuletzt abgefragt 22.3.2009].

[2] E. Aarts and J. Lenstra. *Local Search in Combinatorial Optimization.* Princeton University Press, 2003.

[3] C. Aggarwal, J. Wolf, and P. Yu. The Maximum Factor Queue Length Batching Scheme for Video-on-Demand Systems. *IEEE Transactions on Computers*, 50(2):97–110, 2001.

[4] R. Ahuja, Ö. Ergun, J. Orlin, and A. Punnen. A survey of very large-scale neighborhood search techniques. *Discrete Applied Mathematics*, 123(1-3):75–102, 2002.

[5] R. Ahuja, J. Orlin, and D. Sharma. *New Neighborhood Search Structures for the Capacitated Minimum Spanning Tree Problem.* Sloan School of Management, Massachusetts Institute of Technology, 1998.

[6] R. Ahuja, J. Orlin, and D. Sharma. Very large-scale neighborhood search. *International Transactions in Operational Research*, 7(4-5):301–317, 2000.

[7] R. Ahuja, J. Orlin, and D. Sharma. Multi-exchange neighborhood structures for the capacitated minimum spanning tree problem. *Mathematical Programming*, 91(1):71–97, 2001.

[8] D. Bertsekas. A simple and fast label correcting algorithm for shortest paths. *LIDS Technical Reports, Massachusetts Institute of Technology, Laboratory for Information and Decision Systems*, 1992.

[9] D. Bertsimas and J. Tsitsiklis. *Introduction to Linear Optimization.* Athena Scientific, 1997.

[10] K. Chen, H.-C. Chen, R. Borie, and J. C. L. Liu. File replication in video on demand services. In *ACM-SE 43: Proceedings of the 43rd annual Southeast regional conference*, pages 162–167, New York, NY, USA, 2005. ACM.

[11] L. Cherkasova and M. Gupta. Analysis of enterprise media server workloads: access patterns, locality, content evolution, and rates of change. *IEEE/ACM Transactions on Networking*, 12(5):781–794, 2004.

[12] C. Chou, L. Golubchik, and J. Lui. Striping doesn't scale: how to achieve scalability for continuous media servers with replication. In *Proceedings of the 20th International Conference on Distributed Computing Systems*, pages 64–71, Taipei, Taiwan, 2000.

[13] J. Dahl and L. Vendenberghe. CvxOpt - Python Software for Convex Optimization. `http://abel.ee.ucla.edu/cvxopt`. [Online; zuletzt abgefragt 4.11.2008].

[14] A. Dan, D. Sitaram, and P. Shahabuddin. Scheduling policies for an on-demand video server with batching. In *Proceedings of the second ACM international conference on Multimedia*, pages 15–23. ACM New York, NY, USA, 1994.

[15] A. Duarte, C. Ribeiro, and S. Urrutia. A Hybrid ILS Heuristic to the Referee Assignment Problem with an Embedded MIP Strategy. *Lecture Notes in Computer Science*, 4771:82, 2007.

[16] B. Estellon, F. Gardi, and K. Nouioua. Real-life car sequencing: very large neighborhood search vs very fast local search. *European Journal of Operational Research (EJOR)*, 2007.

[17] R. Fahrion and M. Wrede. On A principle of Chain Exchange for Vehicle Routing Problems (1-vrp). *OR Journal (Journal of the Operational Research Society)*, 41(9):821–827, 1990.

[18] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.

[19] Z. Ge, P. Ji, and P. Shenoy. A Demand Adaptive and Locality Aware (DALA) streaming media server cluster architecture. In *Proceedings of the 12th international workshop on Network and operating systems support for digital audio and video*, pages 139–146. ACM New York, NY, USA, 2002.

[20] D. Ghose and H. Kim. Scheduling Video Streams in Video-on-Demand Systems: A Survey. *Multimedia Tools and Applications*, 11(2):167–195, 2000.

[21] F. Glover and G. Kochenberger. Handbook of Metaheuristics, volume 57 of International Series in Operations Research & Management Science, 2003.

[22] C. Griwodz, M. Bär, and L. Wolf. Long-term movie popularity models in video-on-demand systems: or the life of an on-demand movie. In *Proceedings of the fifth ACM international conference on Multimedia*, pages 349–357. ACM New York, NY, USA, 1997.

[23] D. Guan and S. Yu. A two-level patching scheme for video-on-demand delivery. *IEEE Transactions on Broadcasting*, 50(1):11–15, 2004.

[24] M. Guo, M. Ammar, and E. Zegura. Selecting among replicated batching video-on-demand servers. In *Proceedings of the 12th international workshop on Network and operating systems support for digital audio and video*, pages 155–163. ACM New York, NY, USA, 2002.

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

[26] P. Hansen and N. Mladenovic. *A tutorial on variable neighborhood search*. Groupe d'études et de recherche en analyse des décisions, 2003.

[27] M. Hribar, V. Taylor, and D. Boyce. Choosing a shortest path algorithm. *Technical Report CSE-95-004, Computer Science-Engineering, EECS Department, Northwestern University*, 1995.

[28] B. Hu. *Hybrid Metaheuristics for Generalized Network Design Problems*. PhD thesis, Technische Universität Wien, 2008.

[29] C. Huang, J. Li, and K. Ross. Can internet video-on-demand be profitable? In *Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 133–144. ACM Press New York, NY, USA, 2007.

[30] T. Ibaraki, M. Kubo, T. Masuda, T. Uno, and M. Yagiura. Effective local search algorithms for the vehicle routing problem with general time window constraints. In *Proceedings of the 4th Metaheuristics International Conference (MIC2001)*, pages 293–297, Porto, 2001.

[31] D. E. Knuth. *The art of computer programming, volume 2 (3rd ed.): seminumerical algorithms*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1997.

[32] H. Koop, A. und Moock. *Lineare Optimierung*. Spektrum-Verlag Berlin Heidelberg, 2008.

[33] W. Liao and V. Li. The Split and Merge protocol for interactive video-on-demand. *IEEE MultiMedia*, 4(4):51–62, 1997.

[34] T. Little and D. Venkatesh. Prospects for Interactive Video-on-Demand. *IEEE MultiMedia*, 1(3):14, 1994.

[35] H. Ma and K. G. Shin. Multicast Video-on-Demand services. *SIGCOMM Computer Communication Review*, 32(1):31–43, 2002.

[36] N. Mladenović and P. Hansen. Variable neighborhood search. *Computers and Operations Research*, 24(11):1097–1100, 1997.

[37] G. Nemhauser and L. Wolsey. *Integer and Combinatorial Optimization. 1988*. Wiley, New York, 1988.

[38] M. G. Norman, T. Zurek, and P. Thanisch. Much ado about shared-nothing. *SIGMOD Rec.*, 25(3):16–21, 1996.

[39] M. Palpant, C. Artigues, and P. Michelon. LSSPER: Solving the Resource-Constrained Project Scheduling Problem with Large Neighbourhood Search. *Annals of Operations Research*, 131(1):237–257, 2004.

[40] M. Prandtstetter and G. Raidl. An integer linear programming approach and a hybrid variable neighborhood search for the car sequencing problem. *European Journal of Operational Research*, 191(3):1004–1022, 2008.

[41] G. Raidl and J. Puchinger. Combining (integer) linear programming techniques and metaheuristics for combinatorial optimization. In C. Blum, M. Augilera, A. Roli, and M. Sampels, editors, *Hybrid Metaheuristics*, volume 114 of *Studies in Computational Intelligence*, pages 31–62. Springer, 2008.

[42] G. Raidl, J. Puchinger, and C. Blum. Metaheuristic hybrids. In M. Gendreau and J. Potvin, editors, *Handbook of Metaheuristics*. Springer, 2008 (submitted).

[43] M. Scaparra, S. Pallottino, and M. Scutella. Large-Scale Local Search Heuristics for the Capacitated Vertex p-Center Problem. *Networks*, 43(4):241–255, 2004.

[44] P. Thompson and J. Orlin. The theory of cyclic transfers. *Operations Research Center Working Papers, Massachusetts Institute of Technology*, 1989.

[45] P. Thompson and H. Psaraftis. Cyclic transfer algorithms for multivehicle routing and scheduling problems. *Operations Research*, 41(5):935–946, 1993.

[46] N. Venkatasubramanian and S. Ramanathan. Load management in distributed video servers. In *Proceedings of the 17th International Conference on Distributed Computing Systems (ICDCS '97)*, page 528, Washington, DC, USA, 1997. IEEE Computer Society.

[47] J. S. Vitter. Faster methods for random sampling. *Communications of the ACM*, 27(7):703–718, 1984.

[48] Y. Wang, J. Liu, D. Du, and J. Hsieh. Efficient video file allocation schemes for video-on-demand services. *Multimedia Systems*, 5(5):283–296, 1997.

[49] J. Wolf, P. Yu, and H. Shachnai. Disk load balancing for video-on-demand systems. *Multimedia Systems*, 5(6):358–370, 1997.

[50] Y. Won and J. Srivastava. Strategic Replication of Video Files in a Distributed Environment. *Multimedia Tools and Applications*, 8(2):249–283, 1999.

[51] H. Yu, D. Zheng, B. Zhao, and W. Zheng. Understanding user behavior in large-scale video-on-demand systems. In *Proceedings of the 2006 EuroSys conference*, pages 333–344, New York, NY, USA, 2006. ACM Press.

[52] X. Zhou and C. Xu. Optimal Video Replication and Placement on a Cluster of Video-on-Demand Servers. In *Proceedings of the International Conference on Parallel Processing*, pages 547–555, Washington, DC, USA, 2002. IEEE Computer Society.