

Institut für Architektur von Anwendungssystemen
Universität Stuttgart
Universitätsstraße 38
D-70569 Stuttgart

Diplomarbeit Nr. 2663

Stratifizierte Transaktionen

Olha Danylevych

Studiengang:	Informatik
Prüfer:	Prof. Dr. Frank Leymann
Betreuer:	Dipl. Inf. Ralph Mietzner
begonnen am:	16. August 2007
beendet am:	14. Februar 2008
CR-Klassifikation:	F.2.2, G.1.6, G.2.1, I.2.8

Inhaltsverzeichnis

1. Einführung	1
1.1. Motivation. Problemstellung	1
1.2. Aufbau der Arbeit	2
I. Stratifikation: das Modell	4
2. Grundlagen	5
2.1. Transaktionen	5
2.2. Synchronisationsverfahren: 2-Phase-Commit-Protocol	6
2.3. Asynchrone Kommunikation: Message-basierte Kommunikation	8
3. Stratifizierte Transaktionen: Modellbeschreibung.	11
3.1. Annahmen	11
3.1.1. Ausführungsvoraussetzungen	11
3.1.2. Fehlerbehandlung	12
3.2. Eigenschaften von Ressourcen	12
3.3. Eigenschaften von Transaktionen	13
3.4. Stratifizierung	17
3.4.1. Präzedenzrelation (Abhängigkeitsgraph)	17
3.4.2. Stratifizierungsgraph	19
3.4.3. Die Eigenschaften des Stratifizierungsgraphs	20
3.5. Bewertung der Stratifikation	24
3.5.1. Nachrichtenaufwand des Zwei-Phasen-Commit-Protokolls	24
3.5.2. Nachrichtenaufwand bei Queues	26
3.5.3. Grad der Nebenläufigkeit	27

3.5.4.	Antwortzeit	29
3.5.5.	Ausführungszeit	29
3.5.6.	Kosten der Invokationen	29
3.5.7.	Kosten der Recovery	29
3.5.8.	Zusätzliche Zeitkosten (TimeCosts)	30
3.5.9.	Die gesamte Bewertung	30
3.6.	Azyklität des Stratifizierungsgraphs	30
3.7.	Beispiele für Stratifikation	31

II. Optimale Stratifikation 34

4. Optimierungsverfahren: Grundlagen 35

4.1.	Das Optimierungsproblem: Begriffsbildung	36
4.2.	Optimierungsverfahren	37
4.2.1.	Suchraum	40
4.2.2.	Bewertungsfunktion	41
4.2.3.	Nachbarschaftsstruktur	43
4.3.	Lokale Suche	44
4.3.1.	Hillclimbing	44
4.3.2.	Simulated Annealing	45
4.4.	Evolutionssimulation	48
4.4.1.	Grundbegriffe	49
4.4.2.	Der evolutionäre Algorithmus	50
4.4.3.	Initialisierung	52
4.4.4.	Bewertung der Fitness	52
4.4.5.	Rekombination	53
4.4.6.	Mutation	53
4.4.7.	Selektion	55
4.4.7.1.	Elternselektion	57
4.4.7.2.	Umweltselektion	57
4.4.8.	Terminierungsbedingung	58
4.4.9.	Evolutionäre Standardalgorithmen	58

5. Realisierung der Stratifikation	61
5.1. Suche der optimalen Stratifikation	61
5.1.1. Das Problem der Graphpartitionierung	62
5.1.2. Der Suchraum	63
5.1.3. Die Nachbarschaft	65
5.1.4. Die Nebenbedingungen	67
5.1.4.1. Erkennung der Zyklen nach der Konstruktion eines Nachbargraphen	68
5.1.4.2. Vermeiden von Zyklen. Konstruktion eines bereits zyklenfreien benachbarten Stratifikationsgraphen . .	69
5.2. Realisierung von Hillclimbing	72
5.3. Realisierung von Simulated Annealing	73
5.3.1. Initialbelegung des Parameters T	73
5.4. Hybrider Ansatz: Kombination von Hillclimbing und Simulated Annealing	75
5.5. Realisierung der Evolutionssimulation	77
5.5.1. Initialisierung der Population	77
5.5.2. Mutation	77
5.5.3. Elternselektion und Rekombination	77
5.5.3.1. Allgemeiner Ansatz	78
5.5.3.2. Vereinfachte Version	79
5.5.4. Umweltselektion	80
5.6. Verwendung von Mustern bei der Stratifikation	80
5.6.1. Muster: Stratifizierung nach einer mehrwertigen Eigenschaft (z.B. Lokation)	82
 III. Realisierung	 84
6. Evaluation	85
6.1. Zufallsgraph	85
6.2. Experimenteller Vergleich der Verfahren	86

6.3.	Vergleich der Optimierungsverfahren für einen Zufallsgraph mit einer durchschnittlichen Anzahl von Kanten	88
6.4.	Vergleich der Optimierungsverfahren für einen „dünnen“ Zufallsgraph	88
6.5.	Vergleich der Optimierungsverfahren für einen „dichten“ Zufallsgraph	90
6.6.	Fazit	90
7.	Prototyp	97
7.1.	Aufbau der Anwendung	97
7.2.	Datenstrukturen	99
7.2.1.	Abhängigkeitsgraph	99
7.2.2.	Stratifizierungsgraph	100
7.3.	Erweiterbarkeit	100
7.3.1.	Erweiterung und Änderung der Transaktionseigenschaften . .	100
7.3.2.	Änderung und Erweiterung der Bewertungskriterien	101
7.3.3.	Änderung und Erweiterung der Optimierungsalgorithmen . . .	101
7.4.	Ein-/Ausgaben der Daten	101
7.4.1.	Konfigurationen	101
7.4.2.	Die graphische Benutzeroberfläche (GUI)	102
7.4.3.	Das Format der .xml Ausgabe der Stratifizierung	105
8.	Zusammenfassung und Ausblick	108

1. Einführung

1.1. Motivation. Problemstellung

Zur Realisation von Geschäftsprozessen werden Geschäftsanwendungen verwendet, die oft über heterogene Systeme verteilt sind und verschiedenartige, verteilte Ressourcen manipulieren. Die Robustheit von Geschäftsanwendungen hängt unter anderem von der Konsistenz der verwendeten Ressourcen ab. Ein Konzept für die Aufrechterhaltung der Konsistenz verteilter Ressourcen sind *Transaktionen*. Transaktionen bündeln jeweils eine Untermenge von Aktivitäten oder Operationen zu einer Einheit, für welche gilt, dass sie ganz oder gar nicht durchgeführt werden sollen. Diese Eigenschaft wird Atomizität genannt.

Von verteilten Geschäftsanwendungen, wie z. B. Workflow-Management-Systemen (WFMS) verwendete Transaktionen sind entsprechend systemübergreifend und verteilt, was die Transaktionsverwaltung erschwert. Die Art der Aufteilung von Operationen auf die einzelnen Transaktionen und die eventuelle Vereinigung von Teiltransaktionen in umfassende globale Transaktionen hat direkten Einfluss auf die Performance der gesamten Anwendung. Ein Ansatz zur Lösung dieser zusätzlichen Anforderungen verteilter Transaktionen sind *stratifizierte Transaktionen* ([1], [2]).

Stratifizierte Transaktionen sind eine Verknüpfung von sog. „queued“-Transaktionen und synchronen Transaktionen nach dem Two-Phase-Commit-Modell (2PC). Ein Anwendungsbeispiel für dieses Konzept findet man bei WFMS. Eine globale, eventuell verteilte Transaktion entsteht durch die Anbindung der Transaktionen einer verteilten Anwendung an systeminterne Transaktionen des WFMS. Der dadurch möglicherweise entstehende negative Einfluss auf die Effizienz des Gesamtsystems muss natürlich minimiert werden. Um den Durchsatz und Nebenläufigkeit zu erhöhen,

kann die Menge der Teiltransaktionen nach einer bestimmten Strategie in Gruppen („Strata“) aufgeteilt werden. Die Transaktionen eines *Stratums* werden durch ein 2PC-Protokoll (wie z.B. das XA-Protokoll) synchronisiert; die einzelnen Gruppen kommunizieren miteinander durch persistente Nachrichtenwarteschlangen (Queues). Die Gesamtheit aller Strata bildet eine Graphstruktur und die entsprechend aufgeteilte globale, verteilte Transaktion wird als *stratifizierte Transaktion* bezeichnet. Die Grundannahme für die Stratifizierung ist, dass alle Strata genau dann schließlich ein Commit ausführen, wenn vorher das Wurzelstratum bereits Commit ausgeführt hat. Das wiederholte Scheitern eines Stratums wird als eine echte Ausnahme betrachtet.

Ein aktueller Trend bei der Implementierung verteilter Geschäftsanwendungen ist die Verwendung von *Web Services* (WS), die per se verteilt sind. Web Services bieten bereits Technologien und Standards, die die Anwendung bei der Abwicklung von Transaktionen unterstützen und daher auch mit dem Konzept der stratifizierten Transaktionen in direkten Kontakt kommen. Insbesondere sind das die *Web Service Transactions* (WS-TX), *Web Service Atomic Transactions* (WS-AT) und *Web Service Reliable Messaging* (WS-RM).

1.2. Aufbau der Arbeit

Im konzeptionellen Teil der Arbeit (Kapitel 3) wird eine formale Beschreibung des Modells von stratifizierten Transaktionen eingeführt. Unter anderem werden hier mögliche Kriterien formal beschrieben, anhand deren man die Transaktionen auf „Strata“ (Gruppen von 2PC-Transaktionen) aufteilt. Zuerst werden die Transaktionseigenschaften beschrieben, die für die Aufteilungsentscheidung maßgebend sind. Zu diesen Eigenschaften gehören z.B. die Lokation, die Wiederaufsetzkosten (Recovery), die Aufrufkosten (Invokation), die Verwendung von bestimmten Ressourcen, und die Kommunikation mit einer Teilmenge von Transaktionen. Weiterhin wird eine Graphstruktur beschrieben, die die kausalen Abhängigkeiten zwischen den einzelnen Transaktionen abbildet. Die Menge der Transaktionen mitsamt ihrer Eigenschaften

stellen zusammen mit dem Abhängigkeitsgraph alle benötigten Informationen zur Verfügung, die zur Aufteilung der Transaktion auf Strata nötig sind. Die Aufteilung der Transaktionen auf Strata wird durch einen „Stratifikationsgraph“ repräsentiert, dieser wird ebenfalls in Kapitel 3 beschrieben.

Bei der Aufteilung der Transaktion auf Strata wird die optimale Lösung gesucht, daher ist es notwendig, Bewertungskriterien für die Stratifikation herauszufinden und zu beschreiben. In Kapitel 4 der vorliegenden Arbeit werden Kriterien wie die gesamte Ausführungszeit, die Anzahl der benötigten Nachrichten des 2PC-Protokolls, die Anzahl der benötigten Queues, und der Parallelitätsgrad in dem Stratifikationsgraph beschrieben. Es wird auch eine Zielfunktion eingeführt, die die einzelnen Bewertungskriterien miteinander kombiniert. Zur Suche nach der optimalen Stratifizierung wurden heuristischen Optimierungsverfahren eingesetzt. Es handelt sich um Verfahren der lokalen Suche (Hillclimbing, Simulated Annealing und ein hybrider Ansatz aus beiden) sowie um ein populationsbasiertes Optimierungsverfahren (Evolutionäre Programmierung). Kapitel 4 führt in diese Methoden ein; der Einsatz der Verfahren bei stratifizierten Transaktionen wird dann in Kapitel 5 erläutert.

Für die Evaluation und den Vergleich der Verfahren wurden Experimente durchgeführt. Sie werden zusammen mit den verfahrensspezifischen Einstellungen und ihren Ergebnissen im Kapitel 6 vorgestellt. Im Kapitel 7 wird der im Rahmen der Arbeit realisierten Prototyp präsentiert.

Teil I.

Stratifikation: das Modell

2. Grundlagen

Das Konzept der stratifizierten Transaktionen ist eine Verknüpfung der Konzepte von Zwei-Phasen-Commit- synchronen Transaktionen und „Queued“-Transaktionen. Beide Konzepte werden hier deshalb kurz vorgestellt.

2.1. Transaktionen

Eine Transaktion ist eine Menge von zusammengehörigen Operationen auf dem physischen (und gleichfalls dem abstrakten) Zustand des Systems [3]. Jede Operation manipuliert den Zustand des Systems, indem es seine Ressourcen verändert (wie Datenbanken, Nachrichtensysteme, EJB-Container). Obwohl die Transaktion aus einer Menge von Operationen besteht, wird sie als ein Ganzes verstanden. Das liegt in ihren Eigenschaften begründet.

Eine Transaktion im klassischen Sinne muss folgende Eigenschaften besitzen, die auch als *ACID-Eigenschaften* bezeichnet werden:

- **Atomizität.** Die von der Transaktion vorgenommenen Änderungen werden entweder ganz oder gar nicht durchgeführt. Es ist gewährleistet, dass unter keinen Umständen nur ein Teil der transaktionalen Änderungen ausgeführt und für die Außenwelt sichtbar gemacht wird.
- **Konsistenz.** Eine Transaktion stellt eine korrekte Transformation des Systemzustands dar, d.h. das System wird von einem korrekten Zustand in einen anderen korrekten Zustand überführt. Die von der Transaktion durchgeführten Änderungen verletzen keine Integritätsbedingungen des Systems, sobald sie sichtbar werden.

- **Isolation.** Unabhängig von der Organisation der Durchführung parallel verarbeiteter Transaktionen hat es von außen den Anschein, als wären alle Transaktionen seriell ausgeführt worden. Die transaktionalen Änderungen werden erst nach einem Commit für die Aussenwelt sichtbar.
- **Dauerhaftigkeit.** Sobald die Transaktion erfolgreich (d.h. mit einem Commit) abgeschlossen wurde, können die durch die Transaktion durchgeführten Änderungen nicht mehr zurückgenommen werden und sind daher dauerhaft.

Der Begriff *ACID* für *atomicity, consistency, isolation, durability* wurde von Haerder und Reuter im Jahre 1983 eingeführt [3].

Die wichtigsten Steuerungsbestandteile einer Transaktion sind *begin* (bedeutet den Anfang der Abarbeitung der Transaktion), *commit* (bezeichnet ein erfolgreiches Ende der Abarbeitung und dauerhaftes Speichern der Änderungen) und *rollback* (Befehl, um die Transaktion ungeschehen zu machen).

2.2. Synchronisationsverfahren: 2-Phase-Commit-Protocol

Verteilte Transaktionen sind Transaktionen, die mehrere räumlich verteilte Ressourcen manipulieren. Falls eine verteilte Transaktion zu Ende abgearbeitet wurde, muss sicher gestellt werden, dass alle teilnehmenden Ressourcenmanager ein Commit oder ein Abort durchführen. Um die ACID-Eigenschaften von Transaktionen sicherzustellen, werden Synchronisationsverfahren verwendet. Das meist verwendete Synchronisationsverfahren ist das Zwei-Phasen-Commit-Protokoll.

In Analogie zu vielen anderen Methoden der Informatik wurden auch die Synchronisationsverfahren von Verfahren abgeleitet, die in der Gesellschaft entstanden sind. Das Commit-Protokoll entspricht den Alles-oder-nichts-Abkommen, die von mehreren beteiligten Teilnehmern geschlossen werden. Wie [3] bemerkt, werden beispielsweise bei den Heiratsverfahren unserer Gesellschaft beide Parteien gefragt, und nur bei beidseitiger Zustimmung wird das Verfahren mit einer Zustandsänderung beider Parteien von „ledig“ auf „verheiratet“ abgeschlossen. Stimmt nur eine Partei nicht

zu, wird der Zustand keiner der Parteien durch dieses Verfahren geändert.

Entsprechende Protokolle werden in der Informatik verwendet, um allen teilnehmenden Parteien eine einstimmige Entscheidung darüber zu ermöglichen, ob das System durch eine Transaktion in einen anderen (korrekten) Zustand übergeht. Es bleibt in dem alten Zustand, falls mindestens ein Teilnehmer dagegenstimmte oder gar nicht antwortet. Die Parteien werden durch die Transaktionsmanager (TM) der Knoten vertreten, die an der Transaktion teilnehmen. Beim Zwei-Phasen-Commit-Protocol spielen die Transaktionsmanager folgende mögliche Rollen:

- Koordinator: der Transaktionsmanager des Knotens, von dem aus die Transaktion initiiert wurde.
- Teilnehmer: die Transaktionsmanager/ResourceManager der anderen Knoten, die in die Transaktion eingebunden waren.

Zu Beginn der Transaktion registrieren sich die teilnehmenden Transaktions-/Ressourcen-Manager beim Koordinator. Sobald der Koordinator entscheidet, dass die Transaktion abgeschlossen werden soll, läuft der eigentliche Kernteil des Protokolls ab. Der Koordinator sendet in zwei Runden Nachrichten an die Teilnehmer der Transaktion (siehe Abbildung 2.1).

Phase 1 Der Koordinator befiehlt allen Teilnehmer der Transaktion, den Commit vorzubereiten (prepare Commit) und erhält von den Teilnehmer eine Rückmeldung, ob sie bereit sind, die Transaktion abzuschließen und die Änderungen durchzuführen (Commit) oder nicht. Falls der Koordinator von allen Teilnehmern eine positive Rückmeldung erhalten hat, entscheidet er sich für ein „Commit“. Sonst entscheidet sich der Koordinator mit „Abort“ für das Beenden der Transaktion ohne Veränderung des Systemzustands.

Phase 2 Der Koordinator teilt seine Entscheidung allen Teilnehmern mit. Diese terminieren die Transaktion entsprechend der Entscheidung des Koordinators und senden ihm einen Acknowledgement.

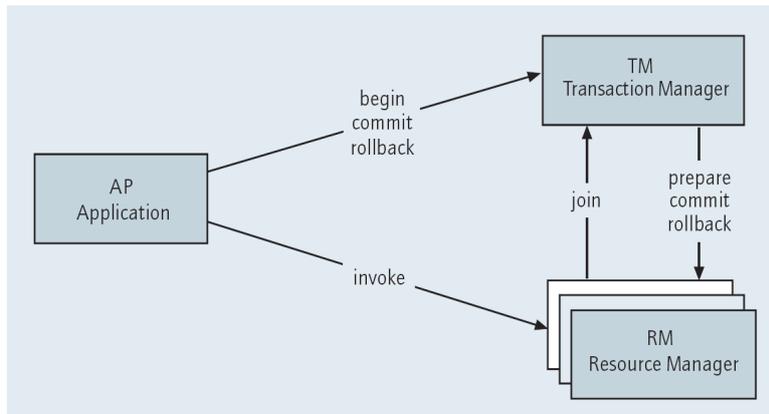


Abbildung 2.1.: Ablauf einer XA-Transaktion [6]

2.3. Asynchrone Kommunikation: Message-basierte Kommunikation

Die Tatsache, dass Anwendungen heterogen sind und über die ganze Welt verteilt sein können, ruft einige Schwierigkeiten hervor. Zum einem ist die Kommunikation durch Netzwerke nicht immer zuverlässig und auch deutlich langsamer als lokale Aufrufe. Weitere Probleme werden dadurch verursacht, dass die interagierenden Systeme heterogen sind: Systeme, die verschiedene Programmiersprachen verwenden und auf verschiedenen Plattformen laufen, müssen miteinander kommunizieren können. Weiterhin ändern sich die einzelnen Anwendungen mit der Zeit, außerdem können bestimmte Systeme ausfallen und sollen dann durch andere ersetzt werden können. Um damit zurechtzukommen, dürfen die Systeme möglichst nur lose durch Anwendungsintegration gekoppelt werden. Zur Lösung dieser Problematik wurden unter anderem folgende Herangehensweisen entwickelt:

- File Transfer: Eine Anwendung schreibt in eine Datei und die andere Anwendung liest daraus. Es müssen Abkommen getroffen werden, die die Namensgebung

und Lokalität der Datei betreffen.

- Die gemeinsame Verwendung einer Datenbank kann als Generalisierung des ersten Verfahrens angesehen werden.
- Remote Procedure Invocation ist eine weitere Art der Koppelung von Anwendungen: eine der Anwendungen stellt ihre Funktionalität bereit, die dann als entfernte Prozedur von einer anderen Anwendung aufgerufen werden kann. Es handelt sich hier normalerweise um synchrone Kommunikation.
- Statt durch gegenseitige Prozeduraufrufe können die Anwendungen durch Nachrichtenübertragung (Messaging) verbunden werden (Queued Transaktion Processing, [4]). Dabei stellt eine Anwendung ihre Nachricht in einen gemeinsamen Kanal ein, die andere liest die Nachricht zu einem ihr passenden Zeitpunkt aus dem Kanal. Die Anwendungen müssen ein Abkommen über den Kanal und das Format der Nachrichten treffen.

Nachrichtenbasierte Kommunikationsformen (Messaging), deren Funktionsprinzip in ihrer Asynchronität und der daraus resultierenden losen Kopplung von Applikationen liegt, sind hierzu am besten geeignet.

Das Prinzip des Queued Transaktion Processing liegt darin, dass jeder der Teilnehmer (Client und Server) eine persistente Nachrichtenwarteschlange besitzt, aus denen er die Nachrichten liest. Anstatt eine Anfrage direkt an den Server zu schicken, schickt der Client sie an die entsprechende Warteschlange. Der Server seinerseits schreibt die Antwort in die Eingangswarteschlange des Clients. Hier ist zu bemerken, dass eine Warteschlange eine transaktionale Ressource ist: das Lesen und Schreiben einer Queue wird als einzelne Transaktion durchgeführt. Ein Request an den Server besteht daher statt aus einer Transaktion aus drei Transaktionen: Der Client schreibt die Anfrage in die Request-Queue des Servers (Transaktion 1), der Server liest aus seiner Request-Queue und schreibt eine Antwort in die Antwort-Queue des Clients (Transaktion 2), der Client liest aus seiner Antwort-Queue (Transaktion 3). Hier entsteht also ein zusätzlicher Aufwand, mit dem man die lose Kopplung des Systems bezahlt.

Die Vorteile des *Messaging* liegen auf der Hand: Das Paradigma erlaubt eine bessere Kapselung als eine Datenbank, und sie ist zuverlässiger als Remote Procedure Invocation. Der Client kann eine Anfrage an den Server senden, auch wenn der Server zusammengebrochen ist und in dem Moment keine Anfragen bearbeiten kann. Der Server kann wiederum einem Client selbst dann antworten, wenn der Client in dem Moment nicht verfügbar ist. Auch Kommunikationsfehler führen nicht zum Verlust von Antworten. Bei Queued-Transaction-Prozessing ist es auch bedeutend einfacher, die Last zwischen mehreren Server zu balancieren: es können mehrere Server Nachrichten aus der selben Queue empfangen, der Server liest eine Anfrage, sobald er dazu in der Lage ist. So wird verhindert, dass einige Server unterbelastet und andere überlastet sind. Weiterhin kann man die Nachrichtenwarteschlangen für prioritäten-basiertes Scheduling verwenden: jede Anfrage kann eine Priorität zugewiesen bekommen und dementsprechend verarbeitet werden [4]. Diese Vorteile erkauft man sich mit höheren Verwaltungskosten und zusätzlichem Aufwand, der mit den transaktional gesicherten Operationen „enqueue“ und „dequeue“ einhergeht.

3. Stratifizierte Transaktionen: Modellbeschreibung.

Im folgenden Abschnitt wird das in der vorliegenden Diplomarbeit aufgestellte Modell für stratifizierte Transaktionen ([1]) beschrieben. Das Modell beschränkt die Sicht auf Transaktionen als Träger gewisser Eigenschaften. Dabei wird eine Transaktion als ein Ganzes gesehen – der interne Aufbau der Transaktion als Menge von Aktionen, die Abhängigkeiten untereinander vorweisen usw., wird nicht betrachtet. Kenntnisse über Eigenschaften von Transaktionen können z.B. aus vorher gesammelten Statistiken des Systems oder durch Abschätzungen (*worst-case* oder *average-case*) ermittelt werden. Zusätzlich zu den Eigenschaften von Transaktionen wird eine Abhängigkeitsstruktur beschrieben. Die Transaktionseigenschaften (Properties) bilden zusammen mit dem Abhängigkeitsgraph eine Sicht auf das System, anhand derer dann die Aufteilung auf Strata durchgeführt wird.

Die Abschnitte 3.4.2 und 3.4.3 befassen sich mit der Definition des durch Stratifizierung entstandenen Stratifikationsgraphen sowie mit der Beschreibung seiner Eigenschaften. Die für die Suche der optimalen Stratifikation benötigte Bewertung der Stratifikation wird im Abschnitt 3.5 behandelt.

3.1. Annahmen

3.1.1. Ausführungsvoraussetzungen

Jedem Stratum wird eine persistente Nachrichtenwarteschlange zugewiesen. In dieser werden alle eingehenden Nachrichten aufgenommen. Es wird angenommen, dass die Ausführung der Transaktionen eines Stratums erst dann gestartet werden kann, wenn

alle Nachrichten, die aufgrund von Abhängigkeiten von anderen Strata erwartet werden, angekommen sind (das entspricht einer Join-Regel mit einer Und-Verknüpfung). Für den Fall, dass eine erweiterte Semantik erwünscht ist, kann das Modell entsprechend erweitert werden – z.B. durch Verwendung einer Oder-Join-Regel (Einsatz von einem Aggregator/Smart Proxy Muster). Weiterhin wird angenommen, dass die Sperren auf Ressourcen vor dem Anfang der Ausführung des Stratums erworben werden müssen. Das ist möglich, wenn die Konkurrenz mehrerer Transaktionen um dieselben Ressourcen als kausale Abhängigkeit dargestellt werden. Die Freigabe der Sperre auf der Ressource kann dann mit einer entsprechenden Nachricht an den Konkurrenten im nächsten Stratum signalisiert werden.

3.1.2. Fehlerbehandlung

Es wird davon ausgegangen, dass bereits die Middleware inkorrekte Nachrichten oder Nachrichten ohne Empfänger auffängt und diese Situationen selbständig behandelt. Beispielsweise können Nachrichten, welche nicht zugestellt werden können oder die ungültig geworden sind, in *Dead-letter queues* eingetragen werden. Auf die Mechanismen, wie eine Middleware dies gewährleistet, wird nicht weiter eingegangen.

Andere Fehler und Ausfälle (Serverausfall, Wegefallen der Verbindung zu einem Server usw.) werden in dem Modell als Wahrscheinlichkeit des Aborts einer Transaktion dargestellt.

3.2. Eigenschaften von Ressourcen

Die Transaktion ist eine Menge von Operationen, die gewisse Ressourcen manipuliert. Wie bereits im Abschnitt 2.1 erwähnt, ist der Begriff *Ressource* eine Verallgemeinerung für so Verschiedenes wie Datenbanken, Nachrichtensysteme, EJB-Container usw. Die Eigenschaften der Transaktionen sind eng mit den Eigenschaften der verwendeten Ressourcen verbunden. Daher wird hier zuerst die Sicht auf eine Ressource beschrieben.

Sei $Res = \{r_1, r_2, \dots, r_m\}$ die Menge der im System vorhandenen Ressourcen. Folgende Eigenschaften der Ressourcen sind in der vorliegenden Arbeit von Bedeutung:

- Die *Art* der Ressource, z. B. Nachrichtenqueues im Gegensatz zu Datenbanken, oder verschiedene Datenbankmodelle. Die Synchronisation von Ressourcen verschiedener Art kann höhere Kosten verursachen als die Synchronisation gleichartiger Ressourcen.
- Die möglichen *Lokation(-en)* von Ressourcen. Es ist auch günstiger, Ressourcen zu synchronisieren, die von derselben Maschine bereitgestellt werden. Wenn es eine Menge L möglicher Lokationen gibt und \mathcal{P} deren Potenzmenge ohne die leere Menge ist, können die tatsächlichen Lokationen von Ressourcen durch eine Abbildung $\mathcal{L} : Res \rightarrow \mathcal{P}(L)$ angegeben werden.

Weitere mögliche Eigenschaften von Ressourcen sind *Zuverlässigkeit* und *Auslastung*. Diese Eigenschaften können verwendet werden, um daraus die Abortwahrscheinlichkeit einer Transaktion zu ermitteln.

3.3. Eigenschaften von Transaktionen

Der Begriff der Transaktion soll im Folgenden die im Abschnitt 2.1 eingeführte Semantik besitzen. Die Transaktionen werden durch eine Menge von Eigenschaften modelliert. Dieser Abschnitt liefert eine Auflistung möglicher Transaktionseigenschaften. Es handelt sich dabei um keine erschöpfende Auflistung.

Sei $Res = \{r_1, r_2, \dots, r_m\}$ die Menge der in dem System vorhandenen Ressourcen. Sei $\mathcal{T} = \{t_1, t_2, \dots, t_n\}$ die Menge der in System durchzuführenden Transaktionen, und sei weiterhin $\pi = \{p_1, p_2, \dots, p_k\}$ die Menge von k möglichen Eigenschaften von Transaktionen.

Die Eigenschaften von Transaktionen können verschiedener Art sein, aber sie können durch eine Menge F spezieller Funktionen dargestellt werden. Im allgemeinen sind die Eigenschaften von Transaktionen durch Abbildungen wie folgt definiert:

$$p_i : \mathcal{T} \rightarrow F$$

Insbesondere werden folgende Eigenschaften von Transaktionen betrachtet und näher analysiert:

- *Lokation(-en)*. Sei L die Menge möglicher Lokationen, und $\mathcal{P}(L)$ sei ihre Potenzmenge. Dann gibt die *Lokations-Eigenschaft* \mathcal{L} einer Transaktion die Lokation(-en) der Server an, auf denen die Transaktion gestartet werden kann.

$$\mathcal{L} : \mathcal{T} \rightarrow \mathcal{P}(L)$$

- *Kosten*, die mit der Invokation der Transaktion verbunden sind.

$$\mathcal{I} : \mathcal{T} \rightarrow \mathbb{R}^+$$

- *Zeitpunkt der Verfügbarkeit*: wann die Transaktion frühestens gestartet werden kann.

$$\tau_{avail} : \mathcal{T} \rightarrow \mathbb{R}^+$$

- *Zeitpunkt der Fälligkeit*: wann die Transaktion spätestens abgeschlossen sein muss.

$$\tau_{maturity} : \mathcal{T} \rightarrow \mathbb{R}^+$$

- *Zeitkosten*. Die Funktion φ besagt, wie teuer es wird, wenn die Transaktion erst zur Zeit τ fertig ist. Jede der Transaktionen hat eine eigene Kostenfunktion.

$$\varphi : \mathcal{T} \rightarrow C \quad \text{wobei} \quad C = \{f : \mathbb{R}^+ \rightarrow \mathbb{R}^+\}$$

Dann beschreibt $\varphi(t)(\tau) = f_t(\tau)$ die Kosten, die bei der Transaktion t anfallen, falls sie zum Zeitpunkt τ fertiggestellt wird. Die Funktion f_t kann je nach der Priorität (Wichtigkeit oder Dringlichkeit) der Transaktion als polynomielle Funktion, exponentielle Funktion oder z.B. als eine abschnittsweise zusammengesetzte Funktion ausgeprägt sein (eine der Möglichkeiten wäre z.B. eine abschnittsweise definierte Funktionen, die vor einem Zeitpunkt durch eine konstante Funktion definiert ist und sich ab dem Zeitpunkt der Fälligkeit $\tau_{maturity}(t)$ wie eine exponentielle Funktion verhält).

- *Erwartete Ausführungsdauer der Transaktion*.

$$\tau_{ExecAvr} : \mathcal{T} \rightarrow \mathbb{R}^+$$

- *Maximale Ausführungszeit der Transaktion.*

$$\tau_{ExecMax} : \mathcal{T} \rightarrow \mathbb{R}^+$$

- *Recovery-Kosten der Transaktion.*

$$Recovery : \mathcal{T} \rightarrow \mathbb{R}^+$$

Die Transaktionen sollen so auf die Strata verteilt werden, dass die Strata mit Transaktionen mit höheren Recovery-Kosten mit einer höheren Wahrscheinlichkeit schon beim ersten Ausführungsversuch erfolgreich beendet werden. Dies wäre z.B. nicht der Fall, wenn eine Transaktion mit hohen Recovery-Kosten in dem selben Stratum mit einer Transaktion ist, die eine hohe Abort-Wahrscheinlichkeit hat.

- *Kommunikation mit gewissen Teilmengen der Transaktionen*

$$\zeta : \mathcal{T} \rightarrow \mathcal{P}(\mathcal{T})$$

Die Funktion $\zeta(t)$ könnte z.B. auf Cluster in dem Abhängigkeitsgraph hinweisen. Unter Umständen könnte man dann je Cluster ein Stratum bilden.

- *Zugriff auf eine bestimmte Art von Ressourcen.* Als eine Art versteht man z.B. ein Datenbankmodell oder ein spezielles Produkt (z. B. DB2, Oracle etc.). Die Ressourcenart kann Auswirkungen auf den Aufwand des 2-Phase-Commit-Protokolls haben, denn die Synchronisation verschiedenartiger Ressourcen ist teurer als Synchronisation von Ressourcen gleicher Art. Sei $KindRes$ die Menge der Ressourcenarten im System und $\mathcal{P}(KindRes)$ sei ihre Potenzmenge. Die Eigenschaft \mathcal{A}_{kind} einer Transaktion, auf eine Menge von Ressourcenarten zuzugreifen, wird beschrieben durch

$$\mathcal{A}_{kind} : \mathcal{T} \rightarrow \mathcal{P}(KindRes)$$

- *Menge der Ressourcen, auf die die Transaktion zugreift.* Sei $Res = \{r_1, r_2, \dots, r_q\}$ eine Menge zur Verfügung stehender Ressourcen. Die Eigenschaft \mathcal{A}_{res} , eine spezifische Menge von Ressourcen zu verwenden, wird beschrieben durch

$$\mathcal{A}_{res} : \mathcal{T} \rightarrow \mathcal{P}(Res)$$

Diese Eigenschaft ist von Belang, da die Manipulation jeder einzelnen Ressource bestimmte Konsequenzen für den Aufwand der Synchronisation nach sich zieht.

- *Wahrscheinlichkeit, dass die Transaktion schließlich mit **commit** abgeschlossen wird*

$$Prob_{Success} : \mathcal{T} \rightarrow [0, 1]$$

Diese Eigenschaft ist wichtig, wenn man den Zeitpunkt der Fertigstellung aller „Wurzelstrata“ verwendet, um die Antwortzeit des gesamten Gebildes anzugeben, das durch die Stratifizierung entsteht. Dabei muss sichergestellt sein, dass alle Transaktionen der nächsten Strata auch schließlich erfolgreich beendet werden. Daher müssen alle Transaktionen, für die es unsicher ist, ob sie schließlich mit **commit** abgeschlossen werden, einer der Wurzelstraten angehören.

- *Wahrscheinlichkeit eines **abort** bei einem einzelnen Durchlauf der Transaktion:*

$$Prob_{Abort} : \mathcal{T} \rightarrow [0, 1]$$

Diese Wahrscheinlichkeit kann zum Beispiel aus Erkenntnissen über die von der Transaktion verwendeten Ressourcen und über die Zuverlässigkeit der Ressourcen berechnet werden; sie sollte aber auch von der Länge der Transaktion (d. h. der Anzahl der enthaltenen Aktionen bzw. Schreibaktionen) abhängig sein.

- *Access Control.* Diese Transaktionseigenschaft beschreibt, ob für den Start der Transaktion eine Autorisierung durch den Benutzer nötig ist. Falls ja, müssen Verschlüsselungsverfahren eingesetzt werden, und durch die Ver- und Entschlüsselung entstehen Kosten. Die Autorisierung erfolgt durch Abstimmung mit einem zuständigen Server – damit sind Kosten verbunden, die auch von der Entfernung zum Server abhängen sollten. Zugangskontrolle kann weiter ausgebaut werden, indem die Art des Verschlüsselungsverfahrens abgefragt wird, mit welcher dann auch entsprechend verschiedene Kosten verbunden sind.

$$AC : \mathcal{T} \rightarrow \{0, 1\}$$

3.4. Stratifizierung

In diesem Abschnitt soll eine formale Beschreibung des Abhängigkeitsgraphes anhand seines Aufbaus eingeführt werden. Außerdem wird beschrieben, was unter einer gültigen Stratifikation verstanden wird.

3.4.1. Präzedenzrelation (Abhängigkeitsgraph)

Sei $\mathcal{T} = \{t_1, \dots, t_n\}$ eine Menge von Transaktionen. Ein *Abhängigkeitsgraph* G_a ist ein gerichteter zyklensfreier Graph $G_a = (V_a, E_a)$ ohne Schlingen und Mehrfachkanten. Die Knotenmenge V_a des Graphs entspricht der Menge von Transaktionen des Systems, und die Kantenmenge E_a des Graphs stellt die Abhängigkeiten zwischen Transaktionen dar:

$$G_a = (V_a, E_a), \text{ wobei } V_a \equiv \mathcal{T} = \{t_1, \dots, t_n\} \text{ und } E_a \subseteq V_a \times V_a$$

Die Kantenmenge wird wie folgt konstruiert:

Es stelle $t_i \rightarrow t_j$ eine Abhängigkeit der Transaktion t_j von der Transaktion t_i dar. Dann gilt

$$\forall t_i, t_j \in V_a : t_i \rightarrow t_j \Leftrightarrow e \in E \text{ mit } e = (t_i, t_j)$$

Der so definierte Graph drückt die partielle Ordnung auf den Transaktionen aus: Die Existenz der Kante $e = (t_i, t_j)$ bedeutet, dass die Transaktion t_i vor der Transaktion t_j ausgeführt werden muss. Die Gründe für das Vorhandensein von Abhängigkeiten können verschiedener Natur sein. Zum Beispiel schreibt die erste Transaktion eine Nachricht in die Nachrichtenqueue der anderen Transaktion, oder die erste Transaktion muss aus semantischen Gründen vor der anderen ausgeführt werden (so wird z. B. bei einer Kreditvergabe zuerst die Kreditwürdigkeit des Kunden geprüft, bevor der Kredit erteilt wird).

Der Abhängigkeitengraph ist zyklensfrei – ein „Richtungsgraph“ – da zyklische Abhängigkeiten nicht existieren dürfen. Andererseits kann der Abhängigkeitsgraph eventuell auch nicht zusammenhängend sein, und zwar wenn zwischen Untermengen der Transaktionen keine direkten oder indirekten Abhängigkeiten bestehen.

Zur Beschreibung der in der Arbeit verwendeten Mechanismen werden noch zusätzliche Hilfsoperatoren benötigt:

- **Vorgänger eines Transaktionsknotens** sind Elemente aus der Potenzmenge der Transaktionsknoten:

$$Vor : \mathcal{T} \rightarrow \mathcal{P}(\mathcal{T})$$

$$\forall t \in \mathcal{T} : Vor(t) = \{ t_i \in \mathcal{T} \mid \exists e = (t_i, t) \in E_a \}$$

$$0 \leq |Vor(t)| \leq |\mathcal{T}| - 1$$

- **Nachfolger eines Knotens** sind Elemente aus der Potenzmenge der Knoten:

$$Nach : \mathcal{T} \rightarrow \mathcal{P}(\mathcal{T})$$

$$\forall t \in \mathcal{T} : Nach(t) = \{ t_i \in \mathcal{T} \mid \exists e = (t, t_i) \in E_a \}$$

$$0 \leq |Nach(t)| \leq |\mathcal{T}| - 1$$

- Menge der „**Wurzeltransaktionen**“

$$w : G_a \rightarrow \mathcal{P}(\mathcal{T})$$

$$w(G_a) = \{ t_i \in \mathcal{T} \mid Vor(t_i) = \emptyset \}, \quad 0 < |w(G_a)| \leq |\mathcal{T}|$$

- Menge der „**Blättertransaktionen**“

$$b : G_a \rightarrow \mathcal{P}(\mathcal{T})$$

$$b(G_a) = \{ t_i \in \mathcal{T} \mid Nach(t_i) = \emptyset \}, \quad 0 < |b(G_a)| \leq |\mathcal{T}|$$

In Extremfall, wenn der Graph keine Kanten besitzt, sind alle seine Knoten Wurzeln und auch Blätter.

3.4.2. Stratifizierungsgraph

Die Idee der Stratifikation besteht darin, dass die Transaktionsmenge (d.h. die Menge der Knoten des Abhängigkeitsgraphes) in Teilmengen aufgeteilt wird, so dass die gebildeten Teilmengen disjunkt sind und ihre Vereinigung die Menge aller Transaktionen bildet. Mit anderen Worten, die Transaktionsmenge wird partitioniert. Eine jede solche Teilmenge von Transaktionsknoten wird **Stratum** genannt. Ein Stratum ist genau dann erfolgreich, falls jede der Transaktionen des Stratums erfolgreich abgeschlossen wurde. Technisch wird dies durch das in 2.2 beschriebene Zwei-Phasen-Commit-Protokoll realisiert.

Die Strata werden nun als Knoten eines Hypergraphes angesehen, der im Folgenden *Stratifikationsgraph* genannt wird. Zwischen je zwei Strata existiert eine Kante genau dann, wenn es mindestens ein Paar von Transaktionen gibt, so dass sich in jedem der beiden Strata eine der Transaktionen befindet und die beiden Transaktionen eine Kante in dem entsprechenden Abhängigkeitsgraph besitzen. Eine Kante in Stratifikationsgraph entspricht also einer Teilmenge der Kanten im Abhängigkeitsgraph. Es wird angenommen, dass jedem Stratum eine persistente *Nachrichtewarteschlange* zugewiesen ist, die alle eingehenden Nachrichten eines Stratums aufnimmt und verwaltet.

Formal wird das wie folgt ausgedrückt. Sei $\mathcal{T} = \{t_1, t_2, \dots, t_n\}$ die Menge aller im System vorhandenen Transaktionen, sei $G_a = (V_a, E_a)$ der Abhängigkeitsgraph der Transaktionen wie in Abschnitt 3.4.1 definiert.

Die Menge der Transaktionen wird auf die Teilmengen (Straten) $\mathcal{S} = \{s_1, s_2, \dots, s_p\}$ aufgeteilt, so dass folgende Bedingungen gelten:

1. $\forall i \in \{1, \dots, |\mathcal{S}|\} : s_i \in \mathcal{P}(\mathcal{T})$ mit $|s_i| \neq 0$
2. $\bigcup_i s_i = \mathcal{T}$
3. $\forall i, j \in \{1, \dots, |\mathcal{S}|\}, i \neq j : s_i \cap s_j = \emptyset$

Der Stratifizierung wird als eine Abbildung des vorgegebenen Abhängigkeitsgraphes

G_a auf Transaktionen in einen Stratifizierungsgraph G_S definiert:

$$\text{Stratification} : G_a \rightarrow G_S$$

Der Graph ist definiert durch $G_S = (V_S, E_S)$ mit $V_S \equiv \mathcal{S} = \{s_1, s_2, \dots, s_k\}$ als Menge der Knoten, welche die Strata repräsentieren¹, und mit der Kantenmenge E_S . Diese wird im Folgenden genauer definiert.

Die Kantenmenge E_S des Stratifikationsgraphs wird wie folgt definiert:

$E_S \subseteq V_S \times V_S$ mit

$$\tilde{e} = (s_i, s_j) \in E_S \Leftrightarrow \exists t_x \in s_i \exists t_y \in s_j : (t_x, t_y) \in E_a$$

(E_a ist die Kantenmenge des Abhängigkeitsgraphs).²

3.4.3. Die Eigenschaften des Stratifizierungsgraphs

- Für jedes Stratum wird durch die Struktur des Stratifikationsgraphs die Menge der **Vorgänger-** und **Nachfolgerstrata** vorgegeben:

$$\mathcal{V} : \mathcal{S} \rightarrow \mathcal{P}(\mathcal{S}) \quad , \quad \mathcal{N} : \mathcal{S} \rightarrow \mathcal{P}(\mathcal{S})$$

Die Anzahl der Vorgänger- und Nachfolgerknoten kann maximal $k - 1$ sein, bei insgesamt k Strata. Es gilt:

$$s_i \in \mathcal{V}(s) \Leftrightarrow \exists \tilde{e} \in E_S \text{ mit } \tilde{e} = (s_i, s)$$

$$s_j \in \mathcal{N}(s) \Leftrightarrow \exists \tilde{e} \in E_S \text{ mit } \tilde{e} = (s, s_j)$$

- Menge der Knoten, die keine Vorgänger haben („**Wurzelstrata**“):

$$\text{Wurzeln} : G_s \rightarrow \mathcal{P}(\mathcal{S})$$

$$\text{Wurzeln}(G_s) = \{s_i \in \mathcal{S} \mid \mathcal{V}(s_i) = \emptyset\}$$

¹Der Vollständigkeit halber könnte man noch eine Abbildung von dem Stratifikationsknoten auf das entsprechende Stratum einführen, aber dies wird hier Einfachheit halber nicht getan. Im folgenden wird die Menge der Stratifikationsknoten und die Menge der Strata als äquivalent behandelt $V_S \equiv \mathcal{S}$.

²Im den folgenden Abschnitten werden die Kanten des Abhängigkeitsgraph mit e bezeichnet und die Kanten des Stratifikationsgraphes mit \tilde{e}

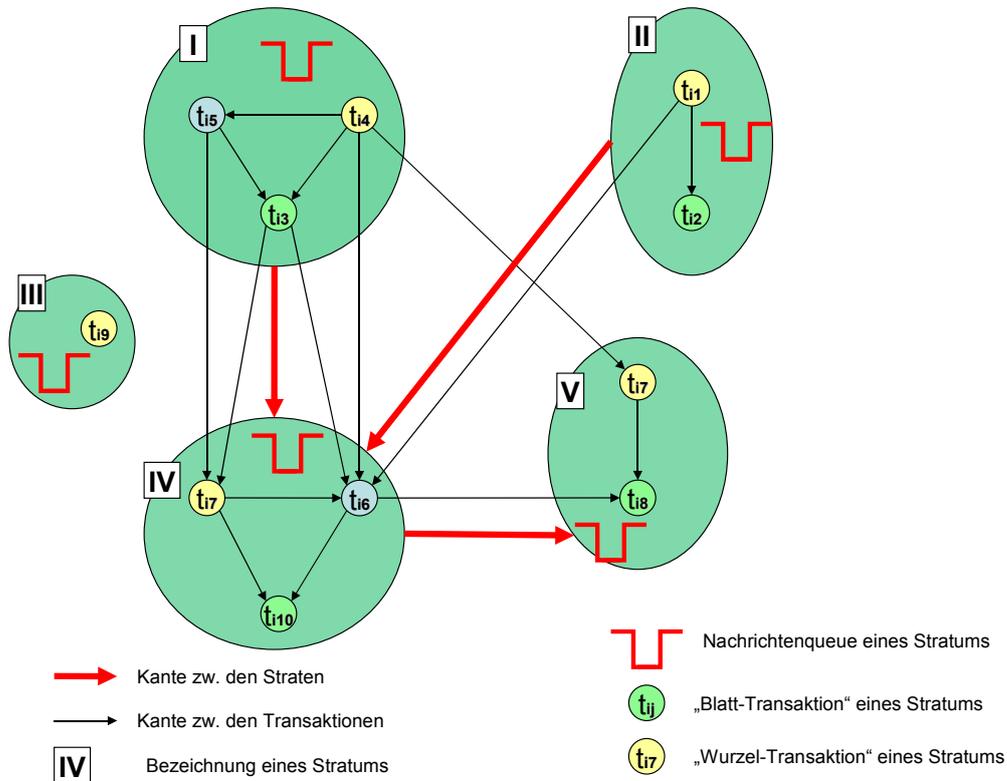


Abbildung 3.1.: Stratifikation

- Menge aller Knoten, die keine Nachfolger besitzen („Blätterstrata“):

$$\mathcal{Blaetter} : G_S \rightarrow \mathcal{P}(\mathcal{S})$$

$$\mathcal{Blaetter}(G_S) = \{s_i \in \mathcal{S} \mid \mathcal{N}(s_i) = \emptyset\}$$

- Die Menge der Wurzeltransaktionen eines Stratum ist eine Teilmenge der Transaktionen eines Stratum, die keine Vorgänger haben, die in dem selben Stratum liegen:

$$\mathcal{W}_T : \mathcal{S} \rightarrow \mathcal{P}(\mathcal{T}) , \quad \mathcal{W}_T(s) = \{t_i \mid \text{Vor}(t_i) \cap s = \emptyset\}$$

- Die Menge der Blättertransaktionen eines Stratum ist eine Teilmenge der Transaktionen eines Stratum, die keine Nachfolger haben, die in dem selben

Stratum liegen:

$$\mathcal{B}_{\mathcal{T}} : \mathcal{S} \rightarrow \mathcal{P}(\mathcal{T}) , \quad \mathcal{B}_{\mathcal{T}}(s) = \{t_i \mid \text{Nach}(t_i) \cap s = \emptyset\}$$

- Jeder Kante des Stratifizierungsgraphes wird eine (nicht leere) Teilmenge der Kanten des Abhängigkeitsgraphs zugewiesen:

$$\text{edgeSet} : E_S \rightarrow \mathcal{P}(E_a) \setminus \{\emptyset\}$$

Für diese Abbildung soll folgendes gelten:

$$\forall \tilde{e}_x, \tilde{e}_y \in E_S : \quad \text{edgeSet}(\tilde{e}_x) \cap \text{edgeSet}(\tilde{e}_y) = \emptyset$$

Es muss aber nicht gelten, dass alle Kanten des Abhängigkeitsgraphes mit den Kanten des Stratifikationskanten so überdeckt sind. Falls zwei Transaktionsknoten in demselben Stratum liegen und eine gemeinsame Transaktionskante besitzen, wird diese in keine der Stratifikationskanten „aufgenommen“:

$$\forall t_i, t_j \in s \ (s \in V_s) \quad \text{mit} \quad \exists e = (t_i, t_j) \in E_a \quad \Leftrightarrow \quad \forall \tilde{e} \in E_S : \quad e \notin \text{edgeSet}(\tilde{e})$$

- Für jede Kante des Stratifizierungsgraphes ist eine Gewichtsfunktion definiert, die der Anzahl der Nachrichten in der jeweiligen Queue entspricht:

$$\phi : E_S \rightarrow \mathbb{N}$$

$$\forall \tilde{e} \in E_S : \quad \phi(\tilde{e}) = |\text{edgeSet}(\tilde{e})|$$

- Frühestmöglicher Zeitpunkt für den Start der Abarbeitung eines Stratums:

$$\hat{\tau}_{start} : \mathcal{S} \rightarrow \mathbb{R}^+$$

Sei a der früheste Zeitpunkt, an dem alle Transaktionen des Stratums verfügbar sind, und sei b der früheste Zeitpunkt, an dem alle Vorgängerstrata abgearbeitet wurden ³:

³Das entspricht den Annahmen aus Abschnitt 3.1

$$a := \max\{\tau_{avail}(t_i) \mid t_i \in s\}$$

$$b := \max\{\hat{\tau}_{finish}(s_{vor}) \mid s_{vor} \in \mathcal{V}(s)\}$$

$$\hat{\tau}_{start}(s) = \begin{cases} a & \text{falls } s \in \mathcal{W}urzeln(G_S) \\ \max\{a, b\} & \text{sonst} \end{cases}$$

- Fertigstellungszeitpunkt des Stratums:

Der Zeitpunkt wird ausgehend von den Eigenschaften der in Stratum enthaltenen Transaktionen berechnet, und zwar aus deren Verfügbarkeitszeitpunkt τ_{avail} , ihrer Ausführungsdauer τ_{exec} und deren kausalen Abhängigkeiten.

$$\hat{\tau}_{finish} : \mathcal{S} \rightarrow \mathbb{R}^+$$

Sei $s := \{t_{i_1}, t_{i_2}, \dots, t_{i_n}\}$

Hilfsfunktion: Zeitpunkt der Fertigstellung der Transaktion t ⁴

$$\tau_{finish} : t \rightarrow \mathbb{R}^+$$

Für jede Transaktion t aus dem Stratum s wird der Fertigstellungszeitpunkt wie folgt beschrieben. Sei $Vor^s(t) := Vor(t) \cap s$ die Menge der Vorgänger der Transaktion, die in dem selben Stratum liegen. Dann ist

$$\tau_{finish}(t) = \begin{cases} \hat{\tau}_{start}(s) + \tau_{exec}(t) & \text{falls } Vor^s(t) = \emptyset \\ \max\{\tau_{finish}(t_i) \mid t_i \in Vor^s(t)\} + \tau_{exec}(t) & \text{sonst} \end{cases}$$

Der Fertigstellungszeitpunkt eines Stratums s , ($\forall s \in \mathcal{S}$), ist dann wie folgt definiert. Man betrachte die Menge der Blättertransaktionen $\mathcal{B}_{\mathcal{T}}(s)$ des Stratums. Der Fertigstellungszeitpunkt des Stratums wird als Maximum der Fertigstellungszeitpunkte seiner Blättertransaktionen definiert:

$$\hat{\tau}_{finish}(s) = \max\{\tau_{finish}(t_b) \mid t_b \in \mathcal{B}_{\mathcal{T}}(s)\}$$

⁴Dieser Wert wird im Kontext einer Stratifikation verstanden und wird daher hier und nicht im Abschnitt 3.3 erläutert.

3.5. Bewertung der Stratifikation

Die Bewertung einer Stratifikation kann anhand mehrerer Kriterien durchgeführt werden. Zu den Bewertungskriterien, die in den folgenden Abschnitten definiert werden, gehören der Nachrichtenaufwand, der mit dem 2-Phase-Commit-Protokoll verbunden ist; der Aufwand der Nachrichten, die auf die Queues der Straten geschrieben werden; der Grad der Nebenläufigkeit des Stratifizierungsgraphes; Antwort- und Ausführungszeit der Struktur, Invokations- und Recoverykosten sowie Kosten, die mit der Überschreitung des Fälligkeitszeitpunktes der jeweiligen Transaktionen verbunden sind. Dies ist keine erschöpfende Auflistung aller möglicher Bewertungskriterien. In den folgenden Abschnitten werden diese Bewertungskriterien detailliert beschrieben.

3.5.1. Nachrichtenaufwand des Zwei-Phasen-Commit-Protokolls

In diesem Unterabschnitt wird der Aufwand des Zwei-Phasen-Commit-Protokolls angegeben, gemessen an der Anzahl und den Kosten von Nachrichten.

Anzahl der benötigten 2PC-Nachrichten Die Anzahl der Nachrichten hängt vom Erwartungswert der Runden des 2-Phasen-Commit-Protokolls ab. Unter einer „Runde“ des 2-Phasen-Commit-Protokolls versteht man einen kompletten Satz von Nachrichten, die für einen Synchronisationsversuch notwendig sind: „Prepare Commit“, „Antworten“, „Commit/Abort“, „Acknowledgement“. Dabei wird eine Runde immer komplett abgeschlossen, sei sie erfolgreich oder nicht. Falls ein Abort stattgefunden hat, wird eine neue Runde in die Wege geleitet. Der Erwartungswert der Anzahl der benötigten Runden von 2PC-Protokoll hängt von der Anzahl der Transaktionen des Stratum s und deren Abort-Wahrscheinlichkeit ab: je mehr Transaktionen in einem Stratum sind, desto höher die Wahrscheinlichkeit, dass mehr „2PC-Runden“ benötigt werden, um es erfolgreich abzuschließen.

Sei $E(\text{RundenAnzahl}_s)$ der Erwartungswert der Anzahl der für Stratum s benötigten

Runden von 2PC.

$$\begin{aligned}
 \text{Prob}(\text{RundenAnzahl}_s = 1) &= \prod_{t_i \in s} \text{Prob}(t_i \text{ erfolgreich}) =: c \\
 \text{Prob}(\text{RundenAnzahl}_s = 2) &= (1 - c) \cdot c \\
 \text{Prob}(\text{RundenAnzahl}_s = m) &= (1 - c)^{m-1} \cdot c \\
 E(\text{RundenAnzahl}_s) &= \sum_j \text{Prob}(\text{RundenAnzahl}_s = j) \cdot j
 \end{aligned}$$

In jeder Runde ist die Wahrscheinlichkeit, dass die Transaktion t_i erfolgreich abgeschlossen werden kann, dieselbe. Die Wahrscheinlichkeit kann berechnet werden, falls es folgende Angaben gibt:

- die Zuverlässigkeit (Verfügbarkeit) der jeweiligen Ressourcen (evtl. abgeleitet aus den Ressourcenarten):
Zuverlässigkeit : $Res \rightarrow [0, 1]$
- evtl. Angaben über die Zuverlässigkeit der Kommunikationswege
- für jede Transaktion die Angaben über deren Zugriff auf bestimmte Ressourcen

Die Wahrscheinlichkeit kann dann wie folgt berechnet werden:

$$\text{Prob}(t_i \text{ erfolgreich}) = \prod_j \text{Zuverlässigkeit}(R_j)$$

Die Kosten der Durchführung des 2-Phasen-Commit-Protokolls hängen des weiteren auch von der Wahl des Koordinators ab sowie von der Entscheidung über den Zeitpunkt, wann ein Commit durchgeführt werden soll. Diese Größen werden aber in der vorliegendem Model nicht berücksichtigt.

Unter Verwendung des Erwartungswerts der Rundenanzahl ergibt sich folgender Nachrichtenaufwand für das 2PC-Protokoll:

$$\begin{aligned}
 f_{2PC} : \mathcal{S} &\rightarrow \mathbb{N} \quad , \quad \mathcal{F}_{2PC} : G_S \rightarrow \mathbb{N} \\
 f_{2PC}(s_x) &= 4 \cdot |s_x| \cdot E(\text{RundenAnzahl}_x) \quad , \quad \mathcal{F}_{2PC}(G_S) = \sum_i f_{2PC}(s_i)
 \end{aligned}$$

Kosten der 2PC-Nachrichten Die Kosten für die Nachrichten des 2PC-Protokolls hängen ab

- vom Nachrichtenaufwand des 2PC-Protokolls
- von der Lokation der jeweiligen Ressource-Managern
- von den Arten der Ressourcen, die miteinander synchronisiert werden sollen.

Die Kostenfunktion für Nachrichten des 2PC-Protokolls ist eine Funktion:

$$C_{2PC} : \mathcal{S} \rightarrow \mathbb{R}$$

$$C_{2PC} = f_{2PC}(s) \cdot (\mathcal{F}_{ArtenDerRessourcen}(s) + \mathcal{F}_{Lokationen}(s))$$

Dabei sind $\mathcal{F}_{ArtenDerRessourcen}(s)$ und $\mathcal{F}_{Lokationen}(s)$ wie folgt definiert:

$$\mathcal{F}_{Lokationen} : \mathcal{S} \rightarrow \mathbb{N} \quad , \quad \mathcal{F}_{Lokationen}(s) = \sum_{t_i \in \mathcal{S}} \left| \bigcup_{r \in A_{res}(t_i)} \mathcal{L}(r) \right|$$

$$\mathcal{F}_{ArtenDerRessourcen} : \mathcal{S} \rightarrow \mathbb{N} \quad , \quad \mathcal{F}_{ArtenDerRessourcen}(s) = \left| \bigcup_i \mathcal{A}_{kind}(t_i) \right|$$

3.5.2. Nachrichtenaufwand bei Queues

Jedem Stratum im Stratifizierungsgraph ist eine Nachrichtenqueue zugeordnet, die die Eingangsnachrichten des Stratums verwaltet. Über die Lokation der Queues kann während des Aufbaus der Strata entschieden werden. Falls nur eine gewisse Menge von Queues verfügbar ist, muss aus den vorhandenen Queues eine für das Stratum zuständige gewählt werden. Die Lokation der Queues kann aber auch von vornherein festgelegt werden.

Der Nachrichtenaufwand bei den Queues ist beschrieben durch:

$$QM : G_S \rightarrow \mathbb{N}$$

$$QM(G_S) = \sum_{e_i \in E_S} \phi(e_i)$$

Grad der Nebenläufigkeit

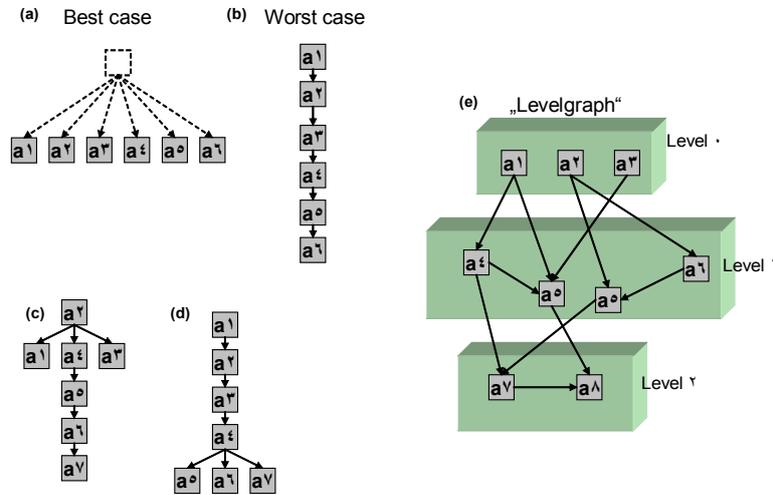


Abbildung 3.2.: Grad der Nebenläufigkeit beurteilt durch die Struktur des Graphes.

3.5.3. Grad der Nebenläufigkeit

Der Grad der Nebenläufigkeit eines Stratifikationsgraphes ist wie folgt definiert:

$$Parallelitaet : G_S \rightarrow \mathbb{R}$$

Bei der Betrachtung des Grads der Nebenläufigkeit sind die Nebenläufigkeit innerhalb der einzelnen Strata sowie die Nebenläufigkeit des gesamten Stratifizierungsbaumes von Interesse. Das Bewertungsprinzip von beiden ist gleich, da beides Graphstrukturen sind. Im folgendem wird von der Nebenläufigkeit im Sinne des gesamten Stratifikationsgraphes gesprochen; bei der Nebenläufigkeit der jeweiligen Strata kann man dasselbe Vorgehen verwenden. Folgende Möglichkeiten für die Bestimmung des Grads der Nebenläufigkeit sind denkbar.

Idee 1: Betrachtung der Graphstruktur.

Der Stratifizierungsgraph wird als ein „Levelgraph“ dargestellt, d.h. alle Nachfolgerstrata eines Stratums befinden sich auf einem gemeinsamen Level. Eine gute Nebenläufigkeit kann erreicht werden, wenn der Baum sehr breit ist (d.h. die Knoten

haben einen hohen Grad); die schlechteste Nebenläufigkeit hat ein degenerierter Baum, dessen Knoten alle maximal den Grad 1 haben. Für jedes Stratum innerhalb des Stratifizierungsgraphs gilt also, dass die Nebenläufigkeit um so besser ist, je höher der Level ist, an dem die Verzweigung in das Stratum stattfindet (siehe dazu die Abbildung 3.2).

Idee 2: Berechnung der durch parallele Ausführung gesparten Zeit.

Sei \mathcal{W} die Menge der Wurzelstrata (wie in Abschnitt 3.4.3 beschrieben).

Sei $Running$ die Menge der laufenden Strata (derer, die gerade ausgeführt werden).

Die aufgesparte Zeit wird wie folgt berechnet:

```

 $\tau_{saved} := 0,$ 
 $\tau_{waiting} := 0$ 
 $\tau_{stepStart} := 0$ 
Initialisierung:  $Running := \text{Wurzeln}(G_S)$ 
WHILE ( $|Running| > 0$ )
  Betrachte  $s_i \in Running$ , so dass  $\forall j : \hat{\tau}_{finish}(s_i) < \hat{\tau}_{finish}(s_j)$ 
   $\forall j$ :
    
$$\tau_{waiting}(s_j) := \begin{cases} 0 & \text{falls } \hat{\tau}_{start}(s_j) < \tau_{stepStart} \\ \hat{\tau}_{start}(s_j) - \tau_{stepStart} & \text{falls } \hat{\tau}_{start}(s_j) < \hat{\tau}_{finish}(s_i) \\ \hat{\tau}_{finish}(s_i) - \tau_{stepStart} & \text{sonst} \end{cases}$$

 $\tau_{waiting} := \sum_j \tau_{waiting}(s_j)$ 
 $\tau_{saved} := \tau_{saved} + \hat{\tau}_{finish}(s_i) \cdot (|Running| - 1) - \tau_{waiting}$ 
 $Running := Running \cup \mathcal{N}(s_i)/s_i$ 
END WHILE

```

Man könnte diese beide Ideen auch kombinieren. Eine der Möglichkeiten wäre zum Beispiel einen Zeitintervall vorzugeben. Die aufgesparte Zeit ist desto „wertvoller“, je früher die Aufspaltung stattgefunden hat: einem früheren Zeitintervall wird also ein höherer Wert zugewiesen. Je später die Zeit eingespart wurde, desto weniger „wertvoll“ ist die Einsparung.

3.5.4. Antwortzeit

Sei G_S der betrachtete Stratifikationsgraph. Die Menge seiner Wurzelstrata ist $Wurzeln(G_S)$. Die Antwortzeit des Stratifikationsgraphs wird durch die Ausführungszeit der Wurzelstrata definiert.

$$AnswerTime : G_S \rightarrow \mathbb{R}^+$$
$$AnswerTime(G_S) = \max\{\tau_{finish}(s_i) \mid s_i \in Wurzeln(G_S)\}$$

3.5.5. Ausführungszeit

Sie G_S der betrachtete Stratifikationsgraph. Die Menge seiner Blätter-Strata ist $\mathcal{B}(G_S)$. Die Ausführungszeit des Stratifikationsgraphs wird wie folgt definiert.

$$ExecTime : G_S \rightarrow \mathbb{R}^+$$
$$ExecTime(G_S) = \max\{\tau_{finish}(s_i) \mid s_i \in \mathcal{B}(G_S)\}$$

3.5.6. Kosten der Invokationen

Die Invokationskosten werden pro Stratum berechnet und für den gesamten Stratifikationsgraph akkumuliert.

$$ic : \mathcal{S} \rightarrow \mathbb{R}^+, ic(s_i) = E(RundenAnzahl_{s_i}) \cdot \sum_{t_j \in s_i} \mathcal{I}(t_j)$$
$$\mathcal{IC} : G_S \rightarrow \mathbb{R}^+, \mathcal{IC}(G_S) = \sum_{s_j \in G_S} ic(s_j)$$

3.5.7. Kosten der Recovery

Die Wiederherstellungskosten werden pro Stratum berechnet und für den gesamten Stratifikationsgraph akkumuliert.

$$rc : \mathcal{S} \rightarrow \mathbb{R}^+, rc(s_i) = (E(RundenAnzahl_{s_i}) - 1) \cdot \sum_{t_j \in s_i} Recovery(t_j)$$
$$\mathcal{RC} : G_S \rightarrow \mathbb{R}^+, \mathcal{RC}(G_S) = \sum_{s_j \in G_S} rc(s_j)$$

3.5.8. Zusätzliche Zeitkosten (TimeCosts)

Hier werden die Kosten berechnet, die durch den Zeitpunkt der Fertigstellung der Transaktionen mitverursacht werden. Falls bei einer Transaktion der Fertigstellungszeitpunkt nach dem Fälligkeitszeitpunkt liegt, können zusätzliche Kosten entstehen.

$$TC : \mathcal{S} \rightarrow \mathbb{R}^+ , \quad TC(s) = \sum_{t_i \in s} \varphi(t_i)(\hat{t}_{finish}(s))$$
$$TimeCosts : G_S \rightarrow \mathbb{R}^+ , \quad TimeCosts(G_S) = \sum_{s_i \in G_S} TC(s_i)$$

3.5.9. Die gesamte Bewertung

Die gesamte Bewertung einer Stratifizierung wird aus den bisher beschriebenen Bewertungskriterien zusammengesetzt. Die einzelnen Kriterien werden dabei jeweils gewichtet, also mit einer Priorität versehen.

Seien $\mathcal{E} = \{\mathcal{E}_1, \mathcal{E}_2, \dots, \mathcal{E}_k\}$ die Menge der Bewertungskriterien.

Seien die Prioritäten der Kriterien wie folgt definiert: $prio : \mathcal{E} \rightarrow \mathbb{R}$.

Dann ist die gesamte Bewertung wie folgt definiert:

$$\mathcal{F}(G_S) = \sum_i prio(\mathcal{E}_i) \cdot \mathcal{E}_i(G_S)$$

Es ist anzumerken, dass zwischen manchen der beschriebenen Bewertungskriterien ein Konflikt besteht. Eine optimale Konstellation bezüglich eines Kriteriums kann die schlechteste bezüglich eines anderen sein. Beispielsweise ist eine Lösung, bei der möglichst jede Transaktion in einem eigenen Stratum liegt, optimal bezüglich der Invokationskosten und der Recoverykosten. Dem entgegengesetzt ist eine Lösung um so besser bezüglich anderer Kriterien wie der Anzahl der benötigten Nachrichtenqueues, je geringer die Anzahl der Strata ist. Daher muss die Vergabe von Prioritäten wohlüberlegt sein.

3.6. Azyklität des Stratifizierungsgraphs

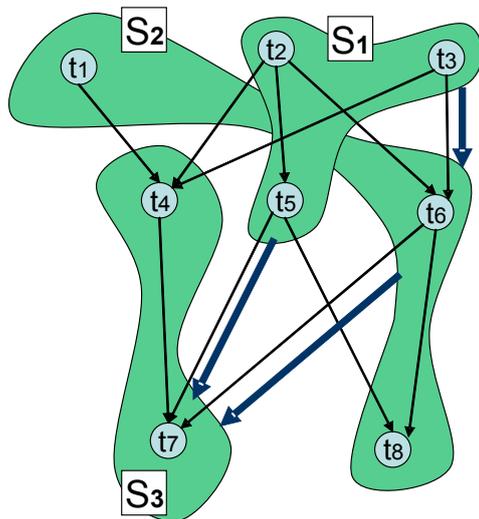
Die Stratifizierung einer Transaktionsmenge bestimmt den Ablauf der Ausführung der Transaktionsmenge. Der vorgegebene Abhängigkeitsgraph der Transaktionen

beinhaltet keine Zyklen: die Abhängigkeiten zwischen den Transaktionen sind eindeutig. Falls der durch Stratifizierung entstandene Stratifikationsgraph einen Zyklus enthält, wird es zu einem Deadlock bei der Ausführung kommen, was natürlich nicht erwünscht ist. Daher soll jede Stratifikation einen zyklensfreien Graph darstellen. Um bei der Konstruktion von Graphen die Azyklität zu wahren, können – abhängig von der Dichte des Abhängigkeitsgraphs (d. h. abhängig von der Anzahl seiner Kanten) – verschiedene Ansätze eingesetzt werden. Diese Ansätze werden genauer im Kapitel 5 beschrieben.

3.7. Beispiele für Stratifikation

Die Abbildung 3.3 liefert eine beispielhafte Stratifikation einer Transaktionsmenge mit einem vorgegebenen Abhängigkeitsgraph.

In der Abbildung 3.4 werden zwei gegensätzliche Stratifizierungen für einen vorgegebenen Abhängigkeitsgraph dargestellt.



$$G_s = (V_s, E_s)$$

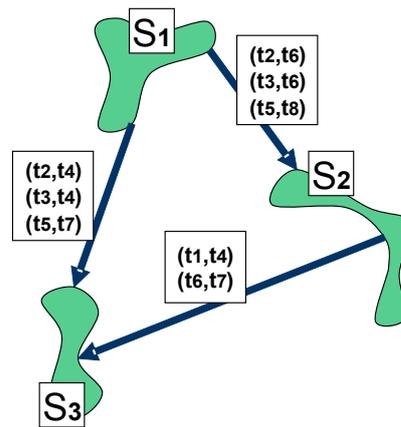
$$V_s = \{S_1, S_2, S_3\},$$

$$E_s = \{(S_1, S_2), (S_1, S_3), (S_2, S_3)\}$$

$$\text{edgeSet}((S_1, S_2)) = \{(t_2, t_6), (t_3, t_6), (t_5, t_8)\}$$

$$\text{edgeSet}((S_1, S_3)) = \{(t_2, t_4), (t_3, t_4), (t_5, t_7)\}$$

$$\text{edgeSet}((S_2, S_3)) = \{(t_1, t_4), (t_6, t_7)\}$$



$$G_a = (V_a, E_a)$$

$$V_a = \{t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8\}$$

$$E_a = \{(t_1, t_4), (t_2, t_4), (t_2, t_5), (t_2, t_6), (t_3, t_4), (t_3, t_6), (t_4, t_7), (t_5, t_7), (t_6, t_7), (t_6, t_8)\}$$

Abbildung 3.3.: Beispiel 1. Rechts die gemeinsame Sicht auf den Abhängigkeitsgraph und den Stratifikationsgraph. Links die Sicht auf den Stratifikationsgraph, mit seinen Kantenbeschriftungen.

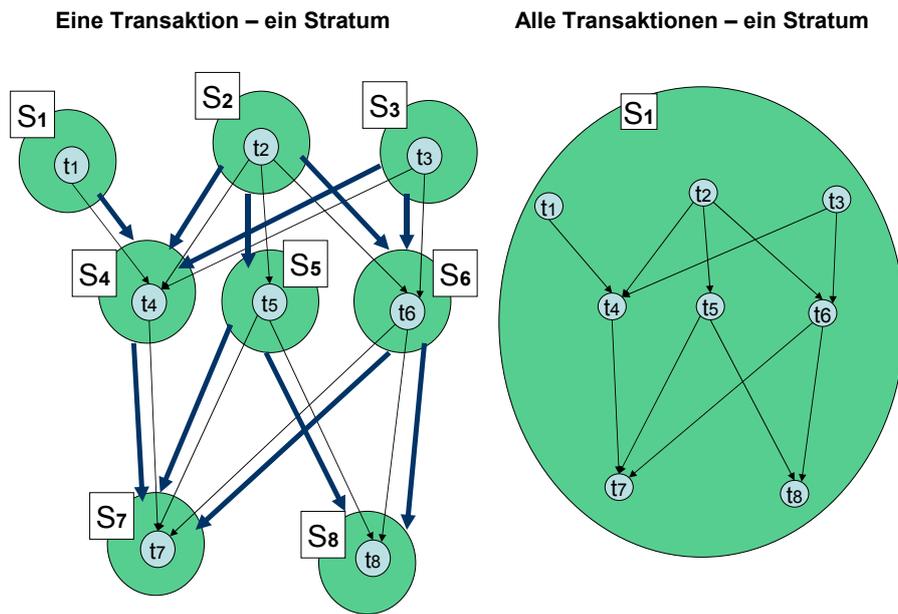


Abbildung 3.4.: Stratifikation

Teil II.

Optimale Stratifikation

4. Optimierungsverfahren: Grundlagen

„Genialität der Natur: selbst einfachste rudimentäre Nachahmungen und Modelle der Prozesse des Lebens sind bereits so mächtig, dass man mit ihnen die erstaunlichsten Resultate erzielen kann“.
[12]

Da das Stratifikationsproblem ein Optimierungsproblem ist – es gehört zur Klasse der kombinatorischen Optimierungsprobleme –, werden in diesem Kapitel die kombinatorischen Optimierungsprobleme und die damit verbundenen Lösungsverfahren eingeführt.

Nachdem der Begriff des Optimierungsproblems definiert worden ist, werden in Abschnitt 4.2 die Klassen der Optimierungsverfahren beschrieben, die für das Lösen eines kombinatorischen Optimierungsproblems eingesetzt werden können. Wichtige Bestandteile der Lösungsdiskussion eines Optimierungsproblems werden erläutert: der Suchraum, die Bewertungsfunktion von Elementen des Suchraums, sowie die Struktur der Gütelandschaft (Vergleich globaler und lokaler Optima) und die Nachbarschaft von Elementen im Suchraum.

Danach werden detailliert Verfahren der Lokale Suche (Hillclimbing und Simulated Annealing) und populationsbasierte Verfahren der Evolutionssimulation in den Abschnitten 4.3-4.4 diskutiert. Im Kapitel 5 sind Erläuterungen dazu enthalten, wie diese Verfahren verwendet wurden, um eine optimalen Stratifikation zu finden.

4.1. Das Optimierungsproblem: Begriffsbildung

Als *Optimierung* wird der Prozess der Modifizierung eines Systems beschrieben, die mit dem Ziel durchgeführt wird, bestimmte Aspekte des Systems effizienter (z. B. bezüglich Kostenaufwand, Zeitaufwand usw.) zu machen. Optimierung ist in vielen Bereichen von Bedeutung, z. B. in der Industrie, wo nach der optimalen Anordnung von Produktionsschritten gesucht wird, aber auch in der Forschung und der Wirtschaft.

Bezüglich des Zeitpunkts, an dem Optimierung einsetzt, unterscheidet man zwischen **statischer** und **dynamischer** Optimierung. Die statische Optimierung des Systems erfolgt während seiner Modellierungsphase – also vor seiner Auslieferung. Eine nachträgliche Anpassung des Systems zur Laufzeit ist dann nicht vorgesehen. Die dynamische Optimierung dagegen erfolgt während des Betriebs des Systems.

Ein Ansatz zur Lösung von Optimierungsproblemen ist die Suche in einem Elementraum, der von dem zu lösenden Problem definiert wird. Gesucht wird nach dem Element, welches bezüglich bestimmter Kriterien das beste ist.

In diesem Abschnitt wird eine formale Definition des eben Gesagten eingeführt.

Definition Ein *Optimierungsproblem* (Ω, f, \succeq) wird durch einen Suchraum Ω , eine *Bewertungsfunktion* $f: \Omega \rightarrow \mathbb{R}$, die jedem *Lösungskandidaten* einen Güterwert zuweist, sowie eine Vergleichsoperation $\succeq \in \{\leq, \geq\}$ definiert.

Dann ist die Menge der **globalen Optima** $\chi \subseteq \Omega$ definiert als

$$\chi = \{x \in \Omega \mid \forall x' \in \Omega \quad f(x) \succeq f(x')\}.$$

Schwierigkeiten beim Lösen von Optimierungsproblemen können aufgrund der Größe des Suchraums oder der zu berücksichtigenden Randbedingungen entstehen. Auch das Aufstellen einer Bewertungsfunktion ist nicht immer trivial.

Eine Unterklasse der Optimierungsprobleme sind die **kombinatorischen Optimierungsprobleme**. Kombinatorische Probleme zeichnen sich dadurch aus, dass die Anzahl der möglichen Lösungen zwar endlich ist, aber schnell mit der Problemgröße

wächst (in der Regel exponentiell). Zu den bekanntesten kombinatorischen Optimierungsproblemen gehören unter anderem das Problem des Handlungsreisenden (TSP), das Rucksackproblem und das Graphpartitionierungsproblem. Die Lösbarkeit dieser Probleme wird in [7] behandelt. Das Problem der Suche nach einer optimalen Stratifizierung gehört zur Klasse der kombinatorischen Optimierungsprobleme. Daher wird in den nächsten Abschnitten die Aufmerksamkeit den Verfahren gewidmet, die kombinatorische Optimierungsprobleme behandeln.

4.2. Optimierungsverfahren

Zur Lösung kombinatorischer Optimierungsprobleme führt ein Optimierungsverfahren auf der Menge aller gültigen Lösungskandidaten (dem *Suchraum*) eine Suche nach denjenigen Elementen durch¹, die nach einer vom Problem bestimmten *Bewertung* optimal sind. Das Verfahren beginnt mit einer initialen Menge von Elementen (die unter Umständen auch nur aus einem Element besteht). In jedem Schritt untersucht das Verfahren eine auf eine bestimmte Art und Weise gewählte Teilmenge des Suchraums. Dies ist oft eine von den aktuellen Elementen abgeleitete Teilmenge des Suchraums; bei der lokalen Suche ist es die *Nachbarschaft* der vorhandenen Menge von Elementen. Ausgehend von dieser Teilmenge entscheidet das Verfahren nach einer bestimmten Strategie, welche der Elemente weiterhin berücksichtigt werden. Die Bewertungsfunktion unterstützt die Strategie und hat den Zweck, die Suche auf eine optimale Lösung hin auszurichten.

Die klassischen Optimierungsverfahren gehören zwei Klassen an: den deterministischen und den nichtdeterministischen Verfahren. Letztere kommen besonders beim Lösen von NP-vollständigen Problemen zum Einsatz, wo oft Heuristiken benutzt werden. Diese Begriffe und die Kriterien zur Beurteilung von heuristischen Optimierungsverfahren werden im Folgenden diskutiert.

Deterministisches und probabilistische Suchverfahren

Die **deterministischen Suchverfahren** haben in jedem Punkt des Suchraums ein

¹Ein Optimierungsverfahren wird daher auch oft als Suchverfahren bezeichnet.

eindeutiges deterministisches Verhalten: bei gleichem Startpunkt kommt die Suche in jedem Suchlauf stets zu denselben Ergebnissen. Bei einem deterministischen Suchverfahren gibt es immer genau einen Lösungsweg. Die Schwäche der deterministischen Optimierungsverfahren liegt darin begründet, dass die Verwendung bestimmter Strategien und Annahmen über die Lage der Optima zum Verhängnis werden können. Die gewählte Strategie kann so gestaltet sein, dass das Suchverfahren gehindert wird, ein globales Optimum zu finden. Es kann auch fatale Auswirkungen auf den Verlauf des Verfahrens haben, wenn die getroffenen Annahmen auch nur sehr wenig von der Realität abweichen. Außerdem kann es für manche Probleme unmöglich sein, deterministische Suchverfahren überhaupt anzuwenden. Deterministische Optimierungsverfahren erfordern meist, dass gewisse Voraussetzungen erfüllt sind, wie die Stetigkeit und die Differenzierbarkeit der Zielfunktion des Problems. Genau das ist aber bei vielen Optimierungsproblemen nicht der Fall.

Die **probabilistischen Verfahren** machen bei der Suche nach einem Optimum in großen Suchräumen vom Zufall Gebrauch. Bei demselben Ausgangspunkt kann die Suche mit einer gewissen Wahrscheinlichkeit mehrere verschiedene Lösungswege begehen. Da bei nichtdeterministischen Verfahren für die Auswahl des Lösungswegs keine Annahmen zugrunde gelegt werden, wird zwar nicht immer der beste Weg gewählt. Dennoch liegt genau hierin die Stärke der Verfahren, da, wie oben bereits erwähnt, die Annahmen ja auch falsch sein können und dann eine optimale Lösung garantiert verhindern. Zwar entscheidet bei probabilistischen Verfahren der Zufall über die Auswahl des Lösungsweges, aber die Suche ist dennoch nicht willkürlich, sondern gerichtet: die Wahrscheinlichkeit, dass ein bestimmter Weg ausgewählt wird, ist proportional zu der *Bewertung* der Elemente auf diesem Weg [12].

Heuristische Methoden

Der Begriff *Heuristik* (aus dem Griechischen: finden, entdecken) bildet den Gegensatz zur Verfahren, die einen vollständigen Durchlauf des Suchraums erfordern. Unter Heuristiken werden Verfahren verstanden, deren Strategie einerseits auf vorhandenem Wissen über das Problem (sowohl theoretischem Wissen als auch Erfahrungsbeobachtungen, aus denen „best practises“ oder Faustregeln abgeleitet wurden), andererseits auf Vermutungen über die Lage der Optima bestimmt wird. Während bei der

vollständigen Enumeration eventuell jeder Punkt des Suchraums geprüft werden muss, wird bei einem Heuristischem Verfahren nur ein Teil des Suchraums betrachtet.

Heuristische Verfahren sind oft iterativ, und sie verfolgen das Ziel, in einer akzeptablen Zeit eine oder mehrere gute Lösungen für eine vorgegebene Problemstellung zu finden. Gute Lösungen sind solche, die möglichst nah am Optimum liegen. Heuristiken geben jedoch keine Garantie für eine bestimmte Güte der erzeugten Lösungen. Für viele der schwierigen Probleme sind Heuristiken jedoch der einzige Weg, um einigermaßen gute Lösungen zu finden.

Heuristiken können sowohl deterministisch als probabilistisch gestaltet sein.

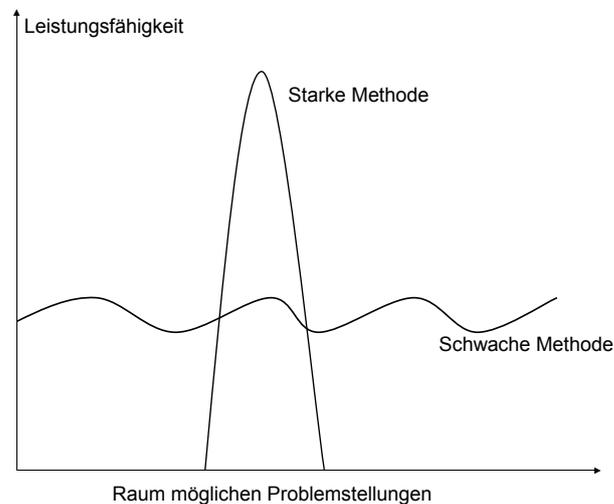


Abbildung 4.1.: Schwache und starke Heuristiken

Kriterien zur Beurteilung von heuristischen Optimierungsverfahren

Mit Kriterien wie der *Allgemeinheit* und der *Leistungsfähigkeit* kann die Qualität einer Heuristik beurteilt werden [10]. Unter dem Grad der Allgemeinheit versteht man die Anzahl der Problemstellungen, bei denen die Methode erfolgreich angewendet werden kann. Die Leistungsfähigkeit beinhaltet selbst weitere Kriterien wie die Lösungswahrscheinlichkeit (die Wahrscheinlichkeit, mit der Heuristik eine einiger-

maßen gute Lösung zu finden), die Lösungsqualität (die mittlere Abweichung der Lösung von dem globalen Optimum), und der Ressourcenbedarf, um eine bestimmte Lösungswahrscheinlichkeit und Qualität der Lösungen zu erreichen. Meist muss man zwischen dem Grad der Allgemeinheit einer Heuristik und ihrer Leistungsfähigkeit abwägen (siehe Abbildung 4.1).

Es gibt eine breite Palette möglicher Suchverfahren. Die Verfahren unterteilen sich – je nachdem, wieviele Elemente gleichzeitig behandelt werden – in **einelementige** und **populationsbasierte** Verfahren. Die in der vorliegenden Arbeit betrachteten Verfahren der *lokalen Suche* verfolgen bei dem Suchprozess jeweils ein Element, die *populationsbasierten* Verfahren haben dagegen in jedem Schritt stets eine Menge von Elementen zur Hand, was den Informationsaustausch über vielversprechende Regionen des Suchraums ermöglicht.

In den folgenden Abschnitten werden die Bestandteile eines Optimierungsverfahrens beschrieben. Es wird der Begriff des Suchraums erklärt, außerdem wird erläutert, was man unter einer Zielfunktion (auch: Bewertungsfunktion, Fitnessfunktion) und der Nachbarschaft eines Elements im Suchraum versteht.

4.2.1. Suchraum

Ein Optimierungsproblem legt fest, wie die möglichen Lösungen des Problems grundsätzlich aussehen könnten. Der Suchraum Ω ist die Menge aller gültigen Lösungskandidaten. Jede spezifische Problemstellung gibt auch vor, welche Nebenbedingungen (Anforderungen) ein Element erfüllen muss, damit es korrekt ist und als ein potenzieller Lösungskandidat in Betracht gezogen werden kann. Der Suchraum beinhaltet also alle strukturell korrekten, aber auch – bezogen auf ihre Güte – schlechte Elemente, die als Lösung des Problems angesehen werden.²

²Beim Lösen eines linearen Gleichungssystems mit zwei Unbekannten zum Beispiel ist der Suchraum die Menge aller reelwertigen Zahlenpaare $\mathbb{R} \times \mathbb{R}$.

4.2.2. Bewertungsfunktion

Durch die Bewertungsfunktion³ wird jedem Element des vom Optimierungsproblem bestimmten Suchraums ein Wert zugewiesen, der ihn als „stärker“ oder „schwächer“ bezüglich gewisser Kriterien bezeichnet:

$$f : \Omega \rightarrow \mathbb{R}^+$$

Bewertet wird nach einem oder mehreren Kriterien, die aus den Eigenschaften der Individuen kombiniert werden. Anhand welcher Kriterien der Lösungskandidat beurteilt wird und wie genau die Zusammensetzung der Eigenschaften in die Bewertungskriterien gestaltet wird, ist problemspezifisch.

Die Wahl der Bewertungsfunktion ist einer der wichtigsten Entscheidungspunkte eines Optimierungsverfahrens. Von der Bewertungsfunktion wird gefordert, dass ein „besseres“ Element des Suchraums auch besser bewertet wird. Allerdings muss noch festgelegt werden, was genau ein „gutes“ Element des Suchraums ist. Normalerweise ist ein besseres Element eines, dessen durch Bewertung festgestellte Güte näher am Optimum liegt. Eine alternative Sichtweise bezeichnet ein Element als besser, wenn das Optimum ausgehend von diesem Element in weniger Schritten erreichbar ist. Diese Definition klingt zwar vielversprechend, die Feststellung eines solchen Kriteriums ist aber häufig nicht oder nur schwer realisierbar.

Die gleichzeitige Optimierung nach mehreren Zielen muss sich entsprechend in der Bewertungsfunktion widerspiegeln. Einerseits können die Bewertungen nach den jeweiligen Kriterien jeweils gewichtet in der Fitnessfunktion akkumuliert werden. Andererseits kann die Fitnessfunktion als ein Vektor aus Bewertungen der Kriterien dargestellt werden.

Trägt man jede Eigenschaft des Lösungskandidaten an einer eigenen Koordinatenachse ein, dann erhält man zusammen mit der zusätzlichen Koordinatenachse für den Fitnesswert eine „Fitnesslandschaft“ (allgemein: „Gütelandschaft“). Jedem Element aus dem Suchraum wird ein bestimmter Punkt auf dieser Gütelandschaft zugewiesen.

³Im Folgenden werden die Begriffe „Bewertungsfunktion“, „Zielfunktion“ und „Fitnessfunktion“ äquivalent verwendet.

In der Abbildung 4.2 werden zwei Extrembeispiele gezeigt: die Variante A stellt ein Problem mit einem „schwierigen“ Suchraum dar, der ein zerklüftetes „Gebirge“ mit vielen lokalen Optima beinhaltet. Die Variante B ist dagegen ein relativ „einfache“ Variante .

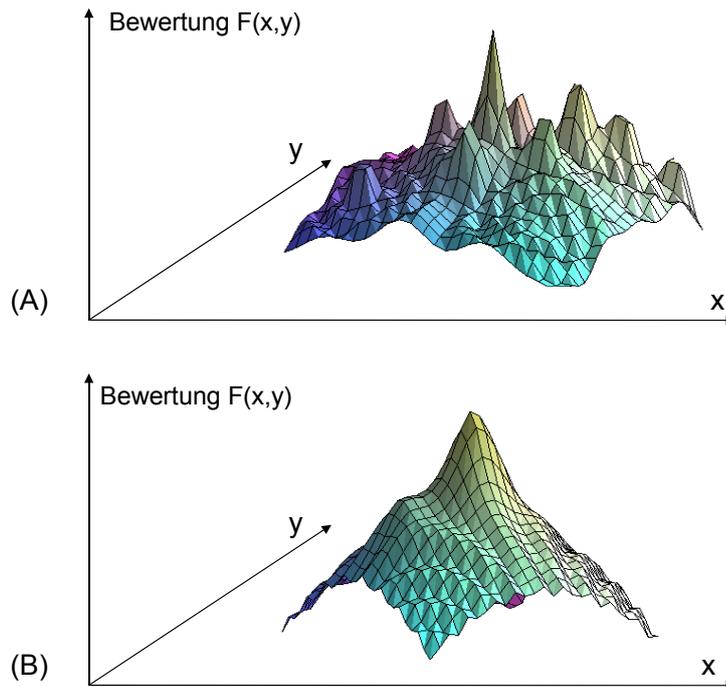


Abbildung 4.2.: Fitnesslandschaft für einen zweidimensionalen Suchraum. Die Variante A stellt eine „schwierige“ Gütelandschaft dar, die Variante B stellt eine relativ einfache Gütelandschaft dar.

Lokales und Globales Optimum. Ein Element wird ein *lokales Optimum* genannt, falls keines der Elemente seiner unmittelbaren Nachbarschaft eine bessere Bewertung hat. Ein *globales Optimum* hat die Eigenschaft, dass kein anderes Element des Suchraums eine bessere Fitness als dieses Element. Ein globales Optimum ist

natürlich auch ein lokales Optimum, und darin besteht eine der Schwierigkeiten der Optimierungsverfahren: ohne Vorwissen kann man nicht erkennen, ob ein gefundenes lokales Optimum auch ein globales ist. Um ein Optimum von vornherein an seinen Eigenschaften als globales Optimum zu erkennen, müsste man den gesamten Suchraum durchlaufen, um festzustellen, ob in der Tat ein globales Optimum gefunden wurde.

Falls eine Mehrzieloptimierung durchgeführt wird und die Fitness als ein Vektor von Eigenschaften dargestellt wird, können allerdings mehrere globale Optima existieren.

4.2.3. Nachbarschaftsstruktur

Eine Nachbarschaftsstruktur ist eine Abbildung, die jedem Element des Suchraums $x \in \Omega$ eine Menge von benachbarten Elementen $Nachbarschaft(x) \subseteq \Omega$ zuweist. Die Nachbarschaftsrelation ist eine symmetrische Relation.

$$Nachbarschaft : \Omega \rightarrow \mathcal{P}(\Omega)$$

Der Begriff der Nachbarschaft ist eng mit dem Begriff der *kleinstmöglichen Veränderungen* eines Elementes des Suchraums verbunden. Die Art dieser kleinstmöglichen Veränderungen des Elements, die zu einem anderen gültigen Lösungskandidaten führen, ist problemspezifisch. Die Anzahl der Nachbarn eines Elements in Suchraum ist nicht immer eine konstante Größe: sie ist von den Anforderungen an die Struktur einer gültigen Lösung und von der Struktur des aktuellen Elements selbst abhängig. Die Größe der Nachbarschaft der Elemente des Suchraums ist mit der Größe des Suchraums selbst und mit dem entstehenden Aufwand der Suche verbunden. Es ist möglich, die Nachbarschaft eines Elementes des Suchraums als Nachbarschaftsgraph darzustellen.

Schrittfunktion

Es gibt verschiedene Strategien, um aus der Menge der Nachbarn ein Element als nächsten Lösungskandidaten auszuwählen. Die Wahl der Strategie hat einen entsprechend großen Einfluss auf die Leistungsfähigkeit des Verfahrens. Folgende Strategien stehen zur Auswahl:

- Random Neighbor. Bei dieser Strategie wird stets eine zufällige Nachbarlösung aus $Nachbarschaft(x)$ generiert.
- Next Improvement. Durchsuche die $Nachbarschaft(x)$ in einer festen Reihenfolge, wähle die erste Lösung, die mindestens so gut wie x ist.
- Best Improvement. Durchsuche die $Nachbarschaft(x)$ vollständig und wähle die beste Nachbarlösung.

4.3. Lokale Suche

Die Verfahren der Lokalen Suche verfolgen auf dem Lösungsweg stets nur ein Element des Suchraums. Dabei versucht das Verfahren, eine bessere Lösung in der Nähe einer bereits gefundenen Lösung zu ermitteln. Das Verfahren ist iterativ. In jedem Schritt des Suchprozesses wird nach einer gewissen Strategie das nächste Element aus der Nachbarschaft des aktuellen Elements ausgewählt. Die einzelnen Verfahren der lokalen Suche unterscheiden sich in der Strategie der Wahl des nächsten Elements. In den folgenden Unterabschnitten werden zwei Vertreter von Verfahren der lokalen Suche vorgestellt: Hillclimbing und Simulated Annealing.

4.3.1. Hillclimbing

Hillclimbing ist ein heuristischer Suchalgorithmus, der zu den Verfahren der lokalen Suche gehört. Man bestimmt für ein Individuum die Nachbarschaft (siehe Abschnitt 4.2.3). Dann wählt man aus der Menge der Nachbarn die n besten Elemente, und aus dieser Teilmenge wird ein Individuum als nächster Kandidat bestimmt. Das Verfahren kann deterministisch realisiert werden, indem man immer den (ultimativ) besten Nachbarn für den nächsten Schritt wählt, oder stochastisch, indem irgendein Nachbarelement als nächstes gewählt wird, das besser als das aktuelle ist.

Idee zur Umsetzung: Wähle $y \in U(x)$. Falls $f(y) \preceq f(x)$, dann $x := y$.

Variante des stärksten Anstiegs (maximales Hillclimbing):

Wähle ein $y \in U(x)$ mit bestem $f(y)$ (dafür muss die gesamte Umgebung getestet werden)

Falls $f(y) \preceq f(x)$, setze $x := y$

ansonsten wird das Verfahren abgebrochen, da man sich in einem lokalem Optimum befindet.

Die Vorteile des Verfahrens bestehen darin, dass das Verfahren oft schnell ein (lokales) Optimum findet. Wenn man das Verfahren mit vielen verschiedenen Anfangspositionen wiederholt, kann man häufig ein „gutes“ Optimum finden.

Falls die Berechnung der Fitnessfunktion sehr aufwändig ist, schneidet das Verfahren im Vergleich zu den populationsbasierten Verfahren schlechter ab. Der Nachteil des Verfahrens liegt darin, dass es aus den lokalen Optima, die keine globalen Optima sind, nicht wieder herausfinden kann. Das Verfahren sollte also nur dann verwendet werden, wenn die Gütelandschaft des Problems keine Zerklüftungen aufweist, sondern aus wenigen, möglichst weit auslaufenden „Hügeln“ besteht.

4.3.2. Simulated Annealing

Bei simuliertem Abkühlen handelt es sich um einen der ersten Ansätze, die die Möglichkeit eröffneten, bei der lokalen Suche einem lokalen Optimum zu entkommen⁴. Das Verfahren ist naturinspiert und beruht auf der Modellierung des physikalischen Abkühlungsprozesses, der aus der Metallschmelze in der Werkstoffphysik bekannt ist. Der Begriff „Annealing“ beschreibt das „Erhitzen und langsame Abkühlen“, um die Struktur der Metalle zu modifizieren (bzw. zu verbessern). Abhängig davon, wie schnell abgekühlt wird, weisen die Metalle nach dem Prozess andere Eigenschaften auf.

Das Verfahren in der Werkstoffkunde läuft folgendermaßen ab. Es wird eine Struktur angestrebt, die in einem möglichst geringen Energiezustand ist. Das Metall wird zuerst erhitzt, was den Molekülen Energie zuführt und ermöglicht, sich neu zu positionieren. Dann wird das Metall langsam abgekühlt, wobei die Moleküle eine spezifische Position einnehmen können, so dass am Ende eine bessere energieärmere Struktur entsteht, als sie vorher vorhanden war.

⁴Kirkpatrick et al. 1983

Würde der Abkühlvorgang zu schnell durchgeführt, blieben die Atome im Metallgitter in energetisch eher ungünstigen Positionen hängen: es entstünden unregelmäßige Strukturen auf einem hohem Energieniveau, was eine Verringerung der Qualität des erzeugten Materials bedeutet. Bei einem langsamen Abkühlen werden dagegen eher regelmäßige Strukturen mit einem geringeren Energieniveau erzeugt: bei der langsamen Abkühlung haben die Moleküle genug Zeit und genug Bewegungsenergie, um schlechte Positionen zu verlassen und sich so zu ordnen, dass stabilere Kristalle entstehen.

Auf die Optimierungsverfahren abgebildet, besteht die Grundidee des Verfahrens darin, dass auch die schlechteren Lösungen mit einer bestimmten Wahrscheinlichkeit akzeptiert werden können (man löst also eine scheinbar gute Struktur auf, um eventuell eine bessere zu finden), wobei die Wahrscheinlichkeit, eine schlechtere Lösung zu akzeptieren, mit jedem Schritt verringert wird. Der genaue Ablauf des „Abkühlens“ hängt von der Problemstellung ab.

Die Verringerung der Temperatur T verläuft nach einem Abkühlungsplan („Kühlschema“), der aus drei möglichen Strategien der Temperaturverringerung zusammengesetzt sein kann (n bezeichne die Nummer des Iterationsschrittes):

- Konstant: $T(n) = \text{const}$
- Arithmetisches Cooling: $T(n) = T(n - 1) - k$, wobei k ein konstanter Reduktionsparameter ist.
- Geometrisches Cooling: $T(n) = \alpha \cdot T(n - 1)$, wobei $\alpha \in [0.8, 0.99]$

Verfahren: Man wählt eine Folge $T_1 \geq T_2 \geq T_3 \geq T_4 \geq \dots \geq 0$ nach einer der beschriebenen Strategien (die Folge $\{T_i\}$ konvergiert gegen 0).

Gegeben sei das Optimierungsproblem (Ω, f, \geq) . Zu jedem $x \in \Omega$ gibt es eine Umgebung $U(x)$.

```
Initialisieren: wähle  $x \in \Omega$ ; berechne  $f(x)$ ; // Energieniveau
k:=1;
while not(Abbruchbedingung) do
  wähle zufällig  $y \in U(x)$ ;
   $\Delta E := f(y) - f(x)$ ;
  if  $\Delta E \leq 0$  then  $x := y$ 
  else if  $\text{random}[0, 1] < e^{-\frac{\Delta E}{T_k}}$  then  $x := y$ ;
    end if
  end if
t := t + 1
end while
```

Wie man an der zufälligen Auswahl des Knotens aus der Umgebungsmenge erkennt, basiert das Verfahren auf der sogenannten „random neighbor“-Schrittfunktion. Das Abbruchkriterium kann das Erreichen einer bestimmter Temperatur sein, oder das Ausbleiben von Verbesserungen des Temperaturniveaus über n Schritte hinweg. Als Anhaltspunkt für die Anzahl der Iterationen auf jedem Temperaturniveau kann z.B. ein Vielfaches der Nachbarschaftsgröße dienen.

Der Initialwert für die Temperatur T ist problemabhängig. Erfahrungsgemäß kann er so gewählt werden, dass ca. 3% aller Züge anfangs abgelehnt werden [11]. Eine andere Variante ist die Wahl

$$T_1 = \max_{x \in \Omega} f(x) - \min_{x \in \Omega} f(x)$$

Das Verfahren gilt als „launisch“. Wählt man als Abkühlungsschema die geometrische Strategie ($T_{t+1} = \alpha \cdot T_t$), so gibt es bei vielen Problemen oft nur ein kleines Intervall für α , mit dem Simulated Annealing gut funktioniert. Der wichtigste Vorteil des Simulated Annealing gegenüber Hillclimbing ist, dass einmal gefundene lokale Optima mit einer Wahrscheinlichkeit, die größer als 0 ist, wieder verlassen werden können (siehe dazu die Abbildung 4.3). Das Simulated Annealing liefert für viele Probleme gute Ergebnisse. Dennoch – im Vergleich zu anderen Verfahren der Lokalen Suche braucht Simulated Annealing oft relativ lange Laufzeiten, bis es gute Lösungen findet.

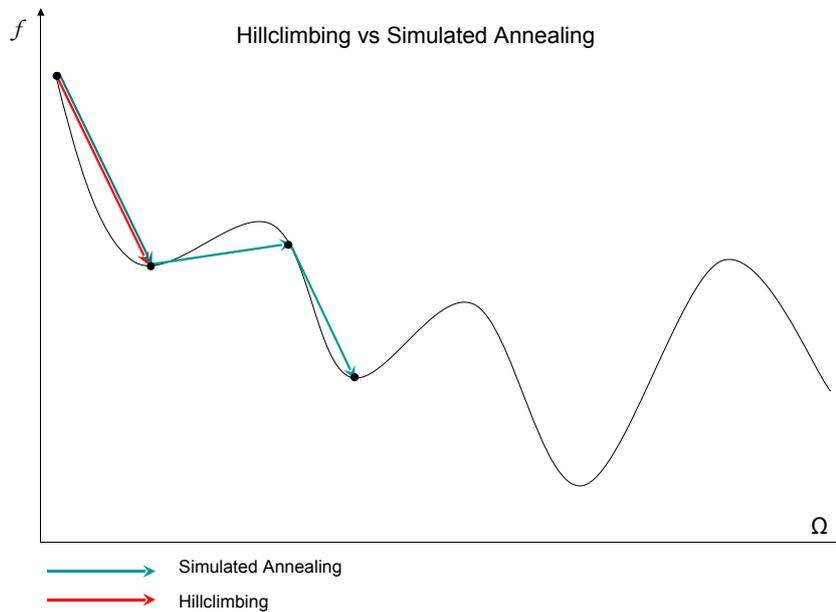


Abbildung 4.3.: Simulated Annealing vs Hillclimbing

Es gibt viele Varianten von Simulated Annealing. Eine Weiterentwicklung des Verfahrens ist z.B. die das Durchführen von wiederholtem „Reheating“.

4.4. Evolutionssimulation

Das Entstehen und die Entwicklung der Lebewesen auf der Erde kann als einer der längsten und aufwändigsten Optimierungsvorgänge angesehen werden, die es je gab. Einige Mechanismen, die vermutlich für diese Entwicklung verantwortlich sind, sind Bestandteile der Evolutionstheorie. Unter einer Entwicklung versteht man intuitiv eine positive Entwicklung. Die in der Natur stattfindende Wandlung der Lebewesen ist dagegen eher als eine Anpassung an die veränderliche Umgebung anzusehen. Als die wichtigsten vorantreibenden Werkzeuge der Evolution werden inzwischen die natürliche Selektion zusammen mit der Mutation und der Rekombination angenommen.

Mit *Evolution* kann man einen iterativen, immer fortlaufenden Vorgang bezeichnen, der Veränderungen in der Häufigkeit der Genallele innerhalb einer Population bewirkt. Durch Mutationen entstehen kleinere oder auch große Änderungen. Durch Rekombination wird das Erbgut (die Eigenschaften) der Elternindividuen neu kombiniert. Die Umweltselektion übt einen Druck aus, indem sie denjenigen Individuen eine höhere Überlebenschance einräumt, welche „bessere“ Eigenschaften aufweisen.

Diese in der Natur beobachteten Vorgänge werden übernommen und als ein Verfahren der Lösungssuche nachgeahmt. Die Umwelt wird dadurch simuliert, dass ihre Einflüsse in Bedingungen ausgedrückt werden. Diese Bedingungen werden so aufgestellt, dass die besseren Lösungen die Bedingungen besser erfüllen und folglich am besten an die Umwelt angepasst sind. Das Verfahren sammelt Informationen über die Struktur des Suchraumes (des Problems), diese Information wird dann implizit in den Individuen der Populationen aufbewahrt.

Der Grundbaustein und Kern eines evolutionären Algorithmus ist der **Evolutionenzyklus**. In diesem Kapitel wird erklärt, wie einige Begriffe in Bezug auf evolutionäre Algorithmen angewendet werden und wie der Evolutionszyklus aufgebaut wird.

4.4.1. Grundbegriffe

In diesem Abschnitt werden die wichtigsten aus der natürlichen Evolution übernommene Begriffe eingeführt.

Lösungskandidat (Individuum): ein einzelnes Element aus einer Gesamtheit – dem **Suchraum** – von Elementen mit gleichen Eigenschaften, aber mit individueller Ausprägung dieser Eigenschaften. Individuen setzen sich aus einzelnen Komponenten (Eigenschaften oder Genen) zusammen.

Population: eine Gruppe von Individuen, die aufgrund ihrer Entstehungsprozesse miteinander verbunden sind und Kinder-Individuen erzeugen können.

Erbgut: Gesamtheit der Erbinformation (Gene). Bei den evolutionären Algorithmen versteht man darunter die Menge der Eigenschaften aller Elemente des

Suchraums.

Diversität der Population: Die Verteilungsart aller Individuen der Population in dem Suchraum. Falls die einzelnen Individuen „weit“ voneinander liegen (sehr unterschiedliche Ausprägung der Eigenschaften), ist die Diversität hoch und der Suchraum relativ gut überdeckt. Sind aber die einzelnen Individuen sehr „ähnlich“, ist deren Abstand klein und man kann sagen, dass die Individuen auf einem relativ kleinem Teil des Suchraums liegen, die Diversität der Population ist also klein.

Vererbung: Als Vererbung bezeichnet man in der Biologie die direkte Übertragung von Eigenschaften der Individuen auf ihre Nachkommen.

Selektion: Ein Mechanismus, der in der Evolution an verschiedenen Stellen zum Einsatz kommt, um aus der Gesamtheit aller Individuen die am besten an die Umgebung angepassten auszuwählen.

Mutation: Die Veränderung von Genen durch äußere Einflüsse. Änderungen, die durch Mutation entstanden sind, sind vererbbar.

Rekombination: Neukombination von Erbgut.

4.4.2. Der evolutionäre Algorithmus

Der Grundverlauf einer simulierten Evolution wird durch einen Evolutionszyklus dargestellt.

Abbildung 4.4 zeigt, wie der Evolutionszyklus durch ein Programm abgebildet wird:

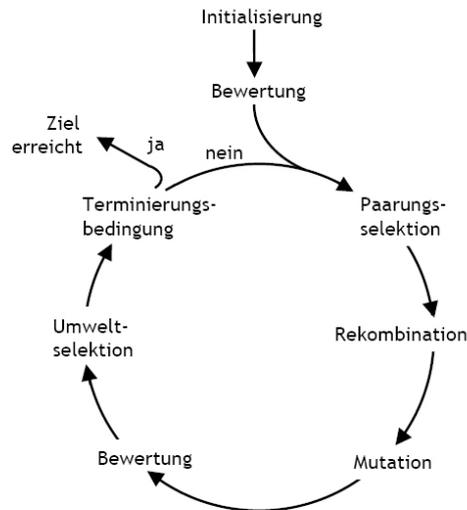


Abbildung 4.4.: Evolutionszyklus

```

1  erzeuge  $Population_0$ ,  $n := 0$ 
2   $\forall p \in Population_0$  : berechne  $fitness(p)$ 
3  while ( $\neg Terminierungsbedingung(Population_n)$ ) do
4       $Eltern := ElternSelektion(Population_n)$ 
5       $Kinder := Rekombination(Eltern)$ 
6       $Kinder := Mutation(Kinder)$ 
7       $\forall p \in Kinder$  : berechne  $fitness(p)$ 
8       $Population_{n+1} := UmweltSelektion(Population_n, Kinder)$ 
9       $n := n + 1$ ;
10 endwhile
  
```

Im Folgenden werden die einzelnen Bestandteile des Evolutionszyklus genauer diskutiert. Es werden die Hauptideen der Realisierung der jeweiligen Mechanismen vorgestellt. Die Hauptrichtungen der evolutionären Algorithmen werden in Abschnitt 4.4.9 erläutert.

4.4.3. Initialisierung

Aufgabe der Initialisierung ist es, eine Menge von Individuen aus dem Lösungsraum auszuwählen, die die Grundlage für die nachfolgende Evolution sein wird. Falls es Vorkenntnisse über die zu erwartende Struktur einer guten Lösung gibt, können diese hier eingebracht werden. Dabei wird eine Teilmenge der Initialpopulation durch Verwendung bestimmter Muster erzeugt, die vermutlich eine positive Auswirkung haben. Dennoch sollte man bei dem Einbringen von Wissen in die Initialpopulation vorsichtig sein, denn die Initialisierung hat einen großen Einfluss auf den gesamten Verlauf der Evolutionssimulation, und eine falsche Annahme kann das Ergebnis verfälschen.

Bei der Initialisierung ist es von Bedeutung, eine Anfangspopulation mit einer möglichst hohen Diversität zu erzeugen. Eine gleichmäßig über den Suchraum verteilte Stichprobenentnahme kann eine ideale Startposition für die Suche sein. Die Auswahl einer solchen Initialpopulation ist aber keine triviale Angelegenheit, insbesondere wenn man über den Suchraum (abgesehen von seiner Größe) nur relativ wenige Kenntnisse hat.

4.4.4. Bewertung der Fitness

Die Bewertung der Fitness bei der Evolutionssimulation ist gleichgestaltet wie bei einer lokalen Suche (siehe 4.2.2). Es ist zu bemerken, dass ein höherer Fitnesswert in der Evolutionssimulation wie auch bei der natürlichen Evolution für ein Individuum eine höhere Überlebenschance bedeutet. Ein Individuum mit einem höheren Fitnesswert hat in der natürlichen Evolution auch eine höhere Wahrscheinlichkeit, an der Fortpflanzung teilzunehmen und somit seine Gene (Eigenschaften) zu erhalten, indem sie an die nächsten Generationen weitergegeben werden. Ob man das entsprechende Prinzip auch bei der Rekombination in der simulierten Evolution realisiert, ist Entscheidungssache.

Der Fitnesswert, der in der Selektion verwendet wird, gibt der Evolutionssimulation genau so wie bei den Verfahren der lokalen Suche eine Entwicklungsrichtung vor. Daher ist die Wahl einer korrekten Bewertungsfunktion ausschlaggebend für den Erfolg der Evolutionssimulation.

4.4.5. Rekombination

Wie in der Natur versteht man unter der simulierten Rekombination die Verteilung und Neuordnung des in der Population bestehenden Erbguts. Die simulierte Evolution hat bei der Rekombination keine Beschränkung der Elternteilanzahl und kann daher auch mehr als zwei Elternteile bei der Rekombination erlauben. Andererseits kann die Rekombination von Erbgut mehrerer Individuen auch durch eine Reihe von Rekombinationen jeweils zweier Elternteile realisiert werden. Daher wird im Folgenden immer nur von Elternpaaren ausgegangen.

Bei der Rekombination zweier „guter“ Elemente erwartet man natürlich, dass der Nachkomme alle „guten“ Eigenschaften der Eltern übernimmt. Tatsächlich ist das aber nicht unbedingt der Fall⁵. Dennoch: je mehr Nachkommen ein Elternpaar hat, desto höher ist die Wahrscheinlichkeit, dass dabei auch ein Kind erzeugt wird, welches die Stärken beider Eltern übernimmt und zumindest eine bessere Fitness als seine Eltern haben wird. Lässt man diese Rekombinationen zu, müssen die Elternindividuen mit größerer Sorgfalt gewählt werden. Je stärker sich die Eltern unterscheiden, desto besser, denn dann wird sehr unterschiedliches Erbgut vermischt, was die Diversität in der Population unterstützt.

4.4.6. Mutation

Unter Mutation versteht man eine spontane Veränderung des Erbgutes. Die Mutation kann als ein Materiallieferant der Evolution angesehen werden.

Der klassischen Evolutionstheorie nach verdankt man der Mutationen die Vielfalt der Arten. In der natürlichen Evolution unterscheidet man ein breites Spektrum an Mutationsarten, je nachdem, welche Ursache und Wirkung sie haben. Es werden generative (weitervererbare) und somatische (nicht vererbare) Arten der Mutationen

⁵Wie eine Anekdote besagt, erhielt Albert Einstein einmal folgenden Brief einer hübschen Dame: „Stellen Sie sich vor, welche Kinder wir haben könnten: mit Ihrer Intelligenz und meiner Schönheit!“ Einstein antwortete darauf: „Was wäre aber, wenn sie meine Schönheit erbten und Ihre Intelligenz?!“

unterschieden. Für die Evolutionssimulation ist hauptsächlich die generative Art der Mutation interessant.

Es wird beobachtet, dass die meisten in der Natur entstehenden Mutationen nach ihrer Wirkung auf die Überlebensfähigkeit der Individuen neutral oder eher negativ sind. Dabei kann die Mutation im negativen Fall fatale Folgen für ein Individuum haben. Mit neutralen Mutationen bezeichnet man Veränderungen an Teilen des Erbguts, die bei der Reproduktion keine Rolle spielen, oder Veränderungen in den nicht kodierenden DNA-Abschnitten. Viele der Mutationen sind auch wegen der Redundanz des genetischen Codes neutral, da sie im Phänotyp nicht zum Ausdruck kommen. Eine positive, zu Selektionsvorteilen führende Auswirkung hat die Mutation der natürlichen Evolution selten. Bei der Evolutionssimulation ist es ähnlich.

Egal, ob die Mutation sofort einen Vorteil bringt oder nicht, sie ist ein bedeutender Bestandteil der simulierten Evolution. Die wichtigste Eigenschaft der Mutation ist die Tatsache, dass sie die Entstehung völlig neuer (noch nicht in der Population vorhandener) Gene (Eigenschaften) bewirkt. Demgegenüber wird bei der Rekombination lediglich das bereits vorhandene Genmaterial neu gemischt. Die Mutation unterstützt also die Erhaltung der Diversität innerhalb einer Population.

Der Zufall spielt bei der natürlichen wie auch bei der simulierten Mutation eine große Rolle. Eine Mutation kann einerseits kleine Veränderungen des Individuums hervorrufen – dies wird **feinabstimmende Mutation** (Mikromutation) genannt. Andererseits kann die Mutation auch sprunghaft wirken und zu großen Änderungen führen; man nennt sie **erforschende Mutation** (Makromutation).

Bei der Durchführung der Mutation muss entschieden werden, welche der Individuen mutiert werden und wie genau die Mutation verläuft, unter anderem auch wie stark. Die **Mutationstärke** (Mutationsrate) besagt, wie stark das Individuum mutiert werden soll (z.B. an wie vielen Stellen das Genmaterial verändert wird und wie tiefgreifend die Veränderungen sind). Diese Größen kann man je nach Problemstellung variieren und anpassen.

Je nach Art der evolutionären Algorithmen und auch abhängig von der Problemstellung spielt die Mutation eine unterschiedliche Rolle. Falls die Rekombination leicht

zu realisieren ist, kann die Rekombination eine erforschende Rolle übernehmen, und die Mutation übernimmt dann die feinabstimmende Rolle. Bei anderen Problemstellungen steht die Mutation an erster Stelle und übernimmt sowohl die erforschende als auch die feinabstimmende Rolle; die Rekombination rückt in den Hintergrund oder wird gar nicht verwendet.

Genau wie bei der Schrittfunktion (Verfahren der Lokalen Suche) und auch bei der Rekombination stellt man an die Mutation die Bedingung, dass jedes neuentstandene Element im Suchraum liegen muss. Eine weitere Anforderung an die Mutation ist, dass jeder Punkt des Suchraums von jedem anderen Punkt in einer endlichen Zahl von Schritten erreicht werden können muss. Die Nichteinhaltung dieser Forderung hätte drastische Auswirkungen für das Verfahren: bestimmte Gebiete des Suchraums würden in der Suche vermieden, und die Chancen auf Erfolg wären von vornherein verringert.

4.4.7. Selektion

Selektion kommt in evolutionären Algorithmen bei der Wahl der Eltern und bei der Konstruktion der neuen Population durch Umweltselektion zum Einsatz.

In der Natur stellt die Umwelt einen Kontext dar, in dem die Population agiert. Die natürliche Umwelt besitzt gewisse Eigenschaften und Charakteristika, und je besser ein Individuum an diese angepasst ist, desto höher sind seine Überlebenschancen. Somit übt die Umwelt auf die Population einen Druck aus: die schlechtesten oder schwächsten Elemente werden aussortiert (ausselektiert): erstens ist ihre Lebensdauer kürzer (ein Individuum übersteht weniger Generationen), und zweitens sind auch die Fortpflanzungschancen bei jedem Generationswechsel geringer. In der simulierten Evolution gilt dasselbe Prinzip. Die Umwelt wird bei der simulierten Evolution von der Problemstellung vorgegeben: das Problem stellt gewisse Ziele und Anforderungen und gibt die Bewertungskriterien der Individuen vor, die, zusammengefasst in der Fitnessfunktion, entscheiden, wie gut „angepasst“ ein Element ist.

Die Selektion übernimmt bei den evolutionären Algorithmen, wie auch in der Natur,

eine lenkende Rolle. Die Selektion macht aus einer sonst zufälligen Suche eine zielgerichtete. Sie lenkt den Entwicklungsprozess in diejenigen Bereiche im Suchraum, wo sich ein Optimum befindet. Im Gegensatz zur Mutation und Rekombination bewirkt die Selektion eine Verringerung der Diversität. Der sogenannte Selektionsdruck stellt ein Maß der „Zielstrebigkeit“ der Selektion dar.

Es gibt eine große Anzahl von Selektionsmethoden. Man unterscheidet zwischen **deterministischen** und **probabilistischen** Arten der Selektion. Bei der deterministischen Selektion wird die Wahl der Individuen ausschließlich und eindeutig durch deren Bewertung bestimmt; bei der probabilistischen dagegen werden auch Individuen mit schlechterem Fitnesswert mit einer gewissen Wahrscheinlichkeit zur Fortpflanzung zugelassen. Die probabilistische Selektionsart hat den Vorteil, dass dabei auch Individuen mit potentiell gutem Erbgut, aber schlechterem Fitnesswert überleben können; dies bewahrt die Diversität in der Population. Eine weitere Entscheidung bezieht sich darauf, wie oft ein Individuum zur Fortpflanzung selektiert werden darf. Bei der **duplikatfreien Selektion** darf ein Individuum lediglich ein einziges Mal gewählt werden, bei der **Selektion mit Duplikaten** auch öfter [9].

Im Folgenden werden einige wichtige Selektionsmethoden bei evolutionären Algorithmen vorgestellt.

Fitnessproportionale Selektion Die fitnessproportionale Selektion ist ein probabilistisches Selektionsverfahren, bei dem die Selektionswahrscheinlichkeit proportional zum Gütewert des Individuums ist. Der Nachteil des Verfahrens liegt darin, dass es sich bei sehr homogenen Populationen (in denen sich die Fitnesswerte der Individuen nur wenig voneinander unterscheiden) wie ein zufälliges Verfahren verhält. Falls die Population dagegen sehr inhomogen ist, kann es passieren, dass ein sehr gutes Individuum alle anderen unterdrückt, was zur Stagnation bei Fortschritten in der Suche führen kann.

Rangbasierte Selektion Die rangbasierte Selektion ist ein probabilistisches Selektionsverfahren, bei dem die Selektionswahrscheinlichkeit des Individuums seiner Position in der nach Fitnesswerten vorsortierten Liste der Individuen entspricht. Im

Gegensatz zur fitnessproportionalen Selektion spielen hier die absoluten Werte der Fitness keine Rolle. Das Vorsortieren verursacht aber einen zusätzlichen Aufwand, was ein Nachteil des Verfahrens ist.

Turnierselektion Eine Turnierselektion $T_{q,k}$ ist ein probabilistisches Selektionsverfahren, bei dem q zufällig und gleichverteilt ausgewählte Individuen in einem Turnier gegeneinander antreten und die k besten Individuen weiterkommen. Die Turnierselektion hat, wie auch die rangbasierte Selektion, auch dann keine Schwierigkeiten, wenn die Populationen sehr homogene Fitnesswerte aufweist. Das Verfahren benötigt aber keine Vorsortierung, was einen Vorteil gegenüber der rangbasierten Selektion darstellt.

4.4.7.1. Elternselektion

Bei der Elternselektion wird jeweils eine Menge der Individuen (Eltern) aus der aktuellen Population ausgesucht, für die dann die Rekombination stattfinden wird. Die Wahl der Eltern kann einerseits zufällig sein, andererseits könnte man an dieser Stelle einen Selektionsdruck ausüben: je höher der Fitnesswert, desto höher die Wahrscheinlichkeit, als Elternteil gewählt zu werden. Eine Methode hierfür ist z. B. die Turnierselektion (detailliert in Abschnitt 4.4.7 beschrieben).

4.4.7.2. Umweltselektion

Die Umweltselektion erzeugt aus der Menge der durch Rekombination und Mutation entstandenen Individuen (eventuell vereinigt mit Individuen der vorherigen Population) eine neue Population, wobei eine neue Generation entsteht. Die Größe dieser „Kandidatenmenge“ ist meist ein Vielfaches der Populationsgröße. Durch Umweltselektion werden einige der Elemente verworfen. Je größer das Verhältnis der Kardinalitäten der „Kandidatenmenge“ und der „Population“ ist, desto höher ist der Selektionsdruck.

Sei $Kinder$ die Menge der Kinderelemente, sei $\lambda = |Kinder|$ ihre Kardinalität.

Sei $\mu = |Population_n|$ die Anzahl der Eltern (d. h. die Größe der Population in der Generation n ist $Population_n$).

Je nachdem wie die alte Population mit der neuen ersetzt wird, unterscheidet man:

- **Plus-Strategie** – eine Ersetzungsstrategie der Umweltselektion, bei der die neue Population $Population_{n+1}$ aus der vorhergehenden Population P_n und der Kinderpopulation auf eine bestimmte Art und Weise zusammengesetzt wird.

$$(\mu + \lambda) \text{ – Plus-Strategie : } Population_{n+1} \subseteq (Kinder \cup Population_n)$$

- **Komma-Strategie** – eine Ersetzungsstrategie, bei der die neue Population $Population_{n+1}$ ausschließlich aus den Kinderindividuen gebildet wird.

$$(\mu, \lambda) \text{ – Komma-Strategie : } Population_{n+1} \subseteq Kinder$$

4.4.8. Terminierungsbedingung

Der Evolutionszyklus wird so lange wiederholt, bis die Terminierungsbedingung erfüllt ist. Die Terminierungsbedingung formuliert bestimmte problemspezifische Voraussetzungen, um den Suchprozess abzuschließen. Terminierungsbedingungen können qualitätsgebundene oder ressourcen- bzw. zeitgebundene Voraussetzungen sein. Eine Terminierungsbedingung könnte zu Beispiel das Erreichen einer bestimmten Güte in der Population sein – gemessen entweder bei einem oder einer bestimmten Anzahl von Elementen, oder über die Population aggregiert. Gegebenenfalls kann die Evolution auch nach einer bestimmten Anzahl von Generationen obligatorisch beendet werden.

4.4.9. Evolutionäre Standardalgorithmen

Die meisten aktuellen evolutionären Standardalgorithmen können in vier Hauptströmungen unterteilt werden. Im folgenden werden ihre jeweiligen Ideen kurz beschrieben. Jede dieser Hauptrichtungen hat ihrerseits eine Fülle von Varianten. Die größten Unterschiede der Hauptrichtungen liegen in der Herangehensweise an die Darstellung der Elemente des Suchraums: wie wird die Information codiert (kürzeste oder breiteste Form), oder wird sie gar durch bestimmte Strukturen repräsentiert. Andere Unterschiede bestehen darin, welche Evolutionsoperatoren die dominierende Rolle übernehmen. Diese Unterschiede ergeben sich allerdings oft als Folgen der gewählten Repräsentation.

Genetische Algorithmen Genetische Algorithmen bilden eine Untergruppe der evolutionären Algorithmen. Eine der Besonderheiten der genetischen Algorithmen ist, dass die Elemente des Suchraums als Zeichenkette mit fester Länge über einem endlichem Alphabet codiert werden. So wird eine Analogie zu den Chromosomen geschaffen. Damit man einen genetischen Algorithmus für Problemstellungen mit einem aus reellen Zahlen bestehenden Suchraum verwenden kann, muss eine Kodierungs-/Dekodierungsfunktion für die Problemstellung definiert werden. Die Rekombination wird bei den genetischen Algorithmen als Hauptoperator behandelt; Mutationen werden bei den genetischen Algorithmen eher als Hintergrundoperator verwendet, der hauptsächlich zu Diversitätserhaltung eingesetzt wird. Die Elternselektion verläuft bei den genetischen Algorithmen fitnessproportional. Als Gründer der genetischen Algorithmen gilt John H. Holland (siehe [15]).

Evolutionstrategien Bei Verwendung von Evolutionstrategien wird ein auf reellen Zahlen basierender Codierungsansatz verwendet. Ein Chromosom besteht dann aus einem Vektor von reellen Zahlen. Die bemerkenswerte Eigenschaft der Evolutionstrategien ist die, dass bei ihnen die Evolutionsparameter selbst der Evolution unterzogen werden; diese ändern sich zusammen mit der eigentlichen Entwicklung der Individuen. Die Elternselektion findet bei den Evolutionstrategien auf einer rein zufälligen (fitnessunabhängigen) Basis statt. Als Erfinder der Evolutionstrategien gelten Bäckmann, Rechenberg und Schwefel.

Evolutionäres Programmieren Die Besonderheit des evolutionären Programmierens im Vergleich zu anderen Verfahren ist, dass hier eine sehr problemnahe Repräsentation der Individuen gewählt wird. Der Phänotyp wird hier von dem Genotyp nicht unterschieden. Aufgrund der so gewählten Repräsentation ist die Mutation der Hauptoperator der Evolution, Rekombination dagegen wird nicht verwendet. Das evolutionäre Programmieren wurde von Fogel, Owens und Walsh ([14]) eingeführt. Das von Fogel beschriebene Verfahren verfolgte das Ziel, möglichst kleine deterministische endliche Automaten (DFAs) zu erzeugen, die ein bestimmtes Verhalten realisieren.

Genetisches Programmieren Beim genetischen Programmieren, das seinerseits auf den genetischen Algorithmen basiert, wird von einer Repräsentation der Individuen als „Chromosomen“ variabler Länge ausgegangen, die durch korrekte (zu einer Grammatik gehörende) Programme interpretiert werden. Oft werden sie als Bäume oder Listen dargestellt. Der Suchraum besteht also aus einer Menge von syntaktisch korrekten Programmen, die ein bestimmtes Schema realisieren. Bewertet wird aufgrund einer vorgegebenen Trainingwertetabelle: das Verfahren soll ein Programm auffinden, das dazu den entsprechenden Rechenmechanismus realisiert. Wie auch bei den genetischen Algorithmen steht die Rekombination im Vordergrund. Bei der Darstellung von Individuen als Bäume werden bei der Rekombination Teilbäume der Eltern miteinander vertauscht. Das genetische Programmieren wurde von Koza Ende der 80er Jahre entwickelt ([16, 17]).

5. Realisierung der Stratifikation

In diesem Kapitel wird der Vorgang der Suche nach einer optimalen Stratifikation erläutert. Der erste Abschnitt beschäftigt sich mit der Spezifik des Problems der Stratifikation. Die weiteren Abschnitte des Kapitels (5.2 - 5.5) erläutern die notwendigen Vorbereitungen und Anpassungen zur Verwendung der Optimierungsverfahren Hillclimbing, Simulated Annealing, eines hybriden Ansatzes aus Hillclimbing und Simulated Annealing und der Evolutionssimulation auf dieses Problem. Der Abschnitt 5.6 beschäftigt sich mit der Frage, wie das Einbringen von Mustern in einen Suchprozess realisiert werden könnte und wie diese Mustern konstruiert werden können.

5.1. Suche der optimalen Stratifikation

Das Problem des Auffindens der optimalen Stratifizierung ist eine Verallgemeinerung des bekannten Problems der Graphpartitionierung, dieses wird zunächst vorgestellt. Danach wird die Semantik und die Größe des Suchraums der Stratifizierung diskutiert. Der Begriff der Nachbarschaft eines Stratifikationselements im Suchraum wird definiert. Die Ansätze zur Vermeidung der Zyklenentstehung sowie einer Zyklererkennung bei der Stratifikation werden beschrieben. Der Vorgang des Auffindens eines benachbarten Elementes unter Berücksichtigung der Nebenbedingung der Azyklichkeit wird erläutert.

Bezüglich des Zeitpunkts der Stratifikation wird angenommen, dass diese während der Modellierungsphase durchgeführt wird. Theoretisch könnte die Stratifizierung auch zur Laufzeit des Systems erfolgen, denn während der Laufzeit können Statistiken ermittelt werden, mithilfe derer man dann die Einstellungen des Verfahrens ändern würde – z. B. indem man andere Prioritäten für die Optimierung setzt. Man

würde dann von einer dynamischen Stratifizierung sprechen. Ein Problem, das dabei gelöst werden müsste, ist die Umstellung des Systems vom früheren Zustand auf die dynamisch angepasste Stratifizierung.

5.1.1. Das Problem der Graphpartitionierung

Die Stratifikation ist eine Abwandlung des bekannten Graphpartitionierungsproblems. Das Graphpartitionierungsproblem geht von einem ungerichteten Graph aus, in dem die Knoten gleichmäßig auf Teilmengen verteilt werden sollen, so dass alle Partitionen in etwa gleich groß sind, aber gleichzeitig je zwei Partitionen nur mit möglichst wenigen Kanten verbunden sind. Es sollen also möglichst wenige Kanten „geschnitten“ werden (siehe dazu die Abbildung 5.1). Das Graphpartitionierungsproblem gehört zur Klasse der NP-vollständigen Probleme.

Das Standard-Graphpartitionierungsproblem kommt in vielen Gebieten als Problembeschreibung zum Einsatz. Für Netzwerke ist es zum Beispiel von Interesse herauszufinden, welche „Gruppen“ des Netzwerks relativ „dünn“ vernetzt sind. Dafür kann man eine wie oben beschriebene „optimale“ Graphpartitionierung durchführen, so dass die stark miteinander verbundenen Knoten des Netzwerks in einer gemeinsamen Partition liegen, während zwischen einzelnen Partitionen möglichst wenige Kanten vorhanden sind. Danach können an den auffällig spärlich vernetzten Stellen weitere Verbindungen hinzugefügt werden, womit die Ausfallsicherheit erhöht werden kann.

Ein anderes Beispiel, bei dem das Graphpartitionierungsproblem zum Einsatz kommt, findet man in der Medizin. Die Knochenmarkstruktur kann mittels eines Graphes modelliert werden, so dass die Knoten des Graphs den Knochenzellen entsprechen und die Kanten die Zellverbindungen darstellen. Falls man in diesem Graph auffällig dünne oder dichte Stellen findet, kann man Aussagen über frühere Frakturen (Knochenbrüche) machen, oder auch Stellen ausmachen, an denen eine Fraktur mit hoher Wahrscheinlichkeit in der Zukunft vorkommen kann.

Der Unterschied zwischen der Stratifikation und dem bekannten Standardgraphpar-

Graphpartitionierung.

Das Ziel: etwa gleich große Partitionen, minimale Anzahl geschnittener Kanten

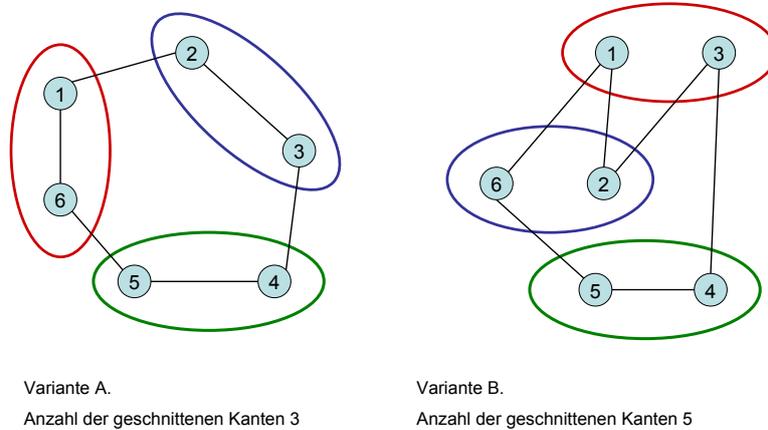


Abbildung 5.1.: Beispiel der Graphpartitionierung: die Variante A ist besser als die Variante B.

tionierungsproblem liegt darin, dass bei der Stratifikation eine Bedingung nicht gilt, nämlich die, möglichst gleichgroße Strata zu erzeugen. Ähnlich sieht es mit der Anzahl der Verbindungen zwischen den Strata aus: je nach den Prioritäten der Bewertungskriterien soll nicht immer die Anzahl der Stratifikationskanten verringert werden. Es sind also vielmehr Mechanismen zu betrachten, die unvoreingenommen bezüglich der Struktur des Graphes sind und für alle Bewertungsarten anwendbar sind.

5.1.2. Der Suchraum

Bei der Suche der optimalen Stratifikation besteht der Suchraum aus der Menge aller Stratifikationen, die bezüglich der aufgestellten Nebenbedingungen gültig sind (auf die Nebenbedingungen bei der Stratifikation wird detailliert im Abschnitt 5.1.4 eingegangen). Manchmal ist die Anzahl aller möglichen Stratifikationen gleich der Anzahl aller möglichen Partitionen der Transaktionsmenge. Das ist der Fall, wenn es im Abhängigkeitsgraph der Transaktionen keine Kanten gibt und somit alle damit

verbundenen Beschränkungen (z.B. Einhaltung der Azyklität) bei der Stratifikation wegfallen.

Die Anzahl aller möglichen Partitionen einer n -elementigen Menge wird durch die **Bellsche Zahl** B_n ¹ angegeben:

$$B_n = \frac{1}{e} \sum_{k=0}^{\infty} \frac{k^n}{k!}$$

$$B_0 := 1, B_1 = 1, B_2 = 2, B_3 = 5, B_4 = 15, B_5 = 52, B_6 = 203, B_7 = 877, \\ B_8 = 4140, B_9 = 21147, B_{10} = 115975, B_{11} = 678570, B_{12} = 4213597, \dots)$$

Die Bellschen Zahlen können mit einer geschlossenen Form näherungsweise angegeben werden:

$$B_n \approx \frac{1}{\sqrt{n}} [\lambda(n)]^{n+\frac{1}{2}} e^{\lambda(n)-n-1}$$

mit $\lambda(n) = e^{W(n)}$, wobei $W(n)$ die Lambert-Funktion ist:

$$W(n) = \sum_{k=1}^{\infty} \frac{(-k)^{k-1}}{k!} n^k$$

Falls aber eine Menge von Nebenbedingungen erfüllt werden muss – wenn ein Abhängigkeitsgraph der Transaktionen vorgegeben ist und Zyklen also vermieden werden müssen, oder falls es Beschränkungen für die Größe der Strata gibt etc. –, ist die Anzahl aller möglichen gültigen Stratifikationen deutlich kleiner.

Bei der Realisierung der Stratifikation in der vorliegenden Arbeit wird zwar angenommen, dass die Anzahl der Transaktionen im System unter hundert ist ($|\mathcal{T}| \ll 100$). Dennoch bleibt das Problem NP-vollständig, womit das Durchlaufen des gesamten Suchraums unmöglich wird. Daher werden heuristische Optimierungsverfahren angewendet.

¹Die Bellschen Zahlen sind nach dem Mathematiker Eric Temple Bell (1883-1960) benannt, der sie untersuchte.

5.1.3. Die Nachbarschaft

Als benachbart werden Stratifikationen angesehen, die sich nur in der Positionierung einer Transaktion unterscheiden. Die kleinstmögliche Änderung bezüglich des Aufbaus eines Stratums, die vorgenommen werden kann, ist das Verlegen einer Transaktion von einem Stratum in ein anderes. Falls die gewählte Transaktion die einzige in ihrem bisherigen Stratum war, wird das bisherige Stratum gelöscht. Eine weitere kleinstmögliche Änderung besteht darin, dass die gewählte Transaktion nicht in ein anderes vorhandenes Stratum verlegt wird, sondern in ein für sie neu erzeugtes Stratum. Auf diese Art und Weise ist jeder Punkt des Suchraums von jedem anderen Punkt nach maximal $2|\mathcal{T}|$ Änderungsschritten erreichbar.

Das Erzeugen eines benachbarten Elementes wird wie folgt formal beschrieben:

$$\text{Nachbarschaft} : G_S \rightarrow G_S$$

Bemerkung. Es wird hier nicht darauf geachtet, ob durch die Verschiebung ein Zyklus in dem neu erzeugten Stratifikationsgraph entsteht, denn hier geht es lediglich darum, was genau passiert, wenn eine Transaktion von einem Stratum in ein anderes verschoben wird. Die Mechanismen zur Zyklenvermeidung und Zyklenerkennung werden in späteren Abschnitten behandelt.

Sei $G_S = (V_S, E_S)$ ein Stratifikationsgraph, und $G_a = (E_a, V_a)$ der dazugehörige Abhängigkeitsgraph. Man wählt eine Transaktion aus der Transaktionsmenge eines Stratums $s_x: t \in s_x$ ($s_x \in V_S$). Man wählt ein anderes Stratum s_y aus dem Graph oder erzeugt es als ein neues (leeres) Stratum. Man erzeugt aus G_S einen modifizierten Stratifikationsgraph G'_S :

$$\text{Nachbarschaft}(G_S) = G'_S, \quad G'_S = (V'_S, E'_S)$$

mit $V'_S = V_S \cup \{s'_x, s'_y\} \setminus \{s_x, s_y\}$ und $s'_x := s_x \setminus \{t\}$, $s'_y := s_y \cup t$.

Die Kanten des Stratifikationsgraphs müssen korrekt umgehängt werden. Im Folgenden wird beschrieben, wie die ausgehenden Stratifikationskanten geändert werden müssen. Bei den eingehenden Kanten ist das Vorgehen analog.

Man betrachtet alle ausgehenden Kanten (\tilde{E}_{out}) aus dem Stratum s_x . Bei dem Entfer-

nen der Transaktion t aus dem Stratum müssen die entsprechenden Kantenmengen der betroffenen Stratifikationskanten auf den aktuellen Stand gebracht werden. Dies betrifft die Stratifikationskanten, in deren Transaktionskantenmenge $edgeSet$ Transaktionskanten enthalten waren, die zu t inzident waren. Alle aus der Transaktion ausgehenden Kanten müssen aus den entsprechenden Listen der Stratifikationskanten gelöscht werden und in die entsprechenden Transaktionskantenmengen der evtl. auch neu entstandenen Stratifikationskanten aufgenommen werden.

Sei \mathcal{E}_{out} die Menge der ausgehenden Kanten des Transaktionsknotens t im Abhängigkeitsgraph G_a :

$$\mathcal{E}_{out} = \{e = (t, t_j) \in E_a | t_j \in V_a\}$$

Sei \tilde{E}_{out} die Menge der ausgehenden Kanten von dem Stratum s_x im Stratifikationsgraph G_S :

$$\tilde{E}_{out} = \{\tilde{e}_{xz} = (s_x, s_z) \in E_S | s_z \in V_S\}$$

Sei $edgeSet$ die Menge der Transaktionskanten, die in einer Stratifikationskante „abgebildet“ sind (wie in Abschnitt 3.4.3 definiert)

$$edgeSet : E_S \rightarrow \mathcal{P}(E_a)$$

Man betrachtet jede ausgehende Kante $\tilde{e}_{xz} \in \tilde{E}_{out}$ des Stratums und überprüft, ob deren Transaktionskantenmenge eine Kante beinhaltet, die von der Transaktion t ausgeht. Falls es eine solche Kante gibt, führt man die Aktualisierung der Transaktionskantenmenge durch.

Sei

$$K := edgeSet(\tilde{e}_{xz}) \cap \mathcal{E}_{out}$$

Falls $K \neq \emptyset$ müssen die Transaktionskanten aus der Menge K in die Transaktionskantenmengen der neuen Stratifikationskanten übernommen werden und aus der Menge $edgeSet(\tilde{e}_{xz})$ gelöscht werden.

Genauer: sei $e \in K$ mit $e = (t, t_j)$, und $t_j \in s_z$ ($s_z \neq s_y$), dann

$$edgeSet(\tilde{e}'_{xz}) := edgeSet(\tilde{e}_{xz}) / K$$

Sei $\tilde{e}_{yz} = (s_y, s_z)$ (falls diese Kante in E_S noch nicht vorhanden ist, wird sie erzeugt).

$$\text{edgeSet}(\tilde{e}'_{yz}) := \text{edgeSet}(\tilde{e}_{yz}) \cup K$$

$$E'_S = E_S \cup \tilde{e}'_{yz} \setminus \tilde{e}$$

Die so definierte Nachbarschaftsbeziehung ist allerdings lediglich struktureller Natur. Sie beschreibt also nur die strukturelle Ähnlichkeit der Elemente – es muss nicht sein, dass zwei benachbarte Individuen auch ähnliche Fitnesswerte besitzen und nahe aneinander auf der Gütelandschaft liegen. Zwar kann beides zutreffen: je nach den Prioritäten der jeweiligen Bewertungskriterien und nach den Eigenschaften der jeweiligen Transaktionen kann einerseits ein kleiner Änderungsschritt bezüglich der Struktur einen großen Sprung in den Gütewerten bedeuten, andererseits könnte der Nachbar auch einen ähnlichen Fitnesswert haben. Es ist schwer, eine Voraussage darüber zu treffen, wie ein struktureller Änderungsschritt die Fitness beeinflusst.

Bezüglich der Wahl des Initialzustands (der Initialelemente) können bei der Stratifikation je nach den Vorkenntnissen über das System verschiedene Strategien verfolgt werden. Die Extrembeispiele eines Initialelements bezüglich seines strukturellen Aufbaus wären Stratifikationen der Art „eine Transaktion – ein Stratum“ (Bottom-Up Ansatz) und „alle Transaktionen – ein Stratum“ (Top-Down Ansatz). Die Anfangsstratifikation kann auch zufällig gewählt werden (besonders, wenn kein Vorwissen vorhanden ist), oder nach einem Muster aufgebaut werden (siehe Abschnitt 5.6).

5.1.4. Die Nebenbedingungen

Es sind eine ganze Reihe Nebenbedingungen - d.h. Anforderungen an die Stratifikation denkbar; hier werden einige davon erläutert.

Beschränkungen bezüglich der Größe eines Stratums In einem Stratum darf nur eine bestimmte maximale Anzahl von Transaktionen liegen.

Beschränkungen bezüglich der Anzahl der Strata in Form einer Angabe der minimalen und maximalen Anzahl der Strata, abhängig von der Anzahl der Transaktionen.

Beschränkungen bezüglich der Kantenanzahl des Stratifikationsgraphes. Diese Nebenbedingung würde zu Problemen bei der Graphpartitionierung führen.

Beschränkungen bezüglich der Anzahl der eingehenden/ausgehenden Kanten eines Stratums. Hierfür könnten wiederum Angaben zur maximalen oder minimalen Anzahl der ein-/ausgehenden Kanten dienen.

Azyklität. Eine notwendige Bedingung der Korrektheit einer Stratifizierung besteht darin, dass der neu entstandene Stratifikationsgraph keine Zyklen enthält (siehe auch Abschnitt 3.6). Es gibt mehrere Alternativen, wie man diese Bedingung erfüllen kann. Es fällt auf, dass dieses Problem Analogien zu einer Deadlock-Erkennung bei Verteilten Systemen hat. Einerseits kann man das Entstehen von einem Zyklus dadurch vermeiden, dass man nur korrekte Änderungen durchführt. Andererseits kann man zuerst die Änderung durchführen und dann prüfen, ob ein Zyklus entstanden ist. Diese beiden Alternativen werden im Folgenden genauer diskutiert.

Die Azyklität eines Stratifikationsgraphes ist im Gegenteil zu anderen Nebenbedingungen eine absolut notwendige Voraussetzung zur Korrektheit des Stratifikationsgraphen. Die restlichen Nebenbedingungen können je nach Bedarf beachtet werden, deren Einhaltung bereitet auch meist keine grösseren Komplikationen. Die Bewahrung der Azyklität ist jedoch nicht so einfach zu bewerkstelligen. Dieses wird in den nächsten Unterabschnitten betrachtet.

5.1.4.1. Erkennung der Zyklen nach der Konstruktion eines Nachbargraphen

Falls der Graph dünn ist, also nur wenige Kanten besitzt, kann die Zyklenerkennung auch erst nach der vorgenommenen Änderung durchgeführt werden. Da nur wenige Kanten vorhanden sind, ist es weniger wahrscheinlich, dass bei der Erzeugung von „benachbarten“ Strukturen ein Zyklus entsteht. Die meisten Änderungsschritte im Sinne von Abschnitt 5.1.3 würden demnach also als „korrekt“ eingestuft. Somit sollte man eine Erkennung der Zyklen besser erst nach der Änderung einsetzen.

Diese Zyklenerkennung kann z.B. mittels Tiefensuche realisiert werden. Es wird eine Markierung eingeführt, die die Knoten in drei Zustände einteilt: „noch nicht bearbeitet“, „in Bearbeitung“, „bereits bearbeitet“. Falls man während des Durchlaufs des

Graphen auf einen Knoten stößt, der sich in dem Zustand „in Bearbeitung“ befindet, so bedeutet das, dass es einen Pfad gibt, der an diesem Knoten anfängt und an ihm auch endet – es existiert also ein Zyklus. Siehe dazu auch [13]. Für jeden der Wurzelknoten des Stratifikationsgraphen wird folgendes Vorgehen angewendet.

```

ZYKLENSUCHE(KNOTEN X)
IF (MARKIERUNG(X) = "IN BEARBEITUNG") THEN ZYKLUS GEFUNDEN
SONST
    IF (MARKIERUNG(X) = "NOCH NICHT BEARBEITET") THEN
        MARKIERUNG(X) := "IN BEARBEITUNG";
         $\forall y: y \in \text{Nachfolger}(x): \text{ZYKLENSUCHE}(Y);$ 
    MARKIERUNG(X) := "BEREITS BEARBEITET";

```

5.1.4.2. Vermeiden von Zyklen. Konstruktion eines bereits zyklensfreien benachbarten Stratifikationsgraphen

Falls der Graph dicht ist (die Anzahl der Kanten ist in etwa quadratisch zur Anzahl der Knoten: $|E| \approx |V|^2$), ist die Wahrscheinlichkeit hoch, dass bei der Konstruktion eines „Nachbargraphen“ Schleifen entstehen. Die Anzahl der fehlerhaften Konstruktionsversuche ist somit unnötig hoch.

Die Einführung von Zyklen kann verhindert und damit dieser Aufwand reduziert werden, wenn von vornherein nur korrekte Graphen erzeugt werden.

Es werden folgende Invarianten beobachtet:

- Eine Transaktion kann aus dem Stratum ohne Verletzung der Azyklitätbedingung dann entfernt werden, falls sie entweder eine „Wurzel-“ oder eine „Blatttransaktion“ innerhalb dieses Stratums ist.

Transaktionen t_x und t_y können in demselben Stratum s liegen falls:

- sie mit einer Kante verbunden sind
- alle Knoten, die auf dem Weg von einem Transaktionsknoten zu der anderen Transaktionsknoten liegen, in demselben Stratum s sind

- zwischen den beiden kein Weg existiert

Die zweite Bedingung beinhaltet eigentlich auch die Bedingung eins und drei, diese werden hier zum besseren Verständnis eingeführt.

Befolgt man bei der Wahl der Transaktion und des „Zielstratums“ (eines aus der Menge der „erlaubten“ Strata) diesen Regel, kann die Verschiebung der Transaktion von dem alten Stratum in das neue durchgeführt werden: der so erzeugte Stratifikationsgraph wird azyklisch sein.

Sei $G_S = (V_S, E_S)$ der Stratifikationsgraph mit $V_S = \{s_1, s_2, \dots, s_n\}$ als Strata.

Man wählt ein Stratum s_a aus der Menge aller vorhandenen Strata zufällig aus.

Aus der Menge der Wurzel- und Blättertransaktionen der Strata wird ein Transaktionsknoten t zufällig gewählt:

$$t \in \mathcal{W}_T(s_a) \cup \mathcal{B}_T(s_a)$$

Der Grund hierfür ist die Einhaltung der oben beschriebenen Bedingung (siehe dazu die Abbildung 5.2).

Für die gewählte Transaktion t wird zunächst probabilistisch entschieden, ob diese Transaktion in ein vorhandenes Stratum oder in ein neu erzeugtes Stratum verlegt werden soll. Falls ein neues Stratum erzeugt werden soll, muss man keine weiteren Massnahmen zur Erhaltung der Azyklität ergreifen. Die Transaktion wird wie im Abschnitt 5.1.3 in das neue Stratum verschoben.

Falls aber entschieden wurde, dass die Transaktion in ein vorhandenes Stratum verschoben werden soll, wird zunächst die Menge der „verbotenen“ Strata berechnet. Aus den restlichen Strata ergibt sich dann die Menge der „erlaubten“ Strata, in die die Transaktion verschoben werden dürfte. Der Menge der erlaubten Strata gehören also solche Strata an, für die der längste Weg von dem gewählten Stratum maximal der Länge 1 ist. Es können also direkte Vorgänger oder Nachfolger des gewählten Stratums sein, aber auch Strata, die mit dem gewählten Stratum gar nicht direkt verbunden sind.

Sei $s := Stratum(t)$. Betrachte die Menge der direkten Vorgänger und Nachfolger dieses Stratums: $\mathcal{V}(s)$, $\mathcal{N}(s)$. Sei \mathcal{S}_{tabu} - die Menge der „verbotenen“ Straten.

Verschiebung der Transaktion t_{i7}

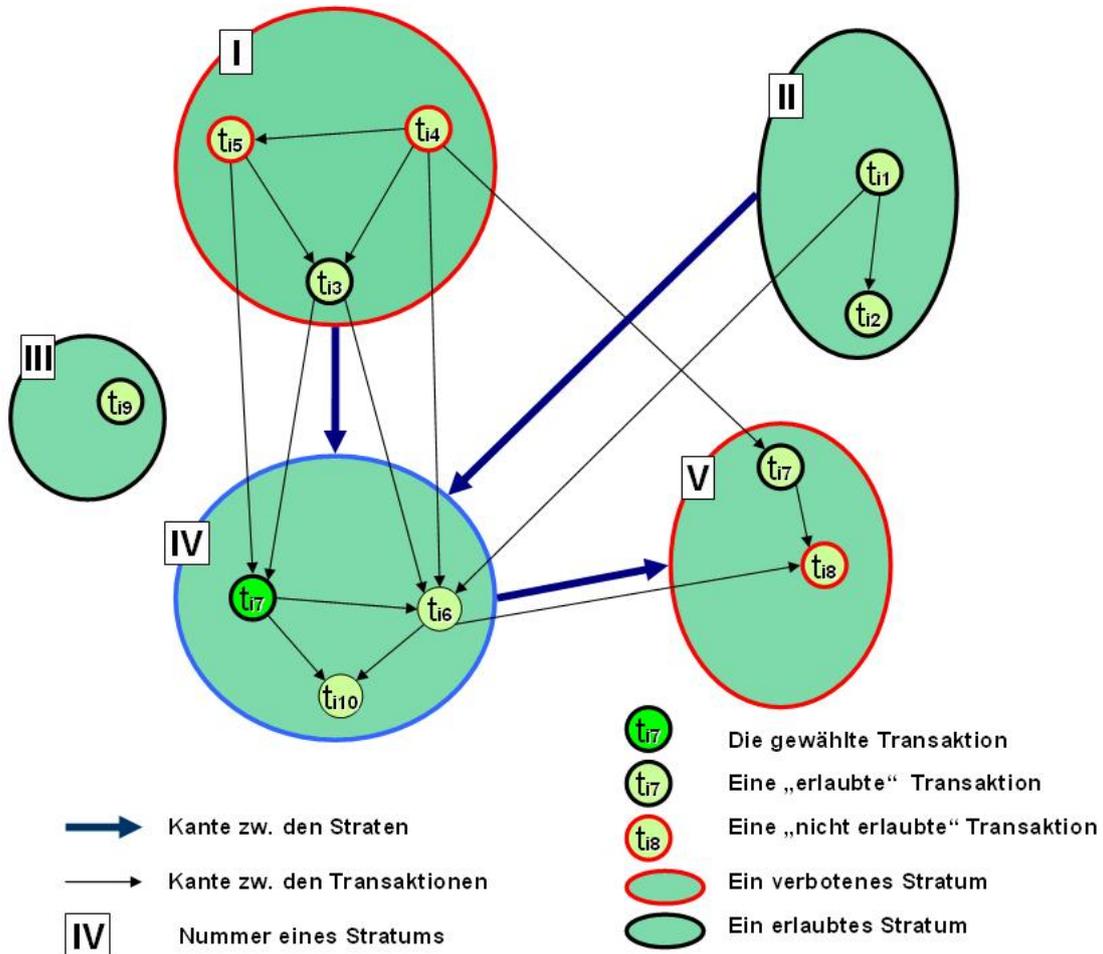


Abbildung 5.2.: Änderungsschritt. Die Transaktion t_{i7} (eine Wurzeltransaktion von Stratum IV) wird verschoben. Die Transaktion darf nicht in die rot markierten Strata verschoben werden, da sonst ein Zyklus im Stratifikationsgraph entstehen würde. Die erlaubten Strata in diesem Fall sind die mit der dicken schwarzen Umrandung: Stratum II oder III.

```

 $\forall s_i \in \mathcal{V}(s):$ 
PROCEDURE TABUVORGÄNGER( $s_i$ )
FALLS  $markierung(s_i) = \text{“false”}$ 
     $markierung(s_i) := \text{„true“};$ 
     $\mathcal{S}_{tabu} := \mathcal{S}_{tabu} \cup s_i$ 
     $\forall s' \in \mathcal{V}(s_i):$  TABUVORGÄNGER( $s'$ )
SONST  $\mathcal{S}_{tabu} := \mathcal{S}_{tabu} \cup s_i$ 
END PROCEDURE

```

```

 $\forall s_j \in \mathcal{N}(s):$ 
PROCEDURE TABUNACHFOLGER( $s_j$ )
FALLS  $markierung(s_j) = \text{“false”}$ 
     $markierung(s_j) := \text{„true“};$ 
     $\mathcal{S}_{tabu} := \mathcal{S}_{tabu} \cup s_j$ 
     $\forall s'' \in \mathcal{N}(s_j):$  TABUNACHFOLGER( $s''$ )
SONST  $\mathcal{S}_{tabu} := \mathcal{S}_{tabu} \cup s_j$ 
END PROCEDURE

```

Die Menge der erlaubten Strata ist dann $\mathcal{S}_{erlaubt}(t) := \mathcal{S} \setminus \mathcal{S}_{tabu}$

Nun wählt man aus der berechneten Menge der erlaubten Strata zufällig ein Stratum aus und verschiebt die Transaktion t dahin so, wie es bereits im Abschnitt 5.1.3 beschrieben wurde.

5.2. Realisierung von Hillclimbing

Als Startelement wird ein zufälliges Element des Suchraums gewählt. Die Schritt-funktion des Hillclimbings wählt ein zufälliges Element aus der Nachbarschaft, und falls dieses besser ist als das aktuelle Element, wird es übernommen. Das Auswahlverfahren des Nachbarelements wurde im Abschnitt 5.1.3 beschrieben. Es wurde die Strategie der Zyklenvermeidung eingesetzt, da es keine genauen Angaben über die Struktur des Abhängigkeitsgraphen (und also der Anzahl seiner Kanten) gab. Es wird auch stets das beste bei der Suche je gesehene Element gespeichert, was auch

dann das Ergebnis der Suche sein wird.

5.3. Realisierung von Simulated Annealing

In diesem Abschnitt wird die Realisierung des Simulated-Annealing-Verfahrens zur Stratifikation erläutert. Die wichtigsten Parameter bei diesem Verfahren sind die Wahl der Abkühlungsplans sowie die Initialbelegung des Temperaturparameters T . Zur Suche nach der optimalen Stratifikation wurde ein geometrisches Abkühlungsschema gewählt mit dem Parameter $\alpha = 0.99$, so dass das Verfahren nur langsam die Schwelle der Akzeptanz verkleinert. Auch bei der Realisierung von Simulated Annealing wird das beste bei der Suche je gesehene Element gespeichert, was auch dann das Ergebnis sein wird.

5.3.1. Initialbelegung des Parameters T

Die initiale Belegung des Parameters T ist für das Verfahren von großer Bedeutung. Falls T zu hoch gewählt wird, ist die Wahrscheinlichkeit, ein schlechteres Element als Nachfolger zu wählen, anfangs bei 1.0. Es werden somit alle Elemente gewählt – egal, wie schlecht diese sind –, was natürlich nicht erwünscht ist.

Falls der Parameter T dagegen zu niedrig ist, werden bereits zu Beginn die schlechteren Lösungskandidaten nicht akzeptiert, und das Verfahren entspricht schon hier dem Hillclimbing.

Ein Verfahren zur Ermittlung des Parameters ist die Abschätzungen der minimalen und maximalen Bewertung für den vorgegebenen Abhängigkeitsgraph. Der Parameter wird dabei initial auf die Differenz der Fitnesswerte des schätzungsweise schlechtesten und schätzungsweise besten Elements des Suchraums gelegt. Dieses wird im folgenden erläutert.

Abschätzung von f_{max} und f_{min}

Man betrachtet die Menge der Bewertungskriterien: $\mathcal{E} = (\mathcal{E}_1, \mathcal{E}_2, \dots, \mathcal{E}_n)$. Für jedes dieser Bewertungskriterien wird im Folgenden der schlechteste Fall beschrieben. Für die Abschätzung der f_{max} wird also für jedes Kriterium sein Worst-Case-Szenario verwendet. Es gibt keine passende Stratifikation, die alle im folgenden beschriebenen

Worst-Case-Szenarios beinhaltet. Aber das ist auch nicht nötig, denn hier wird eine kleinste obere Schranke (Supremum) gesucht. Der so konstruierte f_{max} ist für die Zwecke der Parameterbelegung von T bestens geeignet.

- Für die Invokationskosten sowie für Recoverykosten wäre der schlechteste Fall, wenn alle Transaktionen in einem gemeinsamen Stratum wären. Die Wahrscheinlichkeit, dass eine der Transaktionen aus irgendwelchen Gründen abgebrochen wird, und dadurch auch die Arbeit des gesamten Stratums wiederholt werden muss, ist um so höher, je mehr Transaktionen in einem Stratum liegen, und desto höher sind entsprechend die Invokations- und Recoverykosten.
- Für die Ausführungszeit gilt dasselbe wie bei Invokationskosten. Zwar müssen die Transaktionen nicht auf die anderen Strata warten, aber falls die Arbeit wegen einer abgebrochenen Transaktion wiederholt werden muss, ist die akkumulierte Ausführungszeit länger.
- Die Antwortzeit ist maximal bei einem Stratifikationsgraph, in dem alle Transaktionen in demselben Stratum liegen.
- Die Kosten der 2-Phase-Commit-Protokollnachrichten steigt mit der Anzahl der Transaktionen in Stratum.
- Die Kosten der Queue zwischen den Strata sind dagegen um so größer, je mehr Strata und dazwischenliegende Kanten es insgesamt gibt.
- Die Kosten, die mit dem Überschreiten der Fälligkeitszeitpunkt verbunden sind, werden am höchsten, je mehr Transaktionen in dem selben Stratum liegen; oder falls es in jedem Stratum mindestens eine Transaktion mit hoher Abort-Wahrscheinlichkeit (pro Lauf) gibt.
- Der Grad der Nebenläufigkeit ist im Sinne der in Abschnitt 3.5.3 beschriebenen aufgesparten Zeit definiert. Diese ist im schlechtesten Fall gleich Null.

Es werden also zur der Abschätzung von f_{min} und f_{max} zwei Extremfälle betrachtet: ein Stratifizierungsgraph, in dem alle Transaktionen einem Stratum angehören, und ein Stratifizierungsgraph, in dem jede Transaktion in ihrem eigenen Stratum liegt.

5.4. Hybrider Ansatz: Kombination von Hillclimbing und Simulated Annealing

In der vorliegenden Arbeit wurde auch eine Kombination der zwei bereits vorgestellten Verfahren – Hillclimbing und Simulated Annealing – untersucht. Die Schwäche von Simulated Annealing liegt eben darin, dass es möglicherweise in der Anfangsphase zu sehr von den guten Lösungen abdriftet. In der Kombination von Hillclimbing und Simulated Annealing wird daher versucht, die Stärken der beiden Verfahren zu vereinigen.

Das kombinierte Verfahren besteht aus zwei Phasen, die sich immer wieder abwechseln. In der ersten Phase arbeitet das Verfahren nach dem Hillclimbing-Prinzip. Sobald die Entwicklung zu stocken beginnt und nach n Schritten am Stück keine Verbesserung gefunden wird, geht das Verfahren in die zweite Phase über: die Bedingungen werden gelockert, auch schlechtere Lösungen werden mit einer gewissen Wahrscheinlichkeit akzeptiert. Falls der Entwicklungsprozess an einem lokalen Optimum angelangt ist und das Hillclimbing nicht mehr herausfindet, geht man zu „Plan B“ über – zur Simulated-Annealing-Strategie. Simulated Annealing wird dann für die Dauer von m Schritten verfolgt. Nach den m Schritten wird wieder auf Hillclimbing übergegangen. Vor der „Phase B“ wird eine Kalibrierung vorgenommen, damit ein „Erhitzen“ in Abhängigkeit vom aktuellen Zustand der Suche durchgeführt wird.

Kalibrierung Es werden n Stratifizierungen erzeugt. Dann wird der Medianwert ihrer Bewertungen genommen $f(y_{median})$. Der Parameter T wird so bestimmt, dass ein Element mit der Fitness gleich dem Medianwert mit der Wahrscheinlichkeit β (z.B. $\beta = 0.1$) akzeptiert wird, falls es schlechter als das aktuelle Element x ist.

$$T = \frac{-|(f(y_{median}) - f(x))|}{\log(\beta)}$$

```

RESULT:=MAXREAL;
WHILE (K < ANZAHLVONSCHRITTEN)
/* Phase 1: Hillclimbing */
    HILLCOUNTER:=0;
    WHILE(HILLCOUNTER < MAXHILLCLIMBING UND K < ANZAHLVONSCHRIT-
TEN )
        BEGIN
            WÄHLE  $y \in U(x)$ 
            IF  $f(y) < f(x)$   $x := y$ , RESULT = MIN {RESULT,  $f(y)$  }
            ELSE HILLCLIMBINGCOUNTER++;
             $k++$ ;
        END WHILE

/* Phase 2: Simulated Annealing */
    WÄHLE  $\{y_i\}_{1 < i < h}$  MIT  $\forall y_i \in U(x)$ ,
    BETRACHTE AUS  $\{y_i\}$  DAS MEDIAN BZGL. DER FITNESS  $y_{median}$ 
     $T = -|(f(y_{median}) - f(x))|/\log(\beta)$ ;

/* Konditionierung von Parameter T */
    SACOUNTER:=0;
    WHILE(SACOUNTER < MAXSA UND K<ANZAHLVONSCHRITTEN)
        BEGIN
            WÄHLE  $y \in U(x)$ 
            IF  $f(y) < f(x)$   $x := y$ ; RESULT = MIN {RESULT,  $f(y)$  }
            ELSE
                IF ZUFALLSZAHL[0, 1] <  $e^{-\frac{|f(y)-f(x)|}{T}}$ 
                     $x := y$ ; ,  $T := \alpha \cdot T$ , RESULT = MIN {(RESULT,  $f(y)$  }
                END IF
            END IF
            SACOUNTER++;
             $k++$ ;
        END WHILE
    END WHILE
END WHILE

```

5.5. Realisierung der Evolutionssimulation

Die Vorteile der Evolution – die Erhaltung der Diversität durch Mutation, die durch Selektion gerichtete Entwicklung der Lösungen und die Möglichkeit, positive Eigenschaften durch Rekombination zu bündeln – legen den Schluss nahe, dass evolutionäre Algorithmen wohl die besten Ergebnisse bei der Stratifizierung erzielen können wird.

5.5.1. Initialisierung der Population

Als Startelement des Verfahrens wird eine Stratifikation der Form „eine Transaktion pro Stratum“ gewählt.

Die Population wird ausgehend von einem Startelement initiiert. Jedes Individuum der Population wird aus dem Startelement durch jeweils eine normalverteilte Anzahl von Malen (abhängig von Anzahl der Transaktionen) durch den elementaren Änderungsschritt (siehe dazu den Abschnitt 5.1.3) verschoben. Damit versucht man, eine möglichst gute Verteilung über den gesamten Suchraum zu erzeugen.

5.5.2. Mutation

Es werden m Elemente aus der aktuellen Population ausgewählt und mutiert. Die Mutation ist hier nichts anderes als der im Abschnitt 5.1.3 definierte zufällige Änderungsschritt. Die Mutation kann wie im Kapitel 4.4.6 beschrieben entweder eine erforschende oder eine feinabstimmende Funktion haben. Da aber nicht bekannt ist, wie sich eine strukturelle Änderung auf die Größe des Sprungs in der Fitnesslandschaft auswirkt, kann eine so definierte Mutation in bestimmten Fällen wie eine Makromutation und in anderen Fällen wie eine Mikromutation wirken.

5.5.3. Elternselektion und Rekombination

Die Elternselektion bei der Evolutionssimulation kann zufällig verlaufen.

Die Rekombination im evolutionären Algorithmus muss sehr gut überlegt sein. Es werden im Folgenden zwei Ansätze zur Rekombination vorgeschlagen.

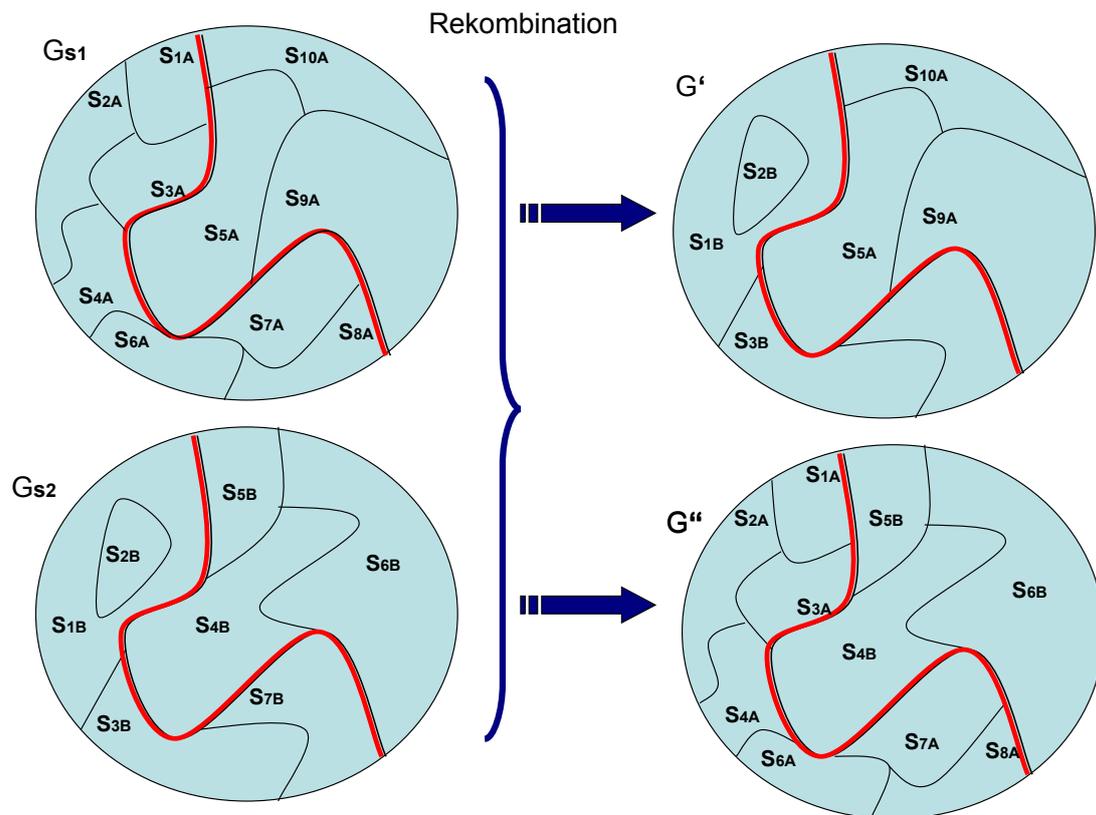


Abbildung 5.3.: Rekombination. Version 1

5.5.3.1. Allgemeiner Ansatz

Der allgemeiner Ansatz zur Rekombination geht von zwei Elternteilen aus. Rekombinationen mit mehr als zwei Elternteilen können durch eine Kette dieser „einfacheren“ Rekombinationsschritte realisiert werden.

Die Idee des Verfahrens ist die folgende. Ein Elternteil wird zufällig aus der Population gewählt. Es wird versucht, einen anderen Elternteil in der Population zu finden, so dass diese beiden Stratifikationsgraphen einen gültigen „gemeinsamen Schnitt“ besitzen – siehe dazu die Abbildung 5.3.

Sei also $P = \{G_{S_1}, G_{S_2}, \dots, G_{S_k}\}$ eine Population bestehend aus k verschiedenen Stratifikationsgraphen.

Man wählt ein Elternteil $R_1 := G'_S = (V'_S, E'_S)$ aus der Population P zufällig aus. Sei $V'_S = \{s'_1, s'_2, \dots, s'_n\}$ die Menge aller Strata des Stratifikationsgraphs G'_S . Man wählt aus der Menge der Strata des ersten Elternteils eine Teilmenge $V' \subseteq V'_S$

Sei also G''_S ein Kandidat für die Auswahl als zweiter Elternteil.

Man betrachtet die Menge der Transaktionen \mathcal{T}' , die zu den ausgewählten Strata gehören: $\mathcal{T}' = \{t'_{i_1}, t'_{i_2}, \dots, t'_{i_k}\}$, das ist die Menge der Transaktionen, die der Menge der Strata V' entsprechen.

Man betrachtet nun die entsprechenden Transaktionen $\mathcal{T}'' (\equiv \mathcal{T}')$ in dem Stratifikationsgraph G''_S : $\mathcal{T}'' = \{t''_{i_1}, t''_{i_2}, \dots, t''_{i_k}\}$.

$\forall t''_{i_j} \in \mathcal{T}''$: falls $\exists t'_x \in \text{Stratum}(t''_{i_j})$, so dass $t'_x \notin \mathcal{T}'$, dann ist der Kandidat G''_S zur Rekombination mit G'_S nicht geeignet.

Falls die Überprüfung erfolgreich beendet wurde und sich der Rekombinationskandidat als tauglich erwiesen hat, können beide Elternteile die gewählten Stratifikationsmengen tauschen, wobei die Kanten zwischen den Strata dann entsprechend umgehängt werden müssen.

Wie hoch ist die Wahrscheinlichkeit, in der Menge der Stratifikationsgraphen ein passendes Rekombinationspaar zu finden? Falls die Diversität der Population gering ist, also alle Individuen durch wenige Änderungsschritte (Mutationsschritte) ineinander überführt werden können, ist die Wahrscheinlichkeit, solche Paare zu finden, höher, als wenn die Individuen der Population in der Entwicklung weit auseinander gedriftet sind.

5.5.3.2. Vereinfachte Version

Dieser Ansatz versucht lediglich aus jedem der Elternteile ein Stratum zu übernehmen, die restlichen Transaktionen werden in eigenen Straten positioniert. Die Bedingung dazu, dass eine Rekombination zweier Elemente möglich ist, ist lediglich, dass diese jeweils ein Stratum beinhalten, deren Transaktionsmengen sich nicht überschneiden.

Seien $R_1 := G'_S = (V'_S, E'_S)$ und $R_2 := G''_S = (V''_S, E''_S)$ zwei Elternteile.

Man wählt ein Stratum s'_i aus dem ersten Elternteil, und man wählt in dem zweiten Elternteil ein Stratum s'_j so, dass die entsprechenden Transaktionsmengen der Strata sich nicht überschneiden:

$$s'_i = \{t_{i_1}, t_{i_2}, \dots, t_{i_n}\} := \mathcal{T}', \quad s'_j = \{t_{j_1}, t_{j_2}, \dots, t_{j_n}\} := \mathcal{T}'' \text{ mit } \mathcal{T}' \cap \mathcal{T}'' = \emptyset$$

Nun erzeugt man ein „Kind“-Individuum G_S^k derart, dass es die beiden Strata beinhaltet, während die restlichen Transaktionen in einzelnen Strata positioniert werden.

Zur Illustration siehe Abbildung 5.4.

5.5.4. Umweltselektion

Als Umweltselektion innerhalb des evolutionären Algorithmus zur Stratifizierung wurde die Turnirselektion ($n : 1$) verwendet. Es wurden jeweils n Individuen zufällig aus der Population gewählt, die dann in einem Turnier gegeneinander angetreten sind. Der Sieger des Turniers wurde dann in die Population der nächsten Generation übernommen. Es wurde die Komma-Ersetzungstrategie verwendet.

5.6. Verwendung von Mustern bei der Stratifikation

Im folgenden Abschnitt werden Überlegungen dargestellt, wie das Wissen über erwartungsgemäß „gute“ Stratifikationen in heuristische Methoden eingebracht werden kann. In spezifischen Fällen, in denen man genaue Angaben über die Eigenschaften einer optimalen Stratifikation hat, kann das Einbringen dieses Wissens in Form von Mustern positiven Einfluss auf die Suche haben. Ein durch die Verwendung von Mustern erstelltes Stratifikationselement könnte z.B. auch als ein vermutlich gutes Startelement für Simulate Annealing oder Hillclimbing verwendet werden. Dennoch, wie bereits in dem Kapitel 4 erwähnt, ist das Einbringen von Mustern bei den heuristischen Verfahren im allgemeinen mit Vorsicht zu genießen.

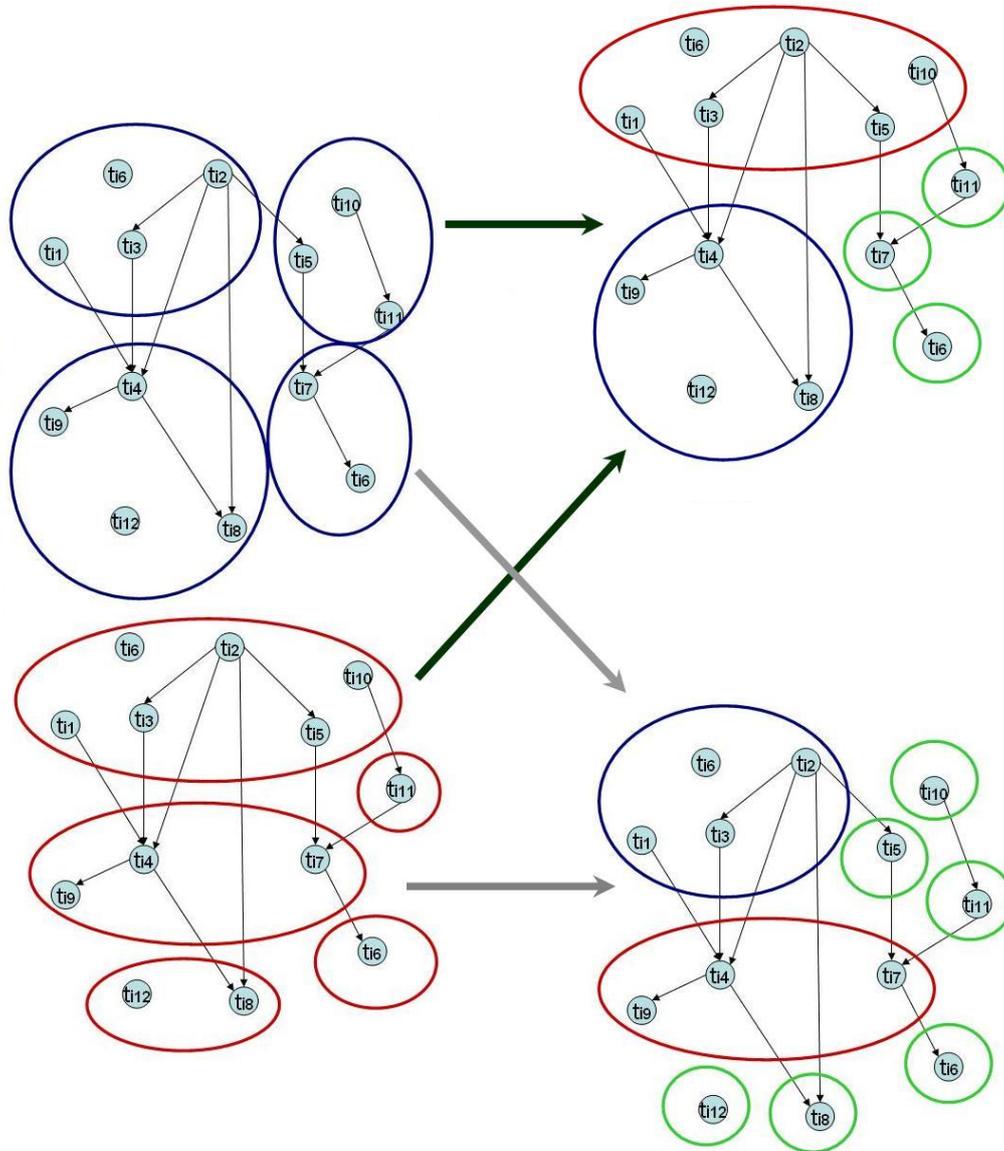


Abbildung 5.4.: Rekombination. Variante 2

5.6.1. Muster: Stratifizierung nach einer mehrwertigen Eigenschaft (z.B. Lokation)

Der Grundgedanke hier ist, diejenigen Transaktionen in einem Stratum zu vereinigen, die dieselben Werte bezüglich einer bestimmten Eigenschaft haben. Eine solche Eigenschaft ist z.B. die Lokationen der Transaktionen und die Lokationen der von Transaktion verwendeten Ressourcen.

Sei $G_S = (V_S, E_S)$ ein Stratifikationsgraph, in welchem jedes Stratum genau einen Transaktionsknoten beinhaltet (also $\forall s_j : |s_j| = 1$). Sei $G_a = (V_a, E_a)$ der dazugehörige Abhängigkeitsgraph, mit Transaktionen $V_a = \mathcal{T} = \{t_1, \dots, t_n\}$.

Es wird zuerst der Fall betrachtet, dass der Abhängigkeitsgraph wenige bis keine Kanten besitzt: $|E_a| \ll |V_a|$.

Aus diesem Abhängigkeitsgraph wird ein Stratifikationsgraph gebildet, in dem die Transaktionen eines Stratums möglichst dieselben Werte bezüglich einer bestimmten Eigenschaft haben.

Sei p die gewählte mehrwertige Eigenschaft der Transaktionen.

Sei $Belegungen = \{b_j\}_{1 < j < |V_a|}$ die Menge der Belegungen der Transaktionen bezüglich dieser Eigenschaft:

$$\forall i \in 1, \dots, j : b_i = p(t_i) = \{v_{i_1}, v_{i_2}, \dots, v_{i_{g_i}}\}$$

wobei $Values = \{v_1, v_2, \dots, v_k\}$, $k \in \mathbb{N}$ die Menge der möglichen Belegungen.

Man erzeugt einen so genannten Schnittgraph, $G_\cap = (V_\cap, E_\cap)$, so dass dessen Knoten den Transaktionsknoten entsprechen $V_\cap = V_a$, und die Kanten wie folgt definiert werden:

$$\exists e = (t_i, t_j) \Leftrightarrow b_i \cap b_j \neq \emptyset, \text{ wobei } b_i = p(t_i), \quad b_j = p(t_j)$$

Sei dann auch eine Beschriftung der Kanten eingeführt:

$$\mathcal{B} : E_\cap \rightarrow \mathcal{P}(Value)$$

$$e = (t_i, t_j) \quad \mathcal{B}(e) = b_i \cap b_j = p(t_i) \cap p(t_j)$$

Man sucht in dem so erzeugtem Schnittgraph nach einer Cliquesüberdeckung², die

²Eine Clique nennt man einen Graph, in dem jeder Knoten mit jedem anderen verbunden ist

bezüglich der Eigenschaft „optimal“ ist. Dies ist wie folgt definiert.
 Eine $Clique(p) = \{b_{i_1}, b_{i_2}, \dots, b_{i_q}\}$ ist eine Menge der Knoten, so dass
 $\forall b_j, b_k \in Clique\ es\ gilt : \exists e \in E_\cap\ mit\ e = (b_j, b_k)$
 und es gilt:

$$\mathcal{B}(b_{i_1}) \cap \mathcal{B}(b_{i_2}) \cap \mathcal{B}(b_{i_3}) \cup \dots \cap \mathcal{B}(b_{i_q}) \neq \emptyset$$

Es wird angestrebt, dass

$$|\mathcal{B}(b_{i_1}) \cap \mathcal{B}(b_{i_2}) \cap \mathcal{B}(b_{i_3}) \cup \dots \cap \mathcal{B}(b_{i_q})| \rightarrow maximal$$

Da es keine oder nur wenige Kanten gibt, kann man den Test auf eventuell entstandene Zyklen hinausschieben, bis man die optimale Cliquesüberdeckung gefunden hat. Am besten baut man diese Cliques schrittweise auf und vergewissert sich dann jedes Mal, dass keine Zyklen entstanden sind.

Jede gefundene Clique kann dann einem Stratum entsprechen. Weiterhin könnten statt Cliques solche Teilgraphen gesucht werden, die einen Cluster (eine Zusammenhangskomponente) bilden.

Bei dichten Abhängigkeitsgraphen ist das Auffinden einer solchen optimalen Stratifizierung deutlich komplizierter. Die Wahrscheinlichkeit, dass bei der Vereinigung zweier Transaktionen, die sich in einer Eigenschaft ähneln, ein Zyklus entsteht, ist hoch. Hier sollte daher eher eine Zyklenvermeidungsstrategie angewendet werden. Zur Feststellung der Cliques eignen sich kleine Schritte. Dafür wählt man zufällig einen Transaktionsknoten, betrachtet seine Vorgängerknoten und Nachfolgerknoten.

Das Finden von guten Musterstratifikationen ist ein Problem für sich. Das Auffinden einer maximalen Clique ist z.B. ein NP-vollständiges Problem. Für die praktische Anwendung eignen sich also solche Musterkonstruktionen, bei denen man auf die Optimalität verzichtet und nach einer einfacheren Struktur fahndet.

Teil III.

Realisierung

6. Evaluation

Dieses Kapitel betrachtet die experimentell ermittelten Ergebnisse der Stratifizierung durch die in Kapitel 5 beschriebenen Verfahren Hillclimbing, Simulated Annealing, den hybriden Ansatz von beiden sowie das evolutionäre Programmieren.

Um die Verfahren vergleichen zu können, arbeiten sie auf demselben zufällig generierten Abhängigkeitsgraph. Es wurden verschiedene Testreihen mit Abhängigkeitsgraphen verschiedener Eigenschaften durchgeführt. Im folgenden wird der Aufbau des Zufallsgraphs beschrieben und die Ergebnisse der jeweiligen Testreihen vorgestellt.

6.1. Zufallsgraph

Zu den Eingabedaten für die Verfahren gehören die Menge der Transaktionen und ein zyklensfreier Abhängigkeitsgraph auf den Transaktionen.

Um einen solchen Graph zu generieren, müssen also zunächst die Transaktionsknoten samt aller ihrer Eigenschaften mit Wertebelegungen generiert werden. Es müssen weiterhin die Kanten des Graphes zufällig erstellt werden.

Generieren von Transaktionsknoten. Die Transaktionen haben Eigenschaften, die genau im Kapitel 3.3 definiert wurden. In dieser spezifischen Realisierung gehören zu den Eigenschaften: Zeit der Verfügbarkeit, Zeitpunkt der Fälligkeit, Invokationskosten, Recoverykosten, sowie auch die Menge der Ressourcen und deren Lokationen und weitere Eigenschaften. Für Testzwecke wurden für die jeweiligen Eigenschaften Wertebereiche gewählt, aus denen dann ein zufälliger Wert erzeugt wurde.

Kanten. Zu Erstellung der Kanten des Abhängigkeitsgraphs wurden jeweils zufällige Paare von Transaktionsknoten gewählt. Hier musste beachtet werden, dass die

der so generierte Abhängigkeitsgraph zyklensfrei bleibt. Dies wurde so realisiert, dass es immer nur Kanten von Transaktionsknoten mit kleinerer Identifikationsnummer zu Transaktionen mit grösserer Identifikationsnummer erzeugt wurden.

Man muss beachten, dass die Bewertung eines solchen Stratifikationsgraphen relativer Natur ist – sie soll nicht den absoluten Wert der anfallenden Kosten darstellen, sondern lediglich den Vergleich zwischen verschiedenen Elementen des Suchraums ermöglichen.

6.2. Experimenteller Vergleich der Verfahren

Um einen statistisch vollständigen experimentalen Vergleich der Verfahren machen zu können, müsste man für eine grosse Menge (mindestens 100) zufällig erzeugter Graphen jeweils mindestens etwa 100 Läufe durchführen. Eine vollständige Abdeckung des möglichen Verhaltens der Verfahren bei allen möglichen Konstellationen geht aber über den Rahmen dieser Diplomarbeit hinaus.

In der vorliegenden Arbeit wurden daher Experimentalläufe für einen dichten Graph (mit vielen Kanten), einen dünnen Graph (einen Graph mit wenigen Kanten) und einen Graph mit einer durchschnittlichen Anzahl von Kanten durchgeführt.

Für jede der Testreihen wurden pro Schritt die Medianwerte des Fitness der Elementen aller Läufe des Verfahrens ermittelt. Diese werden dann für jede Testreihe abgebildet. Es ist zu beachten, dass diese Abbildungen lediglich das Verhalten der Verfahren darstellen, daraus ist nicht das Endergebnis der Suche ersichtlich. Bei der Evolutionssimulation wurden jeweils der beste in der Population vorhandene Fitnesswert ermittelt, die Abbildungen der Evolutionssimulation zeigen jeweils Medianwerte der Fitness von allen Läufen pro Generation (Schritt). Bei allen Verfahren wird in jeder Ausführung das beste während des Suchprozesses gefundene Element gespeichert und am Ende der Suche als Ergebnis ausgeliefert.

Pro Testreihe wurde ein Zufallsgraph generiert, die Fitnesswerte der Testreihen sind also untereinander nicht vergleichbar, weil sie sich aus völlig verschiedenen

Strukturen ergeben. Innerhalb einer Testreihe spricht aber der Fitnesswert für die Qualität der Ergebnisse der jeweiligen Verfahren, die somit untereinander verglichen werden können. Zu korrekten Interpretation der Zahlen ist anzumerken, dass es hier um ein Minimierungsproblem handelte, bei dem Kosten minimiert wurden.

Die Besonderheiten der Struktur der jeweils generierten Graphen werden in der Tabelle 6.1 erläutert.

TestReihe	Abhängigkeitsgraph	Anzahl der Transaktionsknoten	Anzahl der Kanten
1	dünnere Graph	n=20	zufällig($n/4, n$)
2	dichter Graph	n=20	zufällig($\frac{n(n-1)}{4}, \frac{n(n-1)}{2}$)
3	Durchschnittsfall	n=20	zufällig($n/2, \frac{n(n-1)}{2}$)

Tabelle 6.1.: Zufälliger Graph

Pro Verfahren wurden jeweils 30 Läufe gemacht. Bei den Verfahren der lokalen Suche wurden je 10.000 Schritte pro Lauf durchgeführt. Die Evolutionssimulation wurde mit einer Population von 50 Elementen über die Dauer von 500 Generationen durchgeführt.

Die Einstellungen der Varianten des hybriden Ansatzes sind in der Tabelle 6.2 dargestellt.

Art	HillclimbingSchritte (Phase 1)	Kalibrierungsmenge	SA Schritte (Phase 2)
Hybrid30	30	10	30
Hybrid50	30	10	50

Tabelle 6.2.: Einstellungen für die Testläufe der Hybriden Algorithmus

6.3. Vergleich der Optimierungsverfahren für einen Zufallsgraph mit einer durchschnittlichen Anzahl von Kanten

Die Ergebnisse der Testläufe zeigen, dass die Evolutionssimulation für einen „durchschnittlichen“ Abhängigkeitsgraph wie erwartet die besten Ergebnisse erreichte. Unter den Verfahren der lokalen Suche hat das Hillclimbing einen leichten Vorsprung erlangt. Die Verfahren Simulated Annealing und der hybride Ansatz unterscheiden sich kaum in den Ergebnissen. Das Verhalten der Verfahren wird in den Abbildungen 6.2 und 6.1 gezeigt.

Das Verfahren	Medianwerte der Endergebnisse
Simulated Annealing	19337,82415
Hillclimbing	19319,7293
Hybridverfahren30	19330,76295
Evolution	19290,94925

Tabelle 6.3.: Die Medianwerte aus den Ergebniswerten der Verfahren bei einem Abhängigkeitsgraph mit einer durchschnittlichen Anzahl von Kanten

6.4. Vergleich der Optimierungsverfahren für einen „dünnen“ Zufallsgraph

Bei den Läufen dieser Reihe hat aus den Verfahren der lokalen Suche das Hybridverfahren30 einen Vorteil vor den anderen; das Verfahren Hybrid50 ist auf dem zweiten Platz. Hillclimbing hatte bei der Suche nach der optimalen Stratifizierung das schlechteste Ergebnis von allen Verfahren der lokalen Suche. Man könnte dies dadurch erklären, dass ein Graph mit wenigen Kanten geringere Einschränkungen darstellt und daher mehrere Stratifikationsmöglichkeiten erlaubt, was die Struktur des Suchraums beeinflusst. Die Fitnesslandschaft ist in einem solchen Fall für das

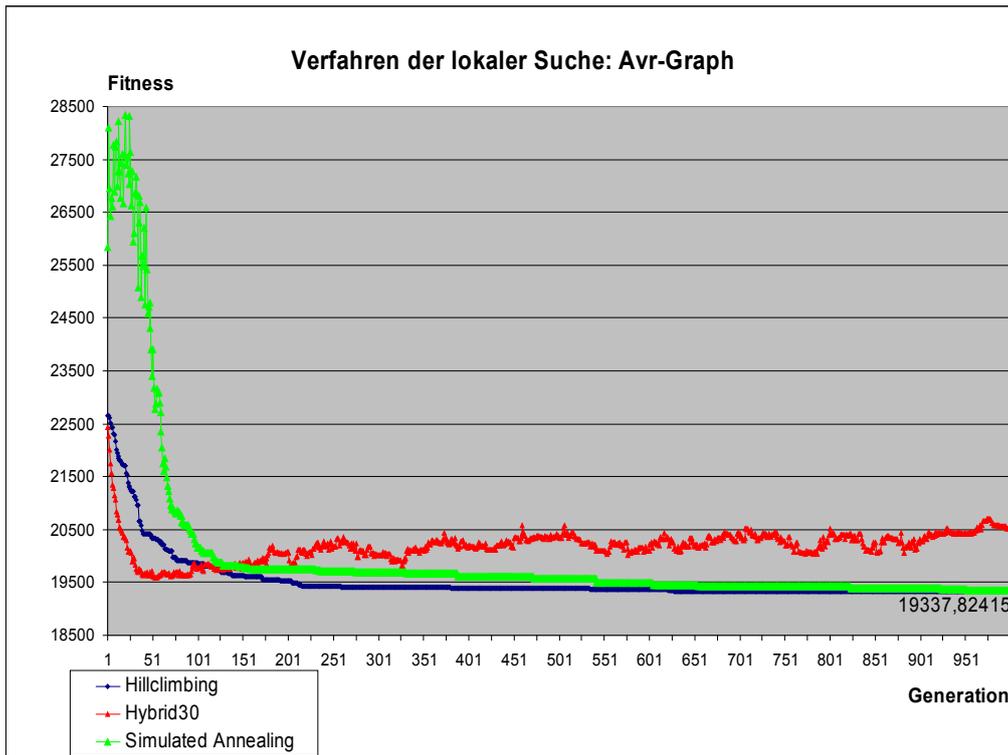


Abbildung 6.1.: Verlauf der Verfahren der lokalen Suche für einen zufällig gewählten Abhängigkeitsgraph mit einer durchschnittlichen Anzahl der Kanten.

Hillclimbing-Verfahren nicht gut beschaffen.

Die Evolutionssimulation erreicht dennoch die besten Ergebnisse, trotz einer vergleichsweise kleineren Generationanzahl.

Das Verfahren	Medianwerte der Endergebnisse
Simulated Annealing	9257,806133
Hillclimbing	9112,039404
Hybridverfahren50	8888,695974
Hybridverfahren30	8768,381544
Evolution	8722,910308

Tabelle 6.4.: Die Medianwerte aus den Ergebnisswerten der Verfahren bei einem dünnen Graph

6.5. Vergleich der Optimierungsverfahren für einen „dichten“ Zufallsgraph

Bei dieser Testreihe hat das Hillclimbing-Verfahren bessere Ergebnisse als Simulated Annealing und als der hybride Ansatz gezeitigt. Die Evolutionssimulation ist bei den Ergebnissen nach wie vor in Führung.

Das Verfahren	Medianwerte der Endergebnisse
Simulated Annealing	12286
Hillclimbing	12261
Hybridverfahren30	12286
Evolution(PopulationGrösse)	12171

Tabelle 6.5.: Die Medianwerte aus den Ergebnisswerten der Verfahren bei einem dichten Graph

6.6. Fazit

Das populationsbasierte Verfahren der Evolutionssimulation hat bei allen Graphentypen die beste Ergebnisse erzielt und somit sich damit eindeutig für einen realen Einsatz qualifiziert. Die Verfahren der lokalen Suche lieferten wechselhafte Ergebnisse: mal war das Hillclimbing besser als die anderen Verfahren, dann war es das Hybridverfahren. Über alle Testreihen hinweg hat von allen Verfahren der lokalen Suche das Hybridverfahren am meisten überzeugt.

Wie man aus den Abbildungen ersehen kann, erfolgt bei dem hybriden Ansatz auch noch zu späteren Zeitpunkten eine Erforschung des Suchraums – es bleibt nicht in den lokalen Optima verfangen, so wie es beim Hillclimbing und dem Simulated

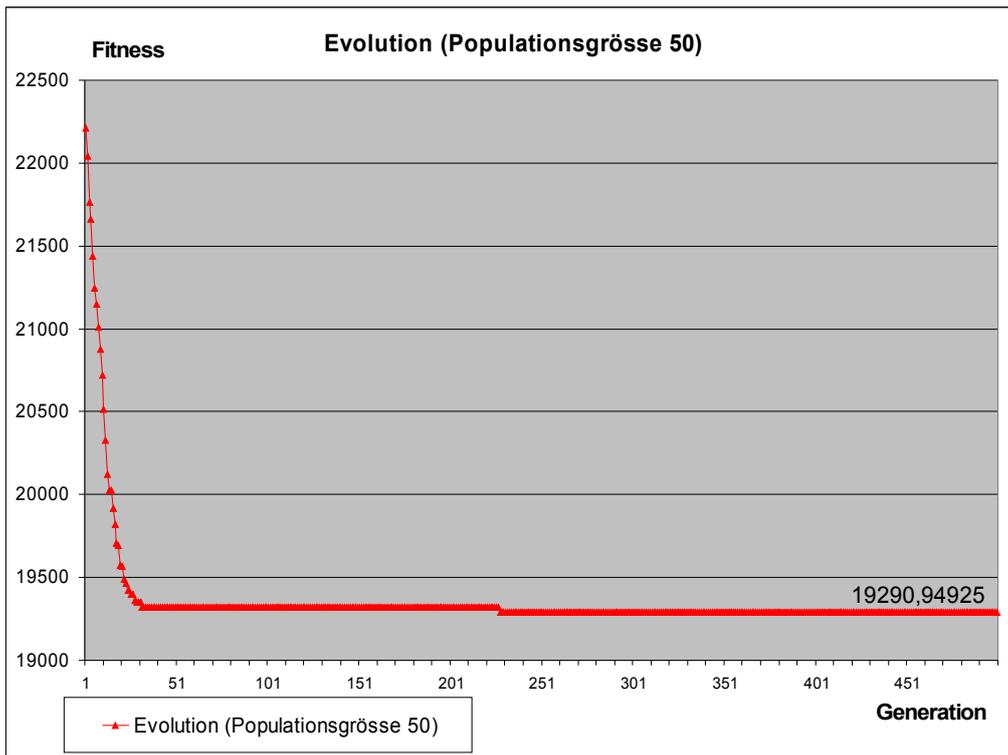


Abbildung 6.2.: Verlauf der Evolutionssimulation für einen zufällig gewählten Abhängigkeitsgraph mit durchschnittlichen Anzahl von Kanten.

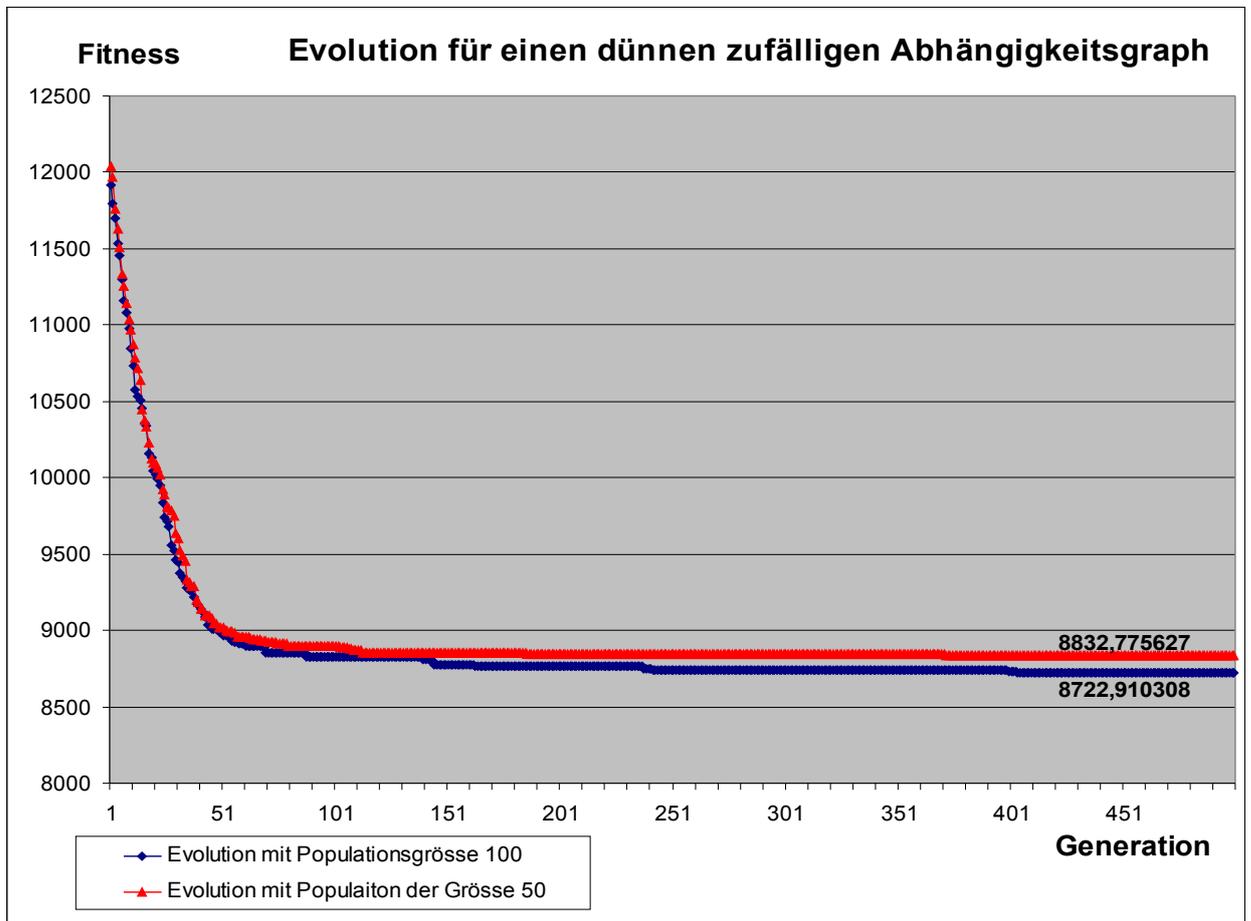


Abbildung 6.3.: Verlauf der Evolutionssimulation für einen zufällig gewählten dünnen Abhängigkeitsgraph.

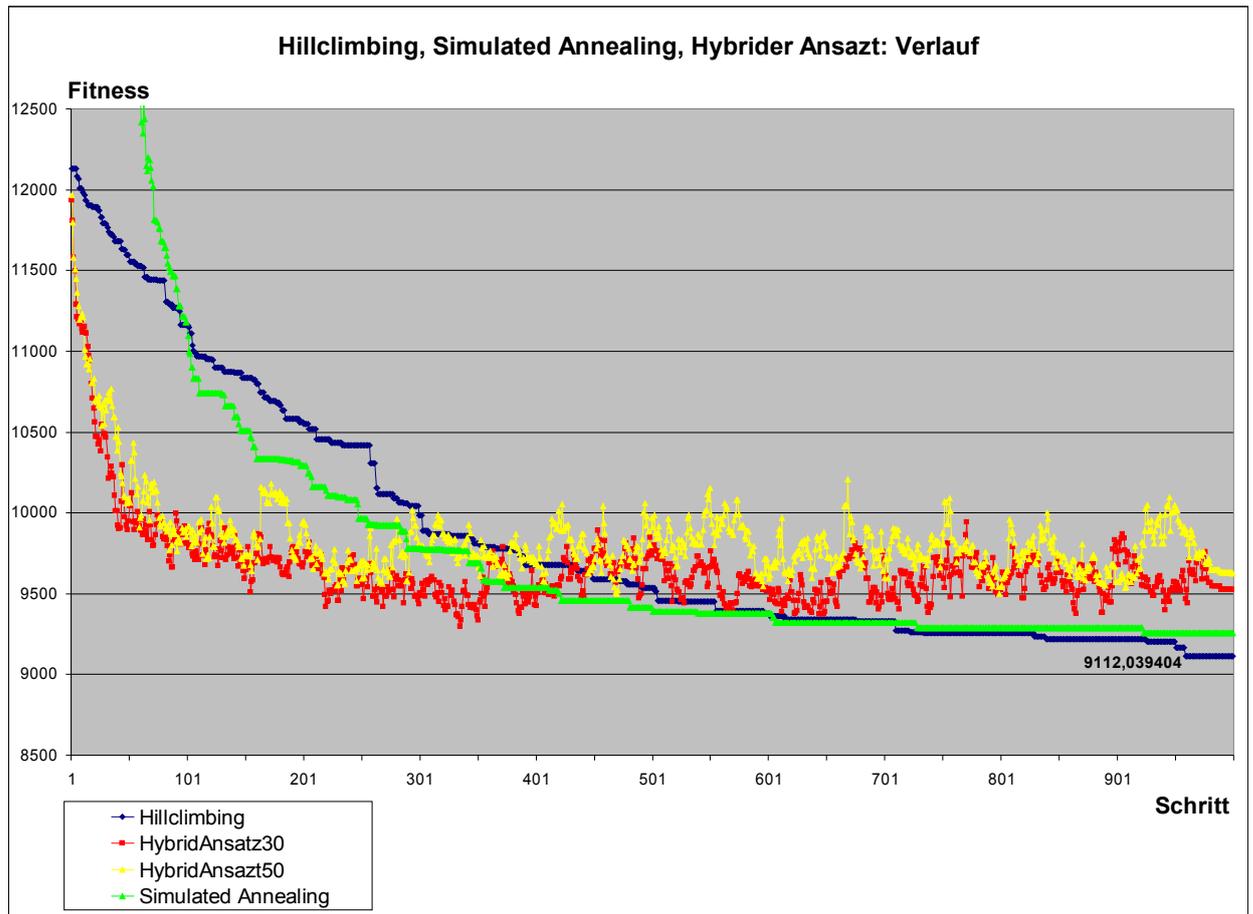


Abbildung 6.4.: Verlauf der Verfahren der lokalen Suche für einen zufällig gewählten dünnen Abhängigkeitsgraph.

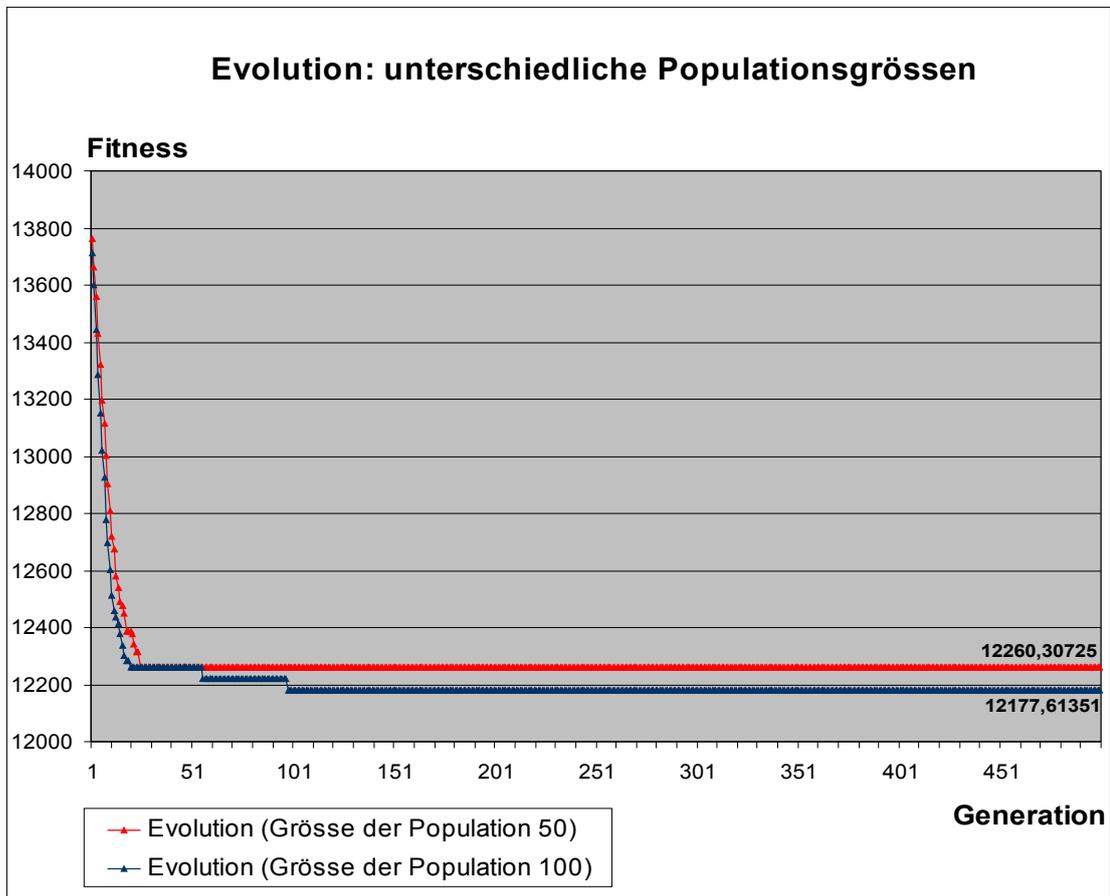


Abbildung 6.5.: Verlauf der Evolutionssimulation für einen zufällig gewählten dichten Abhängigkeitsgraph.

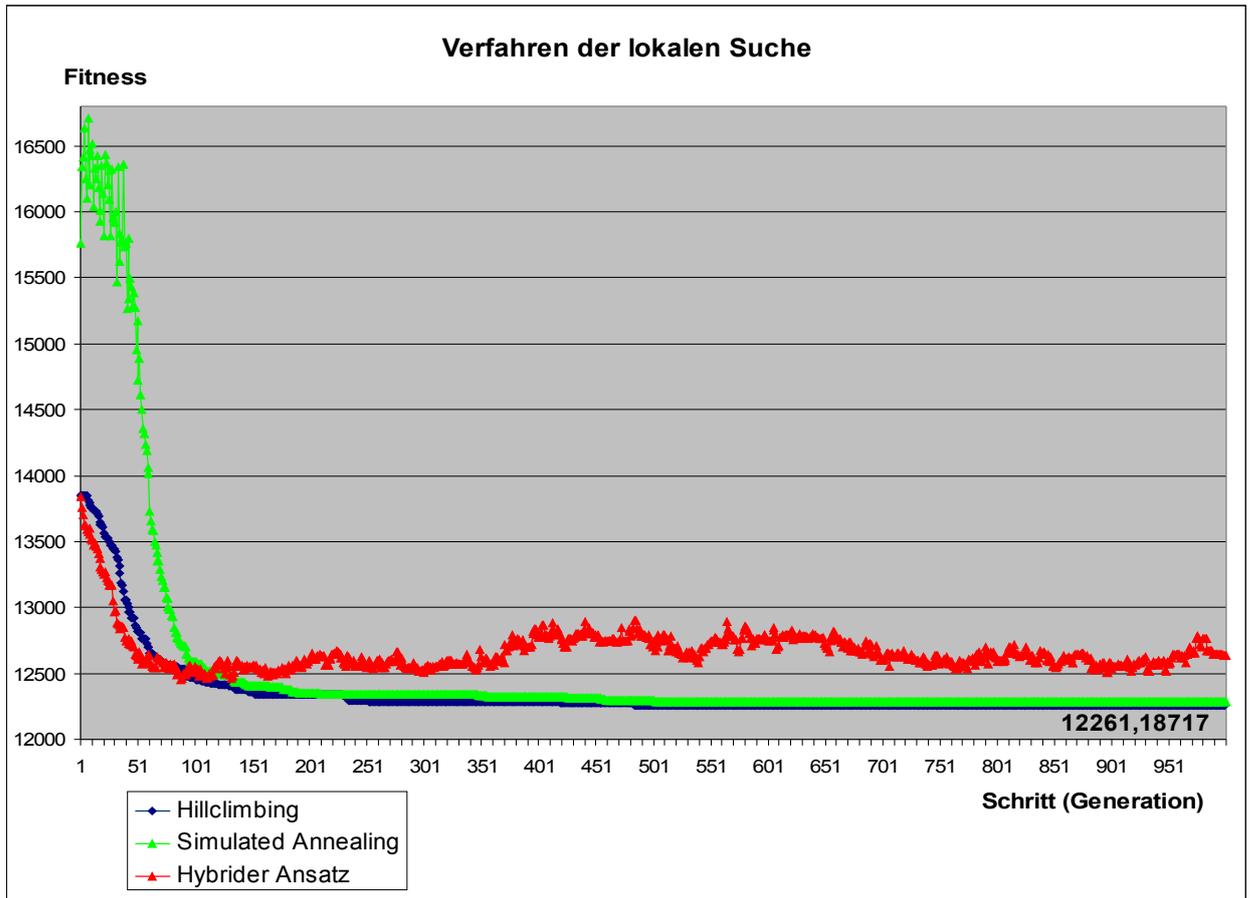


Abbildung 6.6.: Verlauf der Verhalten von Hillclimbing, Simulated Annealing, dem hybriden Ansatz für einen dichten zufälligen Abhängigkeitsgraph.

Annealing stets der Fall ist.

7. Prototyp

In Rahmen der vorliegenden Arbeit wurde ein Prototyp realisiert, in dem Verfahren implementiert wurden, um für eine Menge von vorgegebenen Transaktionen die optimale Stratifizierung zu finden. Den Kern der Anwendung bilden die im Kapitel 4 beschriebenen Verfahren Hillclimbing, Simulated Annealing, der hybride Ansatz aus Hillclimbing und Simulated Annealing sowie die evolutionäre Programmierung. Die Eingabe der Transaktionseigenschaften, der Namen der Konfigurationsdateien, sowie zur Wahl des zu verwendenden Verfahrens geschieht über eine graphische Benutzeroberfläche (GUI). Das Ergebnis der Stratifikation wird in der GUI angezeigt und kann auch in eine XML-Datei exportiert werden. Im Abschnitt 7.1 wird der grundlegende Aufbau der Anwendung erklärt, im Abschnitt 7.4.2 wird die GUI erläutert, der Abschnitt 7.3 beschäftigt sich mit den Erweiterungsmöglichkeiten der Anwendung.

7.1. Aufbau der Anwendung

Anstelle eines Überblicks über die implementierten Komponenten des Prototyps wird hier eine logische Sicht präsentiert, die das System nach der Funktionalität der enthaltenen Komponenten strukturiert.

Datenkomponente . Dazu gehören Transaktionen mit deren Eigenschaften (Properties), der Abhängigkeitsgraph und der Stratifizierungsgraph.

Komponente der Bewertungsmechanismen (EvaluationEngine). Die Komponente besteht aus einer Menge von Bewertungskriterien (EVALUATIONCRITERIA). Dazu gehören die Bewertung der Kosten der 2PC-Nachrichten, die Bewertung des Grads der Nebenläufigkeit, die Bewertung der Antwortzeit, der Ausführungszeit, der Invo-

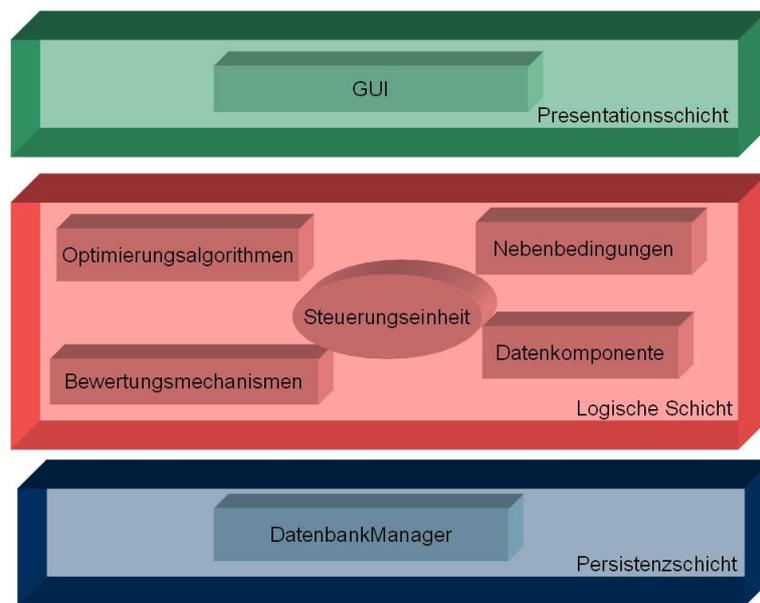


Abbildung 7.1.: Die Schichtenstruktur der Anwendung.

kationskosten, der Recoverykosten sowie der Kosten der Nachrichtenqueues etc. Die Bewertungskriterien wurden in Kapitel 3.5 definiert.

Komponente Optimierungsalgorithmen beinhaltet die Verfahren von Hillclimbing, Simulated Annealing, den Hybrider Ansatz aus dem Hillclimbing und Simulated Annealing, den Evolutionären Algorithmus.

Komponente der Nebenbedingungen (Constraints) . Die Komponente bündelt die Mechanismen zur Erhaltung von Nebenbedingungen wie Azyklität etc.

Datenbankmanager-Komponente . Sie realisiert die Persistenzschicht der Anwendung.

Verwaltungs und Bearbeitungskomponenten:

Steuerungseinheit . Sorgt für die Koordination der Komponenten.

DataCreator . Er besteht aus den Mechanismen, die die Inhalte von der Datenbank auf die Datenstrukturen abbildet und die einen Zufallsgraph für die Experimentalläufe erstellen.

Repräsentation der Ergebnissen der Stratifikation . Funktionalität für die Ausgabe der ermittelten Information, z.B. als XML-Datei.

GUI-Komponente . Zur Darstellung der Graphischen Benutzer-Oberfläche.

7.2. Datenstrukturen

Die Datenstrukturen wurden so gewählt, dass die Abläufe des Optimierungsalgorithmus möglichst effizient gestaltet werden können. Die Transaktionen werden durch die Klasse TRANSACTION dargestellt. Diese Klasse beinhaltet alle Eigenschaften der Transaktionen.

Die Eigenschaften (Properties) der Transaktionen werden wie folgt klassifiziert:

- Eigenschaften, die durch einen einzelnen Wert repräsentiert werden (SINGLEVALUEPROPERTY)
- Eigenschaften, die durch eine Liste von Werten repräsentiert werden (MULTIPLEVALUEPROPERTY)
- Eigenschaften, die durch Funktionen dargestellt werden (FUNKTIONALPROPERTY)

Die Hauptstrukturen, mit denen der Stratifizierungsalgorithmus arbeitet, sind der *Abhängigkeitsgraph* der Transaktionen (Klasse DEPENDANCEGRAPH) und der *StratifizierungsGraph* (Klasse STRATIFICATIONGRAPH).

7.2.1. Abhängigkeitsgraph

Die Hauptbestandteile des Abhängigkeitsgraphs sind die transaktionalen Knoten und die Kanten zwischen diesen Knoten. Sie sind Instanzen der Klasse TRANSACTIONNODE und TRANSACTIONEDGE. Jede Instanz von TRANSACTIONNODE entspricht

einer Transaction, dieser beinhaltet darüberhinaus eine Liste der eingehenden und ausgehenden Kanten der Transaktion, und einen Verweis auf das Stratum, welches die Transaktion beinhaltet.

TRANSACTIONEDGE ist die Klasse, die eine Kante zwischen Transaktionen beschreibt. Eine Instanz der TRANSACTIONEDGE verweist auf die „Vater-“ und „Kind-“ Transaktion, weiterhin verweist sie auf eine Kante des Stratifikationsgraphes, die für die transaktionalen Kanten zuständig ist (siehe Kapitel 3). Der Abhängigkeitsgraph (Klasse DEPENDANCEGRAPH) beinhaltet eine Liste mit den transaktionalen Knoten, und eine Liste der Knoten, die keine Vorgänger haben (die Wurzelknoten).

7.2.2. Stratifizierungsgraph

Die wichtigen Strukturen zum Aufbau von Stratifizierungsgraph sind die Strata (Klasse STRATUM) und die Kanten zwischen den Strata (Klasse STREdge). Die Klasse STRATUM beinhaltet eine Liste der darin enthaltenen Transaktionsknoten sowie eine Liste der eingehenden und ausgehenden Stratifikationskanten (StrEdge). Die Klasse StrEdge verweist auf das „Vater-“ und den „Kind-“ Stratum und beinhaltet eine Liste der dadurch realisierten Transaktionskanten .

7.3. Erweiterbarkeit

Die Erweiterbarkeit der Anwendung entspricht der Problemspezifik der Stratifikationsprozesses. Zum einen kann das Model der Transaktionen (siehe Abschnitt 3.3) erweitert oder geändert werden, indem die einen oder anderen Eigenschaften der Transaktionen hinzugefügt oder entfernt werden. Zum anderen können die Bewertungskriterien an den spezifischen Fall angepasst werden.

7.3.1. Erweiterung und Änderung der Transaktionseigenschaften

Die Transaktionseigenschaften erscheinen in der Anwendung als Attribute der Klasse TRANSACTION. Es wurden verschiedene Kategorien der Transaktionseigenschaften definiert, welche in der Komponente TRANSACTIONPROPERTIES enthalten sind –

siehe Abschnitt 3.3. Es können auch anderen Arten von Transaktionseigenschaften definiert und in der Klasse TRANSACTION hinzugefügt werden.

Da die Eingabedaten (also die Belegungen der Transaktionseigenschaften) auch stets in der Datenbank abgespeichert werden, was je eine Tabelle pro Transaktionseigenschaft erzeugt, muss bei einer Änderung auch die Datenbank (das Datenbankschema) angepasst werden, indem für jede neue Eigenschaft eine neue Tabelle entsprechend hinzugefügt wird.

Die GUI des Prototyps wird automatisch an die Struktur der Klasse TRANSACTION angepasst.

7.3.2. Änderung und Erweiterung der Bewertungskriterien

Jedes Bewertungskriterium ist als Klasse realisiert, die die abstrakte Klasse EVALUATIONCRITERIA implementiert. Jedes verwendete Evaluierungskriterium wird in einer Liste der EvaluationEngine aufbewahrt. Bei der Initialisierung der EvaluierungEngine wird jedes der verwendeten Evaluierungskriterien instantiiert und in die Liste eingefügt. Auch jedes neu hinzukommende Bewertungskriterium soll auf diesem Wege in die EvaluationEngine aufgenommen und dort aufbewahrt werden. Die GUI wird stets an den Inhalt der EvaluationEngine angepasst.

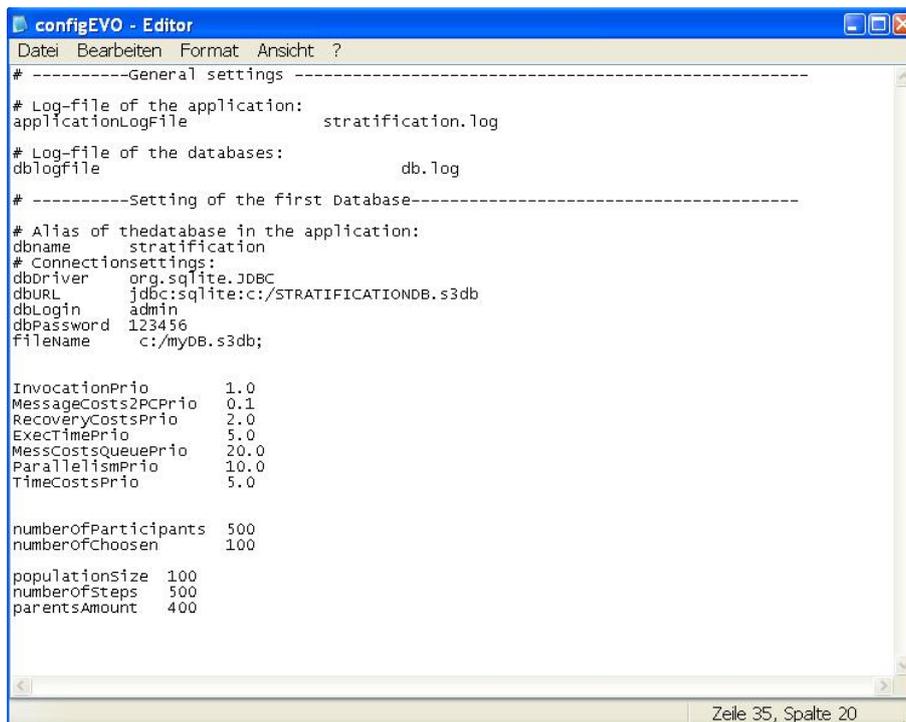
7.3.3. Änderung und Erweiterung der Optimierungsalgorithmen

Die Optimierungsalgorithmen werden in der gleichnamigen Komponente aufbewahrt und von der Steuerungseinheit verwaltet. Wenn also ein neues Optimierungsverfahren hinzugefügt werden soll, muss es in der Komponente aufbewahrt und in der Steuerungseinheit zu Verwaltung freigegeben werden.

7.4. Ein-/Ausgaben der Daten

7.4.1. Konfigurationen

Die Konfigurationen betreffen Angaben über die Datenbankverbindung, Vorgabewerte zu Prioritäten der Bewertungskriterien, sowie algorithmenspezifische Konfigurationen.



```
configEVO - Editor
Datei Bearbeiten Format Ansicht ?
# -----General settings-----
# Log-file of the application:
applicationLogFile      stratification.log
# Log-file of the databases:
dblogfile               db.log
# -----Setting of the first database-----
# Alias of the database in the application:
dbname                  stratification
# Connection settings:
dbDriver                org.sqlite.JDBC
dbURL                   jdbc:sqlite:c:/STRATIFICATIONDB.s3db
dbLogin                 admin
dbPassword              123456
filename                c:/myDB.s3db;

InvocationPrio          1.0
MessageCosts2PCPrio    0.1
RecoveryCostsPrio      2.0
ExecTimePrio           5.0
MessageCostsQueuePrio  20.0
ParallelismPrio        10.0
TimeCostsPrio          5.0

numberOfParticipants    500
numberOfChosen         100

populationSize         100
numberOfSteps          500
parentsAmount          400

Zeile 35, Spalte 20
```

Abbildung 7.2.: Konfigurationsdatei: Datenbank, Bewertung, Evolutionssimulation.

In der Abbildung 7.2 werden verfahrensspezifische Angaben über die Evolution vorgegeben: die Größe der Population, die Anzahl der Eltern etc.

7.4.2. Die graphische Benutzeroberfläche (GUI)

Die GUI wird aus zwei Dialogarten aufgebaut. Der Hauptdialog `MainFrame` (siehe Abbildung 7.4) enthält eine Übersicht der Eingabedaten. Hier können die Namen von Konfigurationsdateien angegeben werden, die den Optimierungsalgorithmus oder die Datenbank konfigurieren, sowie der Name der Ausgabedatei. Im Hauptdialog `MainFrame` können die Prioritäten der Evaluierungskriterien gesetzt werden. Die Eigenschaften der Transaktionen können eingegeben werden: bei Anklicken der in der Liste vorgegebenen Eigenschaften wird eine Tabelle `TableFrame` geöffnet (siehe Abbildung 7.3).

TransactionID	abortProbability
1	0.99
2	0.95
7	0.97

Abbildung 7.3.: TableFrame

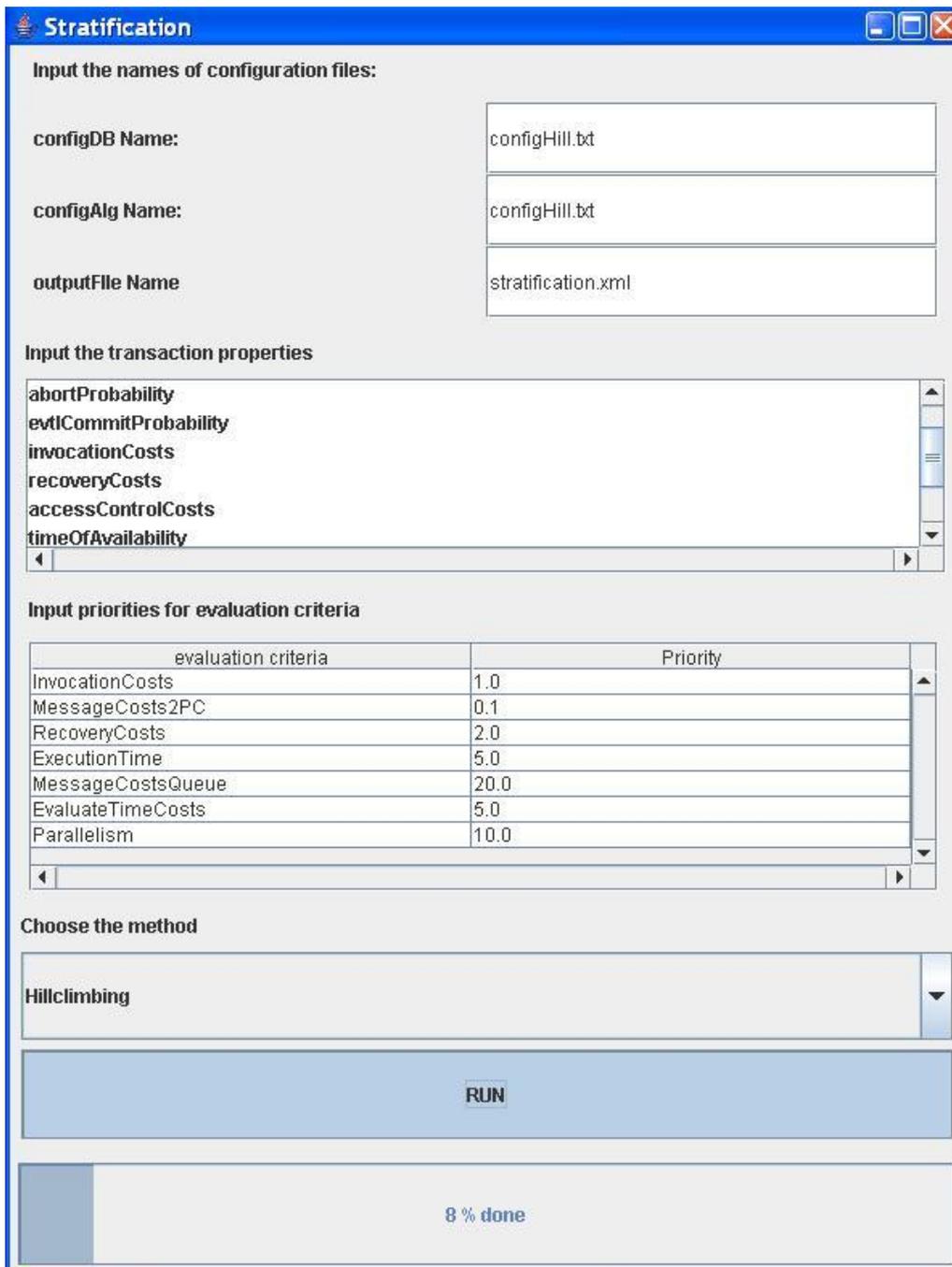


Abbildung 7.4.: Der Hauptdialog.

7.4.3. Das Format der .xml Ausgabe der Stratifizierung

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.w3schools.com"
xmlns="http://www.w3schools.com"
elementFormDefault="qualified">
<xs:element name="STRATIFIKATIONGRAPH">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="NODES">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="STRATUM">
              ...
            </xs:element>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="Edges">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="STRATIFICATIONEDGE">
              ...
              minOccurs="1"
            </xs:element>
          </xs:sequence>
        </xs:complexType>
        <minOccurs="1"
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:schema>
```

```

<xs:element name="STRATUM" >
  <xs:complexType>
    <xs:sequence>
      <xs:element name="TRANSAKTIONID" type="xs:string"
        maxOccurs="n" minOccurs="1" />
    </xs:sequence>
  </xs:complexType>
</xs:element>

```

```

<xs:element name="STRATIFICATIONEDGE" >
  <xs:complexType>
    <xs:sequence>
      <xs:element name="FROMSTRATUM" type="xs:string" />
      <xs:element name="TOSTRATUM" type="xs:string" />
      <xs:element name="TRANSAKTIONEDGES" >
        <xs:complexType>
          <xs:sequence>
            <xs:element name="TRANSAKTIONEDGE" >
              <xs:complexType>
                <xs:sequence>
                  <xs:element name="FROMTR" type="xs:string" />
                  <xs:element name="TOTR" type="xs:string" />
                </xs:sequence>
              </xs:complexType>
              minOccurs="1"
            </xs:element>
          </xs:sequence>
        </xs:complexType>
        minOccurs="1"
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>

```


8. Zusammenfassung und Ausblick

In der vorliegenden Arbeit wurden zwei Konzepte ausgearbeitet und miteinander verbunden: einerseits stratifizierte Transaktionen, andererseits ihre Optimierung.

In dem ersten Teil der Arbeit wurde eine Sicht auf Transaktionen als Träger einer Menge gewisser Eigenschaften beschrieben. Aus Transaktionen wurde ein Abhängigkeitsgraph gebildet, der diese Eigenschaften würdigt. Zusammen mit dem Abhängigkeitsgraph bildeten die Transaktionen eine Struktur, von der ausgehend die Aufteilung der Transaktionen auf Strata durchgeführt wurde. Der durch die Stratifikation entstandene Stratifikationsgraph und seine Bewertungskriterien wurden formal beschrieben.

Der zweite Abschnitt der Arbeit befasste sich mit dem Problem der Suche einer optimalen Stratifikation. Für die Stratifikationsoptimierung wurden Methoden der lokalen Suche (Hillclimbing, Simulated Annealing und ein hybrider Ansatz aus beiden) sowie populationsbasierte Evolutionssimulation angewendet.

Der dritte Teil der Arbeit demonstrierte einige Ergebnisse der Algorithmen aus Teil eins und zwei. Hier wurden Experimentalläufe der Stratifikationsoptimierung durchgeführt sowie der in der Arbeit realisierte Prototyp vorgestellt.

In weiteren Arbeit könnte eine Verfeinerung des Modells der Transaktionen und ihrer Bewertungen durchgeführt werden. Es wäre von großer Interesse, eine Menge weiterer Experimentalläufe durchzuführen, um eine realistische Statistik der Ergebnisse stratifizierter Transaktionen durch die vorgestellten Verfahren zu erhalten. Weiterhin wäre es interessant, das hier verwendete Verfahren zur Stratifizierung mittels Evolution durch die bereits konzeptionell beschriebenen Rekombinationsme-

thoden zu vervollständigen. Außerdem könnten weitere Verfahren und Techniken der Optimierung auf das Problem angewendet werden. Eine solche ist z.B. die Evolutionssimulation mit Verwendung der Inselformationen: es entwickeln sich n Insel-Formationen unabhängig voneinander, alle k Schritte tauschen die Inselformationen miteinander eine gewisse Anzahl von Individuen aus. Für die Gestaltung der Informationsaustausches gäbe es eine Vielzahl von möglichen Topologien: Ring- oder Sterntopologie, vollständiger Graph, Kubus etc.

Bei der Weiterentwicklung des Prototyps kann auch ein Policy-Editor für die Erstellung einer anwendungsfreundlichen graphischen Benutzeroberfläche verwendet werden.

Literaturverzeichnis

- [1] Leymann, F., Transaktionsunterstützung für Workflows. Informatik in Forschung & Entwicklung, 12(1), 1997.
- [2] Leymann, F., Roller, D., Production Workflow. Concepts and Techniques. Prentice Hall PTR. Upper Saddle River. New Jersey 07458, 2000.
- [3] Gray, J., Reuter, A. Transaction Processing: concepts and techniques. San Francisco, California. Morgan Kaufmann Publishers, 1993.
- [4] Bernstein, P., A., Newcomer, E. Principles of Transaction Processing. San Francisco, California. Morgan Kaufmann Publishers, 1997.
- [5] Couloris, G., Dollimore, J., Kindberg, T., Distributed Systems. Concepts and design. Addison-Wesley Publishers Limited, 1994.
- [6] Siedersleben, J., Quasar: Die sd&m Standardarchitektur. Teil 2. sd&m Research. München, 2003.
- [7] Wegener, I. Theoretische Informatik. Eine algorithmen-orientierte Einführung. Leipzig. B.G.Teubner, 1999.
- [8] Wegener, I. Skript zur Spezialvorlesung „Evolutionäre Algorithmen“, SS 2002.
- [9] Weicker, K. Evolutionäre Algorithmen. Teubner-Verlag, 2002.
- [10] Nissen, V. Einführung in Evolutionäre Algorithmen. Optimierung nach dem Vorbild der Evolution. Braunschwei; Wiesbaden. Vieweg, 1997.
- [11] Gerdes, I., Klawonn, F., Kruse, R. Evolutionäre Algorithmen. Wiesbaden. Vieweg, 2004.

- [12] Schöneburg, E., Heinzmann, F., Feddersen, S. Genetische Algorithmen und Evolutionsstrategien: Eine Einführung in Theorie und Praxis der simulierten Evolution. Bonn, Paris. Addison-Wesley, 1994.
- [13] Sedgewick, R. Algorithmen. Addison-Wesley / Pearson, 2002.
- [14] Fogel, L.J., Owens, A. J. & Walsh, M. J. Artificial intelligence through simulated evolution. New York: John Wiley and Sons, 1966.
- [15] Holland, J. H. A new kind of turnpike theorem. *Bulletin of the American Mathematical Society*, 75(6), 1311.1317, 1969.
- [16] Koza, J. R. Hierarchical genetic algorithms operating on populations of computer programs. In N. S. Sridharan (Ed.), Proc. of the 11th Joint Conf. on Genetic Algorithms (pp. 786.774). San Francisco, CA: Morgan Kaufmann Publishers, 1989.
- [17] Koza, J. R. Genetic programming: on the programming of computers by means of natural selection. Cambridge, MA: MIT Press, 1992.

Erklärung

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

(Olha Danylevych)