UNIVERSITÄT DORTMUND

Kreuzungsminimierung für k-seitige Buchzeichnungen von Graphen mit Ameisenalgorithmen

Niels Hendrik Pothmann

Algorithm Engineering Report
TR07-2-006
Juli 2007
ISSN 1864-4503

Universität Dortmund
Fachbereich Informatik
Algorithm Engineering (LS 11)
44221 Dortmund / Germany
http://ls11-www.cs.uni-dortmund.de/

UNIVERSITÄT DORTMUND
FACHBEREICH INFORMATIK

Diplomarbeit

Kreuzungsminimierung für k-seitige Buchzeichnungen von Graphen mit Ameisenalgorithmen

Niels Hendrik Pothmann
12. Februar 2007

INTERNE BERICHTE
INTERNAL REPORTS

Lehrstuhl XI (Algorithm Engineering)
Fachbereich Informatik
der Universität Dortmund

Betreuer:
Erstgutachter Prof. Dr. Günter Rudolph
Zweitgutachter Prof. Dr. Petra Mutzel

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Der Wunsch nach kreuzungsminimalen Zeichnungen beschäftigte bereits Paul Turán [Tur77] während des zweiten Weltkrieges. In einem Arbeitslager wurden Ziegel in Brennöfen gebrannt und auf Lastzügen zu verschiedenen Lagerplätzen transportiert. Jeder Brennofen war mit allen Lagerplätzen durch eine Schiene verbunden. An den Überschneidungen von Schienen entgleisten die Lastzüge jedoch für gewöhnlich, so dass Ziegel herausfielen. Dieses verursachte einen Zeitverlust, der durch die Minimierung der Schienenüberschneidungen hätte vermindert werden können.
Dieses Problem entspricht der Kreuzungsminimierung einer Graphenzeichnung in der Ebene, wobei die Brennöfen die Knoten und die Schienen die Kanten des Graphen darstellen.

Allgemein stellen Graphen ein mächtiges Hilfsmittel zur Repräsentation von Objekten und deren Beziehungen untereinander dar. Für einen Graphen existieren zahlreiche Zeichnungen, die dessen Struktur widerspiegeln, aber geometrisch auf unterschiedliche Weise realisiert sind. Einige von ihnen sind unter Berücksichtigung bestimmter Kriterien zu bevorzugen. Vielfach stellt eine kreuzungsminimale Darstellung das primär zu verfolgende Ziel dar.

Ein Graph kann geometrisch auf diversen Zeichenflächen dargestellt werden. Zahlreich betrachtet man Darstellungen in der Ebene, möglich ist jedoch auch eine Zeichnung auf Oberflächen von Körpern, oder auch als Buch [vgl. Arc96].
Eine spezielle Zeichenmethode ist die *k-seitige Buchzeichnung*[1]. Dabei werden die Knoten entlang einer Linie angeordnet und die Kanten als Halbkreise in k Halbebenen (auch Seiten genannt) gezeichnet. Das *k-seitige Buchkreuzungszahl Problem (BCNP)*[2] besteht darin, eine k-seitige Buchzeichnung eines Graphen mit minimaler Anzahl kreuzender Kanten zu bestimmen. Die Bestimmung kreuzungsminimaler Buchzeichnungen ist u.a. durch folgende Punkte motiviert [siehe auch He06, S.4ff]:

- **Informationsvisualisierung**
  Dient die Darstellung eines Graphen der Informationsvisualisierung, so dass die enthaltenen Informationen von einem Betrachter aufgenommen und interpretiert werden, ist eine kreuzungsminimale Zeichnung anzustreben. Nach Purchase [Pur97] hat die Anzahl der Kantenüberschneidungen einen enorm hohen Stellenwert in Bezug auf das menschliche Verständnis der in einem Graphen repräsentierten Informationen. Eine kreuzungsminimale Zeichnung fördert die Lesbarkeit und die enthaltenen Informationen können von einem Betrachter leichter aufgenommen werden.
  Buchzeichnungen stellen ein lineares Graphenlayout dar, in denen die Knoten entlang einer Linie angeordnet sind. Kreuzungsminimale Buchzeichnungen dienen allgemein der Lesbarkeit und Übersichtlichkeit linearer Layouts von Graphen.

[1] k-page book drawing [SSSV95; SSSV97]. Nähere Informationen finden sich in Kapitel 2.1.2.
[2] k-page Book Crossing Number Problem, in Anlehnung an [He06, S.3].

- **VLSI[3] Design**
  Im Design linearer VLSI Schaltkreise werden die Schaltelemente entlang einer Linie angeordnet und die elektrischen Leitungen verlaufen ober- oder unterhalb dieser Linie. Eine solche Zeichnung korrespondiert zu dem 2-seitigen BCNP. Die Minimierung der Kantenüberschneidungen bedeutet hierbei geringere Produktionskosten. Dieses wird in dem Design linearer VLSI Schaltkreise als erstrangiges Ziel angesehen [vgl. HSM07].

- **Approximation der planaren Kreuzungszahl**
  Über die planare Kreuzungszahl eines Graphen ist bis heute wenig bekannt. Ihre Bestimmung ist NP-schwierig [siehe GJ83].
  Für einen Graphen ist die Anzahl seiner k-seitigen Buchzeichnungen begrenzt und diese können auf einfache Weise aufgezählt werden. Die 2-seitige sowie die 1-seitige Buchkreuzungszahl stellt eine obere Schranke der planaren Kreuzungszahl dar [siehe SSSV95].

Verwandt mit dem BCNP ist das *Bucheinbettungs Problem (BEP)*[4]. Hierbei wird eine Buchzeichnung mit minimaler *Buchdicke* gesucht. Die *Buchdicke* eines Graphen ist die kleinste Zahl k, für die sich in der k-seitigen Buchzeichnung keine zwei auf der gleichen Seite gezeichneten Kanten überkreuzen.
Eine zahlreich genannte Motivation für das BEP ist das Design fehlertoleranter Prozessorenarrays in VLSI Prozessoren [siehe CLR87; Gam86]. Dabei sollen Prozessoren gemäß einer bestimmten Vernetzungsstruktur miteinander verbunden werden. Es ist möglich, dass einige von ihnen nicht korrekt arbeiten. Das Ziel besteht in der ausschließlichen Vernetzung *fehlerfreier* Prozessoren.
Die Prozessorenanordnung erfolgt entlang einer (logischen oder physikalischen) Linie. Verbindungen werden in Bündelungen von elektrischen Leitungen verwaltet, die als Stack arbeiten und oberhalb der Linie verlaufen. Zunächst wird die Linie von links nach rechts durchlaufen und dabei die fehlerfreien Elemente bestimmt. Zur Allokation der Verbindungen werden die Prozessoren anschließend in der umgekehrten Reihenfolge durchlaufen. Von jedem der fehlerfreien Prozessoren kann dabei eine neue Verbindung allokiert werden (PUSH der Verbindung in den Stack), um diesen mit einem weiter links liegenden Element zu verbinden. Weiterhin besteht auch die Möglichkeit, mit einer bereits allokierten Verbindung (von weiter rechts liegenden Elementen) vernetzt zu werden (POP einer Verbindung aus dem Stack).
Wünschenswert ist die Minimierung der Anzahl benötigter Leitungsbündelungen. Das Problem ist äquivalent zur Bestimmung der Buchdicke der Vernetzungsstruktur, wobei jede Leitungsbündelung zu einer Seite korrespondiert (siehe auch Abbildung 1.1). Es dürfen sich dabei keine zwei auf der selben Seite gezeichneten Kanten überschneiden; der beschriebene Vernetzungsaufbau kann keine überkreuzenden Verbindungen verwalten.

Einsatz von Ameisenalgorithmen

Viele kombinatorische Optimierungsprobleme sind NP-schwierig. Das BCNP ist ein statisches kombinatorisches Optimierungsproblem und für $k \in \{1, 2\}$ NP-schwierig. Über die Komplexität des k-seitigen BCNP mit $k > 2$ ist in der Literatur nichts bekannt. Das BEP ist ebenso NP-schwierig, sogar für den Spezialfall einer fest vorgegebenen linearen Knotenanordnung. Zur Optimierung solcher Probleme bieten sich Metaheuristiken[5] an. Für das 1- und 2-seitige

[3] Very Large Scale Integration
[4] Book Embedding Problem [CLR87; Gam86]
[5] [OK96; YI96]

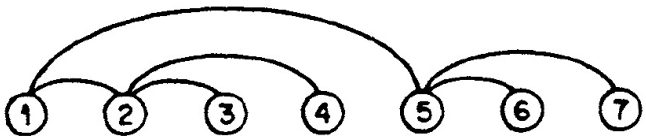


Abbildung 1.1: Beispiel: Die Verbindungsstruktur eines vollständigen binären Baumes mit Tiefe 3 kann in ein 1-seitiges Buch eingebettet werden. Es wird eine Leitungsbündelung benötigt [entnommen aus CLR87].

BCNP konnten bereits mit Hilfe genetischer Algorithmen vielversprechende Resultate gefunden werden [siehe HSM07].

Die Optimierung mit Ameisenkolonien*(ACO)*[6] ist eine nach biologischem Vorbild entwickelte Metaheuristik. Sie orientiert sich an dem Verhalten biologischer Ameisen und ist dem Forschungsgebiet der Schwarm-Intelligenz zugeordnet. Die ACO ist in der Lage, gute, wenn nicht sogar optimale Lösungen für statische sowie dynamische kombinatorische Optimierungsprobleme zu finden. Mittlerweile erfreuen sich diese Algorithmen großer Beliebtheit. Sie sind bereits auf verschiedene Problemklassen erfolgreich angewendet worden [vgl. DS02; DD99]. Für eine Übersicht ist das Buch von Dorigo und Stützle [DS04, S.39f] hilfreich.

1.2 Ziel dieser Arbeit

Das Ziel dieser Arbeit besteht in der Anwendung von Ameisenkolonien zur Optimierung des k-seitigen BCNP. Die Anzahl der zur Verfügung stehenden Seiten, k, muss dabei beliebig festgelegt werden können. Insgesamt sollen verschiedene Ameisenalgorithmen eingesetzt und zur Optimierung einer Lösung für das k-seitige BCNP angepasst werden.
Cordon, Herrera und Stützle [CHS02] veröffentlichten einen 6-Schritte-Plan, der zur Anwendung von Ameisenalgorithmen auf neue Probleme zu realisieren ist. Dessen Umsetzung stellt einen Schwerpunkt dieser Arbeit dar. Dabei wird insbesondere auf die Codierung des Konstruktionsgraphen, die Definition der Pheromonspuren sowie die Bestimmung heuristischer Informationen eingegangen.

Die Performance der eingesetzten Ameisenalgorithmen soll auf unterschiedlichen Klassen von Graphen experimentell verglichen werden. Dieses ist nur möglich, wenn die Algorithmen optimal parametrisiert sind und somit ungünstige Parametereinstellungen ausgeschlossen sind. Aus diesem Grund soll die *SPO Toolbox (SPOT)*[7] zur Optimierung der Parametereinstellungen eingesetzt werden.
Anhand der optimierten Parametersätze soll ferner untersucht werden, ob allgemein empfehlenswerte Einstellungen ausgemacht werden können.
Zur Einschätzung der Performance können die Resultate mit bislang erzielten besten Ergebnissen von He [He06, S.108,133] und Cimikowski [Cim02] auf zirkulären Graphen für $k = 2$ verglichen werden.

[6] *Ant Colony Optimization* [DS04]
[7] Sequential Parameter Optimization Toolbox [BBLP06]

Geringe Anpassungen an die hier entwickelten Algorithmen ermöglichen zudem die Optimierung des BEP. Die notwendigen Anpassungen sind in Anhang B aufgelistet.

1.3 Überblick

Das folgende Kapitel 2 gibt eine Einführung in die benötigten theoretischen Grundlagen dieser Arbeit. Dazu zählen Grundelemente der Graphentheorie, Lösungsmöglichkeiten kombinatorischer Optimierungsprobleme sowie eine Einführung in Ameisenkolonien. Insbesondere wird das zugrundeliegende k-seitige Buchkreuzungszahl Problem erläutert und der Ablauf einzelner Ameisenalgorithmen ausführlich dargestellt.
Kapitel 3 befasst sich mit der Adaption von Ameisenkolonien auf das k-seitige BCNP. Alle dazu notwendigen Schritte werden der Reihe nach umgesetzt.
In Kapitel 4 sind zunächst die zu optimierenden Graphenklassen sowie die Testabläufe dargestellt, bevor die Parameteroptimierung durchgeführt wird. Es folgt eine Analyse der optimierten Parametersätze sowie die Darstellung der auf unterschiedlichen Graphenklassen erzielten Ergebnisse.
Kapitel 5 enthält die Zusammenfassung und den Ausblick dieser Arbeit.

2 Grundlagen

Das Kapitel 2.1 befasst sich mit dem Ursprung und den Einzelheiten des k-seitigen BCNP, wofür insbesondere graphentheoretische Grundlagen eingeführt werden.
Das k-seitige BCNP ist ein kombinatorisches Optimierungsproblem und als solches werden in Kapitel 2.2 die generellen Eigenschaften solcher Probleme und Lösungsmöglichkeiten, u.a. Heuristiken und Metaheuristiken, vorgestellt. Diese Grundkenntnisse werden für das Verständnis der ACO vorausgesetzt, die anschließend in Kapitel 2.3 als Metaheuristik zur Lösung von kombinatorischen Optimierungsproblemen differenziert erläutert wird. Somit werden die benötigten Grundlagen zur Anwendung auf das vorliegende Problem geschaffen.

2.1 Graphentheoretische Grundlagen

Dieses Kapitel gibt eine kurze Einführung in die im späteren Verlauf benötigten Grundlagen der Graphentheorie. Insbesondere wird das in dieser Arbeit zu optimierende *k-seitige Buchkreuzungszahl Problem (BCNP)* vorgestellt. Das Kapitel 2.1.1 richtet sich dabei grundlegend nach Diestel [Die00, S.2ff].

2.1.1 Eine kurze Einführung in Graphen

Ein *(ungerichteter) Graph* besteht aus einem Paar $\mathcal{G} = (V, E)$ mit

(1) V ist eine endliche, nichtleere Menge von Knoten

(2) E ist eine Menge ungeordneter Paare mit $E \subseteq [V]^2$. Die Elemente von E werden *Kanten* genannt und sind 2-elementige Teilmengen von V.

Ein *gerichteter Graph* ist ein Graph $\mathcal{G} = (V, E)$, wobei die Kantenmenge E aus *geordneten* Paaren von Knoten besteht.

Zwei Knoten $v, w \in V$ sind *benachbart* (oder *adjazent*), wenn sie durch eine Kante $e \in E$ verbunden sind. In einem *vollständigen* Graphen $\mathcal{G}$ sind je zwei Knoten von $\mathcal{G}$ benachbart. Ein Graph $\mathcal{G}$ ist *leer*, wenn $\mathcal{G} = (\emptyset, \emptyset)$.
Ein Knoten v heißt mit einer Kante e *inzident*, wenn $v \in e$ gilt.

Definition 2.1.1 (Grad/Durchschnittsgrad)
Der Grad $d(v)$ eines Knotens v ist die Anzahl $|E(v)|$ der mit v inzidenten Kanten.
Der Durchschnittsgrad $d(\mathcal{G}) = \sum_{v \in V} \frac{d(v)}{|V|}$ gibt die ungefähre Anzahl der Kanten pro Knoten an [vgl. Die00, S.5].

Das Verhältnis von Kanten zu Knoten innerhalb eines Graphen $\mathcal{G}$ lässt sich auch direkt ausdrücken: $\varepsilon(\mathcal{G}) = \frac{|E|}{|V|}$. Es gilt $d(\mathcal{G}) = 2 * \varepsilon(\mathcal{G})$, da bei der Berechnung des Durchschnittsgrades jede Kante zweimal betrachtet wird.

Definition 2.1.2 (Weg)
Ein Weg ist ein nicht leerer Graph $P = (V, E)$ der Form

(1) $V = \{v_0, v_1, \ldots, v_k\}$

(2) $E = \{v_0v_1, v_1v_2, \ldots, v_{k-1}v_k\}$

, wobei die v_i paarweise verschieden sind. Oft wird ein Weg durch die natürliche Folge seiner Knoten bezeichnet, etwa $P = v_0v_1 \ldots v_k$ [vgl. Die00, S.7].

Aufeinanderfolgende Knoten in einem Weg sind demnach benachbart. Die *Länge* eines Weges ist die Anzahl der Kanten auf dem Pfad.

Definition 2.1.3 (Kreis)
Ist $P = v_0 \ldots v_{k-1}$ ein Weg und $k \geq 3$, so ist der Graph $C = P + v_{k-1}v_0$ ein Kreis [vgl. Die00, S.8].

Ein Kreis vervollständigt einen Weg um die direkte Verbindung zwischen dem letzten und ersten Knoten.

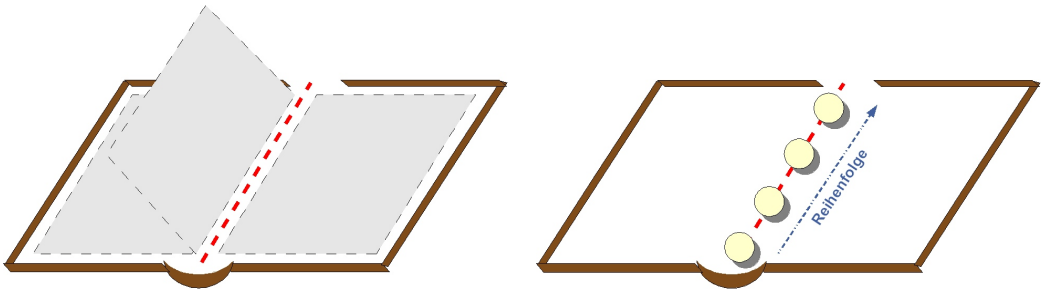
Definition 2.1.4 (Teilgraph)
Ein Teilgraph eines Graphen $\mathcal{G} = (V, E)$ ist ein Graph $\mathcal{G}' = (V', E')$ mit $V' \subseteq V$ und $E' \subseteq E$ [vgl. Die00, S.3].

2.1.2 Buchzeichnungen von Graphen

Die k-seitige Buchzeichnung[1] stellt eine spezielle Zeichenmethode für Graphen dar. Grundlage bildet eine spezifische Zeichenfläche, das *Buch*. Ein Buch besteht aus einem Buchrücken und k Seiten, mit $k \geq 1$. Jede Seite ist mathematisch betrachtet eine Halbebene und alle diese schneiden sich in einer Linie entlang des Buchrückens (Abb. 2.1(a)).
In einer k-seitigen Buchzeichnung werden die Knoten des Graphen in einer bestimmten Reihenfolge auf dieser Linie platziert (Abb. 2.1(b)). Die Kanten zwischen den Knoten müssen jeweils vollständig in einer einzigen der k Seiten gezeichnet werden. Dabei werden die Kanten als Halbkreise gezeichnet [vgl. SSSV95; SSSV97].

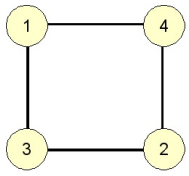
Als Beispiel ist die Buchzeichnung für den ungerichteten Graphen $\mathcal{G}_1$ in Abb. 2.2 mit der linearen Anordnung bzw. Permutation (2, 4, 3, 1) der Knoten illustriert. Die 1-seitige Buchzeichnung ist bereits unter Vorgabe einer Permutation immer eindeutig, da jede Kante nur in dieser einen Seite gezeichnet werden kann. Abb. 2.2(b) stellt die einzig mögliche 1-seitige Buchzeichung von $\mathcal{G}_1$ mit der Permutation (2, 4, 3, 1) dar. In einer 2-seitigen Buchzeichnung (Abb. 2.2(c)) kann jede Kante einer der beiden Halbebenen zugeordnet werden, so dass hier die Permutation allein nicht ausreicht, um eine eindeutige Zeichnung zu definieren.

[1] k-page book drawing [SSSV95; SSSV97]

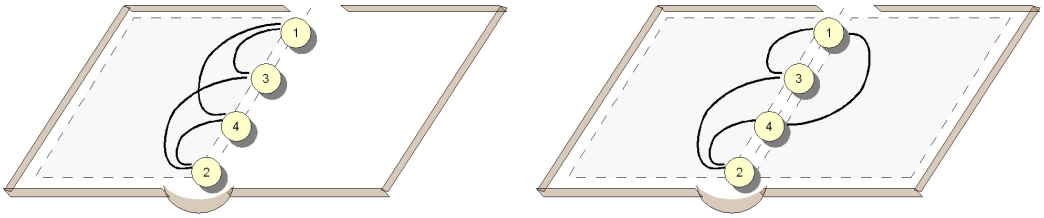


(a) (b)

Abbildung 2.1: Illustration der Buchzeichnung eines Graphen



(a) Gegeben ist der Graph $\mathcal{G}_1$



(b) 1-seitige Buchzeichnung von $\mathcal{G}_1$ (c) eine 2-seitige Buchzeichnung von $\mathcal{G}_1$

Abbildung 2.2: Beispiele für Buchzeichnungen

2.1.3 Kreuzungsminimierung von Graphenzeichnungen

Definition 2.1.5 (Kreuzungszahl)
Die (planare) Kreuzungszahl $cr(\mathcal{G})$ eines Graphen $\mathcal{G}$ ist die kleinste Anzahl von Kantenüberschneidungen aller Zeichnungen von $\mathcal{G}$ in einer Ebene [vgl. BL84].

Definition 2.1.6 (Einbettung)
Ein Graph heißt in eine Fläche S eingebettet, wenn er so auf S gezeichnet ist, dass sich keine zwei Linien schneiden [Har74, S.112]

Ein Graph ist *planar* (oder *plättbar*), wenn er in eine *Ebene* eingebettet werden kann. [vgl. Har74, S.112].

Die Bestimmung der Kreuzungszahl eines Graphen ist NP-vollständig [GJ83] und exakte Werte sind nur für eingeschränkte Klassen von Graphen bekannt. Selbst für vollständige Graphen ist die exakte Kreuzungszahl unbekannt [siehe SSSV96].

2.1.4 Buchkreuzungszahl

Eine k-seitige Buchzeichnung $\mathcal{D}$ des Graphen $\mathcal{G}$ stellt zwei Informationen zur Verfügung:

(1) Die lineare Anordnung (Permutation) der Knoten entlang des Buchrückens.

(2) Die Seite, auf der jede einzelne Kante gezeichnet ist.

Abhängig davon ergibt sich eine unterschiedliche Anzahl von Kantenüberschneidungen, die als *Buchkreuzungszahl* definiert wird.

Definition 2.1.7 (Buchkreuzungszahl)
Die minimale Anzahl von Kantenüberschneidungen $v_k(\mathcal{G})$ eines Graphen $\mathcal{G}$ über alle k-seitigen Buchzeichnungen von $\mathcal{G}$ wird die (k-seitige) Buchkreuzungszahl (BCN)[a] von $\mathcal{G}$ genannt [vgl. SSSV95; SSSV97].

[a] Book Crossing Number

Das *k-seitige Buchkreuzungszahl Problem (BCNP)*[2] besteht darin, eine Ordnung der Knoten sowie eine Verteilung der Kanten auf die k Seiten zu finden, so dass die Kantenüberschneidungen in der k-seitigen Buchzeichnung minimiert werden.
Eine Lösung für das k-seitige BCNP besteht aus einer Permutation der Knoten und einer Seitenzuordnung der Kanten auf die k Seiten. Für $k = 1$ steht nur eine Seite zur Verfügung und die Bestimmung der Seitenzuordnung entfällt. Da die Anzahl möglicher Lösungen endlich ist, ist das BCNP ein *kombinatorisches* Optimierungsproblem.

Die k-seitige Buchkreuzungszahl $v_k(\mathcal{G})$ stellt für $k \in \{1, 2\}$ eine obere Schranke der planaren Kreuzungszahl $cr(\mathcal{G})$ dar. Es gilt stets die folgende Ungleichung [vgl. SSSV95]:

$$cr(\mathcal{G}) \leq v_2(\mathcal{G}) \leq v_1(\mathcal{G}) \tag{2.1}$$

Die Bestimmung der 1- und 2-seitigen Buchkreuzungszahl ist somit ein einfaches Mittel zur Approximation der planaren Kreuzungszahl eines Graphen.

Das 1-seitige und 2-seitige BCNP ist NP-vollständig [MKNF87; MKNF90]. In der Literatur findet sich kein Komplexitätsnachweis für das k-seitige BCNP, was nach Austausch mit Professor Imrich Vrt´o[3] bestätigt wurde. Es ist nicht bekannt, ob das k-seitige BCNP mit $k > 2$ auch NP-vollständig ist, obwohl man versucht sein mag, dieses anzunehmen.

2.1.5 Buchdicke

Definition 2.1.8 (Buchdicke)
Die Buchdicke[a] (oder Seitenzahl) $p(\mathcal{G})$ eines Graphen $\mathcal{G}$ ist der kleinste Wert k, so dass gilt: $v_k(\mathcal{G}) = 0$ [vgl. SSSV97; Gam86].

[a] Book-Thickness

[2] k-page Book Crossing Number Problem, in Anlehnung an [He06, S.3].
[3] Slovak Academy of Sciences, Institute of Mathematics

Das *Bucheinbettungs Problem (BEP)*[4] besteht darin, eine Buch*einbettung* eines Graphen mit minimaler Buchdicke zu finden. Keine zwei auf der gleichen Seite gezeichneten Kanten dürfen sich in der Buchzeichnung überkreuzen. Das *BEP* ist mit dem *BCNP* verwandt.

Das BEP ist NP-vollständig, selbst für den Spezialfall einer fest vorgegebenen Permutation [vgl. CLR87].

2.2 Kombinatorische Optimierung

2.2.1 Kombinatorische Optimierungsprobleme

Optimierungsprobleme gliedern sich in zwei Kategorien [vgl. PS88, S.3f]:

- Optimierungsprobleme, die auf *kontinuierlichen Variablen* definiert sind. Eine Lösung des Problems ist eine Menge reellwertiger Zahlen und es existieren überabzählbar unendlich viele Lösungen.
- Optimierungsprobleme, die auf *diskreten Variablen* definiert sind. Die Menge der möglichen Lösungen ist hierbei endlich und Lösungen stellen nach Reeves [Ree95, S.2] beispielsweise eine Menge oder eine Sequenz von Integer-Werten oder anderen diskreten Objekten dar.
  Diese Probleme nennt man auch *kombinatorische Optimierungsprobleme.*

Ein kombinatorisches Optimierungsproblem wird durch eine Menge von *Probleminstanzen* spezifiziert und ist entweder ein Minimierungs- oder Maximierungsproblem. In einer Probleminstanz sind die Parameter des Problems nicht mehr unspezifiziert, sondern mit konkreten Werten belegt [siehe AL97, S.3; DS04, S.26]. Die folgenden Ausführungen beschränken sich auf Minimierungsprobleme.

Definition 2.2.1 (Instanz eines kombinatorischen Optimierungsproblems)
Eine Instanz eines kombinatorischen Optimierungsproblems ist ein Tupel $(\mathcal{S}, f, \Omega)$.
$\mathcal{S}$ *ist die (endliche) Menge der Lösungskandidaten und* f *die Zielfunktion, die jedem Lösungskandidaten* $s \in \mathcal{S}$ *einen Wert zuweist. Die Menge* Ω *definiert Nebenbedingungen für das Problem.*
$\tilde{\mathcal{S}} \subseteq \mathcal{S}$ *ist die Menge der zulässigen Lösungen und enthält alle Lösungskandidaten, welche die Nebenbedingungen* Ω *erfüllen. [vgl. DS04, S.26]*

Das Ziel der kombinatorischen Optimierung besteht in der Bestimmung einer global optimalen Lösung $s^* \subseteq \tilde{\mathcal{S}}$ mit minimalem Zielfunktionswert, so dass $f(s^*) \leq f(s)$ für alle $s \in \tilde{\mathcal{S}}$.

Kombinatorische Optimierungsprobleme unterteilen sich in *statische* und *dynamische* Probleme. Bei statischen Problemen sind die Eigenschaften des Problems bereits zu Anfang der Optimierung bekannt und verändern sich nicht mit der Zeit. Im Gegensatz dazu variieren die Probleminstanzen dynamischer Probleme während der Laufzeit. Sie sind auf Variablen definiert, die von einem übergeordneten System dynamisch verändert werden können und müssen

[4] Book Embedding Problem [Gam86; CLR87]

sich demnach während der Laufzeit auf die veränderte Umgebung anpassen können [vgl. DS04, S.33 f].

Die Tatsache, dass $\tilde{\mathcal{S}}$ endlich ist, macht es möglich, alle Elemente in $\tilde{\mathcal{S}}$ der Reihe nach zu bestimmen, auszuwerten und schlussendlich dasjenige Element mit den geringsten Kosten zu wählen. Für viele Problemstellungen ist dieser Ansatz jedoch nicht praktikabel, da die Größe des Suchraums ($|\tilde{\mathcal{S}}|$) rapide anwachsen kann [vgl. Ree95, S.7; Stü98, S.15]. Punkt 2.2.2 befasst sich mit diesem Thema.

2.2.2 Komplexität und Lösungsmöglichkeiten

Allgemein besteht das Ziel, kombinatorische Optimierungsprobleme möglichst *effizient* zu lösen, wobei „effizient“ für gewöhnlich mit der Laufzeit einer Lösungsfindung verbunden wird [vgl. Stü98, S.11]. Dieser Laufzeit sind allerdings Grenzen gesetzt.

Viele kombinatorische Optimierungsprobleme sind $NP - schwierig$, so dass sie nur mit sehr hohem Zeitaufwand exakt gelöst werden können. Sie sind für gewöhnlich **einfach** zu beschreiben, aber **schwierig** zu lösen [vgl. AL97, S.1; OK96, S.2f].

Die Rechenzeit verkürzt sich, wenn nicht mehr alle Lösungskandidaten evaluiert werden und auf die Forderung einer exakten Lösung des Problems verzichtet wird. Es besteht sodann das Ziel, möglichst optimale Resultate zu finden.
Approximationsalgorithmen oder auch *Heuristiken* arbeiten nach diesem Schema. Sie untersuchen nicht jede mögliche Lösung, sind meistens aber dennoch in der Lage, gute Ergebnisse sehr viel schneller zu finden als es mit exakten Algorithmen möglich ist. Dieses gilt sogar für sehr große Probleminstanzen [vgl. Stü98, S.16].
Strenggenommen garantiert ein Approximationsalgorithmus, dass sich die ermittelte Endlösung nur um einen konstanten Faktor von dem optimalen Ergebnis unterscheidet, wohingegen Heuristiken keine Garantie für die Qualität des gefundenen Resultates geben [vgl. MC01].
Meistens wird in der Literatur jedoch nicht derart streng unterschieden. Der Terminus *Approximationsalgorithmus* wird oft im Sinne einer Heuristik definiert, so dass nicht notwendigerweise eine Optimalitätsgarantie gegeben sein muss. Sollte dennoch eine Optimalitätsschranke bewiesen werden können, so ist dies ebenfalls ein Approximationsalgorithmus [vgl. DS04, S.29ff].

2.2.3 Heuristiken und Metaheuristiken

Definition 2.2.2 (Heuristik)
„A heuristic is a technique which seeks good (i.e. near-optimal) solutions at a reasonable computational cost without being able to guarantee either feasibility or optimality, or even in many cases to state how close to optimality a particular feasible solution is“ [Ree95, S.6].

Im Bereich der diskreten Optimierung existieren viele Heuristiken, die gute Lösungen für das jeweils zugrundeliegende Problems generieren können. Allerdings weist die Mehrzahl den Nachteil auf, nur auf ein spezielles Problem anwendbar zu sein und nicht ohne Weiteres verallgemeinert werden zu können [vgl. DP06, S.2; Ree95, S.12].
Das (klassische) Prinzip der lokalen Suche ist ein Beispiel für eine Heuristik: Ausgehend von einer initialen Lösung einer speziellen Probleminstanz sucht das Verfahren in der Nachbarschaft

dieser Lösung nach Elementen besserer Qualität. Die Nachbarschaft ist problemabhängig zu definieren und enthält alle Elemente, die sich in bestimmten Eigenschaften von der jeweiligen Lösung unterscheiden. Sobald ein besseres Resultat ermittelt ist, wird der Prozess damit fortgeführt. Das so ermittelte Endergebnis birgt allerdings die Gefahr, lediglich ein lokales Optimum darzustellen [vgl. AL97, S.3; Ree95, S.26f; OK96, S.4f].

Allgemeinere heuristische Methoden, die auf verschiedene kombinatorische Optimierungsprobleme anwendbar sind, werden als *Metaheuristiken* bezeichnet. Sie sind dafür vorgesehen, schwierige kombinatorische Optimierungsprobleme erfolgreich angehen zu können, bei denen „klassische" Heuristiken bezüglich Effektivität und Effizienz versagt haben [vgl. OK96, S.3].

Definition 2.2.3 (Metaheuristik)
„A meta-heuristic is an iterative generation process which giudes a subordinate heuristic by combining intelligently different concepts for exploring and exploiting the search spaces using learning strategies to structure information in order to find efficiently near-optimal solutions" [OK96, S.3].

Metaheuristiken dienen dazu, zugrundeliegende (problemspezifischere) Heuristiken in ihrer Suche zu leiten, mit der Zielsetzung einer effizienten Suche nach vielversprechenden Lösungen. Sie stellen den Anspruch, auf verschiedene kombinatorische Optimierungsprobleme mit geringen Anpassungen anwendbar zu sein [OK96, S.3; Ree95, S.12f; CHS02]. Beispiele für Metaheuristiken sind u.a. *Tabu-Search* [Ree95, S.20ff], *Simulated Annealing* [Ree95, S.70ff] und *Ant Colony Optimization* [DS04] (Kapitel 2.3) .

Der Vorteil von Metaheuristiken besteht in ihrer Einfachheit und Robustheit. Sie können selbst dann angewendet werden, wenn tiefere mathematische Eigenschaften des Problems nicht bekannt sind und somit die Entwicklung einer effizienten Heuristik nicht gewährleistet werden kann. Sogar unter diesen Umständen können Metaheuristiken immer noch gute Lösungen erzielen [vgl. YI96, S.64].

2.3 Optimierung mit Ameisenkolonien

Die Optimierung mit Ameisenkolonien basiert auf Prinzipien der *Schwarm-Intelligenz*. Im Folgenden wird diesbezüglich zunächst das Forschungsfeld der *Schwarm-Intelligenz* beleuchtet, bevor in den weiteren Unterkapiteln auf die Einzelheiten der ACO präziser eingegangen wird.

2.3.1 Schwarm-Intelligenz

Der Zusammenschluss von Individuen zu Schwärmen erschafft biologisch große Vorteile: Bessere Möglichkeiten der Partnerfindung, Nahrungsteilung, sowie gemeinschaftliche Verteidigung vor Eindringlingen. Dies sind nur einige Beispiele für das enorme Potential der Schwarmbildung. Generell sichert das Schwarmpotential ein Überleben der Population und somit der gesamten Art in sehr viel höherem Maße als die Alleinstellung der Individuen auf sich selbst [vgl. KE01, S.96].

Ein *Schwarm* wird als eine Menge von miteinander interagierenden (mobilen) Agenten definiert, die jeweils nur eine lokale (und somit begrenzte) Sicht auf die Umwelt besitzen und

ein Problem gemeinschaftlich lösen [siehe Hof97]. Die Individuen haben eine einfache Struktur, wohingegen das kollektive Verhalten des Schwarms für gewöhnlich verblüffend komplex ist. Dieses komplexe Verhalten des Schwarms ist das Ergebnis von Interaktionen zwischen einzelnen Individuen [vgl. Eng05, S.2].

Auffällig ist, dass vor allem Insektenkolonien in hohem Maße koordiniert erscheinen, obwohl keine für diese Koordination zuständige zentrale Instanz existiert. Allein Prinzipien der Selbstorganisation sorgen dafür, dass der Schwarm als Ganzes eine organisierte Struktur darstellt [vgl. BDT99, S.1ff]. Für diese Selbstorganisation sind Aspekte der Kommunikation und Koordination der einzelnen Individuen untereinander unerlässlich. Auf sie wird in diesem Forschungsfeld ein besonderes Augenmerk gelegt.

Bonabeau, Dorigo und Theraulaz [BDT99] beschränken ihre Ausführungen vorwiegend auf soziale Insekten, was gleichwohl nicht darüber hinwegtäuschen soll, dass der von ihnen definierte Begriff der Schwarm-Intelligenz generell soziale Tiere einschließt:

> „The expression 'swarm intelligence' was first used [...] in the context of cellular robotic systems, where many simple agents occupy one- or two-dimensional environments to generate patterns and self-organize through nearest-neighbor interactions. Using the expressive „swarm intelligence“ to describe only this work seems unnecessarily restrictive: that is why we extend its definition to include any attempt to design algorithms or distributed problem-solving devices inspired by the collective behavior of social insect colonies and other animal societies“. [BDT99, S.7]

Das Forschungsfeld der Schwarm-Intelligenz beschäftigt sich demnach mit dem Entwurf von Algorithmen, die von sozialen Insektenkolonien oder anderen sozialen Tieren inspiriert werden. Das Ziel ist die Modellierung mehrerer (einfach strukturierter) Individuen, die mit der Umwelt und den Nachbarindividuen lokal interagieren, um ein komplexeres Schwarmverhalten zu erzielen. Dieses kann zur Lösung komplexer Probleme verwendet werden [vgl. BDT99, S.7; Eng05, S.3].

Die Optimierung mit Ameisenkolonien zählt neben der Partikelschwarm Optimierung *(PSO)*[5] zu den bekanntesten und erfolgreichsten Vertretern der Schwarm-Intelligenz.

2.3.2 Klassifikation von ACO Algorithmen

Als Ameisenalgorithmen werden generell multi-Agenten Systeme bezeichnet, in denen das Verhalten jedes einzelnen Agenten (künstliche Ameise) von dem Verhalten „realer“ Ameisen inspiriert wird. Ameisen sind soziale Insekten, die in Kolonien leben und aufgrund gemeinschaftlicher Interaktionen fähig sind, ein komplexes Schwarmverhalten zu erschaffen. Ameisenalgorithmen sind eines der erfolgreichsten Beispiele von Systemen der Schwarm-Intelligenz und wurden auf viele Probleme erfolgreich angewendet [vgl. DD99]. Dazu zählen beispielsweise das TSP[6] [SD99] oder das Routing in Netzwerken [DS04, S.223ff]. Eine Übersicht liefert das Buch von Dorigo und Stützle [DS04, S.39f].

Eine spezielle Klasse von Ameisenalgorithmen stellen die so genannten *ACO Algorithmen* dar, auf die sich im Folgenden konzentriert wird. Die von der Schwarm-Intelligenz geforderte

[5] Particle Swarm Optimization [KE01]
[6] Traveling Salesman Problem

biologische Inspiration beruht hierbei ebenfalls auf dem Verhalten „realer“ Ameisen, wobei speziell die Aspekte der kollektiven Nahrungssuche Beachtung finden [vgl. CHS02; DS04, S.25; DD99].

Die ACO ist eine Metaheuristik zur Optimierung statischer, wie auch dynamischer, kombinatorischer Probleme. ACO Algorithmen sind Instanzen der ACO Metaheuristik [siehe DD99].

In Kapitel 2.3.3 wird zunächst die Nahrungssuche biologischer Ameisen als zugrundeliegende Inspiration der ACO dargestellt. Anschließend wird die ACO im Detail erläutert: Die Metaheuristik beschreibt das Schema von ACO Algorithmen und wie sie auf beliebige kombinatorische Optimierungsprobleme anzuwenden sind. Dabei werden Bezüge zum biologischen Ursprung aufgegriffen und verdeutlicht. Daran anschließend werden die einzelnen ACO Algorithmen differenziert vorgestellt.

2.3.3 Biologisches Vorbild

Eine der für Forscher interessantesten Fähigkeiten vieler Ameisenarten ist das Auffinden und Beibehalten kürzester Wege zwischen dem Ameisenbau und diversen Futterquellen. Umso erstaunlicher wird diese Eigenschaft unter Berücksichtigung der Tatsache, dass die Ameisen etlicher Arten nahezu blind sind und visuelle Reize demzufolge nicht aufnehmen können [vgl. CHS02]. Dennoch haben sie eine Möglichkeit miteinander zu kommunizieren: chemische Botenstoffe.

Bedeutung der Pheromonspuren

Zahlreiche Ameisenarten verfügen über eine Drüse, mit Hilfe derer ein spezieller chemischer Botenstoff - das *Pheromon* - an die Umgebung abgesondert werden kann. Abgelegt werden Pheromone z.B. während sich eine Ameise auf dem Weg von einer gefundenen Nahrungsquelle zurück zum Nest bewegt, so dass durch diese Pheromonspur der Weg zur Nahrungsquelle gekennzeichnet wird und sich zurückverfolgen lässt [vgl. BDT99, S.26].
Wenn keine Pheromonspuren vorliegen, bewegen sich Ameisen größtenteils zufallsbehaftet. Anders hingegen verhält es sich bei Existenzen derselbigen: Sie zeigen die Tendenz, mit Pheromon belegten Wegen zu folgen. Aufgrunddessen werden sie von vorliegenden Pheromonspuren beeinflusst und nachfolgende Ameisen auf diese Weise sozusagen „rekrutiert“, diesen Wegen nachzugehen. Eine solche Art der Beeinflussung wird *Massenrekrutierung* genannt und stellt eine Form der *indirekten* Interaktion[7] unter den einzelnen Ameisen dar [vgl. BDT99, S.26; CHS02; CDM92]. Diese Form indirekter Kommunikation ist auch unter der Bezeichnung *Stigmergie* bekannt [siehe BDT99, S.14; DD99].
Mittels dessen werden sehr bald weitere Ameisen zu den erschlossenen Nahrungsquellen gelangen.

Pheromonspuren verdunsten nach einer gewissen Zeit, so dass die Konzentration an den deponierten Stellen nachlässt. Dieser Umstand trägt dazu bei, dass eine hohe Pheromonkonzentration nur dann erhalten bleibt, wenn ein Weg kontinuierlich von Ameisen benutzt wird.

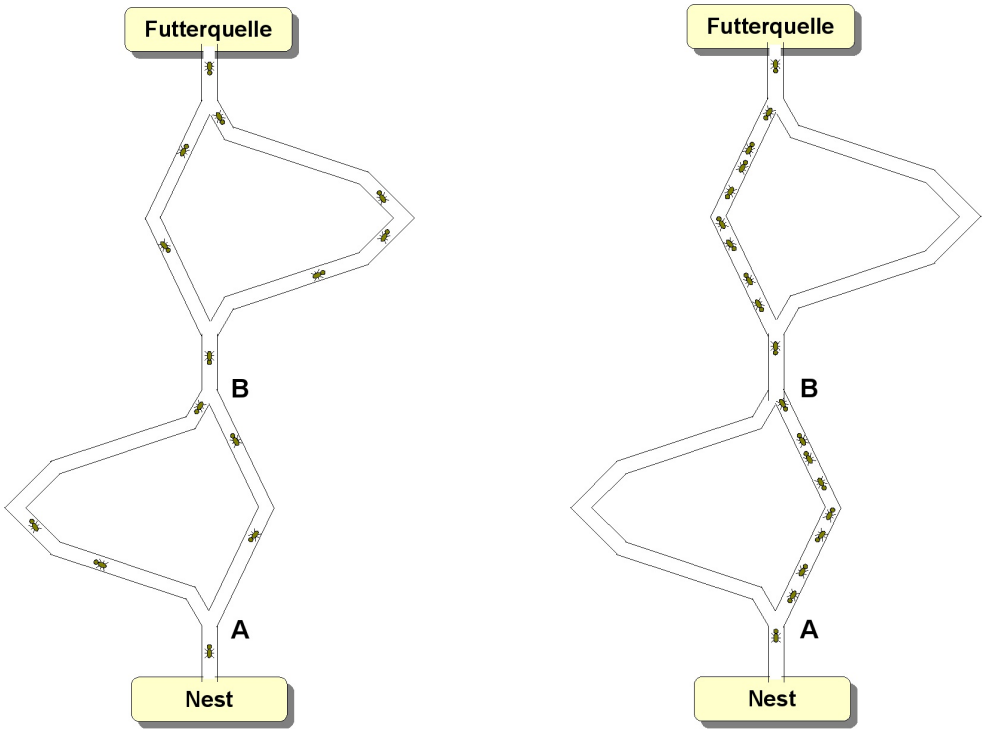
Wege verlaufen in der Natur indes nicht immer nur geradeaus. Sie werden typischerweise von diversen Hindernissen unterbrochen. Lässt sich ein so blockierter Weg nicht mehr fortsetzen,

[7] zwei Individuen interagieren indirekt, wenn eines die Umgebung verändert und das andere zu einem späteren Zeitpunkt auf diese veränderte Umgebung reagiert.

entsteht eine Weggabelung, an der das Hindernis links oder rechts umlaufen werden kann. Hier stehen einer Ameise zur Fortsetzung ihres bislang gegangenen Weges mehrere Möglichkeiten zur Verfügung. Sie kann sich aber nur für eine entscheiden.
Aus Experimenten von Goss [GADP89] und Deneubourg [DAGP90] geht hervor, dass Ameisen diese Wahl von der jeweils vorliegenden Pheromonkonzentration auf den einzelnen Teilwegen abhängig machen. Dieser Auswahlprozess kann als probabilistische Entscheidung aufgefasst und formalisiert werden: Je höher die Pheromonkonzentration, desto höher ist die Wahrscheinlichkeit, dass dieser Weg eingeschlagen wird [vgl. BDT99, S.26 ff; CDM92; CHS02; GADP89; DAGP90].
Schlussendlich führt dieser Umstand zu einem so genannten *selbstverstärkenden Prozess*, der die Ausbildung von Wegen mit sehr hohen Pheromonkonzentrationen zur Folge hat. Die Wahl eines Weges führt dazu, dass die Ameise dort wiederum Pheromon ablegt. Auf diese Weise erhöht sie erneut die Wahrscheinlichkeit zur Wahl in zukünftigen Entscheidungen [siehe CHS02; DD99].

Das *double-bridge*-Experiment

Anhand des „double bridge"-Experimentes von Goss [GADP89] (siehe Abb. 2.3) kann erklärt werden, wie Ameisen kürzeste Wege aufgrund von Pheromonspuren auffinden. Das Phänomen wird nachfolgend nur anhand des unteren Teilstücks (Knotenpunkte A und B) erklärt, für das obige gelten die Ausführungen analog.



(a) Am Anfang wählen die Ameisen beide Abzweigungen gleichwahrscheinlich

(b) Nach einiger Zeit hat sich die Wanderung auf dem kürzeren Weg stabilisiert

Abbildung 2.3: Das „double bridge" - Experiment [vgl. GADP89]

Eine Futterquelle wird durch eine doppelte Brücke vom Ameisennest getrennt. Die beiden Abzweigungen weisen eine unterschiedliche Länge auf. In dem Nest sind Ameisen der argentinische Ameisenart *Linepithea humile* angesiedelt, die nicht nur auf dem Weg von der Futterquelle zurück zum Nest Pheromonspuren ablegen, sondern bereits auch auf dem Hinweg.

Da sich anfangs noch kein Pheromon auf den Wegstrecken befindet, wählen die sich vom Nest zur Futterquelle bewegenden Ameisen am Punkt A beide Abzweigungen mit gleicher Wahrscheinlichkeit (Abb. 2.3(a)). Aufgrund der unterschiedlichen Weglängen werden hingegen die Ameisen auf dem kürzeren Weg die Futterquelle schneller erreichen. Vorausgesetzt wird, dass sich alle Ameisen mit der gleichen Durchschnittsgeschwindigkeit bewegen. Auf dem Rückweg zum Nest ist am Abzweigungspunkt B höchstwahrscheinlich auf dem kürzeren Weg eine höhere Pheromonkonzentration wahrnehmbar. Zum einen mögen die Ameisen auf dem längeren Weg am Punkt B noch nicht angekommen sein (und die Ameisen somit ihre eigene Spur wahrnehmen). Zum anderen sind bis dahin aller Voraussicht nach mehr Ameisen von der (wahrscheinlichkeitstheoretischen) Hälfte, die sich vom Nest aus für den kürzeren Weg entschieden haben, zum Gabelungspunkt B gelangt und die Pheromonkonzentration ist auf diesem Weg somit höher. Die Ameisen werden auf dem Rückweg mit höherer Wahrscheinlichkeit den kürzeren Weg wählen und neues Pheromon darauf ablegen. Hierdurch wird der selbstverstärkende Prozess vorangetrieben und nach einiger Zeit bevorzugen die meisten Ameisen den kürzeren Weg (Abb. 2.3(b)) [vgl. BDT99, S.28 ff; CHS02; CDM92; DD99; GADP89].

2.3.4 Die ACO Metaheuristik

Die Definition der ACO Metaheuristik [DD99] beschreibt verallgemeinernd, wie kombinatorische Optimierungsprobleme zur Anwendung von ACO Algorithmen codiert werden müssen und welches Prinzip diesen Algorithmen allesamt zugrundeliegt.
Mit der Zeit wurden diese Ausführungen in verschiedenen Dokumenten aufgegriffen [siehe CHS02; DS02; SD02; DS04] und teilweise in einer veränderten Notation dargestellt. Sie sind dabei vereinfacht und spezialisiert worden. Im weiteren Verlauf wird die aktuellste Literatur [DS04, S.34ff] als Grundlage verwendet.

Damit eine Instanz eines kombinatorischen Optimierungsproblems $(\mathcal{S}, f, \Omega)$ von ACO Algorithmen optimiert werden kann, muss sie sich auf ein Problem mit folgenden Eigenschaften abbilden lassen:

- Gegeben ist eine endliche Menge von Komponenten $\mathcal{C} = \{c_1, c_2, \ldots, c_{N_C}\}$.
- Die Zustände des Problems werden als Sequenzen $x = \langle c_i, c_j, \ldots, c_h, \ldots \rangle$ über den Elementen von $\mathcal{C}$ definiert.
  $\mathcal{X}$ ist die Menge aller möglichen Sequenzen (oder - äquivalent - Zustände). Die Länge einer Sequenz x ist die Anzahl der Komponenten in der Sequenz und wird mit $|x|$ bezeichnet. Die maximale Länge einer Sequenz wird durch eine positive Konstante $n < +\infty$ begrenzt.
  Die Menge $\mathcal{S}$ der Lösungskandidaten ist eine Teilmenge von $\mathcal{X}$.
- Die Menge der *zulässigen* Zustände $\tilde{\mathcal{X}} \subseteq \mathcal{X}$ wird mittels eines problemabhängigen Tests definiert: Für jede Sequenz $x \in \tilde{\mathcal{X}}$ darf es nicht unmöglich sein, x zu einer Lösung zu

vervollständigen, welche die Nebenbedingungen Ω erfüllt.
Der Test garantiert hierbei jedoch nicht, dass tatsächlich eine Vervollständigung s von x existiert, so dass $s \in \tilde{\mathcal{X}}$ gilt, sondern gibt lediglich an, ob die (Teil-)Sequenz x dieses bereits ausschließt oder noch ermöglicht.

- Die nichtleere Menge der *optimalen* Lösungen wird mit $\mathcal{S}^*$ bezeichnet, wobei $\mathcal{S}^* \subseteq \tilde{\mathcal{X}}$ und $\mathcal{S}^* \subseteq S$ gilt.

- Eine Kostenfunktion $g(s)$ ist jedem Lösungskandidaten $s \in S$ zugeordnet. Zumeist gilt $g(s) = f(s) \ \forall s \in \tilde{\mathcal{S}}$.
Sie kann zusätzlich mit einem Zeitparameter t parametrisiert werden, erforderlich z.B. bei dynamischen Optimierungsproblemen.

Sei $G = (\mathcal{C}, L)$ ein ungerichteter Graph, der anhand einer solchen Repräsentation eines kombinatorischen Optimierungsproblems aufgestellt wird. G wird auch *Konstruktionsgraph* genannt. Die Menge L vernetzt alle Komponenten vollständig miteinander. Es ist zudem generell möglich, die Vernetzung individuell zu gestalten. In den älteren Darstellungen von Dorigo [siehe DD99] wird dieses explizit erlaubt, weshalb es in den weiteren Ausführungen mit eingebracht wird.
Da die Zustände (und damit auch die Lösungen) Sequenzen von Knoten darstellen, können sie als Wege in G interpretiert werden.

Arbeitsweise von ACO Algorithmen

ACO Algorithmen werden dazu benutzt, mit Hilfe einer Kolonie (künstlicher) Ameisen Wege im Konstruktionsgraphen zu finden, die Lösungen möglichst minimaler Kosten entsprechen. An dieser Stelle lässt sich die biologische Inspiration erkennen: „Reale“ Ameisen suchen kostenminimale[8] Wege in ihrer Umgebung, (künstliche) Ameisen suchen kostenminimale[9] Wege im Konstruktionsgraphen G.

Die während des Suchprozesses gewonnenen Informationen werden in (künstlichen) Pheromonspuren τ_{ij} codiert, die in der Regel mit den Verbindungen $(i, j) \in L$ assoziiert sind. Die Pheromonspuren stellen eine Art Langzeitgedächtnis des Suchprozesses dar und leiten zukünftige Ameisen auf ihrem Weg durch den Graphen. Veraltete Pheromoninformationen früherer Ameisen sollten dabei nicht unbegrenzt bestehen bleiben. Dazu wird ein Mechanismus eingesetzt, der (inspiriert vom biologischen Ursprung) in jeder Iteration einen bestimmten Prozentsatz der Pheromonspuren verdunsten lässt. Dieser Mechanismus wird als *Pheromonverdunstung* bezeichnet.

Allgemein besteht das Ziel in der Leitung von Ameisen hin zu solchen Wegen, die Lösungen mit minimalen Kosten repräsentieren. Dazu ist es weiterhin möglich, mit jeder Verbindung einen heuristischen Wert η_{ij} zu assoziieren. Dieser spiegelt im Vorfeld bekannte, problemabhängige Informationen zum Finden guter Lösungen wieder und kann in die Lösungskonstruktion mit einbezogen werden.

[8] bezogen auf die Länge des Weges.
[9] bezogen auf eine Kostenfunktion g bzw. f.

Arbeitsweise der künstlichen Ameisen

Im Wesentlichen stellt eine Ameise k die Ausführung einer probabilistischen Lösungskonstruktion dar. Sie besitzt ein Gedächtnis M^k, das den von ihr zurückgelegten Weg im Graphen speichert und aus dem der Zustand einer Ameise hervorgeht. Dieses Gedächtnis kann für die folgenden Funktionen benutzt werden:

- Generierung zulässiger Lösungen
- Berechnung (dynamischer) heuristischer Werte (siehe Kapitel 3.5)
- Auswertung einer generierten Lösung
- Rückverfolgung des gegangenen Weges

Einer Ameise werden eine oder mehrere Terminierungsbedingungen e^k und ein Startzustand x_s^k zugeordnet. Dieser enthält üblicherweise nur eine Komponente – die Startkomponente. Davon ausgehend bewegt sich die Ameise so durch den Graphen, dass ihr Weg immer einen zulässigen Zustand darstellt, denn nur auf diese Weise kann sie letztendlich eine zulässige Lösung für das Problem erzeugen.
Falls es vom Problem her erforderlich ist, könnte zusätzlich die Bildung einer *unzulässigen* Lösung $s \in \mathcal{S} \setminus \tilde{\mathcal{S}}$ erlaubt werden, die aber sodann abhängig vom Grad der Verletzung durch die Kostenfunktion zu bestrafen ist. In dieser Arbeit ist die Bildung unzulässiger Lösungen nicht notwendig und wird nachfolgend nicht weiter betrachtet.

Wenn sich eine Ameise k in einem Zustand $x_r = \langle x_{r-1}, i \rangle \in \tilde{\mathcal{X}}$ befindet[10], darf sie sich unter der Voraussetzung zur Generierung zulässiger Lösungen vom Knoten i nur zu Knoten j aus ihrer *zulässigen Nachbarschaft* $\mathcal{N}_i^k$ bewegen, die zu $\mathcal{N}_i^k = \left\{ j \,\middle|\, (i,j) \in L \wedge \langle x_r, j \rangle \in \tilde{\mathcal{X}} \right\}$ definiert wird [vgl. DD99]. Die Definition ist den älteren Ausführungen zu entnehmen, da die Darstellungen aus [DS04] keine anpassbare Vernetzung erlauben. Wird eine angepasste Vernetzung verwendet, muss die Bedingung $(i,j) \in L$ noch mit in die Formel einfließen.
Die zulässige Nachbarschaft enthält damit alle Nachbarn von i, unter deren Hinzunahme sich die Ameise wieder in einem zulässigen Zustand befindet.

Die Ameise k muss sich nun für einen dieser Knoten entscheiden. Die Entscheidung wird auf Basis einer Wahrscheinlichkeitsverteilung über den einzelnen Knoten gefällt. Die Wahrscheinlichkeit, mit der sich die Ameise k für den Knoten j entscheidet, wird mit p_{ij}^k bezeichnet und *Übergangswahrscheinlichkeit* genannt.
Dieser Vorgang wiederholt sich, bis mindestens eine Terminierungsbedingung erfüllt ist. Anschließend endet die Lösungskonstruktion.

Wenn sich eine Ameise vom Knoten i zum Nachbarknoten j bewegt, hat sie die Möglichkeit, den Pheromonwert τ_{ij} auf der Kante (i,j) zu verändern. Dieser Mechanismus wird in der englischen Literatur als *online step-by-step pheromone update* - oder auch *local pheromone update* - bezeichnet.
Sobald eine zulässige Lösung für das Problem konstruiert ist (d.h. der Zustand der Ameise repräsentiert eine zulässige Lösung), kann der gegangene Weg sozusagen rückwärts durchlaufen und die Pheromonspuren auf diesen Kanten verändert werden, was man biologisch mit

[10] d.h. die Ameise befindet sich am Knoten i und ist über Sequenz x_{r-1} dorthin gelangt.

dem Weg von der Futterquelle zurück zum Nest vergleichen kann. Diesen Mechanismus bezeichnet man in der englischen Literatur als *online delayed pheromone update* - oder auch *global pheromone update*. Die Ablage solcher (neuen) Pheromonspuren trägt die Bezeichnung *Pheromonverstärkung*.

Im Unterschied zum biologischen Ursprung sind künstliche Ameisen nicht mehr natürlichen Gesetzen unterworfen und können mit einer Reihe zusätzlicher Fähigkeiten versehen werden, um effizientere Algorithmen zu entwickeln. Dazu zählen u.a. das Gedächtnis und die heuristischen Informationen.

Anmerkung
Der erwähnte problemabhängige Test definiert die Menge der zulässigen Zustände und beeinflusst somit direkt die zulässige Nachbarschaft jedes einzelnen Zustands. In der Implementierung dieses Tests ist es möglich, nicht nur sicherzustellen, dass die Nebenbedingungen Ω in keinem Zustand der Ameisen verletzt werden, sondern auch anzugeben, wie sich die Ameisen durch den Konstruktionsgraphen bewegen sollen. Dieses wird als *Konstruktionspolitik* bezeichnet.
Bei Adaptionen auf verschiedene Probleme kann mittels des Tests stets die gewünschte Konstruktionspolitik implementiert werden und muss sich nicht in der Vernetzung widerspiegeln, so dass eine Vollvernetzung keinerlei Einschränkung darstellt. Dadurch können jedoch Verbindungen entstehen, die von vornherein ungewünscht sind und durch den Test stets herausgefiltert werden müssen. Dieser Aufwand lässt sich durch eine angepasste Vernetzung vermeiden. In den älteren Darstellungen von Dorigo [DD99] ist die Menge L auf das jeweilige Problem anpassbar. Einige Implementierungen verwenden eine solche angepasste Vernetzung, wie z.B. in der Anwendung auf das *University Course Timetabling Problem* von Socha, Knowles und Sampels [SKS02].

Diese Grundlagen versteifen sich deshalb nicht nur auf die Darstellungen aus [DS04], sondern im Sinne einer effizienteren Lösungskonstruktion für das BCNP wird die Möglichkeit zu einer individuellen Definition der Vernetzungsstruktur offengehalten.

Ablaufschema von ACO Algorithmen

Es existieren mehrere ACO Algorithmen, die Einzelheiten des gerade beschriebenen Prinzips auf verschiedene Weise umsetzen. Der von Dorigo und Di Caro [DD99] dargestellte Pseudocode eines ACO Algorithmus zeigt ein solches allgemeines Ablaufschema auf. Es ist auf dynamische, wie auch auf statische Optimierungsprobleme, anwendbar.
In der Praxis werden ACO Algorithmen bei statischen Optimierungsproblemen häufig auf einfachere Art realisiert, indem die Ameisen den Konstruktionsgraphen synchronisiert und nacheinander durchlaufen [siehe CHS02; Stü98, S.25; DS04, S.68]. Das BCNP ist ein statisches kombinatorisches Optimierungsproblem. Daher werden sich die weiteren Ausführungen auf ein solches vereinfachtes Ablaufschema berufen.
Algorithmus 2.3.1 orientiert sich an der Literatur [MM05, S.404; DS04, S.68; MC01] und illustriert das algorithmische Skelett von ACO Algorithmen für statische kombinatorische Optimierungsprobleme.

In der Hauptschleife erschafft zunächst jede Ameise eine zulässige Lösung, indem sie wie beschrieben durch den Konstruktionsgraphen wandert. Einigen (oder allen) Ameisen kann in der Phase der lokalen Suche erlaubt werden, ihre gefundene Lösung durch Anwendung einer

Algorithmus 2.3.1 : Pseudocode von ACO Algorithmen für statische kombinatorische Optimierungsprobleme

```
   /* Initialisierung ---------------------                    */
1  Setzen der Parameterwerte
2  Initialisierung der Pheromonspuren
   /* Hauptschleife -----------------------                    */
3  solange (Abbruchbedingung nicht erfüllt) tue
4  |  für Ameise k = 1 bis m tue
5  |  |  Lösungskonstruktion
6  |  ende
7  |  Anwendung einer lokalen Suche                  /* optional */
8  |  Anpassung der Pheromonspuren (globales Pheromonupdate)
9  ende
```

effizienten lokalen Suche zu verbessern. Diese wird auf den Weg der Ameise angewendet. Die Verwendung der lokalen Suche ist nicht zwingend, bringt jedoch für gewöhnlich eine deutliche Performancesteigerung mit sich [vgl. DS02; Stü98, S.24f].
Zuguterletzt werden in der aktuellen Iteration die Pheromonspuren auf allen Kanten des Konstruktionsgraphen G verändert, indem sie um einen gewissen Prozentsatz verdunsten und die Pheromonintensivierung auf die von den Ameisen benutzten Kanten angewendet wird (*globales Pheromonupdate*).

In den folgenden Unterkapiteln werden einzelne ACO Algorithmen detailliert vorgestellt. Beschränkt wurde die Auswahl auf das *Ant System* [DMC96], *Ant Colony System* [DG97] und *MAX-MIN Ant System* [SH00], da diese im Bereich der ACO Optimierung die meistverwendeten Algorithmen darstellen. Zusätzlich wird das Hypercube-Framework [BRD01] aufgenommen. Zunehmend erfreut sich auch der Algorithmus *Approximate Nondeterministic Tree Search (ANTS)* [Man99] großer Beliebtheit. Dieser ist jedoch aufgrund der Notwendigkeit zur Berechnung einer unteren Schranke für die Lösungsqualität bei Hinzunahme einer bestimmten Komponente in die Lösung nicht anwendbar. Während einer Permutationsgenerierung für das k-seitige BCNP können keine Informationen darüber ermittelt werden, welche Lösungsqualität ausgehend von einer spezifischen Teil-Permutation höchstens noch erzielbar ist. Solch eine Schranke hängt von der Seitenzuordnung und der Reihenfolge noch aufzunehmender Komponenten ab. Beides ist zu diesem Zeitpunkt noch nicht determiniert.

2.3.5 Ant System (AS)

Das Ant System [DMC96] wurde von Dorigo, Maniezzo und Colori im Hinblick einer Anwendung auf das TSP entwickelt und war der erste ACO Algorithmus. Weitere Ausführungen über das AS finden sich u.a. in [BDT99, S.41ff; CHS02; Eng05, S.376ff; DS04, S.70ff; SD99], wobei sich die folgenden Darstellungen an der Literatur von Dorigo und Stützle [DS04] sowie Engelbrecht [Eng05] orientieren.

Die initialen Pheromonspuren τ_{ij} können beim AS fest auf eine Konstante τ_0 gesetzt oder aber für jede Kante randomisiert bestimmt werden, wobei der Wert von τ_{ij} auf Grundlage einer Gleichverteilung über dem Intervall $[0, \tau_0]$ basiert. τ_0 ist eine kleine positiv reelle Zahl. Darüber

hinaus gibt es die Möglichkeit, die initialen Pheromonspuren auf Basis einer zuvor ermittelten „guten“ Lösung zu bestimmen, um die Ameisen bereits zu Beginn auf vielversprechende Wege zu führen [vgl. Eng05, S.378 u. 428].
Ungeachtet dessen wird in der Regel die Initialisierung mit konstanten Werten bevorzugt, so dass nur diese Initialisierung künftig Betrachtung findet.

Lösungskonstruktion

Nach den Ausführungen aus Kapitel 2.3.4 muss sich eine Ameise in jedem Konstruktionsschritt probabilistisch für einen nächsten zu besuchenden Knoten aus ihrer zulässigen Nachbarschaft $\mathcal{N}_i^k$ entscheiden.
Die Übergangswahrscheinlichkeit p_{ij}^k berechnet sich im AS wie folgt:

$$p_{ij}^k = \begin{cases} \frac{[\tau_{ij}]^\alpha * [\eta_{ij}]^\beta}{\sum_{u \in \mathcal{N}_i^k} [\tau_{iu}]^\alpha * [\eta_{iu}]^\beta} & \text{falls } j \in \mathcal{N}_i^k \\ 0 & \text{falls } j \notin \mathcal{N}_i^k \end{cases} \tag{2.2}$$

Für alle Knoten, die nicht Teil der zulässigen Nachbarschaft sind, muss die Übergangswahrscheinlichkeit notwendigerweise 0 sein. Auf den übrigen Knoten wird eine Wahrscheinlichkeitsverteilung definiert, wobei die Wahrscheinlichkeit zur Wahl eines Knotens j proportional zur Pheromonspur τ_{ij} und der heuristischen Information η_{ij} auf der Kante (i, j) ist. Je höher diese Werte sind, desto eher wird die Kante gewählt. Die Parameter α und β beeinflussen die Gewichtung der jeweiligen Werte in der Wahrscheinlichkeitsverteilung, d.h. welchen *Einfluss* sie auf die Entscheidungsfindung haben.

Sollte $\alpha = 0$ sein, werden Pheromonspuren bei der Entscheidung nicht mehr berücksichtigt, denn $\tau_{ij}^0 = 1$ für alle $i, j \in \mathcal{C}$. Infolgedessen wird die Entscheidung allein von heuristischen Informationen abhängig gemacht und Knoten mit höheren heuristischen Werten ebenso mit höherer Wahrscheinlichkeit gewählt. Die Suche wird dann zu einem stochastischen Greedy Algorithmus herabgestuft.
Ist hingegen $\beta = 0$, so werden lediglich die Pheromonspuren zur Entscheidungsfindung herangezogen. Dieser Umstand ist mit dem Nachteil einer schnellen *Stagnation* verbunden. Stagnation bezeichnet eine Situation, in der **alle** Ameisen **dieselbe** Lösung bilden, üblicherweise lokale Optima. Eine Stagnation tritt vor allem dann auf, wenn an jedem Entscheidungspunkt die Übergangswahrscheinlichkeit für einen Knoten deutlich höher ist als die der übrigen. Die Ameisen werden nur auf einen Weg geleitet und erkunden den Suchraum nicht ausgiebig.
Insgesamt entsteht daraus die Notwendigkeit zu einer geeigneten Abstimmung der Parameter α und β [vgl. DS04, S.71; CHS02; SD99].

Generell gilt: Je höher α, desto eher wird die Ameise in ihrer Lösungskonstruktion Kanten mit (relativ betrachtet) hohen Pheromonwerten wählen. Analog gilt dies auch für den Parameter β und den Einfluss heuristischer Werte.

Pheromonupdate

Ursprünglich existierten drei unterschiedliche Varianten des AS mit jeweils verändertem Pheromonupdate: *ant-density*, *ant-quantity* und *ant cycle* AS [siehe DMC96]. Heutzutage wird das AS für gewöhnlich nur noch in der Variante *ant-cycle* verwendet. Die übrigen beiden wurden

aufgrund ihrer schlechten Performance verworfen [vgl. DS04, S.70; CHS02]. Die anknüpfenden Darstellungen beziehen sich infolgedessen auf die *ant-cycle* Variante.

Das AS verändert die Pheromonspuren, nachdem alle Ameisen die Lösungskonstruktion beendet haben. Das lokale Pheromonupdate kommt nicht zur Anwendung.
Unter Berücksichtigung von Pheromonverdunstung und Pheromonverstärkung ergibt sich die neue Pheromonkonzentration auf jeder Kante $(i, j) \in L$ wie folgt:

$$\tau_{ij} \longleftarrow (1 - \rho) * \tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij}^{k} \qquad \forall\, (i, j) \in L \tag{2.3}$$

$0 < \rho \leq 1$ gibt die Pheronom-*Verdunstungsrate* an. Die von der Ameise k auf der Kante (i, j) abgelegte Pheromonmenge wird mit $\Delta\tau_{ij}^{k}$ bezeichnet.
Die Verdunstungsrate ρ bestimmt, wie schnell vorherige (gute) Lösungen „vergessen“ werden und besitzt direkten Einfluss auf die Erkundung des Suchraumes: Je größer ρ, desto schneller verdunsten die vorliegenden Pheromonspuren und desto zufälliger bewegen sich die Ameisen im Konstruktionsgraphen. Auf diese Weise wird der Suchraum in höherem Maße erkundet [vgl. Eng05, S.374].

Die von der Ameise k abgelegte Pheromonmenge berechnet sich nach Formel 2.4.

$$\Delta\tau_{ij}^{k} = \begin{cases} \frac{Q}{f(s^k)} & \text{falls } (i, j) \text{ Element der Lösung } s^k \text{ ist} \\ 0 & \text{sonst} \end{cases} \tag{2.4}$$

s^k ist die von der Ameise k konstruierte Lösung. Sie darf nur auf denjenigen Kanten Pheromon ablegen, die sie zur Erstellung ihrer Lösung benutzt. Alle anderen Kanten erhalten eine Pheromonverstärkung von 0.
Die Menge des abgelegten Pheromons ist dabei proportional zur Qualität der generierten Lösung (Minimierungsproblem vorausgesetzt). Q ist eine positive Konstante.

Nach dem obigen Schema ergibt sich der Ablauf des AS wie in Algorithmus 2.3.2 als Pseudocode dargestellt.

Algorithmus 2.3.2 : Pseudocode des Ant Systems

```
   /* Initialisierung ---------------------                    */
1  Setzen der Parameterwerte
2  Initialisierung der Pheromonspuren
   /* Hauptschleife ----------------------                     */
3  solange (Abbruchbedingung nicht erfüllt) tue
4  |  für Ameise k = 1 bis m tue
5  |  |  Lösungskonstruktion anhand Formel 2.2
6  |  ende
7  |  Anwendung einer lokalen Suche                  /* optional */
8  |  Anpassung der Pheromonspuren nach Formel 2.3
9  ende
```

2.3.6 MAX-MIN Ant System (MMAS)

Obwohl das AS beim TSP ermutigende Ergebnisse erlangte, konnte dessen Performance nicht mit den „state-of-the-art"-Algorithmen mithalten. Das AS weist allgemein den Nachteil einer frühen Stagnation des Suchprozesses auf. Experimente haben gezeigt, dass die Leistung durch eine stärkere Orientierung an den während des Suchprozesses besten gefundenen Lösungen gesteigert werden kann. Eine gleichzeitige Vermeidung zu früher Stagnation ist für eine Performancesteigerung unerlässlich [vgl. SH00].

Das von Stützle und Hoos [SH96] entwickelte MMAS greift diese Anforderungen auf und ist momentan eines der leistungsstärksten ACO Algorithmen. Weitere Darstellungen finden sich u.a. in [Eng05, S.383ff; DS04, S.74ff; SH00], wobei fortan die Literatur [SH00] als Grundlage dient.

Das MMAS unterscheidet sich vom AS in drei Aspekten:

- Pheromonspuren werden nicht mehr von allen Ameisen abgelegt, sondern ausschließlich von der Ameise mit der **besten Lösung** s^{best}. Die Lösung s^{best} kann entweder zu der besten Lösung der aktuellen Iteration oder aber zur global besten Lösung seit Beginn des Algorithmus bestimmt werden.
  Da nur eine einzige Ameise Pheromon ablegen darf, wird eine Konzentration der Suche auf Bereiche erreicht, die sich stärker an vorherigen vielversprechenden Lösungen orientieren. Tauchen Lösungselemente hierbei häufig auf, werden sie durch diesen Prozess intensiv verstärkt.

- Um eine frühe Stagnation zu vermeiden, werden die Pheromonkonzentrationen aller Kanten auf ein bestimmtes Intervall $[\tau_{min}, \tau_{max}]$ beschränkt.

- Die initiale Pheromonkonzentration ist nicht mehr variabel, sondern wird fest auf den Wert von τ_{max} gesetzt, was eine stärkere Erkundung des Suchraums zu Beginn des Algorithmus provoziert.

Lösungskonstruktion

Die Lösungskonstruktion geschieht auf die gleiche Weise wie beim AS und die Übergangswahrscheinlichkeiten berechnen sich nach Formel 2.2.

Pheromonupdate

Auch beim MMAS werden die Pheromonspuren erst geändert, nachdem alle Ameisen eine Lösung konstruiert haben (globales Pheromonupdate). Allerdings darf hierbei nur die Ameise mit der besten Lösung s^{best} eine Pheromonintensivierung realisieren. Die Pheromonverdunstung findet nach wie vor auf allen Kanten des Graphen statt. Formel 2.5 verwirklicht das globale Pheromonupdate im MMAS.

$$\tau_{ij} \longleftarrow (1-\rho) * \tau_{ij} + \Delta\tau_{ij}^{best} \qquad \forall\,(i,j) \in L \tag{2.5}$$

Die auf den Kanten abgelegte Pheromonmenge $\Delta\tau_{ij}^{best}$ ergibt sich gemäß Formel 2.6.

$$\Delta\tau_{ij}^{best} = \begin{cases} \frac{1}{f(s^{best})} & \text{falls } (i,j) \text{ Element der Lösung } s^{best} \text{ ist} \\ 0 & \text{sonst} \end{cases} \tag{2.6}$$

Die Lösung s^{best} kann entweder die beste Lösung der aktuellen Iteration darstellen (s^{ib}) oder die beste Lösung seit Beginn des Suchprozesses (s^{gb}). Bei Wahl von s^{gb} besteht die Gefahr, dass sich die Suche sehr schnell auf diese eine Lösung versteift und der Suchraum nicht ausreichend erkundet wird. In der Regel wird im MMAS $s^{best} = s^{ib}$ gesetzt; s^{ib} kann von Iteration zu Iteration differieren.
Stützle und Hoos [SH00] nennen zudem noch kombinierte Strategien zur Wahl von s^{best}, auf die hier nicht weiter eingegangen wird.

Pheromongrenzen

Unabhängig von der Wahl der Lösung s_{best} kann eine Stagnation der Suche auftreten. Um dieses zu vermeiden, werden Pheromongrenzen eingeführt. Sie sollen die Pheromonspuren auf einen festgelegten Wertebereich einschränken, so dass stets die Bedingung

$$\tau_{min} \leq \tau_{ij} \leq \tau_{max} \qquad \forall\ (i,j) \in L \tag{2.7}$$

gilt. τ_{min} stellt die untere und τ_{max} die obere Pheromongrenze dar.
Nach jeder Iteration wurde die Pheromonkonzentration durch das globale Pheromonupdate geändert und sodann muss die Einhaltung dieser Bedingung sichergestellt werden. Falls $\tau_{ij} > \tau_{max}$ ist, so wird τ_{ij} auf den Wert von τ_{max} festgelegt. Analog findet die Umsetzung für die untere Grenze statt. Ist $\tau_{ij} < \tau_{min}$, so setze $\tau_{ij} = \tau_{min}$.

Zu Klären ist nun noch die Bestimmung der Pheromongrenzen. τ_{max} wird gemäß Formel 2.8 determiniert.

$$\tau_{max} = \frac{1}{\rho} * \frac{1}{f(s^{gb})} \tag{2.8}$$

τ_{max} approximiert die maximale Pheromonmenge auf einer Kante im Konstruktionsgraphen[11]. Bei Änderung der global besten Lösung verändert sich auch der Wert von τ_{max}. τ_{max} muss aufgrund dessen nach jeder Iteration neu berechnet werden.

Die Berechnung von τ_{min} gestaltet sich weitaus aufwendiger. Ziel ist die Justierung der Pheromongrenzen mittels der Wahl von τ_{min}, so dass die gewünschte Erkundung des Suchraumes erreicht wird. Je größer das Verhältnis von $\frac{\tau_{max}}{\tau_{min}}$, desto kleiner wird der erkundete Suchraum. Liegen die Pheromongrenzen nahe beieinander, so werden selbst die Kanten mit dem Pheromonwert τ_{min} mit der Zeit nie ganz „unattraktiv“ und die Erkundung des Suchraumes erhalten.

Durch Einführung der Pheromongrenzen soll eine Stagnation der Suche vermieden werden. Im MMAS wird ein anderer Begriff eingeführt: *Konvergenz*. Im Gegensatz zu *Stagnation* bezeichnet die *Konvergenz* eine Situation, in der ein bester Weg existiert, dessen zugehörige Kanten einen Pheromonwert von τ_{max} aufweisen und alle übrigen Kanten haben einen Pheromonwert

[11] für nähere Informationen siehe [SH00].

von τ_{min} [vgl. SH00; SKS02].
τ_{min} sollte so eingestellt werden, dass in einer Konvergenzsituation selbst Kanten mit geringen Pheromonwerten noch eine gewisse Wahrscheinlichkeit aufweisen, von einer Ameise gewählt zu werden. Hierdurch ist eine gezielte Stagnationsvermeidung möglich.

Die untere Pheromongrenze τ_{min} berechnet sich in Abhängigkeit des zugrundeliegenden Konstruktionsgraphen, τ_{max}, und einer (einstellbaren) Wahrscheinlichkeit p_{best} [Stü98; SH00]. p_{best} gibt an, mit welcher Wahrscheinlichkeit der beste Weg (der Weg mit der höchsten Pheromonkonzentration) in einer Konvergenzsituation von einer Ameise konstruiert werden soll. Bestimmt man p_{best} zu einem geringen Wert, so müssen die obere und untere Pheromongrenze nahe beieinander liegen. Auf diese Weise werden alternative Wege häufig gewählt. Je größer p_{best}, desto geringer muss die untere Grenze τ_{min} eingestellt werden. Formel 2.9 setzt dieses um.

$$\tau_{min} = \frac{\tau_{max} * \left(1 - \sqrt[n]{p_{best}}\right)}{(avg - 1) * \sqrt[n]{p_{best}}} \tag{2.9}$$

n ist die Anzahl der Entscheidungen, die eine Ameise zur Konstruktion ihrer Lösung bei der Wanderung durch den Konstruktionsgraphen trifft. avg bezeichnet die durchschnittliche Anzahl der Kanten (oder Knoten), für die sich eine Ameise an jedem Entscheidungspunkt entscheiden kann.
p_{best} steht in direkter Beziehung zu der Erkundung des Suchraumes. Für die Wahl $p_{best} = 1$ ist $\tau_{min} = 0$.

Initialisierung der Pheromonspuren

Die Werte von τ_{min} und τ_{max} sind von der bislang besten Lösung abhängig und stehen aus diesem Grund nicht von Anfang an zur Verfügung.
Jedoch sollen die Pheromonspuren so initialisiert werden, dass sie nach der ersten Iteration einen Wert von τ_{max} aufweisen.

Dieses kann recht einfach erreicht werden, indem die Initialwerte von τ_{ij} auf einen willkürlichen sehr hohen Wert gesetzt werden. Nach der ersten Iteration wird überprüft, ob die Pheromonspuren die von τ_{min} und τ_{max} definierten Grenzen einhalten. Alle höheren Werte werden auf τ_{max} gesetzt und automatisch die gewünschte Bedingung sichergestellt. Eingeführt wurde eine derartige Initialisierung mit τ_{max}, um die Erkundung des Suchraumes während der ersten Iterationen des Algorithmus voranzutreiben.

Reinitialisierung der Pheromonspuren

Die Performance des MMAS kann durch einen zusätzlichen PTS[12]-Mechanismus erhöht werden. Zwar kann eine Stagnation prinzipiell durch die Anpassung der Pheromongrenzen vermieden werden, je höher aber die Wahrscheinlichkeit p_{best} gewählt wird, desto geringer ist τ_{min} und desto geringer wird die Wahrscheinlichkeit, in einer Konvergenzsituation neue Wege zu erkunden.
Der PTS-Mechanismus reinitialisiert die Pheromonspuren in einer Konvergenzsituation proportional zu ihrer Differenz zu τ_{max}:

$$\tau^*_{ij} = \tau_{ij} + \delta * (\tau_{max} - \tau_{ij}) \qquad \text{mit } 0 < \delta < 1 \tag{2.10}$$

[12] *Pheromone Trail Smoothing*

$\tau^*_{ij}(t)$ gibt die Pheromonmenge auf der Kante (i,j) nach Anwendung der Pheromonglättung an. Die grundlegende Idee besteht in der Förderung der Erkundung des Suchraumes nach einer Konvergenz, um nicht ausschließlich in direkter Nähe der aktuell besten Lösung zu suchen. Dazu werden die vorliegenden Pheromonspuren relativ zu dem maximalen Pheromonwert τ_{max} erhöht. Gleichwohl berücksichtigt der Mechanismus eine Gewichtung in Richtung der besten gefundenen Lösung: Für $\delta < 1$ gehen die durch den Suchprozess gewonnenen Pheromoninformationen nicht vollständig verloren. Die Wahl von $\delta = 1$ entspricht einer Reinitialisierung der Pheromonkonzentrationen auf τ_{max}, was einem Neustart gleichkommt. Für $\delta = 0$ ist das PTS vollständig deaktiviert.

Das PTS kann zum einen *nach* einer Konvergenzsituation eingeleitet werden oder *kurz davor*. Als Indikator für „kurz davor“ ist es u.a. möglich, den *durchschnittlichen* λ-*Verzweigungsfaktor*[13] auszuwerten und heranzuziehen. Dieser gibt Aufschluss über die Größe des von den Ameisen erkundeten Suchraumes. Sobald der durchschnittliche λ-Verzweigungsfaktor einen bestimmten Wert unterschreitet, wird das PTS eingeleitet.
Es steht frei, selbst eigene Konvergenzmaße für das vorliegende Problem zu definieren, um Konvergenzsituationen detektieren zu können [siehe z.B. Blu02].

Der Ablauf des MMAS ergibt sich gemäß des Pseudocodes aus Algorithmus 2.3.3.

Algorithmus 2.3.3 : Pseudocode des MAX-MIN Ant Systems

```
   /* Initialisierung ---------------------                    */
 1 Setzen der Parameterwerte
 2 Initialisierung der Pheromonspuren
   /* Hauptschleife -----------------------                    */
 3 solange (Abbruchbedingung nicht erfüllt) tue
 4 |   für Ameise k = 1 bis m tue
 5 |   |   Lösungskonstruktion gemäß Formel 2.2
 6 |   ende
 7 |   Anwendung einer lokalen Suche                  /* optional */
 8 |   Ermittlung der besten Lösung s_best
 9 |   Anpassung der Pheromonspuren nach Formel 2.5
10 |   Anpassung von τ_max und τ_min
11 |   Einhaltung der Pheromongrenzen sicherstellen
12 |   wenn (Suche hat konvergiert) dann
13 |   |   PTS einleiten                              /* optional */
14 |   ende
15 ende
```

2.3.7 Ant Colony System (ACS)

Das Ant Colony System (ACS) wurde von Dorigo und Gambardella [DG97] als ein Nachfolger des AS entwickelt. Weitere Darstellungen sind u.a. in [BDT99, CHS02, Eng05, S.383ff, DS04, S.76ff] zu finden. Die folgenden Ausführungen orientieren sich an Dorigo und Stützle [DS04].

[13] *average* λ-*branching factor*, für nähere Informationen siehe [SH96; DS04, S.87].

Das ACS unterscheidet sich vom AS in folgenden drei Punkten:

- Die Lösungskonstruktion konzentriert sich stärker auf die Sucherfahrung vorheriger Ameisen.
- Das globale Pheromonupdate wird nur auf der besten Lösung angewendet, d.h. selbst die Pheromonverdunstung findet im Gegensatz zum AS und MMAS nicht mehr auf allen Kanten des Graphen statt.
- Das lokale Pheromonupdate wird eingesetzt.

Lösungskonstruktion

Die Lösungskonstruktion geschieht im Vergleich zum AS und MMAS auf veränderte Weise. Eine so genannte *pseudozufällig proportionale* Regel bestimmt den nächsten Knoten j, den die Ameise k vom Knoten i ausgehend wählt:

$$j = \begin{cases} argmax_{l \in \mathcal{N}_i^k} \left\{ \tau_{il} * [\eta_{il}]^\beta \right\} & \text{falls } q \leq q_0 \\ J & \text{sonst} \end{cases} \tag{2.11}$$

q ist eine aus dem Intervall $[0, 1]$ zufällig gleichverteilte Zufallsvariable und $0 \leq q_0 \leq 1$ ist ein einstellbarer Parameter. Die Zufallsvariable J wird auf Basis der bekannten Wahrscheinlichkeitsverteilung aus Formel 2.2 (mit $\alpha = 1$) ermittelt. Es fällt auf, dass auch zur Bestimmung der besten Kante der Parameter α auf 1 gesetzt wird und in der Formel nicht mehr auftritt. Die Ameise wählt mit einer Wahrscheinlichkeit von q_0 die Kante mit der höchst bewerteten Kombination aus Pheromonkonzentration und heuristischen Information. Mit der Gegenwahrscheinlichkeit von $(1 - q_0)$ wird die bekannte Erkundung des Suchraumes realisiert.
Je größer q_0, desto eher wird bei jeder Entscheidung direkt die beste Kante gewählt. So muss sich diese nicht erst in der Wahrscheinlichkeitsverteilung behaupten.

Globales Pheromonupdate

In dem globalen Pheromonupdate des ACS darf einzig die Ameise mit der besten Lösung (s^{gb} oder s^{ib}) eine Veränderung der Pheromonspuren vornehmen.
Eingeschlossen ist dabei, wie erwähnt, auch die Pheromonverdunstung. Diese findet nun nicht mehr auf allen Kanten des Konstruktionsgraphen Anwendung. Das globale Pheromonupdate bestimmt sich zu Formel 2.12.

$$\tau_{ij} \longleftarrow (1 - \rho) * \tau_{ij} + \rho * \Delta\tau_{ij}^{best} \qquad \forall\, (i, j) \in s^{best} \tag{2.12}$$

$$\text{mit } \Delta\tau_{ij}^{best} = \frac{1}{f\left(s^{best}\right)}$$

Auffällig ist überdies, dass die abgelegte Pheromonspur nun mit dem Parameter ρ multipliziert wird. Die neue Pheromonmenge auf der Kante (i,j) stellt damit einen gewichteten Durchschnitt der alten und der neu abgelegten Pheromonmenge dar.

Lokales Pheromonupdate

Nebst dem globalen Pheromonupdate kommt im ACS ebenso das *lokale* Pheromonupdate zur Anwendung. Nachdem die jeweilige Ameise eine Kante (i, j) im Konstruktionsgraphen passiert hat, verändert sie deren Pheromonwert nach der Formel 2.13.

$$\tau_{ij} \longleftarrow (1 - \xi) * \tau_{ij} + \xi * \tau_0 \tag{2.13}$$

Mit ξ ist die Pheromon-Verdunstungsrate für das lokale Pheromonupdate bezeichnet und τ_0 gibt nach wievor den initialen Pheromonwert an.
Das lokale Pheromonupdate fördert eine stärkere Erkundung des Suchraumes nachfolgender Ameisen. Jedesmal, wenn eine Ameise eine Kante benutzt, verringert[14] sie den zugehörigen Pheromonwert und erniedrigt auf diese Weise die Wahrscheinlichkeit zur Wahl dieser Kante in weiteren Lösungskonstruktionen [siehe DS04, S.78].
Zu beachten ist, dass die Pheromonmenge auf einer Kante – bedingt durch die Formeln 2.12 und 2.13 – niemals unter den Wert von τ_0 absinken kann. Es gilt stets die folgende Ungleichung: $\tau_0 \leq \tau_{ij} \leq \frac{1}{f(s^{best})}$. Der Wert von τ_0 muss also immer deutlich kleiner als der Kehrwert der besten Lösungsqualität sein, damit der Algorithmus entsprechend den eben dargelegten Ausführungen arbeiten kann. Die Pheromonspuren können nicht abgesenkt werden, wenn die initiale Pheromonmenge den Kehrwert der besten Lösungsqualität übersteigt.

Algorithmus 2.3.4 : Pseudocode des Ant Colony Systems

```
   /* Initialisierung ---------------------                    */
 1 Setzen der Parameterwerte
 2 Initialisierung der Pheromonspuren
   /* Hauptschleife -----------------------                    */
 3 solange (Abbruchbedingung nicht erfüllt) tue
 4 |   für Ameise k = 1 bis m tue
 5 |   |   Lösungskonstruktion anhand Formel 2.11 und lokales Pheromonupdate
 6 |   |   nach Formel 2.13
 7 |   ende
 8 |   Anwendung einer lokalen Suche                  /* optional */
 9 |   Ermittlung der besten Lösung s_best
10 |   Anpassung der Pheromonspuren nach Formel 2.12
11 ende
```

2.3.8 Hypercube Framework

Bei Anwendung des AS und MMAS können die Pheromonspuren auf den Kanten des Konstruktionsgraphen mit der Zeit verschiedene Werte annehmen. Selbst im MMAS verändert sich das Intervall $[\tau_{min}, \tau_{max}]$ fortlaufend und die Pheromonspuren können zu verschiedenen Zeitpunkten in verschiedenen Wertebereichen liegen. Das von Blum, Roli und Dorigo [BRD01] eingeführte *Hypercube Framework* verwendet einen Mechanismus, der die Pheromonkonzentrationen stets auf den Wertebereich $[0, 1]$ begrenzt. Dabei werden sie automatisch maßstabsgerecht angepasst und auf das Intervall $[0, 1]$ passend skaliert.

[14] gilt nur wenn $\tau_{ij} > \tau_0$.

Das Hypercube Framework wurde für das AS und das MMAS vorgestellt. Es basiert auf der Grundlage, dass für die meisten kombinatorischen Optimierungsprobleme eine Formulierung als mathematisches Programm existiert. In einer solchen Formulierung können die Lösungen als binäre Vektoren (0-1 Integer Programmierung) repräsentiert werden. Es existiert eine Menge von Entscheidungsvariablen, die (typischerweise) mit den Lösungskomponenten (Knoten) des Konstruktionsgraphen assoziiert werden und Werte aus dem Bereich $\{0, 1\}$ annehmen können.
In den Darstellungen von Blum, Roli und Dorigo [BRD01] werden die Entscheidungsvariablen eben nun mit den Komponenten assoziiert. Die Lösungen drücken aus, ob sich eine bestimmte Komponente in der Lösung befindet und die Pheromonspuren sind auf den **Knoten** des Graphen abgelegt.
Bei einigen Problemen, wie beispielsweise dem TSP, befinden sich alle Komponenten des Graphen in **jeder** zulässigen Lösung. Diese werden hierbei durch den *Weg* repräsentiert und nicht allein durch die verwendeten Komponenten, was konform zu den vorigen Darstellungen dieses Kapitels ist. Die Entscheidungsvariablen sind dann mit den Kanten assoziiert und nehmen den Wert 1 an, wenn die jeweilige Kante für die Lösung verwendet wurde und ansonsten den Wert 0 [vgl. DS04, S.82].
Um in Gleichklang mit den vorherigen Ausführungen zu bleiben, in denen die Pheromonwerte den Kanten zugeordnet sind, werden die Darstellungen aus [BRD01] im Folgenden dahingehend angepasst.

Ant System mit Hypercube Framework (HC-AS)

Das globale Pheromonupdate des AS wird im HC-AS nach Formel 2.14 neu definiert, damit die Pheromonspuren stets innerhalb des Intervalls $[0, 1]$ gehalten werden.

$$\tau_{ij} \longleftarrow (1 - \rho) * \tau_{ij} + \rho * \sum_{k=1}^{m} \Delta\tau_{ij}^{k} \qquad \forall\ (i, j) \in L \tag{2.14}$$

Zudem erfolgt eine Anpassung der neu abgelegten Pheromonmenge $\Delta\tau_{ij}^{k}$ entsprechend Formel 2.15.

$$\Delta\tau_{ij}^{k} = \begin{cases} \frac{\frac{1}{f(s^k)}}{\sum_{h=1}^{m} \frac{1}{f(s^h)}} & \text{falls } (i, j) \text{ Element der Lösung } s^k \text{ ist} \\ 0 & \text{sonst} \end{cases} \tag{2.15}$$

Durch das so veränderte Pheromonupdate werden automatisch die gewünschten Grenzen eingehalten. Die Menge des neu abgelegten Pheromons ist für jede Kante in jeder Iteration höchstens 1 (Formel 2.15). Da nach Voraussetzung $\tau_{ij} \in [0, 1]$ gilt, werden die Grenzen nach Anwendung von Formel 2.14 immer noch eingehalten, denn die Faktoren ρ und $(1-\rho)$ ergeben in Summe den Wert 1. Folglich werden sich die Pheromonspuren nach dem Update wieder im Intervall $[0, 1]$ befinden.

MAX-MIN Ant System mit Hypercube Framework (HC-MMAS)

Ebenso wie für das HC-AS wird auch beim HC-MMAS ausschließlich die Formel zum globalen Pheromonupdate angepasst.

$$\tau_{ij} \longleftarrow (1-\rho) * \tau_{ij} + \rho * \Delta\tau_{ij}^{best} \qquad \forall\,(i,j) \in L \tag{2.16}$$

mit

$$\Delta\tau_{ij}^{best} = \begin{cases} 1 & \text{falls } (i,j) \text{ Element der Lösung } s^{best} \text{ ist} \\ 0 & \text{sonst} \end{cases} \tag{2.17}$$

Die Ameise mit der besten Lösung legt nun immer eine feste Pheromonmenge mit dem Wert 1 ab, die Lösungsqualität findet keine Berücksichtigung mehr.

Ein sich im HC-MMAS einstellender Nebeneffekt ist die Tatsache, dass die Pheromongrenzen τ_{min} und τ_{max} nicht mehr zur Laufzeit ermittelt werden müssen. Bereits zu Beginn des Algorithmus liegen alle notwendigen Informationen vor: $\tau_{max} = 1$ und τ_{min} kann im Hinblick einer Konvergenzdetektierung anhand bekannter Formel 2.9 statisch ermittelt werden, da sich der Wert von τ_{max} nicht mehr verändert.

Im MMAS ist die untere und obere Pheromongrenze von dem zu einem Zeitpunkt maximal möglichen Pheromonwert abhängig. Dieser wird zur Laufzeit dynamisch approximiert. Eine solche Anpassung führt zu einem wiederholten Angleich der Pheromongrenzen, was lt. Blum Roli und Dorigo [BRD01] die Dynamik des Systems stören könnte und durch das Hypercube Framework verhindert wird.

3 ACO Algorithmen für das BCNP

In diesem Kapitel wird zunächst eine allgemeine Vorgehensweise zur Optimierung kombinatorischer Probleme anhand von Ameisenkolonien aufgezeigt. Der Aufbau nachfolgender Unterkapitel orientiert sich an diesem Schema und folgt den dargestellten Entwicklungsschritten, so dass insgesamt alle erforderlichen Punkte zur Anwendung der ACO auf das k-seitige BCNP umgesetzt werden.

3.1 Vorgehensweise

Eine allgemeine Vorgehensweise zur Optimierung von kombinatorischen Problemen mit Ameisenkolonien wird von Cordon, Herrera und Stützle [CHS02] angegeben. Abbildung 3.1 stellt die umzusetzenden Schritte dar.

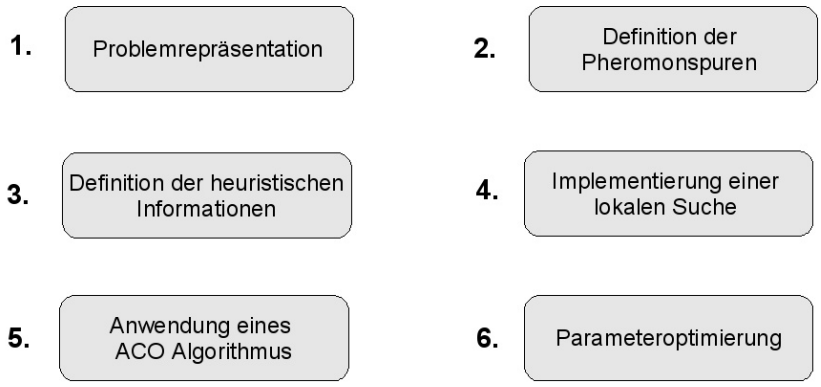


Abbildung 3.1: Vorgehensweise zur Optimierung kombinatorischer Probleme mit Ameisenkolonien

Die Abarbeitungsreihenfolge ist in praktischer Umsetzung nicht strikt sequentiell. Möglich ist, dass bei der Umsetzung gewonnene Erkenntnisse Veränderungen in vorigen Schritten erfordern.

Ziel der Problemrepräsentation ist das Aufstellen eines Konstruktionsgraphen auf Grundlage von Komponenten und Transitionen, den die Ameisen zur Generierung einer Lösung durchwandern. Die Definition der Pheromonspuren ist ein entscheidender Faktor für den Erfolg eines ACO Algorithmus und erfordert typischerweise tieferes Problemverständnis (mehr dazu in Kapitel 3.3).

Nach Cordon, Herrera und Stützle [CHS02] ist die Definition der heuristischen Informationen ebenso entscheidend für die Leistung des Algorithmus, wenn keine lokale Suche eingesetzt wird. Bei Verwendung einer lokalen Suche sind heuristische Informationen hingegen nicht zwingend erforderlich. Die lokale Suche sollte effektiv und effizient arbeiten. Ihr Einsatz ist

nicht vorgeschrieben und wird generell als optional angesehen.
Zwar ist die Verwendung einer lokalen Suche in Kombination mit ACO Algorithmen mittlerweile äußerst populär, dennoch werden Lösungen für das BCNP in der Literatur[1] vielfach anhand von Konstruktionsheuristiken generiert, deren Erkenntnisse bereits nutzbar sind. In dieser Arbeit wird auf den Einsatz einer lokalen Suche verzichtet. Aufgrund der hybriden Optimierungsansätze (siehe Kapitel 3.1.2) liegen bereits Informationen über Heuristiken vor, aus denen Ansätze zur Generierung heuristischer Informationen abgeleitet werden können. Für die Entwicklung einer effizienten lokalen Suche bezüglich des k-seitigen BCNP existieren bislang kaum Anhaltspunkte[2]. Das gilt insbesondere, wenn die lokale Suche nicht nur die Permutation, sondern auch die Seitenzuordnung verändern soll.

Eine gute Ausgangslage für die Parametereinstellungen besteht darin, sich an bereits bestehenden Implementierungen von ACO Algorithmen zu orientieren und bewährte Parameterempfehlungen zu nutzen [vgl. CHS02].
Allerdings sei angemerkt, dass gute Parametersätze generell in hohem Maße problemabhängig sind. Es ist auch möglich, die Parameter mit Hilfe automatisierter Ansätze optimieren zu lassen, wie beispielsweise der *sequentiellen Parameteroptimierung (SPO)*[3] oder eines *genetischen Algorithmus (GA)*, der bereits zur Parameteroptimierung für einen ACO Algorithmus von Botee und Bonabeau [BB98] eingesetzt wurde.

Cordon, Herrera und Stützle [CHS02] weisen auf die besondere Relevanz der ersten vier Schritte hin. Schlechte Entscheidungen auf diesen Ebenen können für gewöhnlich nicht mehr nur durch reine Parameteroptimierung ausgeglichen werden. Im Folgenden wird hierauf besonderes Augenmerk gelegt.

3.1.1 Vorüberlegungen

Eine Probleminstanz des k-seitigen BCNP spezifiziert einen Graphen $\mathcal{G}_{BCNP} = (V, E)$ und die Seitenanzahl k. Das Ziel besteht in der Generierung einer k-seitigen Buchzeichnung für $\mathcal{G}_{BCNP}$ mit minimaler Anzahl von Kantenüberschneidungen.

Zwei Informationen sind zur Aufstellung einer konkreten Buchzeichnung erforderlich:

(1) Die Permutation π der Knoten $v \in V$. Sie gibt ihre Anordnung entlang des Buchrückens an.
Eine Permutation kann als *bijektive Abbildung* $\pi : V \mapsto \{1..|V|\}$ aufgefasst werden, die jedem Knoten genau einen Index in der Permutation zuordnet. Jeder Index ist genau einmal vergeben.

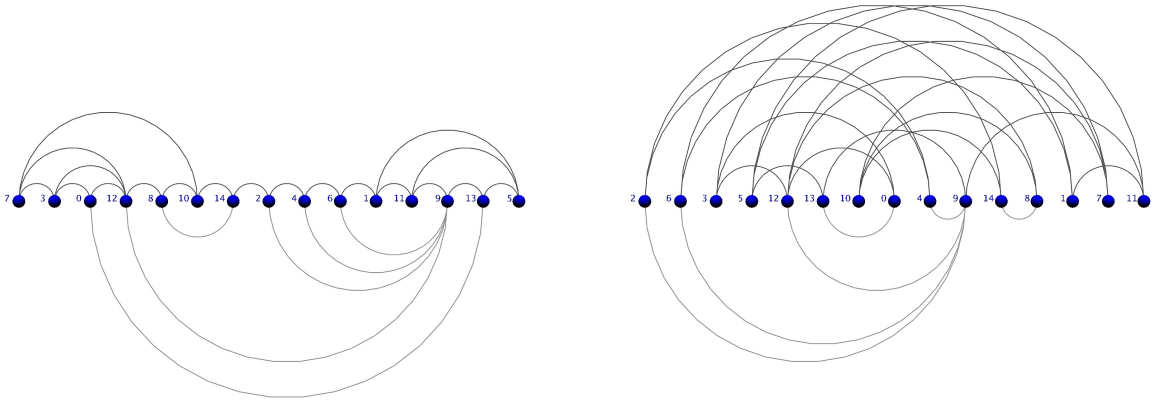
(2) Die Seitenzuordnung σ. Sie ordnet jeder Kante $e \in E$ genau eine der k möglichen Seiten zu. σ ist demnach eine *Abbildung* $\sigma : E \mapsto \{1..k\}$.

Die Menge der Lösungskandidaten einer Probleminstanz bestimmt sich zu allen Tupeln aus möglichen Permutationen und Seitenzuordnungen für diese Instanz: $\mathcal{S} = \{(\pi, \sigma)\}$. Es existieren keine weiteren Nebenbedingungen, so dass gilt: $\Omega = \emptyset \wedge \tilde{\mathcal{S}} = \mathcal{S}$. Die Kostenfunktion

[1] für eine Übersicht ist die Dissertation von He [He06, S.70ff] hilfreich.
[2] bekannt ist ein Tabu-Search-Algorithmus von Winterbach [Win05, S.143ff] für das 1-seitige BCNP.
[3] Sequential Parameter Optimization [BB06]

$f : s \mapsto \mathbb{N}_0$ ordnet jeder Lösung $s \in \mathcal{S}$ die Buchkreuzungszahl der von s repräsentierten k-seitigen Buchzeichnung zu. Die Berechnung von f wird in Kapitel 3.1.3 vorgestellt.

Jede Lösung beinhaltet zwei Teil-Lösungen, die nur **abhängig** voneinander optimiert werden können. Die Zielfunktion ist auf einer Kombination aus beiden Teil-Lösungen definiert. Es gibt keine Möglichkeit, die Qualität einer Permutation losgelöst von einer Seitenzuordnung zu bewerten und umgekehrt.
Hier besteht folgende Schwierigkeit: Gute Lösungen können nur dann entstehen, wenn beide Teil-Lösungen gut aufeinander abgestimmt sind. Eine Seitenzuordnung, die in Kombination mit einer spezifischen Permutation eine geringe Buchkreuzungszahl hervorruft, kann bei einer veränderten Permutation unvorteilhaft sein (siehe Abb. 3.2).
Das Ziel besteht in der Bestimmung einer guten **Kombination** aus Permutation und Seitenzuordnung für die gegebene Probleminstanz.



(a) Eine (optimale) 2-seitige Buchzeichnung mit Buchkreuzungszahl 0

(b) Bei einer Permutationsveränderung wird unter der gleichen Seitenzuordnung die Buchkreuzungszahl 68 erreicht. Die Seitenzuordnung ist nun unvorteilhaft, beispielsweise ließe sich die Kante (9,11) geeigneter auf der unteren Seite einzeichnen.

Abbildung 3.2: Auswirkungen für Buchzeichnungen bei Veränderung der Permutation

Problematische Codierung innerhalb eines Konstruktionsgraphen

Die Optimierung mit Ameisenkolonien ist eine Metaheuristik auf Basis einer *Konstruktionsheuristik* [siehe DS04, S.29ff; CHS02]. Lösungen werden sequentiell von Grund auf neu gebildet. Gespeichert wird die Information, wie diese vielversprechend generiert werden können. Diese Informationen werden während des Suchprozesses dynamisch angepasst und verändert.

Greift man das Problem „als Ganzes“ an und codiert beide Teil-Probleme innerhalb **eines** Graphen, so stellt der Weg einer Ameise eine Lösung $s = (\pi, \sigma)$ dar.
Nachdem die Ameise ihre Wanderung durch den Konstruktionsgraphen beendet hat, legt sie eine Pheromonspur auf dem von ihr benutzten Weg ab. Die Stärke der Pheromomspur repräsentiert die Qualität der generierten Lösung. Von dieser Qualität können wie beschrieben keine Rückschlüsse auf die Qualität der Teil-Lösungen gezogen werden, da diese nur für eine Kombination betrachtet werden kann. Genau diese Rückschlüsse könnten nun von nachfolgenden

Ameisen gezogen werden. Die Intention, Ameisen zur Bildung von „guten“ Wegen zu leiten, würde verfälscht werden: (Teil-)Wege mit hohen Pheromonspuren müssen nicht zwangsweise zu einer guten Lösung führen.

Abbildung 3.3 verdeutlicht diesen Umstand. Es wird angenommen, dass der Weg einer Ameise zunächst eine Permutation und erst danach eine Seitenzuordnung codiert.

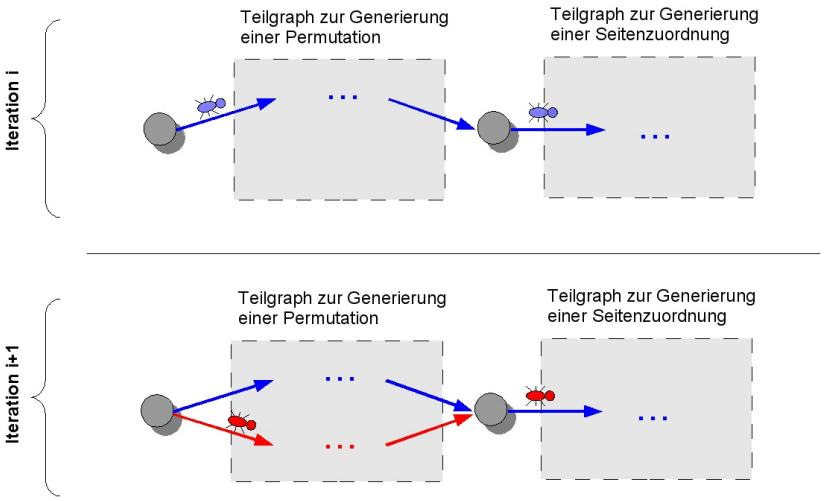


Abbildung 3.3: Potentiell problematische Beeinflussung durch Pheromonspuren

Als zugrundeliegender ACO Algorithmus sei das MMAS verwendet, die Problematik stellt sich jedoch auch bei den anderen Algorithmen. Hier darf nur die Ameise mit der besten Lösung eine Pheromonspur ablegen. In der nächsten Iteration haben die zugehörigen Kanten eine höhere Wahrscheinlichkeit, gewählt zu werden.
Wenn eine Ameise in der nächsten Iteration (rot dargestellt) beispielsweise bereits einen veränderten ersten Teil-Weg durchlaufen hat, so würde eine Beeinflussung der Suche in Richtung der letzten Seitenzuordnung nicht notwendigerweise nutzbringend sein; es würde von der Qualität der letzten Lösung auf die Qualität der Seitenzuordnung geschlossen, so dass diese isoliert bewertet wird. Die Pheromonspuren könnten in solch einem Fall ihre Bestimmung verfehlen und Ameisen eventuell - salopp gesprochen - „auf die falsche Fährte“ führen.

Das Verhalten von ACO Algorithmen ist generell nur begrenzt erforscht und insbesondere für eine solche Strategie unbekannt, so dass mögliche Folgen nur spekulativ angenommen werden können. Die vorigen Ausführungen sind ebendaher bewusst im Konjunktiv gehalten. In der Literatur[4] existiert keine Implementierung, die zwei Zuordnungen innerhalb eines Graphen codiert.

3.1.2 Optimierungsansätze

Die Bestimmung einer k-seitigen Buchzeichnung kann wie beschrieben über die Generierung von zwei Zuordnungen (Abbildungen) realisiert werden.
Das *University Course Timetabling Problem* [SKS02] besteht ebenfalls in der Optimierung

[4] bezogen auf die Übersicht aus [DS04, S.39f].

zweier voneinander abhängiger Zuordnungen. Es existiert eine Umsetzung mit ACO Algorithmen von Socha, Knowles und Sampels [SKS02]. Das Problem besteht aus der Zuweisung vorliegender Ereignisse zu bestimmten Zeitintervallen. Jedes der so ermittelten Ereignis/Zeitintervall-Tupel findet in genau einem Raum statt und erfordert die Generierung einer zweiten Zuordnung, die diese Tupel mit den zur Verfügung stehenden Räumen assoziiert. Dabei muss eine Kostenfunktion minimiert und bestimmte Nebenbedingungen eingehalten werden.
Die Implementierung codiert jedoch nicht beide Teilprobleme innerhalb eines Konstruktionsgraphen. Es wird eine hybride Strategie verfolgt: Der ACO Algorithmus optimiert nur die Assoziation von Ereignissen zu Zeitintervallen, wohingegen die (noch fehlende) Raum-Zuordnung separat und deterministisch anhand der generierten Zeitintervall-Zuordnung bestimmt wird. Diese steht nicht mehr unter dem Einfluss des ACO Algorithmus.

Es besteht generell auch die Möglichkeit, nur eine Teil-Lösung für das k-seitige BCNP von einem ACO Algorithmus zu bestimmen und zu optimieren. Die noch fehlende Zuordnung kann auf andere Art und Weise generiert werden. Sie muss möglichst gut zu der ersten Teil-Lösung passen. Die Qualität der (Gesamt-)Lösung repräsentiert die Qualität der ersten Teil-Lösung und somit auch die Stärke der abgelegten Pheromonspuren an. Umso wichtiger wird es nun, eine möglichst optimal auf die erste abgestimmte zweite Teil-Lösung zu finden.

Auch He implementierte einen hybriden Ansatz für das 2-seitige BCNP bestehend aus einem GA und einer *SLOPE*-Heuristik [siehe HSM07]. Die *SLOPE*-Heuristik ist nicht ohne Anpassungen auf das k-seitige BCNP anwendbar. In Kapitel 3.4 wird sich der Adaption zweier Heuristiken zur Generierung von Seitenzuordnungen mit k Seiten ausführlich gewidmet.

Einsatz von ACO Algorithmen

Allgemein ist die Generierung einer guten Permutation auf Grundlage einer gegebenen Seitenzuordnung beschwerlich zu realisieren. Eine Veränderung der Permutation hat zudem einen größeren Einfluss auf die Buchkreuzungszahl als eine Veränderung der Seitenzuordnung [vgl. He06, S.127]. Zudem sind bereits heuristische Ansätze zur Bestimmung der Seitenzuordnung auf Grundlage einer Permutation bekannt[5]. Aufgrunddessen wird bei einer hybriden Optimierung im Folgenden stets die Permutation von einem ACO Algorithmus optimiert und die Seitenzuordnung von anderen Verfahren generiert.

Es existieren allerdings zwei Sonderfälle, bei denen sich diese Problematik nicht stellt, da dann eines der beiden Teilprobleme bereits gegeben ist, und zwar

1. **$\mathbf{k = 1}$**
   Hierbei existiert nur eine Seite und die Seitenzuordnung ist trivialerweise bereits determiniert. Alle Kanten müssen auf Seite 1 gezeichnet werden. Die Bestimmung einer Seitenzuordnung erübrigt sich und es ist ausschließlich notwendig, die Permutation zu optimieren.

2. **Der Graph ist vollständig**
   Die Permutation π hat in diesem Fall keinen Einfluss auf die Lösungsqualität, da sich die 1-seitige Buchzeichnung bei beliebiger Wahl von π nicht verändert. π kann zu Anfang der Optimierung beliebig gesetzt werden. Das Ziel besteht sodann einzig in der Generierung

[5] siehe dazu [Cim02; HSV05]

einer optimalen Seitenzuordnung für diese festgelegte Permutation.
In dieser Arbeit werden vollständige Graphen nicht betrachtet und von daher ist dieser Fall nicht zu berücksichtigen.

Weiterhin wäre es zusätzlich möglich, die Permutation zunächst für das 1-seitige BCNP zu optimieren. Hierfür kann die Lösungsqualität direkt bestimmt werden. Auf Basis der so optimierten Permutation ist ein weiteres Verfahren zur Optimierung der k-seitigen Seitenzuordnung anzuwenden. Voraussetzung dafür ist, dass eine „gute“ Permutation für das 1-seitige BCNP auch für eine Zeichnung mit mehr zur Verfügung stehenden Seiten vorteilhaft ist. In Vorab-Tests hat sich jedoch gezeigt, dass die anderen Ansätze vielversprechender sind und insofern werden folgend nur diese betrachtet.

Damit bestehen folgende Möglichkeiten zur Optimierung des k-seitigen BCNP mit ACO Algorithmen:

1. **Codierung innerhalb eines Konstruktionsgraphen**
   Ein Weg durch den Konstruktionsgraphen codiert eine Permutation sowie eine Seitenzuordnung für eine Probleminstanz.

2. **Hybride Optimierungsansätze**
   Hierbei wird die Permutation π mit einem ACO Algorithmus optimiert und deren Qualität indirekt bewertet. Dazu wird nach der Lösungskonstruktion eine passende Seitenzuordnung generiert (siehe Algorithmus 3.1.1). Die Qualität der Gesamt-Lösung dient als Maßstab für die Qualität der Permutation.

   a) Die Seitenzuordnung σ wird mit einer Heuristik auf Basis von π generiert. Die Qualität der Permutation bestimmt sich zu der Qualität der Lösung (π, σ), so dass sie indirekt bestimmt wird.

   b) Analog zu 2a, nur dass die Seitenzuordnung σ von einer Metaheuristik separat für die Permutation π optimiert wird, z.B. von einem weiteren ACO Algorithmus. Die Ausführungen dieser Arbeit geben Details darüber preis, wie eine Seitenzuordnung von einem ACO Algorithmus generiert werden kann, auch wenn in den Tests in Kapitel 4 diese Möglichkeit nicht betrachtet wird.

Algorithmus 3.1.1 : Pseudocode von ACO Algorithmen für hybride Optimierungsansätze

```
   /* Initialisierung ---------------------                              */
1  ...                                                 /* siehe Kapitel 2.3 */
   /* Hauptschleife -----------------------                              */
2  solange (Abbruchbedingung nicht erfüllt) tue
3  |  für Ameise k = 1 bis m tue
4  |  |  Lösungskonstruktion                  /* Lösung ≙ Knotenpermutation π */
5  |  |  Generierung der Seitenzuordnung σ auf Basis von π
6  |  |  Bewertung der Lösung unter Berücksichtigung von σ
7  |  ende
8  |  ...                                                /* siehe Kapitel 2.3 */
9  ende
```

3.1.3 Kostenfunktion

Die Kostenfunktion f ordnet jeder Lösung $s \in \mathcal{S}$ die Buchkreuzungszahl der repräsentierten k-seitigen Buchzeichnung zu. Es wird $g(s) = f(s)$ gesetzt.

Zwei Kanten $(t, u) \in E$ und $(v, w) \in E$ kreuzen sich in einer Buchzeichnung genau dann, wenn sie auf derselben Seite gezeichnet sind und $t <_\pi v <_\pi u <_\pi w$ gilt. Die Ordnung $<_\pi$ bezieht sich auf die Positionen in der Permutation π (siehe Abb. 3.4) [vgl. Woo02; HSV05].

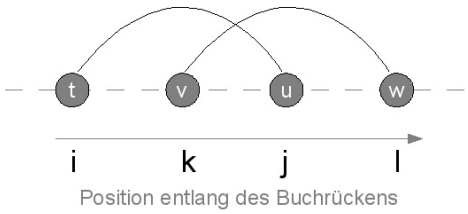


Abbildung 3.4: Situation, in der sich zwei auf der gleichen Seite gezeichnete Kanten kreuzen

Mathematische Formeln zur Berechnung der k-seitigen Buchkreuzungszahl finden sich in der Literatur [HSV05; Cim02]. Dabei werden alle *Positionen* i, j, k und l durchlaufen, für die sich zwei Kanten $(\pi^{-1}(i), \pi^{-1}(j)), (\pi^{-1}(k), \pi^{-1}(l)) \in E$ schneiden, wenn sie auf der gleichen Seite eingezeichnet sind. Zur Erinnerung: $\pi : V \mapsto \{1..|V|\}$ bildet jeden Knoten auf eine Position entlang des Buchrückens (Index in der Permutation) ab. π^{-1} ist die Inverse zu π und ermittelt zu einer Position den entsprechenden Knoten. Ein Beispiel: In Abbildung 3.4 ist $\pi(t) = i$ und $\pi^{-1}(i) = t$.
In ersten Implementierungen der ACO Algorithmen wurde ein Verfahren eingesetzt, das sich an diesen Formeln orientiert und die Buchkreuzungszahl auf diese Weise berechnet. Dafür ist die Implementierung der Instanzgraphen als Adjazenzmatrix[6] sinnvoll. So kann effizient geprüft werden, ob eine Kante zwischen zwei bestimmten Knoten verläuft. Die Laufzeit einer solchen Berechnung beträgt $O\left(|V|^4\right)$ [vgl. Cim02]. Sie ist damit ausschließlich von der Knotenanzahl abhängig. Auf Instanzgraphen mit hohem Durchschnittsgrad mag dieses ein adäquates Mittel zur Berechnung der Buchkreuzungszahl sein. Bei Graphen mit geringem Durchschnittsgrad hingegen lässt sich die Laufzeit verbessern, wenn nicht etwa über die Positionen, sondern die vorhandenen Kanten iteriert wird. Da diese Berechnung innerhalb eines ACO Algorithmus zahlreich aufgerufen wird, kann die Rechenzeit auf Graphen mit geringem Durchschnittsgrad insgesamt deutlich verringert werden.

Alle Kanten des Graphen müssen dazu paarweise miteinander verglichen werden. Hiermit wird nun die Implementierung der Instanzgraphen als Kantenliste erforderlich (Anhang C.1), um die Kanten effizient zu durchlaufen. Für jedes Kantenpaar werden die verbundenen Knoten und deren Position in der Permutation ermittelt. Jedes Kantenpaar wird genau einmal betrachtet. Erfüllen die Positionen die Bedingung einer Kreuzung, so wird die Buchkreuzungszahl um den Wert 1 inkrementiert. Insgesamt müssen $\frac{|E|*(|E|-1)}{2}$ Kantenpaare miteinander verglichen werden. Algorithmus 3.1.2 realisiert diese Berechnung. Die Laufzeit beträgt also $O\left(\frac{|E|*(|E|-1)}{2}\right) = O(|E|^2)$.
Der Algorithmus arbeitet nicht optimal. Mögliches Verbesserungspotential bestünde in der Einsparung von Vergleichen derjenigen Kantenpaare, die auf unterschiedlichen Seiten eingezeichnet sind. Auf diese Weise ließe sich der Rechenaufwand weiter verringern.

[6] [Tur04, S.25f]

Algorithmus 3.1.2 : Berechnung der Buchkreuzungszahl

Eingabe : Ein Graph G_{BCNP}, die Knotenpermutation π und die Seitenzuordnung σ
Ausgabe : Buchkreuzungszahl der repräsentierten Buchzeichnung

```
1  int kreuzungen = 0;
2  für alle u = 1 bis |E| -1 tue
3  |  e1 = u-te Kante in Kantenliste;
4  |  i = min(π(e1.knoten1), π(e1.knoten2));
5  |  j = max(π(e1.knoten1), π(e1.knoten2));
6  |  für alle v = u+1 bis |E| tue
7  |  |  e2 = v-te Kante in Kantenliste;
8  |  |  wenn σ(e1) == σ(e2) dann
9  |  |  |  k = min(π(e2.knoten1), π(e2.knoten2));
10 |  |  |  l = max(π(e2.knoten1), π(e2.knoten2));
11 |  |  |  wenn (i < k < j < l) oder (k < i < l < j) dann
12 |  |  |  |  kreuzungen++;
13 |  |  |  ende
14 |  |  ende
15 |  ende
16 ende
17 return kreuzungen;
```

3.2 Problemrepräsentation

Kapitel 3.2.1 befasst sich zunächst mit der Codierung von Informationen innerhalb des Konstruktionsgraphen. Im weiteren Verlauf werden die Konstruktionsgraphen für die beiden Teil-Probleme aufgestellt, so dass ein ACO Algorithmus zur Generierung einer Permutation oder aber einer Seitenzuordnung angewendet werden kann. Dies ist eine Voraussetzung für die hybride Optimierung des Problems.
Die Bestimmung eines Gesamtgraphen zur Codierung einer *Lösung* für das k-seitige BCNP erfolgt danach aufgrund der Erkenntnisse, die bei der Codierung der Teil-Probleme gewonnen wurden.
Im Sinne einer effizienten Lösungskonstruktion bietet es sich an, die Konstruktionsgraphen als Adjazenzliste zu verwalten (siehe auch Anhang C).

3.2.1 Kategorisierung der Komponenten

Die codierten Informationen müssen aus dem Weg einer Ameise ablesbar sein, so dass sie entweder aus den verwendeten Komponenten (Knoten des Konstruktionsgraphen), Transitionen (Kanten des Konstruktionsgraphen), oder beidem hervorgehen.
An dieser Stelle wird für weitere Ausführungen festgelegt, die Informationen in den Komponenten zu codieren. Es ist nur entscheidend, welche Komponenten (und ggf. in welcher Reihenfolge) die Ameise in ihrer Lösungskonstruktion besucht, was überdies ein Standard-Vorgehen bei der Anwendung von Ameisenkolonien auf neue Probleme darstellt.

Komponenten werden nachfolgend in zwei Kategorien eingeteilt:

1. **Komponenten mit Informationen**
   Diese Komponenten enthalten Informationen über die Buchzeichnung von G_{BCNP} und sind bei der Auswertung des Weges zu berücksichtigen.

2. **Komponenten ohne Informationen**
   In einem Graphen, der nur aus Informationskomponenten besteht, kann die Vernetzungsstruktur nicht sonderlich flexibel angepasst werden.
   Mehr Flexibilität in der Strukturgestaltung wird durch zusätzliche Komponenten ohne enthaltene Informationen erreicht. Sie sind bei der Interpretation des Weges zu übergehen.

3.2.2 Repräsentation der Knotenpermutation

Orientierung für eine Permutationsbestimmung bietet bereits die Anwendung der ACO auf das TSP[7]. Das Problem besteht darin, eine Rundreise von Städten mit minimaler Gesamtlänge zu finden, wobei jede Stadt genau einmal besucht werden muss. Für jede Stadt wird eine Komponente in einem vollvernetzten Graphen angelegt. Die Ameisen beginnen ihre Suche wahlweise an dem Knoten, der die Heimatstadt des Handlungsreisendendem repräsentiert, oder aber bei einer zufällig gewählten Stadt. Dabei dürfen sie keine Komponente mehr als einmal anlaufen.
Die von einer Ameise generierte Rundreise ergibt sich aus der (relativen) Reihenfolge der besuchten Städtekomponenten in dem Weg durch den Konstruktionsgraphen. Eine solche Reihenfolge kann als Permutation über der Komponentenmenge interpretiert werden. Das Prinzip ist folglich für das BCNP nutzbar.

Codierung des Konstruktionsgraphen

Für jeden Knoten $v \in V$ wird eine Informationskomponente - nachfolgend als Knotenkomponente bezeichnet - angelegt. Sie repräsentiert diesen Knoten. Alle diese Knotenkomponenten müssen untereinander erreichbar sein. Sie werden dazu vollständig miteinander vernetzt.
Alle Ameisen beginnen beginnen ihre Lösungskonstruktion an einer separaten Startkomponente, die keine Informationen codiert. Diese ist mit allen Knotenkomponenten durch eine Kante verbunden.
Dadurch wird die erste Komponente in der Permutation von einer Ameise selbst bestimmt und muss nicht extra noch separat „von außen“ ausgewählt werden. Zudem würde sich hiermit die Möglichkeit ergeben, mit Pheromonspuren sinnvolle zuerst aufzunehmende Komponenten zu lernen. Dieses ist jedoch nicht notwendig, wie hieran in Kapitel 3.3.3 erörtert wird.

Lösungskonstruktion einer Ameise

Die Ameise muss sich bei der Lösungskonstruktion so durch den Graphen bewegen, dass sie keine Knotenkomponente zweimal besucht und auch die Startkomponente kein weiteres Mal anläuft. In einer Permutation darf jedes Element nur genau einmal vertreten sein. Ein Umweg

[7] Traveling Salesman Problem [DS04, S.65 ff; SD99]

über die Startkomponente würde die Struktur einer Lösung verändern und Pheromonspuren zwischen Knotenkomponenten umgehen. Eine Vermeidung dessen ist allerdings möglich.
Die Bedingung kann entweder zusätzlich im problemabhängigen Test berücksichtigt oder aber durch die Vernetzungsstruktur eingebracht werden. Indem die Startkomponente nur durch *gerichtete* Kanten mit den Knotenkomponenten verbunden wird, ist diese im weiteren Weg der Ameise nie wieder Teil der Nachbarschaft einer Knotenkomponente. Der Weg dorthin braucht somit nicht extra evaluiert (und dann verworfen) zu werden. Diese Möglichkeit findet im Folgenden Verwendung.

Der problemabhängige Test definiert einen Zustand x genau dann als zulässig, wenn keine Knotenkomponente in x mehrfach vorkommt. Somit wird die zulässige Nachbarschaft eines Zustands stets alle noch nicht besuchten Knotenkomponenten enthalten und es ist durch die Vollvernetzung immer möglich, eine zulässige Lösung zu erschaffen.
Wenn alle Knotenkomponenten von der Ameise besucht wurden, ist eine Permutation aus dem Weg einer Ameise auslesbar und die Lösungskonstruktion soll enden. Die Terminierungsbedingung e^k kann anhand der zulässigen Nachbarschaft definiert werden: Sie enthält dann, und nur dann, keine Elemente mehr. Demnach bestimmt sich e^k zu $\mathcal{N}_i^k = \emptyset$.

Ein auf diese Weise generierter Konstruktionsgraph wird im Folgenden mit G_π bezeichnet.

Beispiel

Die Struktur des Konstruktionsgraphen $\mathcal{G}_\pi$ ist in Abb. 3.5 an einem Beispiel mit $|V| = 5$ visualisiert: An oberster Position befindet sich die Startkomponente (gelb) und darunter sind die fünf Knotenkomponenten (rot) kreisförmig angeordnet, die jeweils einen bestimmten Knoten (bezeichnet mit 0..4) des Instanzgraphen $\mathcal{G}_{BCNP}$ repräsentieren.

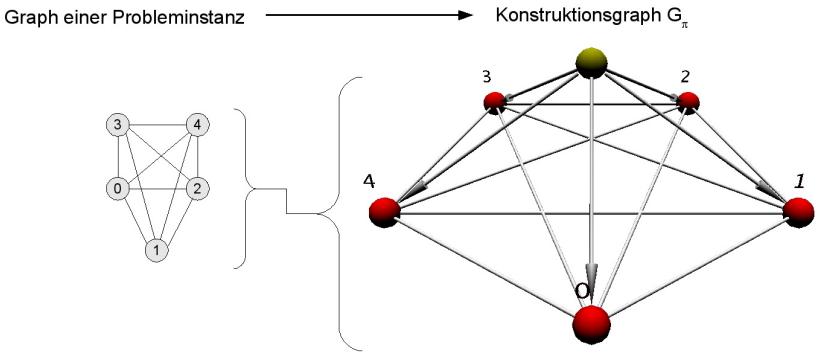


Abbildung 3.5: Konstruktionsgraph G_π mit fünf Knotenkomponenten

Abbildung 3.6 zeigt einen beispielhaften Weg einer Ameise auf. Dieser Weg repräsentiert die Permutation (1,2,0,3,4) der Knotenmenge $\{0, 1, 2, 3, 4\}$, da die Informationskomponenten der Knoten in dieser Reihenfolge besucht wurden.

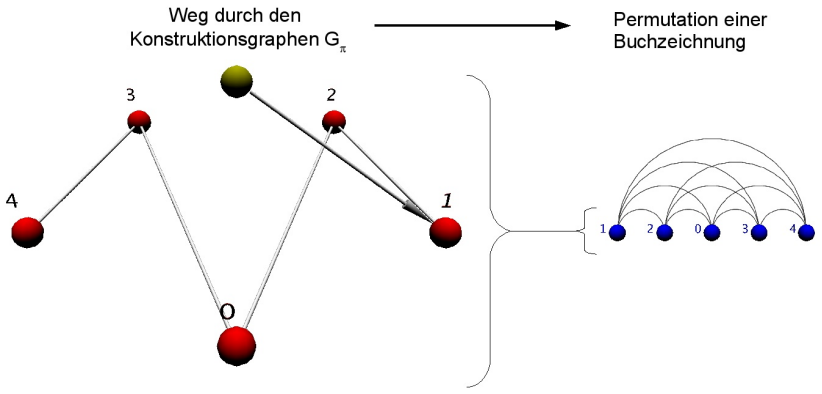


Abbildung 3.6: Ein Weg einer Ameise in einem Konstruktionsgraphen G_π

Komplexität

Für den auf Basis eines Instanzgraphen $\mathcal{G}_{BCNP} = (V, E)$ erstellten Konstruktionsgraphen $G_\pi = (\mathcal{C}, L)$ gelten folgende Komplexitätsgrößen:

$$\begin{aligned} |\mathcal{C}| &= 1 + |V| \\ |L| &= \frac{(|V|+1) * (|V|)}{2} \end{aligned}$$

3.2.3 Repräsentation der Seitenzuordnung

Eine Seitenzuordnung kann wie beschrieben als Abbildung formalisiert werden. Es geht nunmehr darum, jeder Kante $e \in E$ genau eine Seite zuzuordnen. Dieses ist ein *Zuordnungsproblem*[8].

Codierung von Zuordnungsproblemen in der Literatur

In der Literatur finden sich bereits ACO Implementierungen von Zuordnungsproblemen, beispielsweise für das *University Course Timetabling Problem* [SKS02] und das *Generalized Assignment Problem* [LS98]. Das Buch von Dorigo und Stützle [DS04, S.39,159ff] gibt eine Aufstellung über die bislang implementierten Probleme.
Seien A und B die zuzuordnenden Mengen. Die Elemente der Menge A sollen dabei jeweils einem Element der Menge B zugeordnet werden. Unter Berücksichtigung der Darstellungen aus [DS04, S.43] wird ersichtlich, dass es auf folgende Weisen möglich ist, Zuordnungen zu codieren:

I) Die Informationskomponenten repräsentieren ein Tupel $(a, b) \in A \times B$ [vgl. SKS02].

II) Die Informationskomponenten repräsentieren die Elemente aus $A \cup B$ [vgl. DS04, S.43].

[8] assignment problem [siehe DS04, S.159ff]

Variante I

Die Reihenfolge, in der die Elemente der Menge A den Elementen der Menge B zugeordnet werden, ist von vornherein festzulegen (siehe dazu Abb. 3.7). Unproblematisch ist dieses vor allem dann, wenn die Reihenfolge für die Lösungsqualität irrelevant ist oder aber eine feste Reihenfolge erwünscht ist.
Eine Informationskomponente (a, b) repräsentiert die Zuordnung des Elementes a zu dem Element b. Da jedes Element aus A nur genau einem Element aus B zugeordnet wird, darf die Lösung ausschließlich Komponenten enthalten, für die sich die Elemente aus A unterscheiden. Der Konstruktionsgraph wird dazu sozusagen in Abschnitte eingeteilt, die von einer Ameise nacheinander durchlaufen werden. Jeder Abschnitt beinhaltet $|B|$ Informationskomponenten $(a, b_1), (a, b_2), \ldots, (a, b_{|B|})$ für ein Element $a \in A$. Diese geben jeweils an, welchem Element $b \in B$ das Element a zugeordnet wird. Somit existieren $|A|$ Abschnitte und in jedem Abschnitt muss genau eine Komponente besucht werden. Jede Komponente ist mit jedem Element der direkt nachfolgenden Ebene durch eine gerichtete Kante verbunden.
Eine Ameise bewegt sich bei der Lösungskonstruktion nun immer von einem Abschnitt in den nächsten. Auf diese Weise werden die Elemente der Menge A jeweils einem Element der Menge B zuordnet. Da die Konstruktion des Graphen einen Weg in „Rückrichtung“ aufgrund der Vernetzung mit gerichteten Kanten nicht zulässt, erübrigt sich der problemabhängige Test und es sind stets alle Zustände zulässig.
Die Lösungskonstruktion endet, wenn eine bestimmte Komponente - der Zielknoten - erreicht wurde.

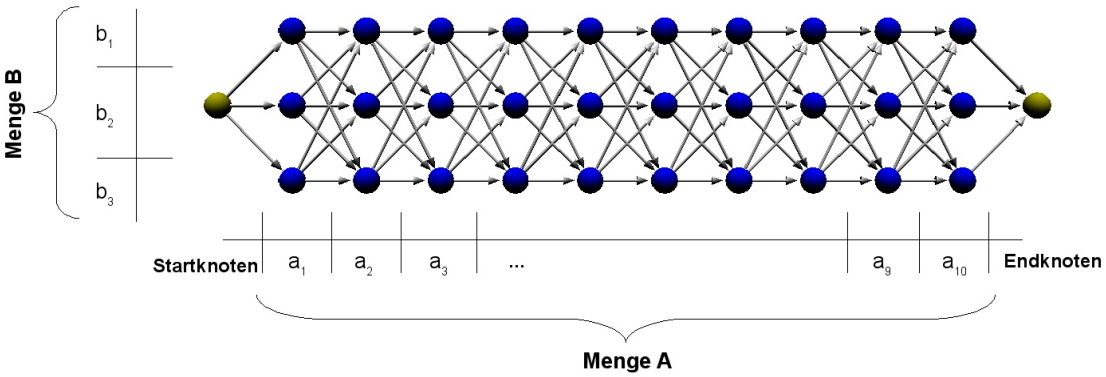


Abbildung 3.7: Beispiel eines Konstruktionsgraphen für Variante I mit $|A| = 10$ und $|B| = 3$

Variante II

Für jedes Element aus A und für jedes aus B existiert eine Informationskomponente, die dieses Element repräsentiert. Sie sind allesamt vollständig miteinander vernetzt (siehe Abb. 3.8).
In der Lösungskonstruktion bewegt sich eine Ameise *abwechselnd* zwischen den Komponenten beider Mengen hin und her. Zuallererst muss immer eine Komponente der Menge A angesteuert werden. Interpretiert wird der (Teil-)Weg von einer A-Komponente zu einer B-Komponente als Zuordnung des repräsentierten Elementes aus A mit dem repräsentierten Element aus B. Wenn jedes Element aus A nur genau einem Element aus B zugeordnet werden soll, darf die

Ameise die bereits besuchten Informationskomponenten der Elemente aus A in der Lösungskonstruktion nicht mehr anlaufen.
Der problemabhängige Test definiert hierbei einen Zustand als zulässig, wenn er keine Komponente der Menge A mehrfach enthält und die Komponenten der Menge A und B stets im Wechsel auftreten.
Eine solche Codierung erfordert keine zuvor definierte Ordnung der Elemente. Diese kann gleichwohl, falls gewünscht, anhand von Pheromonspuren erlernt werden [vgl. DS04, S.43].

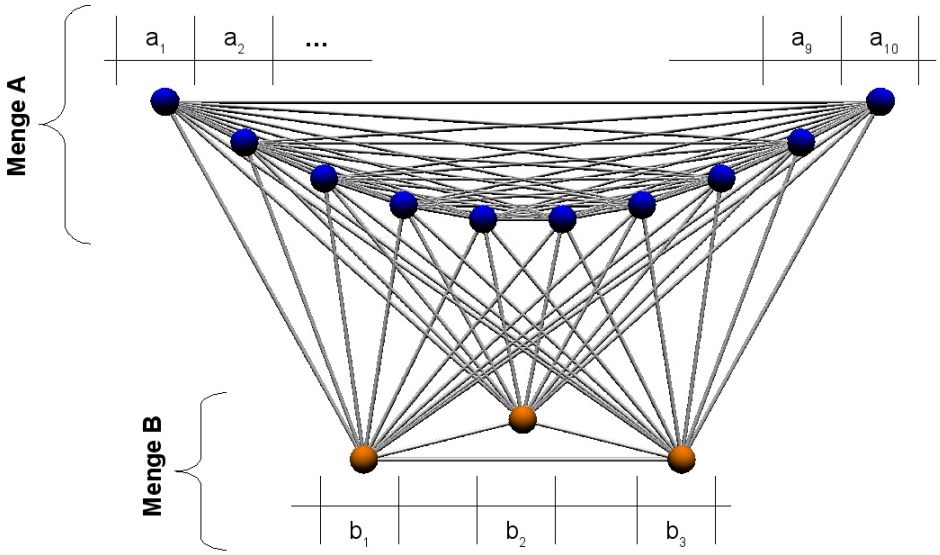


Abbildung 3.8: Beispiel eines Konstruktionsgraphen für Variante II mit $|A| = 10$ und $|B| = 3$

Auswahl und Adaption einer Codierung

Die Zuordnungsreihenfolge der Kanten ist für die Zielfunktion nicht entscheidend. Prinzipiell ist demnach die Verwendung der Variante I möglich, jedoch könnte dieses bei Einbeziehung dynamischer heuristischer Informationen (siehe Kapitel 3.5) von Nachteil sein: Für Greedy-Heuristiken ist die Reihenfolge der Seitenzuordnung von entscheidender Bedeutung. Diese wäre hier für jeden Lauf gleich und ließe sich nicht anpassen.

Die Variante II bietet diesbezüglich einen stärker anpassbaren Ansatz, auch wenn dafür ein längerer Lösungsweg der Ameisen in Kauf genommen werden muss (die nächste Kantenkomponente muss zusätzlich ausgewählt und angelaufen werden).
Im Folgenden wird daher eine Anpassung von Variante II zur Bestimmung einer Seitenzuordnung verwendet.

Adaptiert auf das k-seitige BCNP stellt die Menge A die Kanten und B die Seiten dar. Die Vernetzung kann allerdings einfacher gestaltet werden: Die Konstruktionspolitik einer Ameise lässt es ohnehin nicht zu, dass zwei Seitenkomponenten oder zwei Kantenkomponenten nacheinander angelaufen werden. Um die Nachbarschaft einer Komponente zu verkleinern (im Sinne gesteigerter Effizienz) kann deshalb darauf verzichtet werden, die Informationskomponenten für Kanten und Seiten *untereinander* zu vernetzen. Keine Seitenkomponente ist mit einer anderen Seitenkomponente verbunden. Gleiches gilt für Kantenkomponenten. Es muss

nun zudem nicht mehr gesondert darauf geachtet werden, dass die Ameise die beiden Komponententypen im Wechsel ansteuert; das stellt die Vernetzungsstruktur jetzt bereits sicher. Ein Zustand ist also genau dann zulässig, wenn er keine Kantenkomponente mehrfach enthält. Zusätzlich wird noch eine Startkomponente eingefügt, die ausschließlich mit den Kantenkomponenten vernetzt ist. Bevor eine Seitenkomponente angelaufen werden kann, muss zunächst eine Kantenkomponente besucht werden. Äquivalent zu den vorigen Ausführungen darf die Startkomponente während der Lösungskonstruktion nicht mehrfach besucht werden. Von dieser gehen demzufolge gerichtete Kanten aus.

Lösungskonstruktion einer Ameise

Eine Ameise bewegt sich nun während der Lösungskonstruktion so durch den Graphen, dass sie keine Kantenkomponente mehrfach besucht. Sobald der zurückgelegte Weg eine Seitenzuordnung repräsentiert, endet die Lösungskonstruktion. Das ist der Fall, wenn nach dem Besuch einer Seitenkomponente keine unbesuchte Kantenkomponente mehr existiert. Jeder Kante ist dann genau einer Seite zugeordnet worden. In dieser Situation enthält die zulässige Nachbarschaft des aktuellen Zustands wiederum keine Elemente und die Terminierungsbedingung e^k wird erneut zu $\mathcal{N}_i^k = \emptyset$ definiert.

Abbildung 3.9 zeigt den Konstruktionsgraphen einer Codierung dieser Art für einen speziellen zugrundeliegenden Graphen auf. Ein solcher Konstruktionsgraph sei mit G_σ bezeichnet.
Die *visuelle* Darstellung eines Lösungsweges ist nicht eindeutig interpretierbar. Kantenkomponenten können innerhalb eines Weges zwei Verbindungen zu Seitenkomponenten aufweisen. Die Seitenzuordnung wäre aus einer Grafik nicht ersichtlich. Aus diesem Grund wird auf ein Beispiel für die Decodierung eines Weges an dieser Stelle verzichtet.

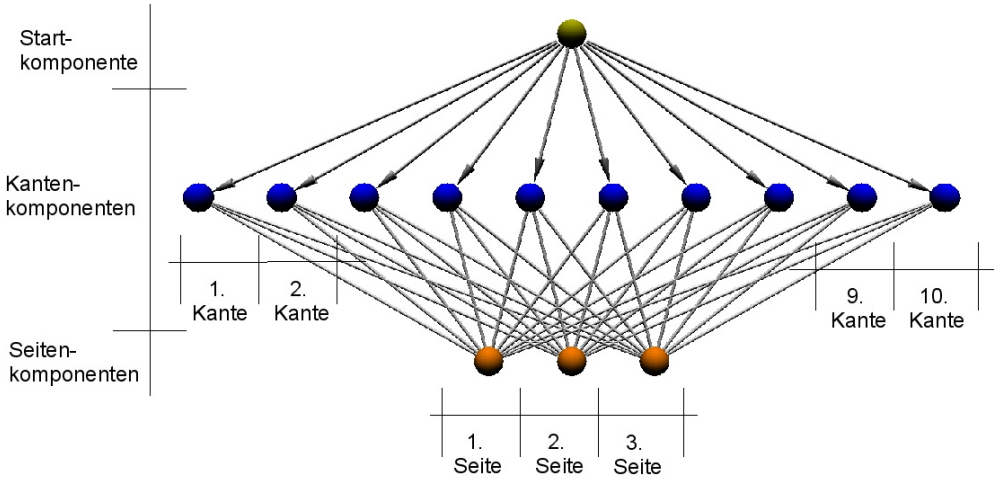


Abbildung 3.9: Codierung der Seitenzuordnung – hier im Beispiel mit 10 Kanten und k = 3

Komplexität

Für den auf Basis eines Graphen $\mathcal{G}_{BCNP} = (V, E)$ erstellten Konstruktionsgraphen $\mathcal{G}_\sigma = (\mathcal{C}, L)$ (siehe Abb. 3.9) gelten folgende Komplexitätsgrößen:

$$\begin{aligned} |\mathcal{C}| &= 1 + |E| + k \\ |L| &= |E| + (|E| * k) \\ &= |E| * (k + 1) \end{aligned}$$

Anmerkung
Eine Permutation kann wie beschrieben als bijektive Abbildung aufgefasst werden. Damit wird es grundsätzlich möglich, Konstruktionsgraphen für die Permutation auch entsprechend der hier vorgestellten Codierungen aufzustellen, wobei dann allerdings die Informationskomponenten der Positionen nicht mehrfach angelaufen werden dürfen. Der Konstruktionsgraph G_π stellt aber eine wesentlich einfachere Möglichkeit zur Codierung einer Permutation dar, die zudem aufgrund kürzerer Lösungswege schneller auszuwerten ist.
Aus diesem Grund wurde in Kapitel 3.2.2 nicht auf die weiteren Codierungsmöglichkeiten hingewiesen.

3.2.4 Codierung innerhalb eines Graphen

Ziel dieses Kapitels ist die Aufstellung eines Konstruktionsgraphen, so dass der Weg einer Ameise nun eine Permutation sowie eine Seitenzuordnung codiert. Die Konstruktionsgraphen der vorigen Unterkapitel können dazu herangezogen und miteinander vernetzt werden.

Gekoppelte Vernetzung

Grundsätzlich besteht die Möglichkeit, die Konstruktionsgraphen so zu vernetzen, dass zunächst eine Permutation und erst dann eine Seitenzuordnung generiert wird (Abb. 3.10). Die Ameise durchläuft dazu die zuvor beschriebenen Konstruktionsgraphen *nacheinander*. Wenn der Weg durch den Graphen $\mathcal{G}_\pi$ beendet ist und eine Permutation generiert wurde, muss die Wanderung auf $\mathcal{G}_\sigma$ fortgesetzt werden. Die beiden Teilgraphen werden durch gerichtete Kanten miteinander verbunden, wobei von jeder Knotenkomponente eine gerichtete Kante zu der Startkomponente des zweiten Teilgraphen angelegt werden muss. Ausgehend von jeder Knotenkomponente ist es dann möglich, den Weg fortzusetzen. Diese Verbindung darf allerdings erst dann benutzt werden, wenn eine Permutation vollständig generiert ist. Ansonsten bestünde keine Möglichkeit, nachträglich die Permutation zu vervollständigen.
Der problemabhängige Test muss die Menge der zulässigen Zustände also so definieren, dass diese Bedingung stets eingehalten wird. Die Startkomponente des Konstruktionsgraphen zur Seitenzuordnung darf erst dann Teil der zulässigen Nachbarschaft einer Knotenkomponente sein, wenn jede Knotenkomponente genau einmal besucht wurde. Diese stellt dann das einzig zulässige Element dar. Zusätzlich darf keine Knoten- und Kantenkomponente mehrfach besucht werden.
Die Lösungskonstruktion soll enden, wenn der zurückgelegte Weg eine Knotenpermutation und eine Seitenzuordnung für den Instanzgraphen repräsentiert. Die zulässige Nachbarschaft enthält in einer solchen Situation keine Komponenten, da analog zu den vorigen Ausführungen jede Kantenkomponente bereits besucht wurde und die Ameise keine weitere Möglichkeit

zur Fortsetzung ihres Lösungsweges hat. Die Terminierungsbedingung e^k ist demnach auch in dieser Codierung erfüllt, wenn die zulässige Nachbarschaft keine Elemente mehr enthält.
Ein auf diese Weise generierter Konstruktionsgraph sei mit $G_{\pi,\sigma}$ bezeichnet.

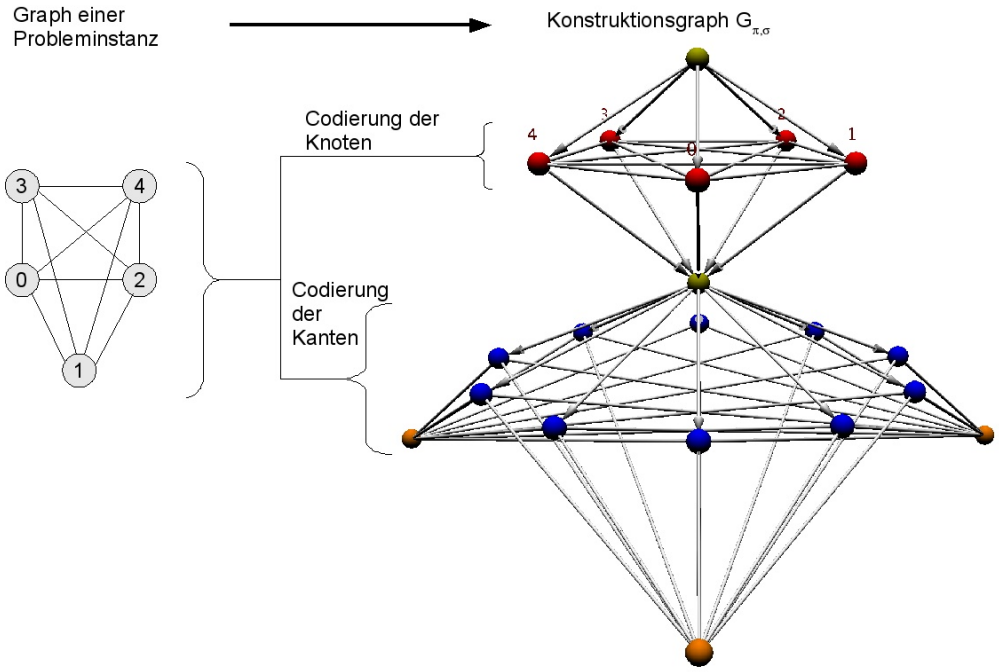


Abbildung 3.10: Konstruktionsgraph zur Codierung einer 3-seitigen Buchzeichnung eines voll vernetzten Graphen mit 5 Knoten

Die Komplexität eines solchen Graphen ergibt sich zu

$$\begin{aligned} |\mathcal{C}| &= 2 + |V| + |E| + k \\ |L| &= \frac{(|V|+1) * (|V|)}{2} + |V| + |E| * (k+1) \\ &= \frac{(|V|+3) * (|V|)}{2} + |E| * (k+1) \end{aligned}$$

Verallgemeinerung: Fusionierte Vernetzung

Eine weitere Möglichkeit besteht darin, die Generierung einer Permutation und einer Seitenzuordnung ineinander zu verflechten. Dieses stellt eine Verallgemeinerung der vorigen Vernetzungsvariante dar. Das bedeutet nun, dass sich die Ameise nach dem Besuch einer Knotenkomponente dafür entscheiden kann, eine weitere Knotenkomponente (sofern vorhanden) oder eine Kantenkomponente im Seitenzuordnungsgraphen anzulaufen. Ähnliches gilt für die Seitenkomponenten: Von hier ausgehend muss es dann möglich sein, entweder eine weitere Kante zu belegen oder eine weitere Knotenkomponente in die Permutation aufzunehmen. Auf diese Weise kann jederzeit eine zulässige Lösung generiert werden. Es darf bei der Lösungskonstruktion keine Knoten- und Kantenkomponente mehrfach besucht werden.
Dazu wird die Startkomponente des Seitenzuordnungs-Graphen entfernt und jede Knotenkomponente mit jeder Kantenkomponente durch eine gerichtete Kante verbunden. Die Kante

muss gerichtet sein, weil die Ameise die Gegenrichtung nicht durchlaufen darf. Ausgehend von einer Kantenkomponente muss immer direkt als nächstes eine Seitenkomponente angelaufen werden. Aus diesem Grund dürfen keine neuen Verbindungen von einer Kantenkomponente ausgehen. Gleiches gilt auch für die Seitenkomponenten: Sie sind zusätzlich durch gerichtete Kanten mit den Knotenkomponenten verbunden, da in der Rückrichtung zunächst eine Seite gewählt, vorher aber keine Kantenkomponente besucht wurde. Dieses ist unzulässig.
Zudem darf die Startkomponente nun auch auf die Kantenkomponenten verweisen, so dass nicht mit der Generierung einer Permutation begonnen werden muss. Der problemabhängige Test muss nach wie vor sicherstellen, dass keine Knoten- und Kantenkomponenten mehrfach besucht werden.
Die so entstehende Struktur ist in Abb. 3.11 visualisiert. Vorgreifend sei erwähnt, dass diese Codierung nur in Verbindung mit einem ausgeklügelten Pheromonmodell Vorteile bietet. Ein adäquates Pheromonmodell ist allerdings schwierig zu bestimmen – sofern es überhaupt möglich sein sollte, ein solches aufzustellen. Diese Variante wird daher nicht eingesetzt. Ergänzende Informationen finden sich in Kapitel 3.3.5.

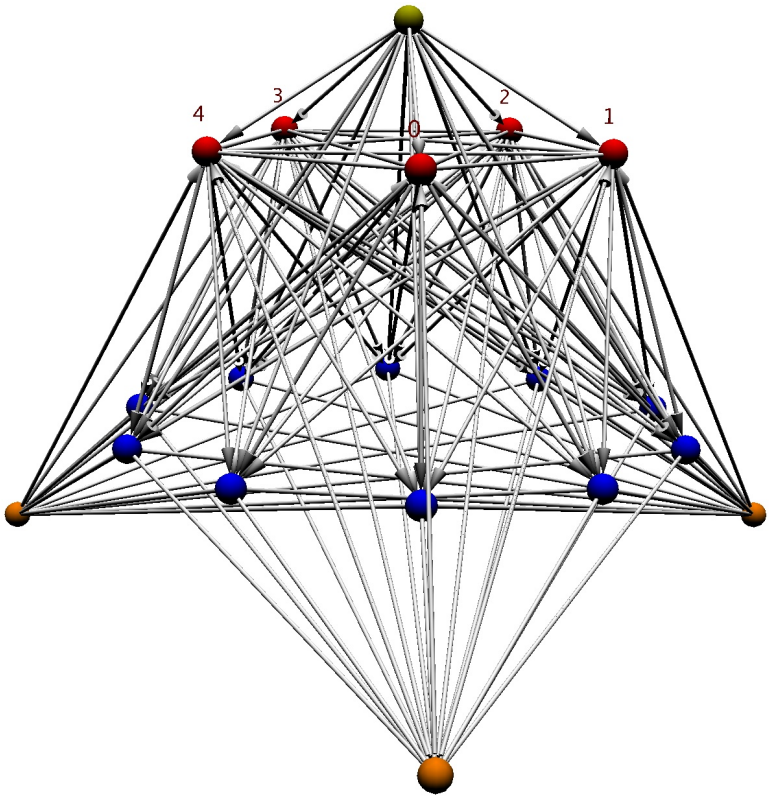


Abbildung 3.11: Fusionierte Vernetzung: Konstruktionsgraph zur Codierung einer 3-seitigen Buchzeichnung eines voll vernetzten Graphen mit 5 Knoten

Die Komplexität der Codierung aus Abb. 3.11 ergibt sich zu

$$
\begin{aligned}
|\mathcal{C}| &= 1 + |V| + |E| + k \\
|L| &= \frac{(|V|+1)*(|V|)}{2} + |E|*k + |E| + k*|V| + |V|*|E| \\
&= \frac{(|V|+1)*(|V|)}{2} + |E|*(k+1) + |V|*(k+|E|)
\end{aligned}
$$

3.3 Definition der Pheromonspuren

Pheromonspuren repräsentieren die während des Suchvorgangs gewonnenen Informationen und werden zur Generierung von Lösungswegen herangezogen, so dass Ameisen in nachfolgenden Iterationen auf Wege geleitet werden, die sich bereits als sinnvoll herausgestellt haben. Da der Erfolg einer ACO Implementierung stark von dem verwendeten Pheromonmodell abhängt, ist es äußerst entscheidend, ein mit der Natur des Problems konformes Pheromonmodell zu wählen [siehe DD99; LD04; DS04, S.212f].
Es wird nun kurz erläutert, warum das Pheromonmodell so entscheidend für die Qualität eines ACO Algorithmus ist und insbesondere darauf eingegangen, welche Pheromonmodelle für die beschriebenen Codierungen der Konstruktionsgraphen sinnvoll somit einzusetzen sind.

3.3.1 Standardvariante: Ablage von Pheromonspuren auf Kanten

Die Darstellungen in Kapitel 2.3 basieren auf der Annahme, dass die Pheromonspuren τ_{ij} die Bedeutung für gute Lösungen codieren, Komponente j direkt nach der Komponente i zu besuchen. Demzufolge lernt der Algorithmus die *relative* Positionierung einzelner Komponenten [siehe auch LD04]. Die Pheromonspuren können so gedeutet werden, dass sie auf den Kanten des Graphen abgelegt sind. Die in Kapitel 2.3 vorgestellten Formeln, insbesondere diejenigen zum Pheromonupdate, sind aus folgendem Grund auf genau dieses Pheromonmodell abgestimmt: Der erste ACO Algorithmus - das AS - wurde zunächst für eine Anwendung auf das TSP entwickelt. Erst darauffolgend sind weitere ACO Algorithmen entworfen und zudem auch auf andere Probleme adaptiert worden. Das TSP eignet sich allerdings hervorragend, um die Arbeitsweise von ACO Algorithmen auf einfache Art und Weise zu verdeutlichen. Bei der Beschreibung von ACO Algorithmen wird in nahezu allen Publikationen das beschriebene Pheromonmodell berücksichtigt.

Eine Lösung für das TSP ist eine Rundreise von Städten. Die Speicherung der relativen Komponentenpositionierung ist für dieses Problem sinnvoll, da eine Rundreise (im Spezialfall des symmetrischen TSP) eine zyklische Permutation darstellt, d.h. unabhängig an welcher Stadt die Tour begonnen und in welcher Richtung sie abgearbeitet wird, werden sich die Kosten nicht verändern. Mehrere Permutationen können ein und dieselbe Rundreise repräsentieren, wenn sie deren *relative* Positionierung der Komponenten untereinander beibehalten. Das beschriebene Pheromonmodell drückt genau diese relative Positionierung aus [siehe LD04]. Im Spezialfall des symmetrischen TSP ist die Pheromonmatrix dabei ebenfalls symmetrisch [siehe SD99], was der Interpretation entspricht, jeder Kante innerhalb des Konstruktionsgraphen *genau einen* Pheromonwert zuzuordnen. Für das asymmetrische TSP hingegen ruft die Richtung einer Verbindung zwischen zwei Städten unterschiedliche Kosten hervor. In solch einem Fall muss der Algorithmus die Möglichkeit haben, diese unterschiedlichen Informationen zu lernen. Die Pheromonmatrix ist nun nicht mehr symmetrisch. Abbildung 3.12 visualisiert die Unterschiede.

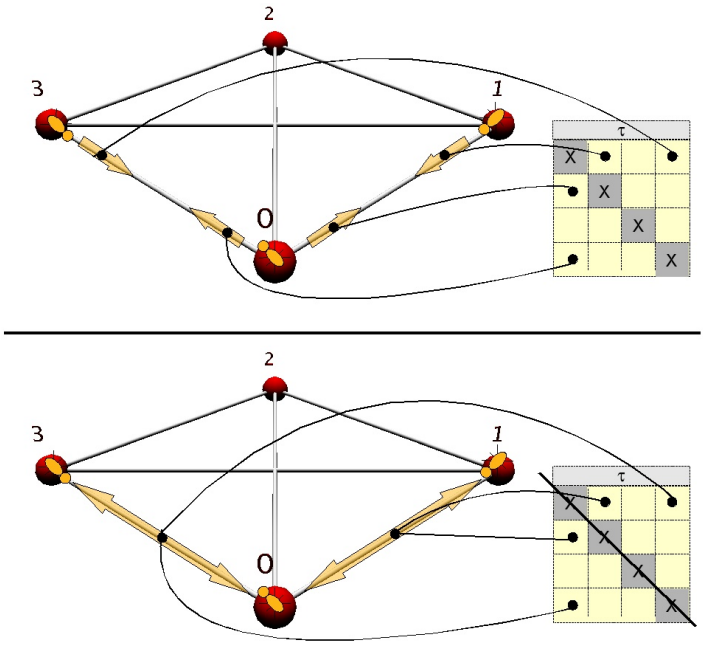


Abbildung 3.12: Unterschied zwischen asymmetrischer (oben) und symmetrischer (unten) Pheromonmatrix

3.3.2 Möglichkeiten zur individuellen Definition der Pheromonspuren

Bei Adaption der ACO auf ein spezifisches Problem ist man jedoch nicht an das beschriebene Pheromonmodell gebunden: Es ist möglich, die Bedeutung der Pheromonspuren individuell zu definieren und damit sehr genau beeinflussen zu können, auf welche Grundlage sich die Lernerfahrung des Algorithmus stützt. Zunächst sollte man festlegen, welche Informationen über das Problem zu speichern sind. Dazu ist typischerweise ein tieferes Problemverständnis erforderlich.

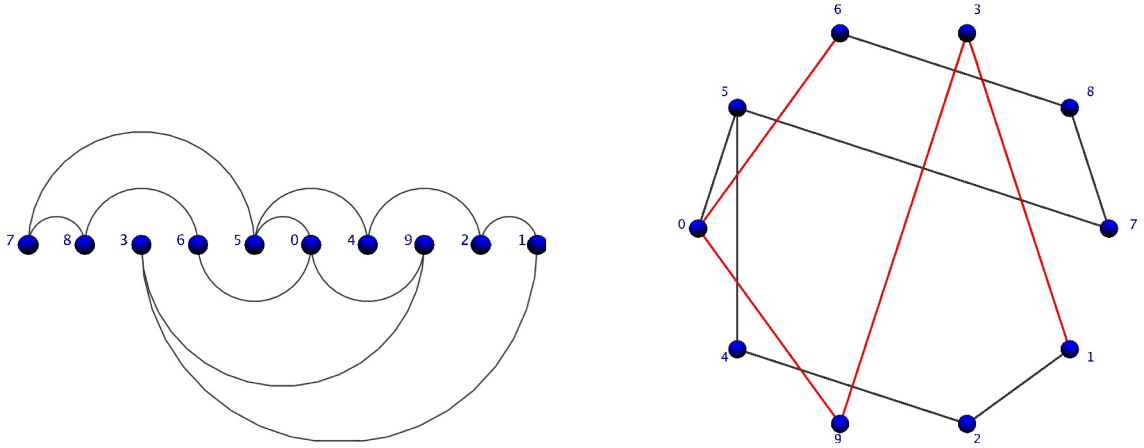
In der Anwendung von ACO auf das *k-cardinality tree problem* [Blu02] präsentiert Blum ein Pheromonmodell, das als Ablage von Pheromonspuren auf Kanten*paaren* innerhalb des Konstruktionsgraphen betrachtet werden kann. Abhängig davon, welche Kante vorher gewählt wurde, um zu der aktuellen Komponente zu gelangen, werden die Pheromonspuren für die weiterführenden Kanten verschiedene Werte annehmen.
Die Vielfalt möglicher Pheromonmodelle ist allerdings noch weitaus größer und die Pheromonspuren müssen nicht immer mit dem Konstruktionsgraphen verbunden sein. Es ist auch möglich, diese anhand des Zustands einer Ameise und zusätzlicher Speicherstrukturen zu berechnen [siehe dazu SKS02].

Bedauerlicherweise bestehen keine allgemeinen Erkenntnisse über die Wahl der Modelle, da das Forschungsfeld der ACO noch recht jung ist. Allerdings scheint es zusätzlich erstrebenswert zu sein, die Pheromonmodelle möglichst einfach und klein in Bezug auf die Komplexität zu halten. Bei Blum [Blu02] scheiterte das Pheromonmodell mit Ablage auf Kantenpaaren, da im Vergleich zu dem Standard-Modell deutlich mehr Zeit für die Ermittlung von guten Lösungen benötigt wurde. Ein Grund dafür ist die Tatsache, dass die Anzahl der zu speichernden Pheromonspuren in Bezug auf das Standard Pheromonmodell quadratisch (anstatt linear) bezüglich der Kantenanzahl ist und mehr Informationen zur Verfügung stehen.

Damit spätere Phasen des Algorithmus von der gespeicherten Sucherfahrung profitieren können, sollten möglichst viele Informationen aus einer Lösung extrahiert werden. Je spezieller die zu lernenden Informationen sind, desto eher werden in späteren Phasen für noch nicht betrachtete Teil-Lösungen keine Informationen zur Verfügung stehen. Die Ameisen können bei ihrer Lösungskonstruktion nicht so stark von vorherigen Erfahrungen profitieren.
Eine alleinige Orientierung an der Natur des Problems ist damit nicht immer auch automatisch ein Garant für gute Performance. Die nachfolgenden Punkte befassen sich mit den zu realisierenden Pheromonmodellen für die zuvor aufgestellten Konstruktionsgraphen.

3.3.3 Pheromonmodell für den Graphen G_π

Die Knotenpermutationen von k-seitigen Buchzeichnungen weisen erstaunliche Ähnlichkeit zu den Eigenschaften der Lösungen des symmetrischen TSP auf: Beides sind zyklische Permutationen. In einer Buchzeichnung können die Positionen der Knoten beliebig getauscht werden, sofern ihre *relative* Reihenfolge beibehalten wird. Die Buchkreuzungszahl verändert sich dadurch nicht.
Dieses lässt sich auf einfache Weise verdeutlichen: Zu einer k-seitigen Buchzeichnung existiert eine äquivalente *Kreiszeichnung* [siehe HSM07; He06, S.42; Win05, S.33f]. Hierbei werden die Knoten des Graphen auf der Oberfläche eines Kreises angeordnet, und zwar gemäß ihrer Anordnung in der Permutation. Die Kanten sind als gerade Linien in k Farben gezeichnet. Jede Farbe ist mit genau einer Seite assoziiert. Abbildung 3.13 veranschaulicht die Kreiszeichnung beispielhaft für eine gegebene 2-seitige Buchzeichnung.



(a) Beispielhafte 2-seitige Buchzeichnung eines Graphen

(b) Äquivalente Kreiszeichnung zu der Buchzeichnung links

Abbildung 3.13: Äquivalente Darstellungen von Buchzeichnungen

In der Kreiszeichnung ergibt sich die Kreuzungszahl zu der Anzahl der sich kreuzenden Kanten in *gleicher Farbe*. Da beide Zeichnungen äquivalent sind, ist auch die (Buch-)Kreuzungszahl gleich. Die Kreiszeichnung verwischt jedoch die Bedeutung der absoluten Komponentenpositionen. Es ist nicht mehr nachvollziehbar und somit irrelevant, bei welcher Komponente die Zeichnung begonnen und in welcher Richtung sie erstellt wurde. Die einzige auslesbare Information ist die *relative* Knotenanordnung. Alle Buchzeichnungen mit einer zyklisch veränderten Permutation entsprechen der gleichen Kreiszeichnung und weisen insbesondere die

gleiche Buchkreuzungszahl auf.
Daraus lässt sich folgern, dass die Permutation entlang des Buchrückens zyklisch ist.

Das Pheromonmodell für den Permutationsgraphen sollte aus diesem Grund die relative Anordnung der Komponenten codieren. Die Pheromonspuren werden als Matrix $\tau \in \mathbb{R}^{\mathcal{C}_V \times \mathcal{C}_V}$ gespeichert. $\mathcal{C}_V$ bezeichnet die Knotenkomponenten des Konstruktionsgraphen. Die Pheromonmatrix ist dabei symmetrisch.
Für die Verbindungen zwischen der Startkomponente und den Knotenkomponenten liegen keine Pheromoninformationen vor. Aufgrund der zyklischen Permutation kann mit jeder Knotenkomponente begonnen werden. Die Lösungskonstruktion der Ameisen wird so implementiert, dass in solchen Fällen eine Komponente aus der zulässigen Nachbarschaft randomisiert ausgewählt wird. Das entspricht zudem der biologischen Inspiration: Wenn keine Pheromonspuren vorliegen, bewegen sich die Ameisen zufallsbehaftet.

3.3.4 Pheromonmodell für den Graphen G_σ

Während der Lösungskonstruktion bewegt sich die Ameise abwechselnd zwischen Kanten- und Seitenkomponenten hin und her. Sinnvoll ist die Einbeziehung von Pheromonspuren in die Auswahl der nächsten Seitenkomponente. Anhand dieser Entscheidung wird die aktuell betrachtete Kante zugewiesen. Zu dem Zeitpunkt befindet sich die Ameise an einer Kantenkomponente i. Die Pheromonspuren τ_{ij} sollten codieren, wie sinnvoll es für gute Ergebnisse ist, die von Komponente i repräsentierte Kante auf die von der Seitenkomponente j repräsentierte Seite zu zeichnen. Dazu bestehen verschiedene Möglichkeiten.

Pheromonmodell zum Lernen der absoluten Seitenpositionen

Als Standardvariante codieren die Pheromonspuren τ_{ij} die Information, wie sinnvoll es ist, die Seitenkomponente j direkt nach Kantenkomponente i zu besuchen und auf diese Weise die von i repräsentierte Kante auf der von j repräsentierten Seite zu zeichnen. Die Pheromonspuren können so betrachtet werden, dass sie mit den Kanten von Kantenkomponenten zu Seitenkomponenten innerhalb des Konstruktionsgraphen assoziiert sind. Abhängig von der jeweiligen Kantenkomponente werden die vorliegenden Pheromonspuren verschiedene Werte annehmen. Die Pheromonmatrix τ bestimmt sich zu $\tau \in \mathbb{R}^{\mathcal{C}_E \times \mathcal{C}_k}$. $\mathcal{C}_E$ ist die Menge der Kantenkomponenten und $\mathcal{C}_k$ die Menge der Seitenkomponenten des Konstruktionsgraphen. Die Pheromonspuren sind dabei nur für die Richtung von Kantenkomponenten zu Seitenkomponenten definiert (siehe Abb. 3.14).
Diese Variante codiert eine *absolute* Zuordnung der Seiten. Für jede Kante wird gelernt, auf welcher speziellen Seite sie gezeichnet werden sollte, d.h. es wird nicht berücksichtigt, welche weiteren Kanten dieser Seite zugeordnet sind.

Befindet sich die Ameise an einer Seitenkomponente, so besteht die Möglichkeit, auch die Wahl der nächsten Kantenkomponente von Pheromonspuren abhängig zu machen. Bei der Codierung einer Seitenzuordnung für das k-seitige BCNP ist die Reihenfolge der Zuordnungen für die Lösungsqualität nicht relevant. Es ergeben sich jedoch im Zusammenspiel mit heuristischen Informationen (siehe Kapitel 3.5 für nähere Informationen) neue Möglichkeiten: Für den Erfolg einer Greedy-Heuristik ist die Reihenfolge der Seitenzuordnung ausschlaggebend. Da sich in Experimenten von Cimikowski [Cim02] gezeigt hat, dass eine randomisierte Auswahl der als nächstes zu betrachtenden Kante für die eingesetzte Greedy-Heuristik vorteilhaft

ist, wird für die Rückrichtung kein Pheromon abgelegt. Die Intention ist, dass alle Ameisen von einer Seitenkomponente ausgehend stets alle Kantenkomponenten der zulässigen Nachbarschaft gleich attraktiv einschätzen sollen. Die Lösungskonstruktion der Ameisen soll wie beschrieben einen Knoten aus der zulässigen Nachbarschaft zufällig auswählen, wenn keine Pheromoninformationen für die Entscheidung vorliegen. Die randomisierte Auswahl bietet einer dynamischen Heuristik vergrößerten Spielraum. In mehreren Durchläufen können unterschiedliche Kantenreihenfolgen erreicht werden. Wenn für diesen Graphen nicht immer die gleiche Permutation zugrundeliegt, könnte das Lernen der Reihenfolge somit nachteilig sein.
Für die Verbindungen zwischen der Startkomponente und den Kantenkomponenten sind ebenso keine Pheromonspuren gespeichert, so dass die zuerst zuzuordnende Kante ebenfalls zufällig ausgewählt wird.
Abbildung 3.14 verdeutlicht die Pheromonablage anhand eines Beispiels. Auf den Wegen von Kantenkomponenten zu Seitenkomponenten liegen Pheromonspuren vor. Für die Gegenrichtung werden keine Informationen gespeichert. Die Wahrscheinlichkeit zur Wahl einer Kante ist dann für alle Elemente der zulässigen Nachbarschaft gleich. Im Beispiel ist die rechte Kantenkomponente kein Element der zulässigen Nachbarschaft und die Wahrscheinlichkeit zur Wahl einer Alternative demnach 1/3.

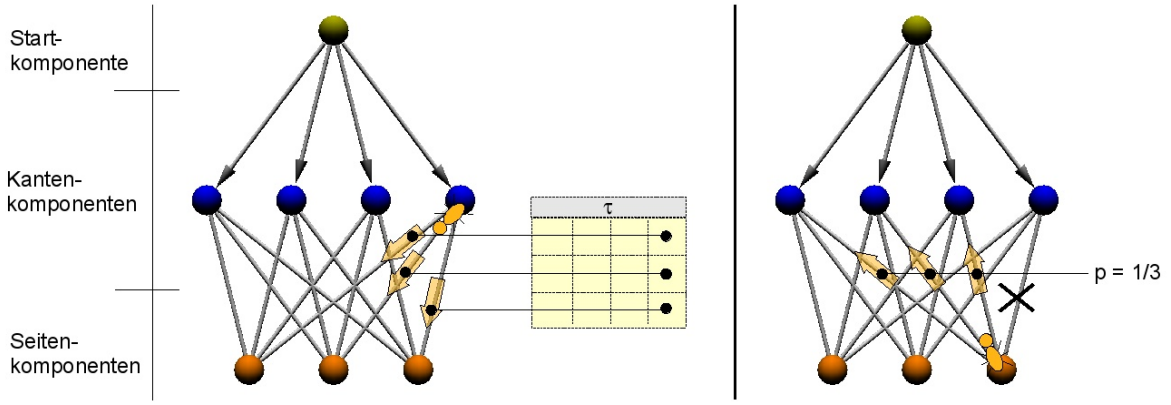


Abbildung 3.14: Beispiel für das Pheromonmodell eines Graphen $\mathcal{G}_\sigma$.

Ergänzung: Pheromonmodell zum Lernen relativer Seitenpositionen

Ausgehend von der Natur des Problems ist vorerst nicht entscheidend, *welcher* Seite die einzelnen Kanten zugeordnet sind. Relevant für die Buchkreuzungszahl ist die Information, welche Kanten *zusammen* auf einer Seite gezeichnet sind.
Socha, Knowles und Sampels [SKS02] entwickelten für das *University Course Timetabling Problem* aus vergleichbarem Grund ein anderes Pheromonmodell. Seien A und B wie gehabt die zuzuordnenden Mengen. Dieses Modell lernt, welche Elemente aus A nicht zusammen dem gleichen Element aus B zugeordnet werden sollten.
Die Informationen sind in einer zusätzlichen Matrix $\mu \in \mathbb{R}^{A \times A}$ abgelegt. Nach der Lösungskonstruktion werden die Informationen in der Matrix μ gemäß des Pheromonupdates verändert. Dazu werden die Einträge aus μ für alle Elemente $a \in A$, die in der Lösung dem *gleichen* Element $b \in B$ zugeordnet sind, paarweise entsprechend der Formel zum Pheromonupdate

geändert. Die Pheromonspuren τ_{ij} sind nicht mehr in einer Matrix abgelegt, sondern für jede Entscheidung in Abhängigkeit des bisherigen Teilweges einer Ameise aus μ zu berechnen. Nähere Informationen finden sich in [SKS02].

Adaptiert auf das k-seitige BCNP wäre dieses Pheromonmodell in der Lage zu speichern, welche Kanten nicht zusammen auf der gleichen Seite gezeichnet werden sollten. Durch die Einführung einer zusätzlichen Matrix wird jedoch besonders die Anwendung dieses Modells für das ACS erschwert. In der Durchführung des Pheromonupdates (lokal oder global) muss für jede Entscheidung nicht mehr nur ein Pheromonwert geändert werden, sondern mehrere Werte in der zusätzlichen Matrix. Socha, Knowles und Samples verwendeten ausschließlich das MMAS und das Pheromonmodell ist auf dessen Eigenschaften (im Speziellen die obere Pheromongrenze) abgestimmt. Das lokale Pheromonupdate des ACS ist hinsichtlich der Pheromonablage auf Kanten oder Komponenten abgestimmt und nicht auf die tiefere Veränderung einer zusätzlichen Pheromonmatrix.
Zudem würde das globale Pheromonupdate und die Lösungskonstruktion im Vergleich zum Standard-Modell längere Zeit benötigen, da Pheromonspuren errechnet werden müssen und nicht aus einem Matrix-Eintrag direkt auslesbar sind.
In den Experimenten von Socha, Knowles und Sampels hatte dieses Pheromonmodell keine gute Performance im Vergleich zum Standard-Modell, wenn zusätzlich eine lokale Suche eingesetzt wird. In ersten Testversuchen mit den in dieser Arbeit betrachteten ACO Algorithmen wurde es zwar adaptiert und implementiert, brachte aber höchstens vergleichbare Ergebnisse zum Standard-Modell.
Insgesamt spiegelt die Intention dieses Pheromonmodells zwar die Natur des zugrundeliegenden Problems besser wieder, bringt aber zu viele Nachteile mit sich. Es wird deshalb nicht weiter betrachtet und nur die absoluten Seitenpositionen berücksichtigt.

Pheromonspuren können aus diesem Grund stets so betrachtet werden, dass sie für jeden betrachteten Konstruktionsgraphen G_π bzw. G_σ auf den Kanten abgelegt sind (sofern Pheromoninformationen vorliegen). Dieser Umstand ermöglicht eine einfache Adaption der Algorithmen, da die Pheromonmodelle konform zu den Darstellungen aus Kapitel 2.3 sind.

3.3.5 Pheromonmodell für den Gesamtgraphen $G_{\pi,\sigma}$

Der Gesamtgraph $G_{\pi,\sigma}$ besteht aus der Vernetzung zweier Teilgraphen, für die in den vorigen Unterkapiteln bereits einzeln Pheromonspuren definiert wurden. Diese Erkenntnisse können ohne Veränderungen übernommen werden, so dass die Pheromonspuren für jeden Teilgraphen wie beschrieben verwaltet werden. Dazu beinhaltet der Konstruktionsgraph $G_{\pi,\sigma}$ zwei Pheromonmatrizen. Während der Lösungskonstruktion wird jeweils nur die Matrix des aktuellen Teilgraphen ausgewertet. Das Pheromonupdate berücksichtigt beide Matrizen.
Ist der nächste Knoten eine Knotenkomponente, so speichern die Pheromonspuren die Erfahrung, wie sinnvoll es ist, diese Knotenkomponente als nächste (unter Berücksichtigung der vorher gewählten Komponente) in der Permutation einzuordnen (siehe Kapitel 3.3.3). Die Pheromonspuren für die Kanten- und Seitenkomponenten ergeben sich aus Kapitel 3.3.4.

Ergänzung: Potential des fusioniert vernetzten Gesamtgraphen

Die deutlich komplexere Vernetzung und aufwendigere Lösungskonstruktion bietet keinerlei Vorteile, wenn für den fusioniert vernetzten Gesamtgraphen (siehe S. 47) das gleiche Pheromonmodell wie für den Graphen $G_{\pi,\sigma}$ verwendet wird. Dabei wären die Pheromoninformationen für die Wahl der nächsten Knotenkomponenten jedoch nicht mehr mit den Kanten assoziiert, sondern berechnen sich anhand des aktuellen Zustands. Die Reihenfolge der Zuordnung von Knoten und Kanten ist nicht relevant für die Zielfunktion und auch nicht für die Pheromonspuren. Es ist demnach bei diesem Pheromonmodell irrelevant, ob die Teilgraphen nacheinander durchlaufen oder miteinander kombiniert werden.
Die Möglichkeit der wechselseitigen Zuordnung von Knoten und Kanten würde nur dazu führen, dass mehr Ressourcen aufgrund der höheren Komplexität dieser Variante verbraucht werden. Zudem würde die Lösungskonstruktion entscheidend länger benötigen, da die Nachbarschaft mehr zu evaluierende Komponenten enthält und somit größer ist.
Die fusioniert vernetzte Variante ist z.B. nur dann interessant, wenn die Pheromoninformationen für die Wahl der Knotenkomponenten in Abhängigkeit von der bisherigen Teil-Seitenzuordnung codiert würden. Dieses könnte die Chance zur Generierung besserer Lösungen bieten, erfordert demgegenüber aber auch ein äußerst tiefes Problemverständnis und erhöht die Komplexität der Pheromonspuren, so dass der Erfolg von vornherein fraglich ist (siehe Kapitel 3.3.2).
Solange kein Modell gefunden ist, das die erweiterten Möglichkeiten ausschöpfen könnte, ist die Codierung gemäß des Graphen $G_{\pi,\sigma}$ zu bevorzugen. Aufgrunddessen wird in dieser Arbeit der fusioniert vernetzte Gesamtgraph nicht weiter behandelt. Eine Verallgemeinerung des Graphen $G_{\pi,\sigma}$ ist somit nur unter Verwendung deutlich komplexerer Pheromonmodelle interessant.

3.4 Heuristiken zur Generierung einer Seitenzuordnung

Wie bereits in Kapitel 3.1.1 erwähnt, ist es möglich, hybride Optimierungsansätze aus ACO Algorithmen und Heuristiken zu verfolgen. Daher werden zunächst Heuristiken entwickelt, die eine Seitenzuordnung mit k Seiten auf Basis einer Permutation generieren können und sich für einen hybriden Einsatz qualifizieren.
Weiterhin besteht die Möglichkeit der Einbeziehung heuristischer Informationen in die Lösungskonstruktion einer Ameise. In Kapitel 3.5 wird darauf eingegangen, wie heuristische Informationen ergänzend zu den Pheromonspuren ermittelt werden können.

3.4.1 Vorüberlegungen

Die zu verwendende Heuristik muss eine Seitenzuordnung bestimmen, die in Kombination mit der vorgegebenen Permutation eine möglichst minimale Buchkreuzungszahl hervorruft.

Anhaltspunkte zur Bestimmung solcher Heuristiken bietet ein Artikel von Cimikowski [Cim02]. Cimikowski entwickelte und testete acht Heuristiken für das *Fixed Linear Crossing Number Problem (FLCNP)*. Dies entspricht der Kreuzungsminimierung einer 2-seitigen Buchzeichnung mit fest vorgegebener Permutation.
Die Heuristiken können nicht direkt verwendet werden, sondern sind auf k-seitige Buchzeichnungen zu erweitern.

Prinzipiell sind Verfahren interessant, die für das 2-seitige BCNP gute Resultate liefern. Diese weisen nach einer Adaption auch für den allgemeinen Fall ein hohes Potential zur Generierung von vielversprechenden Lösungen auf.
Aus den Testergebnissen von Cimikowski geht hervor, dass ein neuronales Netz die besten Ergebnisse liefert. Allerdings wäre dieses nur für jeweils einen festen Wert des Parameters k anwendbar – k ist fest in der Struktur codiert. Es müsste daher für jeden Wert ein Netz eingesetzt und trainiert werden.
In den Testergebnissen fällt weiterhin auf, dass eine randomisierte Greedy-Heuristik äußerst zufriedenstellende Resultate erbrachte. Diese Heuristik wird nun erweitert, so dass sie eine Seitenzuordnung für das k-seitige BCNP generieren kann.

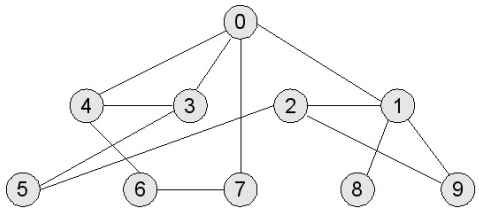
Drei weitere Heuristiken zur Bestimmung einer Seitenzuordnung für das 2-seitige BCNP werden von He vorgestellt [siehe HSV05; He06, S.98ff]. Die *SLOPE*-Heuristik wurde in Kombination mit einem genetischen Algorithmus eingesetzt und erzielte dabei vielversprechende Resultate [siehe HSM07]. Sie arbeitet zeitlich äußerst effizient, was ihren Einsatz zusätzlich motiviert.
Nachstehend wird ebenso die *SLOPE*-Heuristik zur Generierung einer k-seitigen Buchzeichnung erweitert.

3.4.2 Erweiterung der randomisierten Greedy-Heuristik

Gegeben ist der Graph einer Probleminstanz $G_{BCNP} = (V, E)$ und eine Permutation π der Knoten $v \in V$.
Cimikowski [Cim02] betrachtet nur zwei Seiten für die randomisierte Greedy-Heuristik. Diese arbeitet folgendermaßen:
Die Heuristik startet mit einer Buchzeichnung, in der noch keine Kanten eingezeichnet sind. Die Knotenreihenfolge entspricht dabei der Permutation π. In jedem Schritt wählt die Heuristik randomisiert eine noch nicht eingezeichnete Kante $e \in E$ aus und berechnet die 2-seitigen Buchkreuzungszahlen, die bei Zeichnung von e in jeder der beiden Seiten erreicht werden. Die Kante wird dann derjenigen Seite zugeordnet, auf der die geringste Buchkreuzungszahl erzielt wurde. Falls beide Seiten die gleiche Kreuzungszahl hervorrufen, wird die obere bevorzugt. Die aktualisierte (lokale) Buchzeichnung dient als Ausgangslage für den nächsten Schritt.
Sobald alle Kanten $e \in E$ eingezeichnet sind, endet das Verfahren.

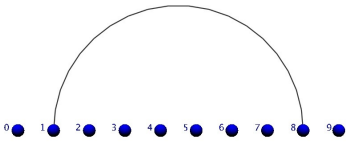
In dieser Diplomarbeit sollen kreuzungsminimale k-seitige Buchzeichnungen generiert werden. Die Heuristik wird nun dahingehend angepasst, dass nach der Auswahl einer Kante e die k-seitigen Buchkreuzungszahlen ermittelt werden, die bei Zeichnung von e in jeder der k Seiten entstehen. Anschließend wird diejenige Seite ermittelt, bei der die geringste k-seitige Buchkreuzungszahl entstand und e dieser zugeordnet. Sollten mehrere Seiten die gleiche (lokal) minimale Buchkreuzungszahl hervorrufen, bietet es sich an, unter diesen jeweils die Seite mit der kleinsten Seitennummer[9] zu wählen.
So wird erreicht, dass gerade am Anfang die erste Seite verstärkt belegt und die weiteren vorerst freigehalten werden. Eine Seite kann erst dann belegt werden, wenn eine Zeichnung auf allen vorigen Seiten eine höhere Buchkreuzungszahl hervorruft. Für die noch ausstehenden Kanten besteht in späteren Schritten eine höhere Wahrscheinlichkeit, auf einer Seite mit höherer Nummerierung ohne Kreuzungszunahme gezeichnet werden zu können. Je weniger

[9] zur Identifikation seien die Seiten durchnummeriert, etwa $0 \triangleq$ Seite $1, 1 \triangleq$ Seite $2, ..., k-1 \triangleq$ Seite k

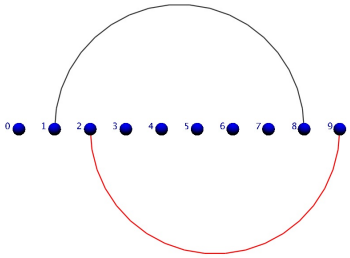


(a) Gegeben sei der Graph einer Probleminstanz, $\mathcal{G}_3$. Die randomisiert bestimmte Kantenreihenfolge beginne mit den Kanten $(1,8)$, $(2,9)$ und $(0,1)$.

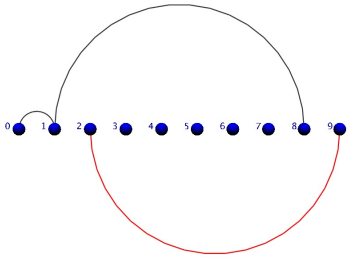
(b) Ausgangslage: Buchzeichnung mit Permutation $(0,\ldots,9)$ ohne eingezeichnete Kanten



(c) Zeichnung der zuerst ausgewählten Kante auf der ersten Seite



(d) Die nächste Kante wird auf der zweiten Seite eingezeichnet, da sie auf der ersten bereits eine Kreuzung hervorruft



(e) Die dritte Kante kann wieder auf der ersten Seite kreuzungsfrei eingezeichnet werden

Abbildung 3.15: Beispielhafte Erstellung einer Seitenzuordnung mit der randomisierten Greedy-Heuristik für die ersten drei Kanten

Kanten bereits auf einer Seite eingezeichnet sind, desto eher wird eine kreuzungsfreie Zeichnung möglich sein.

Performancesteigerung

Es bietet sich ein Potential zur Performancesteigerung, wenn die jeweilige Kante nicht erst auf einer Seite eingezeichnet und die Buchkreuzungszahl gemäß Algorithmus 3.1.2 (Seite 38) berechnet wird. Zur Ermittlung der Seite mit minimaler Buchkreuzungszahl ist lediglich die Bestimmung der Kreuzungs**zunahme** nach Hinzufügen der Kante erforderlich [siehe auch Cim02]. Algorithmus 3.4.1 vergleicht dazu die bereits eingezeichneten Kanten ausschließlich mit der aktuell betrachteten.
Zusätzlich ist es auf diese Weise möglich, die Kreuzungszunahme simultan für alle k Seiten zu bestimmen. Die Komplexität ist dann unabhängig vom Parameter k.

Die Berechnung der Kreuzungszunahme nach Algorithmus 3.4.1 setzt dieses Prinzip um und

hat eine Komplexität von $O(|E|)$. Sie muss für jede Kante einmal aufgerufen werden. Somit ergibt sich für die randomisierte Greedy-Heuristik eine Komplexität von insgesamt $O(|E|^2)$.

Algorithmus 3.4.1 : Berechnung der Kreuzungszunahme einer Kante e

Eingabe : Die (lokale) Buchzeichnung (repräsentiert durch die Permutation π und eine Seitenzuordnung σ) und die aktuell betrachtete Kante e.
Daten : int[] kreuzungszunahme
Ausgabe : kreuzungszunahme

```
1  kreuzungszunahme = new int[k];
2  i = min(π(e.knoten1), π(e.knoten2));
3  j = max(π(e.knoten1), π(e.knoten2));
4  für alle u = 1 bis |E| tue
5  |  e2 = u-te Kante in Kantenliste;
6  |  k = min(π(e2.knoten1), π(e2.knoten2));
7  |  l = max(π(e2.knoten1), π(e2.knoten2));
8  |  wenn (i < k < j < l) oder (k < i < l < j) dann
9  |  |  kreuzungszunahme[σ(e2)]++;
10 |  ende
11 ende
12 return kreuzungszunahme;
```

Da die Heuristik randomisiert ist, können in Abhängigkeit von der ausgewählten Kantenreihenfolge Lösungen unterschiedlicher Qualität erzielt werden. Sinnvoll ist damit die Realisierung von Multistarts und Wahl der besten generierten Lösung. Die Anzahl der Multistarts sollte dabei im Hinblick auf eine effiziente Rechenzeit möglichst gering gehalten werden. Gleichzeitig verringert sich dadurch aber auch die Chance zur Generierung besserer Lösungen - es werden weniger Seitenzuordnungen betrachtet. Hierbei muss ein angemessener Kompromiss gefunden werden.

Verhalten bei hoher Seitenzahl k

Die erweiterte Heuristik bringt einen angenehmen Nebeneffekt mit sich: Es wird nicht immer die maximale Anzahl von Seiten ausgenutzt. Dieser Umstand ermöglicht eine hybride Optimierung des *BEP* mit ACO Algorithmen. Sollte es unter der (randomisiert) bestimmten Reihenfolge möglich sein, eine k-seitige Buchzeichnung mit Buchkreuzungszahl 0 zu generieren (*Bucheinbettung*), so sind nur die *ersten* h von k Seiten mit Kanten belegt. Da noch freie Seiten bis zum Schluss nur belegt werden, wenn diese zur Generierung einer Zeichnung mit minimaler Kreuzungszunahme nützlich sind, bleiben in einer kreuzungsfreien Zeichnung gegebenenfalls Seiten mit höherer Seitennummer ungenutzt. Ist die generierte Zeichnung nicht kreuzungsfrei, so ist auf jeder Seite mindestens eine Kante gezeichnet; es wäre sonst während der Konstruktion eine noch freie Seite gewählt worden, auf der eine Buchkreuzungszahl von 0 hätte erreicht werden können.

Abbildung 3.16 visualisiert diesen Effekt. Der Übersichtlichkeit halber wird jede Seite durch eine spezifische Farbe repräsentiert. Für die Optimierung wurde $k = 6$ gesetzt. Auffällig ist, dass nur (die ersten) vier Seiten mit Kanten belegt sind, wobei die erste Seite (dunkelgrau) die

größte Anzahl beinhaltet. Die Kanten auf den nachfolgenden Seiten können nicht mehr auf einer vorigen Seite kreuzungsfrei eingezeichnet werden. Möglicherweise ist unter dieser Permutation mit einer anderen Kantenreihenfolge eine weitere Dezimierung verwendeter Seiten realisierbar.

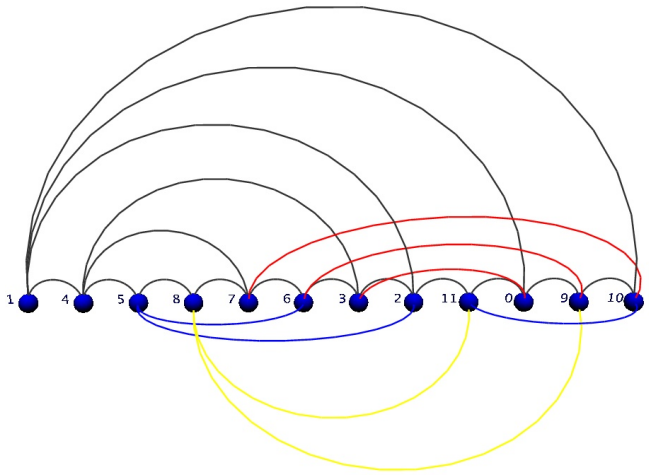


Abbildung 3.16: Eine 6-seitige Buchzeichnung, wobei die Seitenzuordnung von der randomisierten Greedy-Heuristik generiert wurde. Es sind nur 4 Seiten mit Kanten belegt, die verbleibenden 2 Seiten sind nicht belegt.

Eine Adaption von ACO Algorithmen auf das BEP mit den hier beschriebenen Ansätzen wird in Anhang B beschrieben und ist nicht direkter Bestandteil dieser Arbeit.

3.4.3 Erweiterung der *SLOPE*-Heuristik

Die *SLOPE*-Heuristik [HSV05; He06, S.98ff] betrachtet die äquivalente Kreiszeichnung des Graphen, nicht die Buchzeichnung (vgl. S. 50).
Die Kanten werden in der Kreiszeichnung entsprechend ihrer Winkel zu der Horizontalachse auf die Seiten verteilt. Im Folgenden beziehen sich die Rechnungen auf das Bogenmaß. Stehen zwei Seiten zur Verfügung, so werden alle Kanten mit einem Winkel zwischen 0 und $\frac{\pi}{2}$ der ersten Seite zugeteilt. Alle anderen werden Seite zwei zugeordnet. Diese bilden einen Winkel zwischen $\frac{\pi}{2}$ und π mit der Horizontalachse (siehe Abb. 3.17).

Da He nur zwei Seiten betrachtet, ist die explizite Bestimmung des Winkels überflüssig und die Heuristik kann auf effiziente Art umgesetzt werden, wenn direkt mit den Positionen der Start- und Endknoten einer Kante gerechnet wird [vgl. HSM07]. Für k Seiten wird diese Bestimmung aufgrund einer höheren Anzahl von Fallunterscheidungen aufwendiger und unübersichtlicher. So werden die Winkel im Folgenden explizit berechnet, auch wenn zur Berechnung der trigonometrischen Funktionen ein höherer Rechenaufwand anfällt als für einen direkten Vergleich der Positionen. Die mathematischen Grundlagen können in Bronstein [BSMM05] nachgelesen werden.
Der Winkel jeder Kante zur Horizontalachse wird exakt bestimmt und daraufhin die Seite determiniert. Dazu wird der Bereich $[0, \pi]$ in k Abschnitte unterteilt und jede der k Seiten ist mit genau einem dieser Bereiche assoziiert. Die zuzuweisende Seite bestimmt sich nun deterministisch, je nachdem, in welchem Abschnitt der Winkel der Kante liegt.

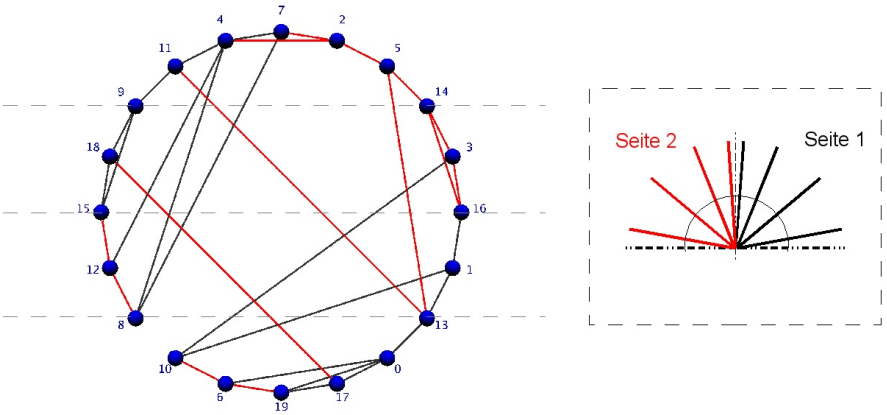


Abbildung 3.17: Beispiel für die Kanteneinteilung der *SLOPE*-Heuristik mit k = 2. Die Kanten in roter Farbe sind der 2. Seite zugeordnet.

Gegeben ist eine Permutation π der Knoten $v \in V$ des Instanzgraphen $G_{BCNP} = (V, E)$. Die Winkel werden auf Grundlage der Kreiszeichnung in Abb. 3.18 berechnet, in der die Knoten gemäß ihrer Position in der Permutation entlang der Oberfläche des Kreises anordnet werden. Alle in der Permutation benachbarten Knoten haben dabei den gleichen Abstand ℓ voneinander. Der Ursprung des (kartesischen) Koordinatensystems befindet sich im Zentrum des Kreises.

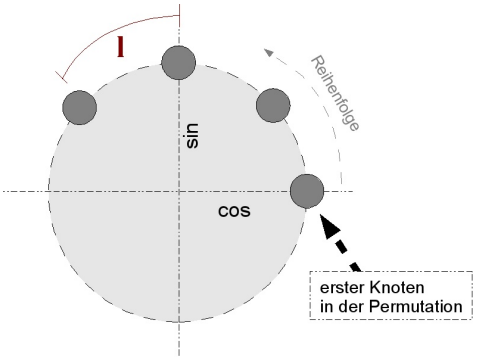


Abbildung 3.18: Anordnungsreihenfolge der Knoten in der Kreisdarstellung gemäß einer zugrundeliegenden Permutation

Für die Berechnung der Winkel ist der Radius des umschließenden Kreises uninteressant. Daher wird dieser im Folgenden zum Einheitskreis bestimmt.
ℓ bezeichnet die Länge des zwei benachbarte Knoten trennenden Kreisbogens und bestimmt sich nach Formel 3.1.

$$\ell = \frac{2 * \pi}{|V|} \tag{3.1}$$

Die Koordinaten des Knotens an i-ter Position[10] in der Permutation, können anhand der

[10] die Zählung beginne bei 0.

folgenden Formeln berechnet werden:

$$x = cos(i * \ell) \tag{3.2}$$
$$y = sin(i * \ell) \tag{3.3}$$

Der Winkel zwischen der Horizontalachse und einer Kante lässt sich mit Hilfe trigonometrischer Funktionen berechnen. Die Positionen von beiden verbundenen Knoten auf dem Kreis sind bekannt (Formel 3.2, 3.3).

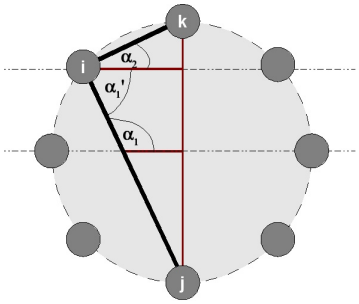


Abbildung 3.19: Berechnung des Winkels einer Kante

Für alle gesuchten Winkel α gilt stets die Gleichung 3.4.

$$\begin{aligned} & \tan(\alpha) = \tfrac{\Delta y}{\Delta x} \\ \Leftrightarrow \quad & \alpha = arc\tan\left(\tfrac{\Delta y}{\Delta x}\right) \end{aligned} \tag{3.4}$$

Seien i und j die miteinander verbundenen Knoten. Δy und Δx werden nach Formel 3.5 bestimmt.

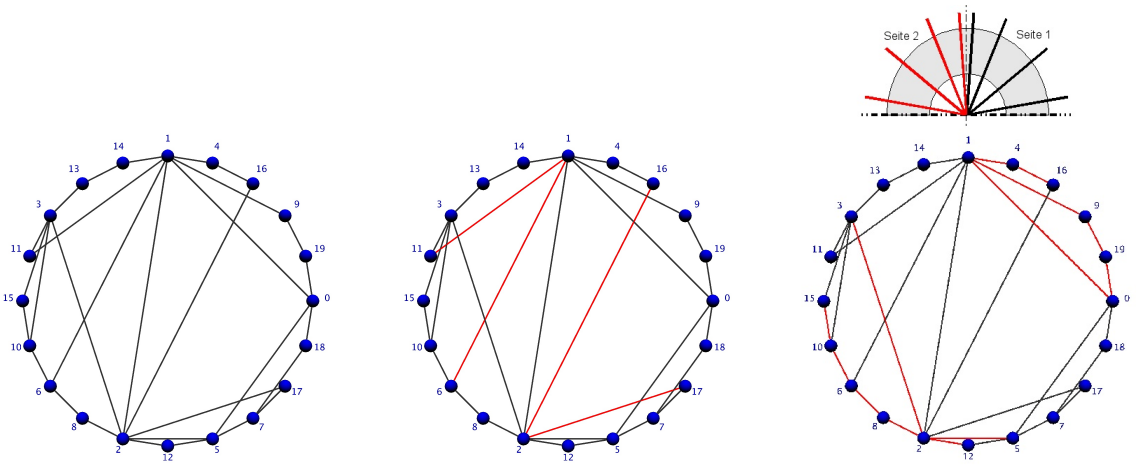
$$\begin{aligned} \Delta y = i.y - j.y \wedge \Delta x = i.x - j.x \\ (\text{oder } \Delta y = j.y - i.y \wedge \Delta x = j.x - i.x) \end{aligned} \tag{3.5}$$

Die $arc\tan$-Funktion berücksichtigt allein den Quotienten beider Werte. Insofern ist nicht relevant, ob die Positionen des ersten Knotens von den Positionen des zweiten Knotens subtrahiert werden oder umgekehrt. Die Funktion $arc\tan$ hat einen Bildbereich von $\left[-\frac{\pi}{2}, +\frac{\pi}{2}\right]$, so dass Winkel $> \frac{\pi}{2}$ negativ in Bezug zur Horizontalachse angegeben werden. Um dieses zu korrigieren, muss in solch einem Fall der Wert π hinzuaddiert werden. Dadurch wird stets der gewünschte Winkel im Bereich $[0, \pi]$ ermittelt.

Beispiel:
In Abb. 3.19 sind folgende Daten gegeben: $|V| = 8$, $\ell = \frac{\pi}{4}$. Zu Bestimmen sei der Winkel der Kante $(i, j) \in E$ mit der Horizontalachse. Der Knoten i hat in der Permutation Position 3 und Knoten j Position 6. Gemäß Formel 3.4 und 3.5 gilt damit: $arc\tan\left(\frac{1.707}{-0.707}\right) \approx -1.178$. Da dieser Wert < 0 ist wird der Winkel α_1' angegeben. Gesucht ist aber der Gegenwinkel α_1. Daher ergibt sich $\alpha_1 = \pi + \alpha_1'$. α_2 hingegen ist positiv und spiegelt direkt den gesuchten Winkel wieder.

Im Gegensatz zur Greedy-Heuristik ist die *SLOPE*-Heuristik nicht versucht, „seitensparend“ zu arbeiten und demzufolge nicht zur Generierung einer seitenminimalen Bucheinbettung geeignet. Die *SLOPE*-Heuristik ist deterministisch und hat eine Komplexität von $O(|E|)$, da jede Kante genau einmal evaluiert wird. Sie arbeitet deshalb äußerst effizient.

Abbildung 3.20 und 3.21 zeigen Beispiele für die Anwendung der Heuristiken auf.

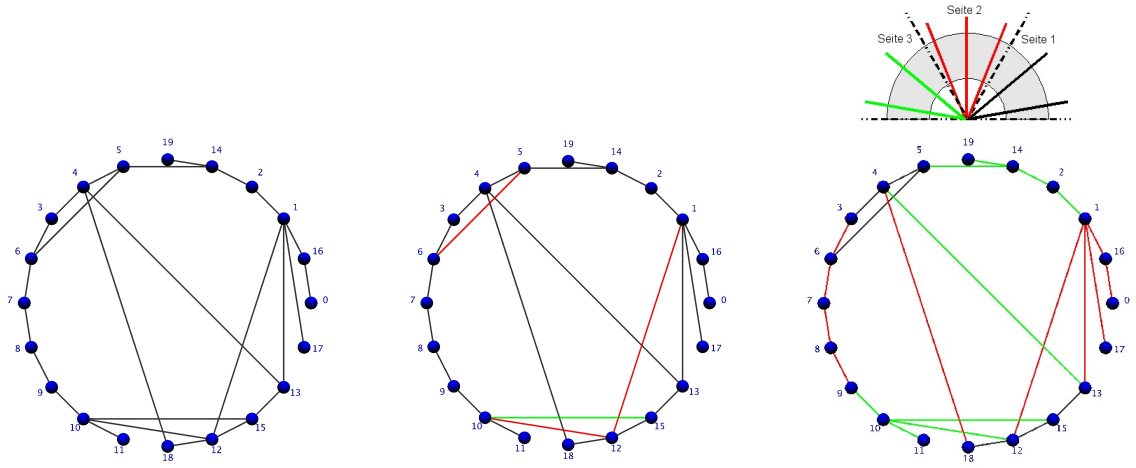


(a) Gegeben ist ein Graph G_1 (als Kreiszeichnung) und $k = 2$

(b) Seitenzuordnung unter Anwendung der Greedy-Heuristik bei unveränderter Permutation. Die Kreuzungszahl ist 0.

(c) Seitenzuordnung unter Anwendung der *SLOPE*-Heuristik bei unveränderter Permutation. Die Kreuzungszahl ist 4.

Abbildung 3.20: Ein Beispiel der Greedy- und *SLOPE*-Heuristiken für $k = 2$



(a) Gegeben ist ein Graph G_2 (als Kreiszeichnung) und $k = 3$

(b) Seitenzuordnung unter Anwendung der Greedy-Heuristik bei unveränderter Permutation. Die Kreuzungszahl ist 0.

(c) Seitenzuordnung unter Anwendung der *SLOPE*-Heuristik mit unveränderter Permutation. Die Kreuzungszahl ist 0.

Abbildung 3.21: Ein Beispiel der Greedy- und *SLOPE*-Heuristiken für $k = 3$

3.5 Heuristische Informationen

Heuristische Informationen unterstützen die Ameisen bei der Auswahl von Lösungskomponenten und repräsentieren *a priori Wissen* über das Problem. Man unterscheidet dabei zwischen zwei Arten [vgl. DS04, S.215]:

- **Statische Informationen**
  Statische heuristische Informationen werden in der Initialisierung des Algorithmus *einmalig* berechnet und in einer Matrix η gespeichert. Sie verändern sich nicht mehr und bleiben über alle Lösungskonstruktionen unverändert. Dies ermöglicht einen geringen Rechenaufwand.

- **Dynamische Informationen**
  Dynamische heuristische Informationen sind von der bislang generierten Teil-Lösung einer Ameise abhängig, die aus deren Zustand hervorgeht. Sie müssen daher für jede Entscheidung in der Lösungskonstruktion neu berechnet werden. Demnach ist ein höherer Rechenaufwand im Vergleich zu statischen Informationen erforderlich, der jedoch durch eine höhere Genauigkeit der berechneten Informationen kompensiert werden könnte.

Heuristische Informationen müssen für jedes Teilproblem getrennt spezifiziert werden. Vor allem unterscheiden sich heuristische Informationen für die Wahl der nächsten Knotenkomponente von denjenigen, die zur Wahl der nächsten Seitenkomponente vorliegen. Sie basieren auf differenzierenden Berechnungsgrundlagen.
Insofern können sie bei der Optimierung innerhalb des Gesamtgraphen $G_{\pi,\sigma}$ eine unterschiedliche Relevanz für die Lösungskonstruktion aufweisen. Der Parameter β gibt vor, wie wünschenswert Komponenten mit hohen heuristischen Werten auszuwählen sind. Aufgrund unterschiedlicher Berechnungsgrundlagen mag es notwendig sein, zwei Parameterwerte für β zu berücksichtigen und die Gewichtung getrennt einzustellen.
Unterschiedlich ist weiterhin die Struktur beider Teilgraphen. Diesbezüglich sollten auch für die Gewichtungen der Pheromonwerte zwei getrennt einstellbare Parameter von α herangezogen werden. Ergänzende Informationen finden sich in Kapitel 3.6.1.

3.5.1 Unterstützung der Bestimmung einer Permutation

Eine Übersicht über Algorithmen für das 1- und 2-seitige BCNP findet sich in [HS04; He06, S.70ff]. Ziel des 1-seitigen BCNP ist ausschließlich die Bestimmung einer vorteilhaften Knotenpermutation. Aus den Heuristiken für das 1-seitige BCNP können Anhaltspunkte zur Bestimmung angemessener heuristischer Informationen für eine Permutationsgenerierung abgeleitet werden.

Ein mögliches zu verfolgendes Ziel ist die Maximierung der Kantenanzahl entlang der Oberfläche des umschließenden Kreises, bezogen auf die äquivalente Kreiszeichnung des Graphen (siehe Kapitel 3.13). Der *CIRCULAR*-Algorithmus von Six und Tollis [ST99] setzt dieses Prinzip um.
Das hat folgenden Grund: Je größer der Abstand zwischen den verbundenden Knoten, desto mehr Kreuzungen kann eine Kante potentiell hervorrufen. Das Ziel besteht in der Vermeidung von Kanten, die durch die Mitte verlaufen. Die Kanten zwischen den in der Permutation benachbarten Knoten können sogar niemals eine Überschneidung mit einer anderen Kante

hervorrufen (siehe Abb. 3.22). In einigen Implementierungen wird aus diesem Grund nach einem *Hamiltonkreis* im Graphen der Probleminstanz gesucht und dessen (relative) Knotenreihenfolge als Permutation verwendet [siehe z.B. Cim02]. Ein Hamiltonkreis ist ein **Kreis**[11] innerhalb eines Graphen $\mathcal{G}$, der alle Knoten von $\mathcal{G}$ (genau einmal) enthält [vgl. Die00, S.209].

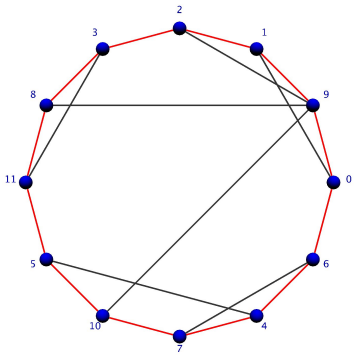


Abbildung 3.22: Anordnung entsprechend eines Hamiltonkreises (rot)

Aus diesen Erkenntnissen lassen sich statische heuristische Informationen ergänzend zu den Pheromonspuren definieren. Die Pheromonspuren τ_{ij} codieren für Knotenkomponenten die Erwünschtheit, Komponente j direkt nach der Komponente i auszuwählen. Heuristisch zu bevorzugen sind dabei Knotenkomponenten j, deren repräsentierter Knoten im Graphen der Probleminstanz durch eine Kante mit dem von Knotenkomponente i repräsentierten Knoten verbunden ist. Solche Knotenkomponenten sollten einen höheren heuristischen Wert aufweisen. Auf diese Weise wird die Möglichkeit zur Generierung von Knotenordnungen geschaffen, die einen Hamiltonkreis darstellen.

Die heuristischen Informationen werden wie folgt definiert:

$$\eta_{ij} = \begin{cases} 1.0 & \text{falls } (k(i), k(j)) \in E \\ 0.5 & \text{sonst} \end{cases} \tag{3.6}$$

$k(i)$ bezeichnet den von Knotenkomponente i repräsentierten Knoten.
Relevant für die Wahl der Konstanten ist die Tatsache, dass miteinander verbundene Knoten den Wert 1.0 zugewiesen bekommen. Dieser verändert sich durch beliebige Wahl des Exponenten β nicht. Anders hingegen verhält es sich mit Konstanten $\neq 1.0$. Hierbei hat die Wahl von β entscheidenden Einfluss. Ausschlaggebend ist, dass der heuristische Wert von nicht verbundenen Knoten < 1.0 ist. Je größer β ist, desto kleiner werden die heuristischen Werte für nicht verbundene Knoten. Umso eher werden jetzt in der Lösungskonstruktion Komponenten angesteuert, deren repräsentierte Knoten adjazent sind.
Da sich die heuristischen Informationen während der Lösungskonstruktion nicht verändern, können sie statisch in der Initialisierungsphase berechnet werden (siehe auch Abb. 3.23).

Weiterführende Idee

Bei dem beschriebenen Ansatz werden alle Komponenten, deren repräsentierte Knoten zu dem von der aktuellen Komponente repräsentierten Knoten adjazent sind, gleich „attraktiv“

[11] siehe Kapitel 2.1.

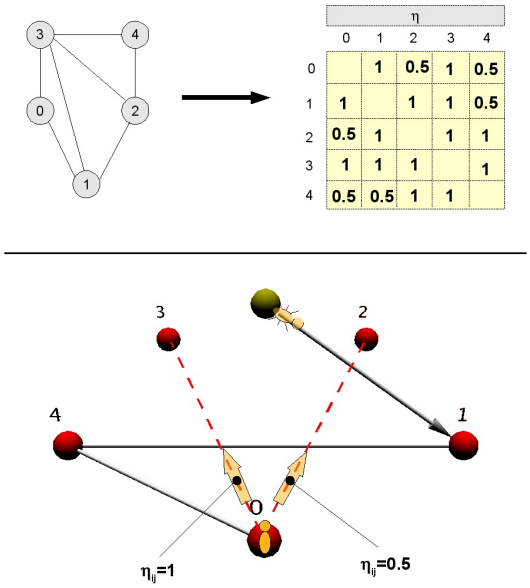


Abbildung 3.23: Beispiel für Berechnung statischer heuristischen Informationen anhand eines Instanzgraphen. Die Matrix η bezieht sich auf Komponenten des Konstruktionsgraphen, die hier ebenfalls anhand einer Nummer - der des repräsentierten Knoten - identifiziert seien.

bewertet.
Erweiternd könnte zusätzlich noch eine Abstufung der heuristischen Werte dieser Komponenten vorgenommen werden. Resende und Ribeiro [RR01] stellen Erkenntnisse über eine zwei-Phasen Heuristik zusammen und beschreiben dort u.a. eine heuristische Strategie von Goldschmidt und Takvorian [GT94] zur Suche eines Hamiltonkreises. Der erste Knoten wird zu demjenigen mit minimalem Grad bestimmt. Im weiteren Verlauf wird immer derjenige von den verbundenen Knoten gewählt, welcher mit einer minimalen Anzahl bereits belegter Knoten verbunden ist.
Hieraus abgeleitet könnten Knotenkomponenten mit höheren heuristischen Werten versehen werden, die mit möglichst wenig der bislang platzierten Knotenkomponenten verbunden sind. Es entstünde die Notwendigkeit zur dynamischen Bestimmung heuristischer Informationen, da die bislang platzierten Knoten nur aus dem jeweiligen Zustand der Ameise auslesbar sind. In dieser Arbeit wird jedoch aufgrund des höheren zeitlichen Aufwandes zur Berechnung dynamischer Informationen nicht von einer solchen Erweiterung Gebrauch gemacht.

3.5.2 Unterstützung der Bestimmung einer Seitenzuordnung

Wenn eine Ameise durch den Konstruktionsgraphen der Seitenzuordnung wandert, bewegt sie sich wechselseitig zwischen Kanten- und Seitenkomponenten. Heuristische Informationen müssen nur für solche Situationen herangezogen werden, in denen sich die Ameise an einer Kantenkomponente befindet; nur dann liegen auch Pheromonspuren vor. Von hier aus muss sie sich für die nächste zu besuchende Seitenkomponente entscheiden und ordnet auf diese Weise die Kante einer Seite zu.
An dem bisherigen Teil-Weg bzw. Zustand der Ameise können die bislang zugeordneten Kan-

ten ausgelesen und eine (lokale) Buchzeichnung generiert werden, analog zu den Ausführungen über die Greedy-Heuristik. Um die Suche besser leiten zu können, wäre es sinnvoll, diejenigen Seitenkomponenten zu bevorzugen, auf deren repräsentierter Seite die Zeichnung der zu belegenden Kante unter Berücksichtigung der lokalen Buchzeichnung möglichst wenige Kreuzungen verursachen würde.
Zur Generierung der lokalen Buchzeichnung muss die Permutation im Vorfeld bekannt sein. Das ist hingegen für alle angesprochenen Optimierungsvarianten der Fall: Sogar für die Optimierung mit dem Gesamtgraphen $G_{\pi,\sigma}$ wird durch den ersten Teil-Weg zunächst eine Permutation codiert. Sie steht demzufolge fest bzw. kann aus dem Zustand einer Ameise extrahiert werden, bevor dynamische heuristische Informationen für die Seitenzuordnung zu ermitteln sind (siehe Abb. 3.24).

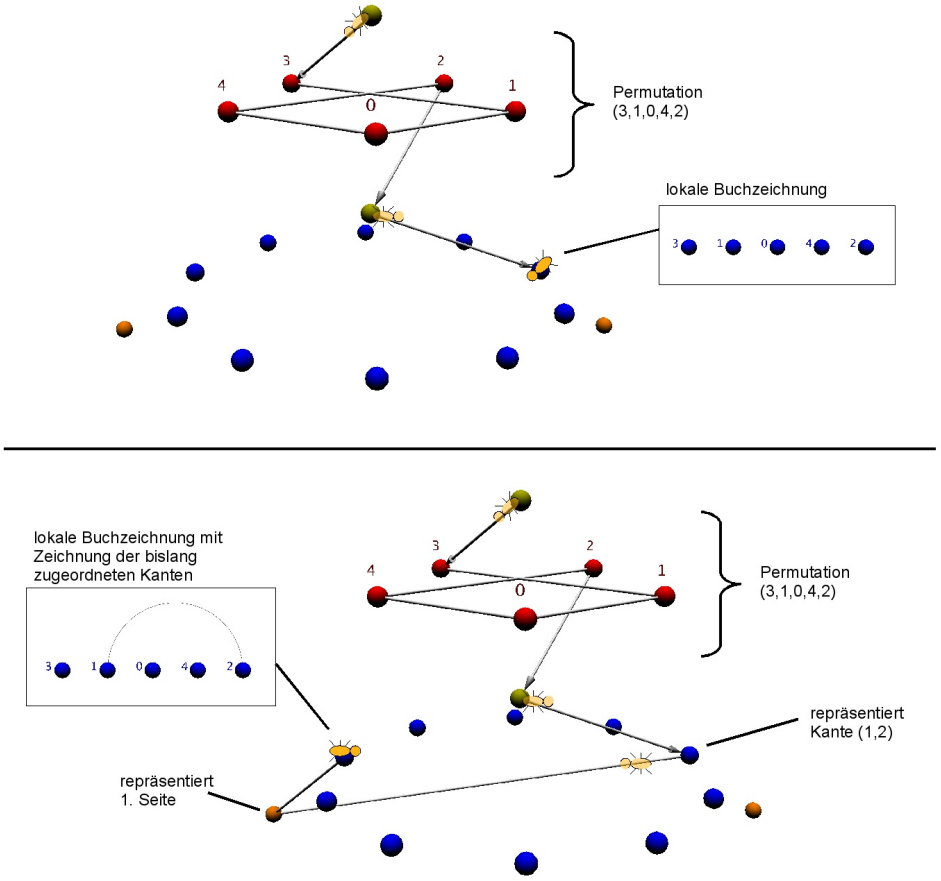


Abbildung 3.24: Generierung der lokalen Buchzeichnung anhand des bisherigen Weges einer Ameise. Sie stellt die Basis zur Ermittlung heuristischer Informationen dar.

Generell entsteht die Notwendigkeit zur *dynamischen* Bestimmung der heuristischen Informationen, da diese von allen vorigen Entscheidungen abhängig sind. Basis dieser Daten kann nur eine Greedy-Heuristik sein. Die Ameise entscheidet sich an jeder Komponente aufgrund eines randomisierten Prozesses für die nächste zu besuchende Komponente. Es dürfen zur Generierung heuristischer Informationen nur Daten verwendet werden, die bereits durch den vorherigen Weg der Ameise determiniert sind und nicht mehr verändert werden können. Alle weiteren

Annahmen, die beispielsweise auf noch ausstehenden Entscheidungen beruhen, könnten durch die randomisierte Lösungskonstruktion nicht sichergestellt werden und die Basis der heuristischen Informationen wäre hinfällig. Somit eignen sich ausschließlich Greedy-Heuristiken: Sie wählen die in der momentanen Situation „beste“ Alternative.

Zur Berechnung der heuristischen Informationen wird die lokale Teil-Buchzeichnung während der Lösungskonstruktion mitgespeichert. Befindet sich eine Ameise an einer Kantenkomponente i, wird für jede Seitenkomponente j in der zulässigen Nachbarschaft[12] die Kreuzungs**zunahme** nach Algorithmus 3.4.1 auf Seite 57 ermittelt. Sie wird dann in einen heuristischen Wert umgewandelt:

$$\eta_{ij} = \frac{1}{1 + cr_{ij}(x^k)} \tag{3.7}$$

$cr_{ij}(x^k)$ bezeichnet die Kreuzungszunahme bei Zeichnung der von i repräsentierten Kante auf der von j repräsentierten Seite. Der Zustand x^k der Ameise k beinhalte die lokale Teil-Buchzeichnung, auf die sich die Berechnung bezieht. Im Nenner wird die Kreuzungszahl mit 1 addiert, da sie auch den Wert 0 annehmen kann und die Division durch 0 nicht definiert ist. Seitenkomponenten, die bei Ansteuerung eine niedrigere Buchkreuzungszahl in der anhand des Weges generierten Buchzeichnung hervorrufen, erlangen eine höhere heuristische Präferenz. Auch hier gilt: Je höher β, desto wichtiger wird ein hoher Wert der heuristischen Informationen bei der Lösungskonstruktion.

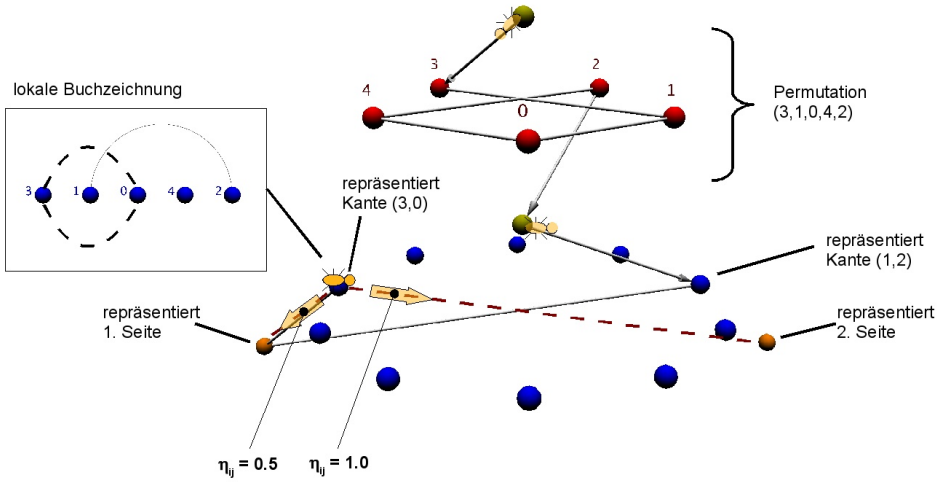


Abbildung 3.25: Beispiel für die Berechnung der dynamischen heuristischen Informationen. Die Kante (3,0) verursacht bei Zeichnung auf der ersten Seite eine Kreuzung mit der bereits eingezeichneten Kante (2,1). Daher ist die heuristische Information für diese Alternative mit 0.5 bewertet. Bei Zeichnung auf der zweiten Seite bleibt die Buchkreuzungszahl auf dem Wert 0 und $\eta_{ij} = 1$.

Wenn der Wert von β sehr hoch ist und $\alpha = 0$, wird jede Ameise in ihrer Lösungskonstruktion beständig diejenige Seitenkomponente mit der geringsten (lokalen) Kreuzungszunahme wählen. Der Algorithmus wird sich in diesem Fall wie ein randomisierter Greedy Algorithmus verhalten. Das Verhalten ist aber nicht ganz mit der Greedy-Heuristik aus dem vorigen

[12] lt. Definition aus Kapitel 3.2.3 enthält die zulässige Nachbarschaft stets **alle** Seitenkomponenten.

Unterkapitel vergleichbar, denn diese bevorzugt zusätzlich noch die Seite mit der kleinsten Seitennummer. Hierbei würde allerdings unter den Seiten mit der gleichen lokal optimalen Kreuzungszunahme zufällig ausgewählt.

3.6 Anwendung eines ACO Algorithmus

In den folgenden Unterkapiteln wird beschrieben, wie die einzelnen der in Kapitel 2.3 vorgestellten abstrakten Algorithmus-Schemata konkret für dieses Problem umgesetzt werden. Dieses stellt den vorletzten Schritt zur Anpassung von ACO Algorithmen auf das k-seitige BCNP dar. Desweiteren werden die zu optimierenden Parameter herausgestellt.

3.6.1 Globale Einstellungen

Dieses Unterkapitel fasst zunächst Einstellungen und Parameter zusammen, die in der Implementierung jedes ACO Algorithmus angepasst werden müssen.

Abbruchbedingung

Die Abbruchbedingung sei erfüllt, wenn eine der folgenden Gültigkeit hat:

- Es wurde eine Lösung s mit $f(s) = 0$ gefunden
- Eine bestimmte Iterationsanzahl n_{max} ist erreicht worden

Da das globale Optimum der Buchkreuzungszahl einer k-seitigen Buchzeichnung in der Regel nicht bekannt ist, wird keine Rücksicht darauf genommen, ob die Lösungsqualität 0 überhaupt erreicht werden kann. Sollte das globale Optimum einen Wert > 0 aufweisen, so wird der Algorithmus erst nach Erreichen der maximalen Iterationsanzahl terminieren.
n_{max} ist dabei problemabhängig zu wählen. Die Bestimmung erfolgt in Kapitel 4.2.1, nachdem die zu testenden Probleminstanzen feststehen. Auf komplexeren Graphen sollte eine höhere Iterationsanzahl erlaubt werden, damit die Lösungskonstruktion nicht vorzeitig abgebrochen wird und vielversprechende Lösungen gefunden werden können. Die Operationen zur Adaption der Pheromonspuren und somit der Steuerung des Suchprozesses werden nach jeder Iteration getätigt. Ein Vergleich über die Iterationen stellt sicher, dass die Algorithmen die gleichen Chancen aufweisen, gute Lösungen zu finden.

Parametereinstellungen

Die in allen ACO Algorithmen einzustellenden Parameter sind:

- die Anzahl der Ameisen m
- die Pheromon-Verdunstungsrate ρ
- β und α (ausgenommen ACS)

Die Parameterempfehlungen für das TSP [siehe DS04, S.71] lassen darauf schließen, dass die Anzahl der Ameisen m für das AS und MMAS linear mit der Größe des Konstruktionsgraphen zunehmen sollte. Auf diese Weise werden mehr Lösungen generiert und evaluiert, die zum Pheromonupdate herangezogen werden können und zukünftige Entscheidungen beeinflussen. Nachteilig wirkt sich das jedoch auf die benötigte Rechenzeit auf.
In Vorabexperimenten mit den implementierten Algorithmen konnten durch den Einsatz von mehr Ameisen insgesamt bessere Lösungen gefunden werden, so dass nachstehend für alle in den Grundlagen vorgestellten ACO Algorithmen die Ameisenanzahl linear in Abhängigkeit der Größe des Konstruktionsgraphen eingestellt wird (siehe auch S.72). Die k Seitenkomponenten werden dabei nicht aufgegriffen.
Unberücksichtigt bliebe bei einer zusätzlichen Optimierung der Ameisenanzahl die damit einhergehenden Laufzeitveränderung, sofern diese nicht zur Beurteilung der Lösungsqualität mit eingerechnet wird.

Die Anzahl der Ameisen für den Konstruktionsgraphen $\mathcal{G}_\pi$ bestimmt sich folgend zur Hälfte der Problemgröße. Damit die Ameisenanzahl bei der Optimierung auf dem Graphen $\mathcal{G}_{\pi,\sigma}$ nicht zu stark anwächst, wird die Problemgröße hierbei nur mit dem Faktor $\frac{1}{4}$ multipliziert. Wenn der Instanzgraph einen Durchschnittsgrad von mindestens 2.0 aufweist, werden nicht weniger Ameisen als für den Graphen $\mathcal{G}_\pi$ eingesetzt. Die sich daraus ergebende Ameisenanzahl m ist in Tabelle 3.1 für die jeweiligen Konstruktionsgraphen angegeben.

	m
$\mathbf{G}_\pi$	$\frac{1}{2} * \lvert\mathcal{C}_V\rvert$
$\mathbf{G}_{\pi,\sigma}$	$\frac{1}{4} * (\lvert\mathcal{C}_V\rvert + \lvert\mathcal{C}_E\rvert)$

Tabelle 3.1: Berechnung der Ameisenanzahl m anhand der zugrundeliegenden Problemgröße. $|\mathcal{C}_V|$ ist die Anzahl der Knotenkomponenten und $|\mathcal{C}_E|$ die Anzahl der Kantenkomponenten des Konstruktionsgraphen.

Einführung zusätzlicher Parameter

Sollte der vorliegende ACO Algorithmus die Lösungskonstruktion auf dem Konstruktionsgraphen $G_{\pi,\sigma}$ durchführen, so wird zusätzlich zu β noch ein zweiter Parameter β_2 verwendet. Die bei der Lösungskonstruktion verwendeten unterschiedlichen heuristischen Informationen können auf diese Weise entsprechend berücksichtigt werden.
Analog mag es ebenso sinnvoll sein, zwei Parameterwerte zur Gewichtung der Pheromonspuren heranzuziehen, da die Struktur des zugrundeliegenden Pheromonmodells beider Teilgraphen unterschiedlich ist. Zwar sind alle Pheromonspuren (bildlich) auf den Kanten des Konstruktionsgraphen $G_{\pi,\sigma}$ abgelegt, dennoch verhält sich der Lauf einer Ameise auf beiden Teilgraphen unterschiedlich. Für $k = 2$ stehen einer Ameise im Teilgraphen der Seitenzuordnung je zwei Alternativen für die Wahl der nächsten Seite zur Verfügung. Im Teilgraphen der Permutation hingegen verringert sich die Anzahl der Auswahlmöglichkeiten mit jeder weiteren Entscheidung stets um den Wert eins. Der Parameter α bestimmt den Einfluss der Pheromonspuren bei der Entscheidungsfindung. Somit sollten die α-Werte getrennt einstellbar sein, um die unterschiedliche zugrundeliegende Struktur berücksichtigen zu können.

Künftig wird für den Graphen $G_{\pi,\sigma}$ der Parameter α (α_2) den Einfluss der Pheromonspuren

innerhalb des Teilgraphen $\mathcal{G}_\pi$ ($\mathcal{G}_\sigma$) angeben. Analog verhält es sich mit den Parametern β und β_2 für die heuristischen Informationen.

Änderung der Lösungskonstruktion

Entgegen den Darstellungen aus Kapitel 2.3 sind durch die in Kapitel 3.3 definierten Pheromonmodelle nicht mehr auf allen Kanten Pheromonspuren abgelegt. Sie sind nunmehr ausschließlich mit den Kanten zwischen Knotenkomponenten und von Kantenkomponenten zu Seitenkomponenten assoziiert. Die Übergangswahrscheinlichkeit für die Wahl der nächsten Komponente muss auf Basis dieser Veränderungen neu definiert werden. Wenn keine Pheromonspuren vorliegen, soll die Ameise zufällig gleichverteilt auswählen. Dabei ist darauf zu achten, dass die Formel auch nach den Veränderungen eine Wahrscheinlichkeitsverteilung darstellt, d.h. die Summe der Übergangswahrscheinlichkeiten aller zur Verfügung stehenden Alternativen muss stets dem Wert 1 entsprechen. Bedingt durch die Vernetzungsstrukturen wird eine Ameise von einer beliebigen Komponente ausgehend entweder für *alle* Kanten zu den Komponenten der zulässigen Nachbarschaft Pheromonspuren vorfinden oder für *keine*. Insofern beeinflussen sich die definierten Übergangswahrscheinlichen nicht und in jeder Situation wird eine Wahrscheinlichkeitsverteilung definiert.
Das globale Pheromonupdate kann zudem nicht mehr auf alle Kanten des Graphen angewendet werden, sondern nur noch auf denjenigen, zu denen Pheromonspuren gespeichert sind.

Beispielhaft seien die Veränderungen für das AS verdeutlicht. Die Änderungen für das MMAS und ACS sowie das Hypercube Framework ergeben sich dementsprechend.
Die Übergangswahrscheinlichkeit nach Formel 2.2 ändert sich zu Formel 3.8.

$$p_{ij}^k = \begin{cases} \frac{[\tau_{ij}]^\alpha * [\eta_{ij}]^\beta}{\sum_{u \in \mathcal{N}_i^k} [\tau_{iu}]^\alpha * [\eta_{iu}]^\beta} & \text{falls } j \in \mathcal{N}_i^k \ \wedge (i,j \in \mathcal{C}_V \ \vee \ i \in \mathcal{C}_E) \\ \frac{1}{|\mathcal{N}_i^k|} & \text{falls } j \in \mathcal{N}_i^k \ \wedge (i,j \notin \mathcal{C}_V \ \wedge \ i \notin \mathcal{C}_E) \\ 0 & \text{falls } j \notin \mathcal{N}_i^k \end{cases} \tag{3.8}$$

Wenn die Komponente i eine Kantenkomponente ist, sind Pheromonspuren für die nächste Entscheidung abgelegt. Ebenso verhält es sich, falls i und j Knotenkomponenten sind.
Das globale Pheromonupdate wird in gleicher Weise zu Formel 3.10 verändert.

$$\mathcal{CP} = \{(u,k) | (u,k) \in L \wedge (u,k \in \mathcal{C}_V \ \vee \ u \in \mathcal{C}_E)\} \tag{3.9}$$

$$\tau_{ij} \longleftarrow (1-\rho) * \tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij}^k \qquad \forall\,(i,j) \in \mathcal{CP} \tag{3.10}$$

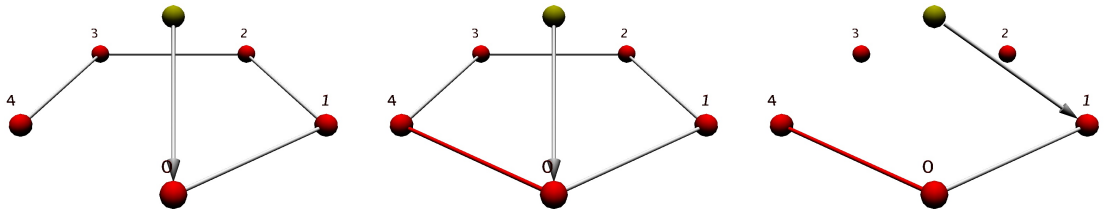
Der Konstruktionsgraph $\mathcal{G}_{\pi,\sigma}$ verwaltet zwei Pheromonmatrizen für die jeweiligen Teilgraphen. Beide Matrizen sind gemäß dieser neuen Formel zu verändern, wobei ausschließlich Knoten- bzw. Kantenkomponenten in einem Teilgraphen eingesetzt sind und auch nur die betroffenen Verbindungen berücksichtigt werden müssen.

Verstärkung der Verbindung zwischen erster und letzter Knotenkomponente

Falls die Ameise in ihrer Lösung eine Permutation codiert[13] ist zu beachten, dass sie jede Knotenkomponente genau einmal aufgenommen hat (Abb. 3.26(a)). Somit wurde die Kante

[13] das ist der Fall, wenn die Ameise einen Graphen G_π oder $G_{\pi,\sigma}$ durchwandert. In dieser Arbeit werden keine anderen Konstruktionsgraphen eingesetzt, so dass die Lösungen stets eine Permutation codieren.

zwischen der zuletzt und zuerst besuchten Knotenkomponente nicht gewählt (Abb. 3.26(b)). Der Pheromonwert dieser Verbindung wird während des Pheromonupdates nicht verstärkt. Da dieser aber aufgrund der Tatsache, dass die Permutation zyklisch[14] ist, mit der Lösung s^k assoziiert ist, muss zusätzlich noch die Verbindung zwischen der zuerst und zuletzt besuchten Knotenkomponente mit berücksichtigt werden. Beginnt eine Ameise die Lösungskonstruktion in der nächsten Iteration mit einer anderen Komponente und folgt dem Weg mit der höchsten Pheromonspur, so ist für die zuerst und zuletzt besuchte Komponente der vorigen Lösung keine Information zur Fortsetzung des Weges aktualisiert worden (Abb. 3.26(c)). Diese Verbindung ist also zusätzlich zu beachten.
Die Lösungen für das TSP sind ebenso zyklische Permutationen. In den Implementierungen von ACO Algorithmen auf das TSP wird von Dorigo und Stützle [DS04, S.67] erwähnt, dass eine Ameise nach Aufnahme der letzten Lösungskomponente die erste Komponente erneut ansteuert. Somit wäre die fehlende Verbindung ein Element der Lösung.
Ob die Verbindung nun Teil der Lösung selbst ist oder nachträglich mit eingerechnet wird, stellt für die Funktionalität der Algorithmen keinerlei Unterschied dar. Die Implementierungen in dieser Arbeit berücksichtigen diese Verbindung nun nebst dem Lösungsweg nachträglich im Pheromonupdate.



(a) Eine (beispielhafte) Lösung s^k einer Ameise

(b) Die Verbindung zwischen der zuerst (0) und zuletzt (4) besuchten Knotenkomponente ist nicht Teil der Lösung s^k

(c) In einer späteren Lösungskonstruktion würde die Verbindung zwischen Knotenkomponente 0 und 4 nicht verstärkt berücksichtigt werden

Abbildung 3.26: Verbindung zwischen der zuerst und zuletzt besuchten Knotenkomponente, beispielhaft für einen Konstruktionsgraphen $\mathcal{G}_\pi$ mit $\mid \mathcal{C}_V \mid = 5$

3.6.2 Ant System

Die zusätzlich einzustellenden Parameter für das AS sind

- der initiale Pheromonwert τ_0
- der Faktor Q zur Steuerung der abgelegten Pheromonmenge

Die Initialisierung der Pheromonwerte wird durch Setzen aller Pheromonspuren auf $\tau_{ij} = \tau_0$ realisiert.
Der Startzustand einer Ameise beinhaltet die Startkomponente des Konstruktionsgraphen. Die Suche nach einem Weg geschieht wie im Kapitel 2.3.4 beschrieben, wobei die Übergangswahrscheinlichkeiten aus Formel 3.10 Verwendung finden.

[14] siehe Kapitel 3.3.3.

Für das globale Pheromonupdate ist zu beachten, dass die Zielfunktion f den Wert 0 annehmen kann. Dadurch wäre die Formel 2.4 nicht definiert. Um diesen Umstand auszugleichen, wird der Zielfunktionswert für das Pheromonupdate um den Wert 1 inkrementiert, was die folgende veränderte Formel für die Ablage von Pheromonspuren auf Kanten ergibt:

$$\Delta\tau_{ij}^{k} = \begin{cases} \frac{Q}{\mathbf{1}+f(s^k)} & \text{falls } \begin{array}{l}(i,j) \text{ ist Element der Lösung } s^k \vee \\ \text{i,j sind zuerst und zuletzt besuchte Knotenkomp. in } s^k\end{array} \\ 0 & \text{sonst} \end{cases} \tag{3.11}$$

Wie bereits erwähnt, fließt zusätzlich zu der Lösung s^k die Verbindung zwischen zuerst und zuletzt besuchter Knotenkomponente in das Pheromonupdate mit ein.

3.6.3 Ant System mit Hypercube-Framework

Das HC-AS unterscheidet sich vom AS durch einen veränderten Pheromonupdate-Mechanismus. Der Parameter Q entfällt hierbei. Alle weiteren Veränderungen können gemäß den vorigen Ausführungen über das AS übernommen werden.
Zur Berechnung neu abgelegten Pheromonmenge ist insbesondere stets der Zielfunktionswert f mit dem Wert 1 zu addieren.

3.6.4 MAX-MIN Ant System

Für das MMAS sind die folgenden Parameter geeignet einzustellen:

- Auswahl zwischen s^{ib} oder s^{gb} für das Pheromonupdate
- p_{best}
- δ

Die Initialisierung der Pheromonwerte geschieht wie beschrieben durch Setzen τ_{ij} auf einen fiktiven hohen Wert.

Die Lösungskonstruktion ist unverändert, für das globale Pheromonupdate muss jedoch wieder die Formel für die abzulegende Pheromonmenge unter Berücksichtigung der Division durch 0 und der zusätzlichen Verbindung angepasst werden:

$$\Delta\tau_{ij}^{best} = \begin{cases} \frac{1}{\mathbf{1}+f(s^{best})} & \text{falls } \begin{array}{l}(i,j) \text{ ist Element der Lösung } s^{best} \vee \\ \text{i,j sind zuerst und zuletzt besuchte Knotenkomp. in } s^{best}\end{array} \\ 0 & \text{sonst} \end{cases} \tag{3.12}$$

Die für das Pheromonupdate herangezogene Lösung wird zu s^{ib} bestimmt, da sich hierdurch die Gefahr zur Konzentration auf lokale Optima verringert.

Berechnung der Pheromongrenzen

Zur Bestimmung der unteren Pheromongrenze τ_{min} ist die Kenntnis über die Anzahl der Entscheidungspunkte n und die durchschnittliche Anzahl von Entscheidungen an jedem Entscheidungspunkt, avg, erforderlich. Diese Werte sind von dem zugrundeliegenden Konstruktionsgraphen abhängig und müssen nun für jeden Konstruktionsgraphen speziell definiert werden. Berücksichtigt werden hierbei nur Entscheidungen, die für die Aufstellung der Buchzeichnung relevant sind, d.h. insbesondere ist die Wahl der nächsten zu belegenden Kante keine Entscheidung. Dieses spiegelt den Konvergenzzustand weit besser wieder, da alle nicht berücksichtigten Entscheidungen keinen Einfluss auf die Pheromonspuren haben und nicht zur Konvergenzmessung berücksichtigt werden sollten. Tabelle 3.2 zeigt die sich ergebenden Werte auf, wobei $\mathcal{C}_V$ die Menge der Knotenkomponenten und $\mathcal{C}_E$ die Menge der Kantenkomponenten des Konstruktionsgraphen bezeichnet.

	n	avg
$\mathbf{G}_\pi$	$\lvert\mathcal{C}_V\rvert$	$\frac{\lvert\mathcal{C}_V\rvert}{2}$
$\mathbf{G}_\sigma$	$\lvert\mathcal{C}_E\rvert$	k
$\mathbf{G}_{\pi,\sigma}$	$\lvert\mathcal{C}_V\rvert + \lvert\mathcal{C}_E\rvert$	$\frac{\frac{1}{2}\lvert\mathcal{C}_V\rvert^2 + \lvert\mathcal{C}_E\rvert * k}{n}$

Tabelle 3.2: Einstellung von n und avg bzgl. des jeweiligen Konstruktionsgraphen

Für den Konstruktionsgraphen G_π bestimmt sich n zu $|\mathcal{C}_V|$, da die Ameise während der Lösungskonstruktion jede Knotenkomponente genau einmal anläuft und dabei bestimmt wird, an welcher Position die repräsentierten Knoten in der Permutation auftreten. Durchschnittlich stehen ihr dazu an jedem Entscheidungspunkt $\frac{|\mathcal{C}_V|}{2}$ Komponenten zur Verfügung – zu Anfang kann sie aus n Komponenten wählen und am Ende nur noch eine einzige.
Im Graphen G_σ muss sich die Ameise $|\mathcal{C}_E|$-mal entscheiden, welcher Seite sie die repräsentierte Kante zuordnet. Zur Auswahl stehen ihr dabei stets alle k Seiten.
Für $G_{\pi,\sigma}$ bestimmt sich die Weglänge n zu der Summe aus den Weglängen beider Teilgraphen und avg berechnet sich zu

$$\begin{aligned} avg &= \frac{|\mathcal{C}_V| * \frac{|\mathcal{C}_V|}{2}}{|\mathcal{C}_V| + |\mathcal{C}_E|} + \frac{|\mathcal{C}_E| * k}{|\mathcal{C}_V| + |\mathcal{C}_E|} \\ &= \frac{\frac{1}{2}|\mathcal{C}_V|^2}{n} + \frac{|\mathcal{C}_E| * k}{n} \end{aligned} \tag{3.13}$$

In $|\mathcal{C}_V|$ (von $|\mathcal{C}_V| + |\mathcal{C}_E|$) Entscheidungen stehen durchschnittlich $\frac{|\mathcal{C}_V|}{2}$ Komponenten zur Verfügung und in $|\mathcal{C}_E|$ Entscheidungen sind dies k Komponenten.

Reinitialisierung der Pheromonspuren

Der PTS Mechanismus wird eingeleitet, sobald alle Pheromonspuren außer denjenigen, die zur besten Lösung korrespondieren, einen Wert von τ_{min} aufweisen. Das ist dann der Fall, wenn die Pheromonmatrix maximal n Einträge (siehe Tabelle 3.2) mit einem höheren Wert als τ_{min} beinhaltet. Auf dem Pfad der besten Lösung wird stets eine Pheromonspur abgelegt, so dass die betreffenden Verbindungen den Wert τ_{min} überschreiten. Für den Gesamtgraphen

$\mathcal{G}_{\pi,\sigma}$ müssen dazu die Pheromonmatrizen beider Teilgraphen zusammen betrachtet werden. Eine Konvergenzmessung auf Basis des durchschnittlichen λ-Verzweigungsfaktors[15] hat sich nicht als geeignet erwiesen. Eine Konvergenzsituation liegt vor, wenn dieser Faktor einen bestimmten Schwellenwert unterschreitet. Den gilt es geeignet zu bestimmen. Problematisch ist dieses insbesondere aufgrund der Gegebenheit, dass die Graphen $\mathcal{G}_{\pi,\sigma}$ und $\mathcal{G}_{\pi}$ unterschiedlich strukturiert sind. Somit ändert sich auch der minimale Wert des durchschnittlichen λ-Verzweigungsfaktors. Die Schwellenwerte müssen abhängig vom Konstruktionsgraphen bestimmt werden.
Die hier gewählte Konvergenzdetektierung ist auf jeden Konstruktionsgraphen ohne Anpassungen anwendbar und bringt keine zusätzlichen Parameter mit ein.

3.6.5 MAX-MIN Ant System mit Hypercube Framework

Das HC-MMAS unterscheidet sich vom MMAS darin, dass zur Berechnung der abgelegten Pheromonmenge (Formel 2.17) keine Qualität der generierten Lösung berücksichtigt wird. Das globale Pheromonupdate bedarf demnach keiner Anpassung.
Die weiteren Einstellungen und Ausführungen aus dem vorigen Unterkapitel über das MMAS werden unverändert übernommen.

3.6.6 Ant Colony System

Für das ACS sind folgende Parameter anzupassen:

- Auswahl zwischen s^{ib} und s^{gb} für das globale Pheromonupdate
- τ_0
- q_0
- ξ

Die initialen Pheromonwerte werden fest auf τ_0 gesetzt – äquivalent zum AS. Das globale Pheromonupdate für das ACS wird anhand der global besten Lösung bestimmt, also $s^{best} = s^{gb}$, da es sich in den Experimenten mit dem TSP als vorteilhaft erwiesen hat [vgl. DS04, S.77] und allgemein als bevorzugte Einstellung gilt.

Analog zu den vorigen Ausführungen wird die Menge des abzulegenden Pheromons neu definiert:

$$\Delta\tau_{ij}^{best} = \frac{1}{1 + f\left(s^{best}\right)} \tag{3.14}$$

Das lokale Pheromonupdate sowie die Lösungskonstruktion müssen bis auf den Umstand, dass die Ameise nach dem Besuch der letzten Knotenkomponente zusätzlich noch die Verbindung zwischen zuletzt und zuerst besuchter Knotenkomponente in das lokale Pheromonupdate mit einbezieht, nicht weiter angepasst werden. Auch für das globale Pheromonupdate ist diese Verbindung mit einzubeziehen.

[15] average λ-branching factor [SH96]

3.6.7 Auflistung der zu optimierenden Parameter

In diesem Kapitel wurden die einzustellenden Parameter aufgezeigt und entschieden, einige davon fest vorzugeben. Somit müssen diese nicht mehr optimiert werden.
Um die Ergebnisse dieses Kapitels zusammenzufassen, listet Tabelle 3.3 die noch nicht automatisiert berechneten Parameter für verschiedene Kombinationen aus ACO Algorithmen und Konstruktionsgraphen auf. Sie sind nun von einer Optimierungsmethode geeignet einzustellen. Kapitel 4.4 widmet sich dieser Thematik.
Die Parameterbedeutung kann in der Zusammenstellung aus Anhang A nachgelesen werden.

	$\mathbf{G}_\pi$ oder $\mathbf{G}_\sigma$	$\mathbf{G}_{\pi,\sigma}$
AS	$\alpha, \beta, \rho, \tau_0, Q$	$\alpha, \alpha_2, \beta, \beta_2, \rho, \tau_0, Q$
HC-AS	$\alpha, \beta, \rho, \tau_0$	$\alpha, \alpha_2, \beta, \beta_2, \rho, \tau_0$
MMAS/HC-MMAS	$\alpha, \beta, \rho, p_{best}, \delta$	$\alpha, \alpha_2, \beta, \beta_2, \rho, p_{best}, \delta$
ACS	$\beta, \rho, q_0, \xi, \tau_0$	$\beta, \beta_2, \rho, q_0, \xi, \tau_0$

Tabelle 3.3: Zusammenstellung der Algorithmus-Parameter spezieller ACO Algorithmen auf den jeweiligen Konstruktionsgraphen

4 Testergebnisse und Performance

Zunächst werden innerhalb dieses Kapitels die Graphenklassen bestimmt, auf denen die zuvor adaptierten ACO Algorithmen getestet werden. Danach folgt der Aufbau der empirischen Tests sowie eine Gegenüberstellung der von den Algorithmen benötigten Rechenzeiten. Da ACO Algorithmen über diverse einzustellende Parameter verfügen, befasst sich Kapitel 4.4 mit der Anwendung der SPO-Toolbox (SPOT)[1] zur Parameteroptimierung für die eingesetzten Algorithmen auf unterschiedlichen Probleminstanzen. In Punkt 4.5 wird im Anschluss daran untersucht, ob aus den optimerten Parametersätzen Empfehlungen abgeleitet werden können. Kapitel 4.6 trägt die mit optimierten Parametern erzielten Resultate auf den eingesetzten Graphenklassen zusammen.

4.1 Testgraphen

Zum Testen der Algorithmen werden drei Klassen von Graphen verwendet: Zirkuläre Graphen, Petersen Graphen, sowie Rome Graphen.
Zirkuläre Graphen wurden von Cimikowski [Cim02] und He [He06, S.108; HSM07] zum Vergleich von Algorithmen für das 2-seitige BCNP (bzw. FLCNP) verwendet. Dabei konnten mit genetischen Algorithmen die besten Resultate erreicht werden [siehe HSM07]. Die Ergebnisse sind für einige Probleminstanzen veröffentlicht, so dass die Leistung der ACO Algorithmen damit verglichen werden kann.
Die 2-seitige Buchkreuzungszahl stellt eine obere Schranke der planaren Kreuzungszahl dar (siehe Kapitel 2.1.4). Petersen Graphen werden eingesetzt, um die Abweichung zwischen der besten ermittelten 2-seitigen Buchkreuzungszahl und der planaren Kreuzungszahl betrachten zu können. Für einige dieser Graphen ist die planare Kreuzungszahl bekannt.
Rome Graphen wurden von He zum Performancevergleich genetischer Algorithmen verwendet [siehe HSM07]. Konkrete Ergebnisse für einzelne Probleminstanzen sind jedoch nicht veröffentlicht.

Halin Graphen und planare Rome Graphen haben sich für einen Performancevergleich als ungeeignet erwiesen, da sie in der zur Verfügung stehenden Größe nicht in der Lage sind, die Algorithmen entsprechend zu fordern. In Kapitel 4.6.1 sind die erzielten Resultate beispielhaft für jeweils einen Graphen dieser Klassen aufgeführt.

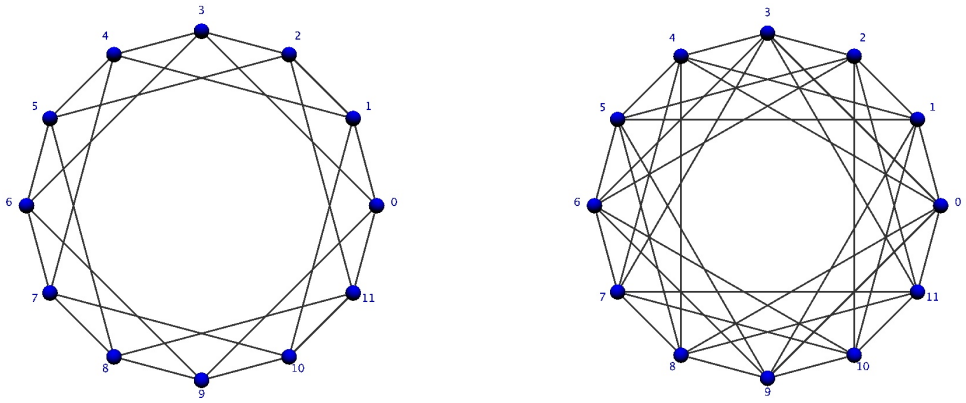
Weitere Graphenklassen finden in dieser Arbeit keine Betrachtung. Die ACO Algorithmen werden somit ausschließlich auf zirkulären Graphen mit den Ergebnissen anderer Verfahren verglichen. Es sind allgemein nur wenig Ergebnisse für konkrete Probleminstanzen mit $k = 2$ veröffentlicht. Die wichtigsten Dokumente in diesem Bereich stellen ein Artikel von Cimikowski [Cim02] und die Dissertation von He [He06] dar. Cimikowski betrachtet eine feste lineare Ordnung der Knoten, so dass die ACO Algorithmen diesbezüglich bereits einen Vorteil aufweisen. He konnte zeigen, dass eine feste Permutation für das 2-seitige BCNP zu restriktiv ist

[1] [BBLP06]

und genetische Algorithmen bessere Resultate hervorbrachten [vgl. HSM07; He06, S.132].
Strategien zur Lösung des 2-seitigen BCNP basieren vielfach darauf, dass zunächst eine Permutation für das 1-seitige BCNP bestimmt wird. Auf Grundlage dieser Permutation werden Heuristiken zur Generierung einer Seitenzuordnung angewendet. Ergebnisse dazu finden sich in [He06, S.213ff]. Die hier eingesetzten Optimierungsvarianten mit ACO Algorithmen sind allgemeiner gehalten und prinzipiell im Vorteil.
Für Ergebnisse auf Probleminstanzen mit $k > 2$ konnte keine Literatur gefunden werden.

4.1.1 Zirkuläre Graphen

Ein zirkulärer Graph[2] $\mathcal{C}_n(a_1, \ldots, a_k)$, $0 < a_1 < \cdots < a_k < \frac{n+1}{2}$, ist ein Graph mit n Knoten. Der i-te Knoten ist mit den $i \pm a_1, \ldots, i \pm a_k \, (mod\, n)$-ten Knoten durch eine Kante verbunden. Die Anzahl der Kanten eines zirkulären Graphen berechnet sich zu $|E| = n * k$.
Abbildung 4.1 liefert zwei Beispiele für zirkuläre Graphen.



(a) Der Graph $\mathcal{C}_{12}(1,3)$ (b) Der Graph $\mathcal{C}_{12}(1,3,4)$

Abbildung 4.1: Beispiele für zirkuläre Graphen

4.1.2 Petersen Graphen

Der *generalisierte Petersen Graph*[3] $P(m, \ell)$ mit $\ell \in \mathbb{Z}^+$ besteht aus $2 * m$ Knoten und wird auf folgende Weise bestimmt:
$C = (v_0, v_1, \ldots, v_{m-1})$ ist ein Kreis mit m Knoten. Die verbleibenden m Knoten sind mit $v'_0, v'_1, \ldots, v'_{m-1}$ bezeichnet. Jeder Knoten v_i wird mit v'_i durch eine Kante verbunden. Zusätzlich werden die Kanten zwischen v'_i und $v'_{i+\ell}$ eingefügt, wobei die Indizes modulo m zu berechnen sind.
Ein Beispiel für einen Petersen Graphen ist in Abb. 4.2 visualisiert.

Für Petersen Graphen mit $\ell = 3$ ist die planare Kreuzungszahl bekannt [RS02]:

$$cr(P(3k,3)) = k \quad \text{für } k \geq 4 \tag{4.1}$$
$$cr(P(3k+1,3)) = k+3 \quad \text{für } k \geq 3 \tag{4.2}$$
$$cr(P(3k+2,3)) = k+2 \quad \text{für } k \geq 3 \tag{4.3}$$

[2] *circulant* graphs [BT84]
[3] [RS02]

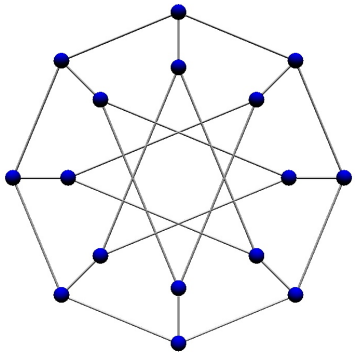


Abbildung 4.2: Eine Zeichnung des Petersen Graphen P(8,3)

Im Folgenden finden die Petersen Graphen $P(n, 3)$ für die weiteren Tests Verwendung. Hintergrund dieser Auswahl ist die Möglichkeit, die Abweichung zwischen der besten 2-seitigen Buchkreuzungszahl und der planaren Kreuzungszahl des Graphen betrachten zu können.

4.1.3 Rome Graphen

Die Rome Graphen sind dem *Graph Drawing Toolkit (GDToolkit)*[4] entnommen. Verwendet wird die Bibliothek *ALF/CU*, die auch zum Performancevergleich genetischer Algorithmen für das 2-seitige BCNP eingesetzt wurde [siehe HSM07]. Es sind über 10000 ungerichtete Graphen mit 10 bis 100 Knoten enthalten. Die Graphen tragen die Benennung „ug[a].[b]“, wobei [a] eine fortlaufende Nummer und [b] die Knotenanzahl des Graphen darstellt. Diese Bezeichnung wird im Folgenden zur Identifikation der Graphen übernommen.
Rome Graphen weisen allgemein einen geringen Durchschnittsgrad zwischen ca. $2,0$ und $3,0$ auf.

4.2 Testaufbau

4.2.1 Bestimmung der Iterationsanzahl n_{max}

Ein ACO Algorithmus terminiert, wenn die maximale Iterationsanzahl n_{max} oder eine Lösung mit Buchkreuzungszahl 0 erreicht wurde (vgl. Kapitel 3.6). Diese maximale Iterationsanzahl wird nun auf die zu optimierenden Graphenklassen sowie die Größe des jeweiligen Graphen abgestimmt, damit zum einen hinsichtlich einer angemessenen Rechenzeit (siehe auch Kapitel 4.3) nicht zu lange gerechnet wird. Zum anderen zeigen die ACO Algorithmen auf unterschiedlichen Graphenklassen ein anderes Laufverhalten. Daher ist eine Abstimmung der maximalen Iterationsanzahl auf die Graphenklasse sinnvoll.
Die Festlegung erfolgte zunächst für die ersten getesteten Probleminstanzen. Um die Ergebnisse dieser Läufe nutzen zu können und die Iterationsanzahl weiterer Probleminstanzen darauf abzustimmen, wurde ein lineares Gleichungssystem aufgestellt. Hierzu ist die Iterationszahl

[4] http://www.dia.uniroma3.it/~gdt/

für jeweils zwei unterschiedliche Probleminstanzen vorgegeben worden. Die Lösung des Gleichungssystems ergab die in Tabelle 4.1 dargestellten Formeln. Diese geben nun die Berechnung der Iterationsanzahl für beliebige Probleminstanzen an.

Graphenklasse	n_{max}
Zirkulär	$10* \mid V \mid +75 * \frac{\mid E \mid}{\mid V \mid}$
Petersen	$4* \mid V \mid +27 * \frac{\mid E \mid}{\mid V \mid}$
Rome	$4.5* \mid V \mid +75 * \frac{\mid E \mid}{\mid V \mid}$

Tabelle 4.1: Berechnung von n_{max} in Abhängigkeit von der Graphenklasse und der Größe eines Instanzgraphen $\mathcal{G}_{BCNP} = (V, E)$.

4.2.2 Testszenarien

Die Auswahl der zu testenden ACO Algorithmen wird auf das AS, ACS und MMAS beschränkt. Das HC-AS und HC-MMAS sind zwar implementiert und prinzipiell einsetzbar, finden in den Tests hingegen aus Zeitaspekten und in Anbetracht der steigenden Datenmengen keine Berücksichtigung. Zunächst sind die Standard-Algorithmen interessant zu vergleichen, das Hypercube Framework wird als Variante vorerst nicht betrachtet.
Insgesamt bestehen damit neun Optimierungsvarianten: Drei ACO Algorithmen, die jeweils in Kombination mit der randomisierten Greedy- bzw. $SLOPE$-Heuristik oder aber direkt auf dem Gesamtgraphen $\mathcal{G}_{\pi,\sigma}$ Lösungen konstruieren.

Die randomisierte Greedy-Heuristik kann bei mehreren Durchläufen Lösungen unterschiedlicher Qualität hervorbringen. Wie erwähnt ist es sinnvoll, diese zur Generierung einer Seitenzuordnung für eine Knotenpermutation π mehrfach zu starten und das beste erzielte Ergebnis zu betrachten. Es mag vorteilhaft sein, die Aufrufzahl in Abhängigkeit von der Kantenanzahl des Graphen zu bestimmen, so dass mit steigender Kantenanzahl mehr zufällige Reihenfolgen evaluiert werden und die Chance zur Generierung besserer Lösungen gegeben ist. Da die Rechenzeit der Heuristik jedoch bereits quadratisch mit der Kantenanzahl ansteigt und ein entscheidender Faktor ist, wurde eine stets fixe Aufrufanzahl vom Wert drei festgelegt. Dadurch ist die Rechenzeit dieser Variante im Vergleich zum Einsatz der $SLOPE$-Heuristik höher.

Auswahl der Probleminstanzen

Als Einschränkung berücksichtigt der weitere Verlauf dieser Arbeit nur die 2-seitige Buchkreuzungszahl, also $k = 2$. Je größer die Seitenanzahl k, desto geringer ist die optimale Buchkreuzungszahl eines Graphen. Zum Vergleich der Algorithmen müssten für höhere Werte von k ebenso größere Graphen eingesetzt werden. Ansonsten würden sie nicht entsprechend gefordert und könnten nicht verglichen werden. Allerdings erhöht sich auf größeren Graphen auch die benötigte Rechenzeit. Die Größe der Probleminstanzen muss jedoch diesbezüglich beschränkt werden, um die Parameteroptimierung in akzeptabler Zeit realisieren zu können. Diese ruft die Algorithmen mehrfach auf. Ergänzend findet sich in Kapitel 4.6.5 eine exemplarische Untersuchung für $k = 3$ und $k = 4$ bezüglich einer speziellen Probleminstanz, die

bei einer Erhöhung der Seitenanzahl stets noch ausreichend schwierig zu optimieren ist.
Für $k = 1$ ist ausschließlich die Optimierung einer Permutation erforderlich und keine Verfahren zur Generierung einer Seitenzuordnung einzusetzen, so dass dieser Fall ebenso keine Berücksichtigung findet.

Für jede Graphenklasse wurden ca. zehn Graphen unterschiedlicher Größe ausgewählt, deren 2-seitige Buchkreuzungszahl im weiteren Verlauf von den Algorithmen optimiert wird. Bei der Auswahl sind insbesondere sehr kleine Probleminstanzen außer Acht gelassen, für die alle Algorithmen nahezu identische Ergebnisse liefern und deren Optimierung somit nicht schwierig genug ist. Die größte Probleminstanz wird so bestimmt, dass die Parameteroptimierung eine akzeptable Zeit beansprucht. „Akzeptabel“ sei dabei eine Rechenzeit von ca. sechs Stunden für die aufwendigste Optimierungsvariante. Die zur Verfügung stehende Testplattform (siehe auch Kapitel 4.3.1) ist auf Jobs mit einer maximalen CPU Zeit von sechs Stunden begrenzt. Längere Rechenzeiten sind zwar prinzipiell möglich, müssen jedoch in einem separaten Scheduling-Verfahren mit eingeschränktem Parallelisierungsgrad durchgeführt werden.

Zirkuläre Graphen sind bezüglich des Durchschnittsgrades anpassbar. Die Auswahl orientiert sich hierfür an den von Cimikowski [Cim02] eingesetzten zirkulären Graphen, um die von ACO Algorithmen erzielten Ergebnisse damit vergleichen zu können. Dabei wurden Graphen mit unterschiedlicher Knotenanzahl und Durchschnittsgrad selektiert. Petersen und Rome Graphen haben im Vergleich zu den gewählten zirkulären Graphen einen geringeren Durchschnittsgrad. Ebenso wurde für diese Graphen eine kleinere Iterationsanzahl gewählt, so dass insgesamt zur Gewährleistung einer akzeptablen Rechenzeit die maximale Knotenanzahl höher sein darf.

Die so ausgewählten Graphen sind in Anhang D nebst optimierten Parametereinstellungen aufgelistet. Im Speziellen sind dies Zirkuläre Graphen bis zu 40 Knoten, Petersen Graphen bis zu 88 und Rome Graphen bis zu 80 Knoten.

4.3 Rechenaufwand und Zeitvergleich

Die Größe der Konstruktionsgraphen ist von dem jeweiligen Instanzgraphen abhängig. Diesbezüglich nimmt die Lösungskonstruktion der ACO Algorithmen auf verschiedenen Probleminstanzen eine unterschiedliche Rechenzeit in Anspruch. Zur Verwirklichung einer Entscheidung müssen viele rechenintensive Operationen (insbesondere die Potenz) durchgeführt werden. Bei größeren Konstruktionsgraphen ist die durchschnittliche Größe der zulässigen Nachbarschaft höher, so dass die Entscheidungsfindung einen höheren zeitlichen Aufwand beansprucht.

Die Berechnung heuristischer Informationen ist trotz der eingebrachten Beschleunigungen ein einflussreicher Faktor in der Gesamtperformance. Das gilt sowohl für die Evaluierung dynamischer heuristischer Informationen während der Lösungskonstruktion als auch für den Aufruf der randomisierten Greedy-Heuristik zur separaten Generierung einer Seitenzuordnung. Diese wird darüber hinaus mehrfach gestartet.

Kandidatenlisten
Das Problem möglicher Ineffizienz von ACO Algorithmen auf großen Probleminstanzen ist bekannt und führte zur Einführung von *Kandidatenlisten*[5]. Diese nehmen für jede Komponente eine Auswahl „vielversprechender“ als nächstes zu besuchender Komponenten auf. Während

[5] [DS04, S.217]

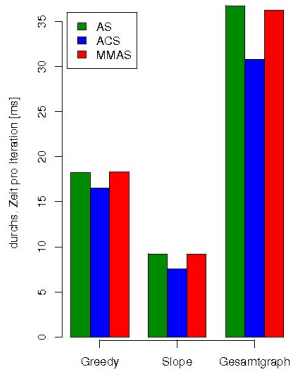
der Lösungskonstruktion wählen die Ameisen zunächst aus den Elementen der Kandidatenliste einer Komponente und betrachten keine weiteren Alternativen. Ein Element, das nicht in der Kandidatenliste einer Komponente ist, wird erst gewählt, nachdem alle Elemente in der Kandidatenliste besucht wurden [siehe auch SD99]. Somit wird die Anzahl der zur Verfügung stehenden Entscheidungsalternativen beschränkt und ermöglicht eine effizientere Lösungskonstruktion. Dabei gilt es zu bedenken, dass ebenso die Erkundung des Suchraumes eingeschränkt wird. Daher ist bei der Implementierung von Kandidatenlisten besondere Vorsicht geboten, zumal diese für einen generellen Einsatz bislang wenig erforscht sind [vgl. DS04, S.218]. In dieser Arbeit wird aus diesem Grund auf ihren Einsatz verzichtet.

4.3.1 Zeitvergleich

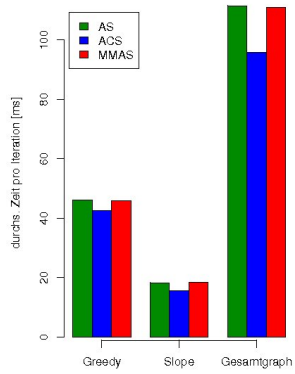
Der Lehrstuhl stellte eine Testplattform mit sieben Rechnern unterschiedlicher Ausstattung und Rechenleistung zur Verfügung. Anhand eines Schedulingverfahrens konnte ein Experiment auf jedem dieser Rechner ausgeführt werden, die zudem noch weitere nicht vorhersehbare Prozesse bearbeiteten. Die gespeicherten Laufzeiten der Algorithmen können aus diesem Grund nicht zur weiteren Auswertung berücksichtigt werden. Sie sind von der aktuellen Auslastung abhängig. Insgesamt belief sich die Rechenzeit für die Parameteroptimierung und die anschließenden Experimente auf über einen Monat. Die Verfahren können somit als rechenaufwendig betrachtet werden.

Für einen Zeitvergleich wurde jeder Algorithmus auf einem separaten Rechner mit Athlon 1000 GHZ Prozessor und Ubuntu 6.06 Betriebssystem gestartet. Die ermittelte Gesamtzeit ist durch die Anzahl der Iterationen dividiert worden, um die durchschnittliche Zeit pro Iteration zu berechnen.
Abbildung 4.3 stellt die Ergebnisse der Zeitmessung zusammen.

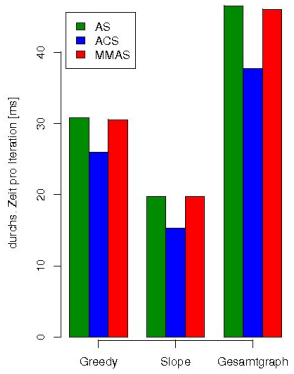
Es zeigt sich, dass die relative Laufzeitdifferenz der Optimierungsvarianten von dem Durchschnittsgrad des Graphen abhängt. Je höher der Durchschnittsgrad, desto stärker wächst der Rechenzeitaufwand in Kombination mit der randomisierten Greedy-Heuristik im Vergleich zur *SLOPE*-Heuristik. Das liegt an der höheren algorithmischen Komplexität sowie der Eigenschaft, dass die Greedy-Heuristik für jede Permutation dreimal gestartet wird.
Auch ist ersichtlich, dass der Rechenaufwand mit steigender Graphengröße insgesamt stark anwächst. Eine Erklärung dafür liefert die höhere Ameisenanzahl und die größeren Konstruktionsgraphen, so dass in der Lösungskonstruktion mehr Entscheidungsalternativen zu berücksichtigen sind. Die Heuristiken müssen zudem mehr Kanten betrachten. Da die Ameisenanzahl für den Gesamtgraphen $\mathcal{G}_{\pi,\sigma}$ zusätzlich mit dem Durchschnittsgrad ansteigt, erhöht sie sich bei steigendem Durchschnittsgrad und somit auch die relative Zeitdifferenz zu den anderen beiden Optimierungsvarianten.
Auffällig ist überdies, dass die ACS-Algorithmen im Vergleich zu den anderen ACO-Algorithmen immer eine geringere Laufzeit erfordern. Dieser Umstand lässt sich anhand der fehlenden Notwendigkeit zur Potenzierung der Pheromonspuren mit dem Parameter α für jede Entscheidungsalternative begründen. Hiermit wird eine kostspielige und zahlreich aufzurufende Operation eingespart.



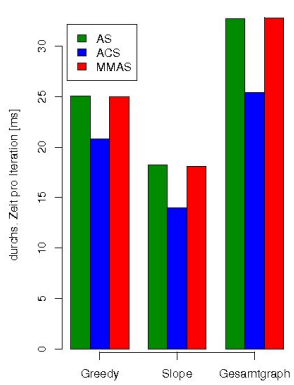
(a) $\mathcal{G} = \mathcal{C}_{20}(1,2,3),\ d(\mathcal{G}) = 6$



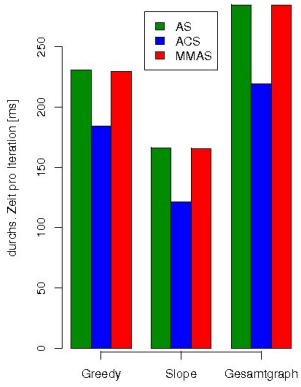
(b) $\mathcal{G} = \mathcal{C}_{24}(1,3,5,7),\ d(\mathcal{G}) = 8$



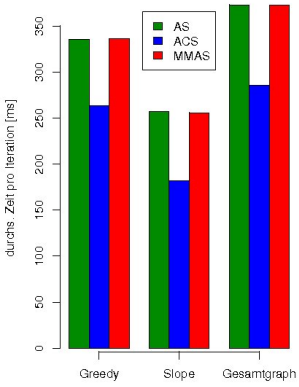
(c) $\mathcal{G} = \mathcal{C}_{28}(1,3),\ d(\mathcal{G}) = 4$



(d) $\mathcal{G} = P(14,3),\ d(\mathcal{G}) = 3$



(e) $\mathcal{G} = P(30,3),\ d(\mathcal{G}) = 3$



(f) $\mathcal{G} = ug63.70,\ |V| = 70 \wedge |E| = 89,\ d(\mathcal{G}) = 2.54$

Abbildung 4.3: Durchschnittliche Rechenzeit pro Iteration für ausgewählte Graphen und k = 2

4.3.2 Aufschlüsselung der Rechenzeit

Die Rechenzeit setzt sich maßgeblich aus der Lösungskonstruktion, dem Zeitaufwand der Heuristiken, sowie der Auswertung der Kostenfunktion zusammen. Innerhalb der Lösungskonstruktion müssen von einer Ameise mehrere Entscheidungen verwirklicht werden, deren Zeitaufwand einzeln kaum messbar ist; das Messverfahren löst nur in Millisekunden auf. Die einzelnen Schritte werden jedoch zahlreich aufgerufen, so dass insgesamt ein hoher Rechenaufwand entsteht. Messbar ist z.B. erst die Ausführung der gesamten Lösungskonstruktion. Damit kann insbesondere der Rechenaufwand für den Gesamtgraphen nicht aufgeschlüsselt werden, da hier Heuristik und Lösungskonstruktion nicht getrennt angewendet werden.
Möglich ist jedoch eine Aufschlüsselung für die hybriden Optimierungsvarianten. Dazu kann die Zeit der Lösungskonstruktion einer Ameise und die Zeit für die Anwendung der Heuristik zusammen mit der Kostenfunktionsauswertung gemessen werden. Insgesamt ergibt sich

der hohe Rechenaufwand, da mehrere Ameisen pro Iteration eine Lösung konstruieren und auswerten, die Algorithmen einige Iterationen bis zur Terminierung ausführen und zudem mehrfach gestartet werden (z.B. von der SPOT).

Die Laufzeit der Heuristiken und die Auswertung der Kostenfunktion ist von der Kantenanzahl abhängig, wohingegen die Zeit zur Lösungskonstruktion von der Knotenanzahl des Instanzgraphen bestimmt wird. Davon abhängig verschiebt sich der benötigte Zeitanteil. Tabelle 4.2 listet den Zeitanteil von Lösungskonstruktion (ACO Algorithmus) und der Heuristik samt Auswertung der Kostenfunktion für drei der in Kapitel 4.3.1 untersuchten Probleminstanzen auf. Für die randomisierte Greedy-Heuristik sind dabei die drei Multistarts eingeschlossen. Die verbleibenden Prozente werden von weiteren Operationen der Verfahren in Anspruch genommen und sind vernachlässigbar.
Detailliertere Erklärungen für diese Daten wurden bereits in Kapitel 4.3.1 angeführt.

	+GR						+SL					
	AS		**MMAS**		**ACS**		**AS**		**MMAS**		**ACS**	
	L	H	L	H	L	H	L	H	L	H	L	H
$\mathcal{C}_{24}(1,3,5,7)$	17	81	17	81	12	86	44	53	45	52	34	62
$P(30,3)$	58	40	57	40	47	50	79	17	79	17	72	22
$ug63.70$	64	33	64	33	53	44	83	14	83	14	76	20

Tabelle 4.2: Anteil an der Rechenzeit in % für die Lösungskonstruktion (L) und Heuristik inkl. Funktionsauswertung (H) für die hybriden Optimierungsansätze

4.4 Parameteroptimierung

Parametersätze für ACO Algorithmen werden üblicherweise von Hand eingestellt. Allgemein existieren bereits Parameterempfehlungen für die Anwendung von ACO Algorithmen auf spezifische Probleme, wie beispielsweise das TSP [DS04, S.71]. In einer Anwendung auf neue Probleme mag eine Abstimmung erforderlich sein. Sollten die Parameter nicht gut eingestellt sein, lassen sich nahezu beliebig schlechte Ergebnisse provozieren.
Aus diesem Grund wird die von Bartz-Beielstein, Lasarczyk und Preuß entwickelte *SPO Toolbox (SPOT)*[6] eingesetzt. Sie basiert auf der sequentiellen Parameteroptimierung (SPO)[7], die im Buch von Bartz-Beielstein [BB06, S.126ff] im Detail beschrieben wird. Die SPOT ist in Matlab implementiert und ermöglicht eine einfache Anwendung auf neue Algorithmen. Die Kommunikation basiert auf dem Austausch von ASCII Textdateien.

Die Parameter sind getrennt für jede Optimierungsvariante zu betrachten. In den hybriden Varianten generieren die ACO Algorithmen ausschließlich eine Permutation. Die Heuristiken erfordern keine Parameter. Da sich die Heuristiken jedoch unterschiedlich verhalten, ist es im Sinne der Generierung von guten Kombinationen aus Permutation und Seitenzuordnung angebracht, die Parameter für jede Optimierungsvariante separat zu optimieren.

[6] Sequential Parameter Optimization Toolbox [BBLP06]
[7] Sequential Parameter Optimization

4.4.1 Anwendung der SPOT

Die ACO Algorithmen sind in der Programmiersprache *Java* entsprechend den Ausführungen in Kapitel 3.6 implementiert worden (siehe auch Anhang C). Zur Parameteroptimierung werden sie von der SPOT mit verschiedenen Parametersätzen (so genannten *Designpunkten*) aufgerufen. Die implementierten Algorithmen müssen in der Lage sein, die Designpunkte aus einer Designdatei auszulesen und die Ergebnisse in einer ebensolchen Ergebnisdatei zu speichern. Die SPOT wertet die Ergebnisdatei anhand eines internen Modells aus und zieht sie zur Generierung neuer Designpunkte heran, mit denen ein Algorithmus erneut aufgerufen wird. Die besten[8] Designpunkte sind dabei in chronologischer Reihenfolge in einer Datei abgespeichert. Mit dem zuletzt eingetragenen Designpunkt wurden insgesamt die besten Ergebnisse erzielt. Dieser gilt als optimierter Parametersatz.

Zur Determinierung der Designpunkte des ersten Durchlaufs verwendet die SPOT das *Latin Hypercube Sampling (LHS)*[9]. Dieses arbeitet folgendermaßen:
Sei p die Anzahl der LHS Designpunkte. Der zu untersuchende Bereich jedes Parameters wird in p Intervalle gleicher Größe untergliedert und die Designpunkte getreu folgendem Schema bestimmt:

- Jeder Designpunkt wird zufällig innerhalb eines Intervalls platziert
- In allen eindimensionalen Projektionen befindet sich in jedem Intervall genau ein Designpunkt

Abbildung 4.4 verdeutlicht dieses Schema beispielhaft für zwei Parameter und vier Designpunkte.

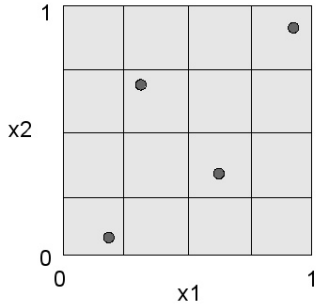


Abbildung 4.4: Beispiel eines Latin Hypercube Sampling für zwei Parameter x_1 und x_2 im Wertebereich von $[0, 1]$ und vier Designpunkten [vgl. GWE03]. Die vier generierten Designpunkte sind zufällig innerhalb jedes Intervalls platziert.

Die Größe des Initialdesigns legt fest, wieviele Designpunkte im ersten Durchlauf von der SPOT ausgewertet werden. Die SPOT nutzt zur Vorhersage von vielversprechenden Designpunkten in weiteren Schritten ein internes Modell, das mit den Daten der Ergebnisdatei initialisiert wird. Damit dieses angewendet werden kann, ist eine bestimmte minimale Größe des Initialdesigns erforderlich. Diese ist von der Parameteranzahl abhängig.

[8] bezogen auf das arithmetische Mittel aus den Resultaten aller mit diesem Designpunkt gestarteten Läufe.
[9] [GWE03]

Für die Optimierung auf dem Konstruktionsgraphen G_π ist das Initialdesign zu 35 Designpunkten bestimmt worden. Dieser Wert hat sich zur Anwendung des internen Modells als ausreichend herausgestellt. Die Optimierungsläufe auf dem Graphen $G_{\pi,\sigma}$ erfordern mehr Parameter (α_2, β_2), so dass dafür ein größeres Initialdesign zu realisieren ist. Hier wurden 50 Designpunkte vorgegeben.

Grundsätzlich steigert sich die Güte der optimierten Parametersätze mit längerer Optimierungsdauer. Den größten Performancegewinn erhält man allerdings bereits nach den ersten Durchläufen. Um mit Blick auf die Zeit nicht zu viele Läufe durchzuführen, lässt sich die Anzahl der Algorithmenaufrufe durch den Parameter *budget* begrenzen. Dieser wurde für die hybriden Optimierungsvarianten auf den Wert 210 und für die Optimierung mit dem Gesamtgraphen $G_{\pi,\sigma}$ auf 250 gesetzt, d.h. zur Bestimmung von optimalen Parametern wird jeder Algorithmus insgesamt maximal 210 bzw. 250 mal aufgerufen. Im Verhältnis zum Rechenaufwand hat sich dieses als angemessener Kompromiss herausgestellt.

Die SPOT erfordert die Vorgabe der zu untersuchenden Intervallgrenzen[10], innerhalb derer Designpunkte generiert werden. Diese werden im nächsten Abschnitt bestimmt.

Parameterintervalle

Eine Orientierung für die Bestimmung der Parameterintervalle können bereits bewährte Parametereinstellungen sowie die von Botee und Bonabeau [BB98] zur Optimierung des ACS festgelegten Intervalle bieten. Die Aufstellung der folgenden Tabelle nutzt diese Daten als Grundlage.
Die heuristischen Informationen sind neu eingeführt worden, von daher existieren bislang keine Anhaltspunkte zu einer angemessenen Gewichtung. Generell hat sich aber der Wert 20 als geeignete obere Grenze erwiesen.
Für das ACS muss der Wert von τ_0 immer kleiner als der Kehrwert der besten Lösungsqualität sein, da das lokale Pheromonupdate ansonsten keine Wirkung hervorrufen würde (vgl. Kapitel 2.3.7). Um die Parameteroptimierung von vornherein in diese Richtung zu lenken, wurde die obere Grenze für τ_0 in Verbindung mit dem ACS auf $0,01$ gesetzt. Dies hat sich auch für Probleminstanzen mit Optimum nahe bei 0 nicht als nachteilig herausgestellt.

Parameter	Wertebereich	Min	Max
α /α_2	$\mathbb{R}_+$	1.0	5.0
β /β_2	$\mathbb{R}_+$	1.0	20.0
ρ	$\mathbb{R}_+$	0.01	0.5
ξ	$\mathbb{R}_+$	0.01	0.5
τ_0 *(AS)*	$\mathbb{R}_+$	0.0	1.0
τ_0 *(ACS)*	$\mathbb{R}_+$	0.0	0.01
Q	$\mathbb{N}_+$	1	100
q_0	$\mathbb{R}_+$	0.0	1.0
p_{best}	$\mathbb{R}_+$	0.0	0.5
δ	$\mathbb{R}_+$	0.0	1.0

Tabelle 4.3: Intervallgrenzen

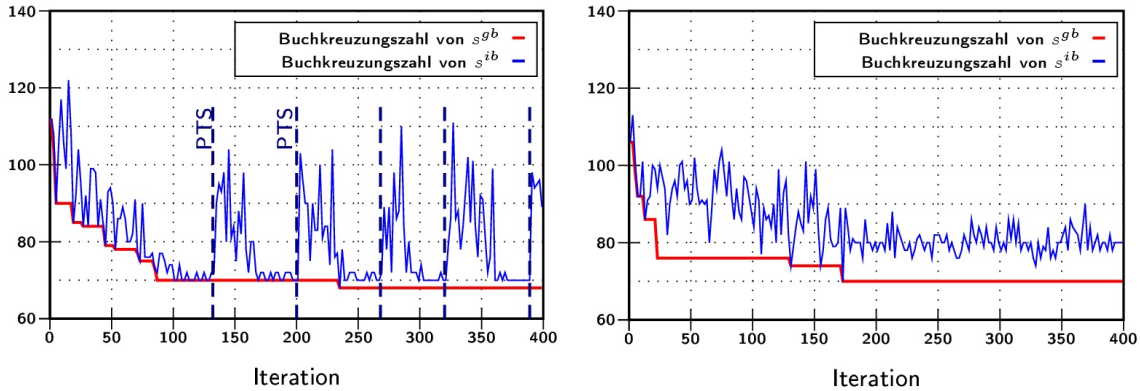
[10] auch mit *region of interest* bezeichnet.

Die Parameternamen müssen mit den zugehörigen Intervallgrenzen und Wertebereich in einer Datei entsprechend der von der SPOT geforderten Notation gespeichert werden. Die SPOT liest die Einstellungen vor der Optimierung ein und zieht diese zur Generierung der Designpunkte heran.

4.4.2 Vergleich der MMAS Varianten hinsichtlich des PTS

Die Beschränkung der maximalen Iterationsanzahl mag die Frage nach dem Nutzen des PTS aufwerfen. Ist n_{max} zu gering, so könnte das PTS nicht eingeleitet werden und der Parameter δ kann beliebig gewählt werden. Darüber hinaus ist das Einleiten des PTS vom jeweiligen Lauf eines MMAS-Algorithmus abhängig. Die Parametereinstellungen könnten weiterhin von der SPOT so angepasst worden sein, dass nicht genügend Pheromonspuren vor Erreichen der maximalen Iterationsanzahl auf einen Wert von τ_{min} gefallen sind und somit keine Konvergenz detektiert wurde. Hierauf besitzen z.B. die Parameter ρ und α entscheidenden Einfluss.

Abbildung 4.5(a) zeigt einen Lauf des MMAS mit Nutzung des PTS auf, dargestellt bis Iteration 400. Der Algorithmus optimiert dabei ausschließlich eine Permutation. Durch Einleiten des PTS wurde das Verlassen eines lokalen Optimums ermöglicht. Es zeigt sich, dass die beste Lösung der jeweiligen Iteration aufgrund der Pheromon-Reinitialisierung zunächst eine enorm schlechtere Qualität aufweist, sich dann aber wieder auf einem lokalen Optimum versteift. Durch den hohen Wert von p_{best} weisen τ_{min} und τ_{max} eine große Differenz auf. Der Suchraum wird dann stark eingeschränkt.
Ist p_{best} hingegen gering, muss das PTS nicht zwangsläufig eingeleitet werden. Die minimale Pheromonkonzentration liegt nun sehr nahe bei τ_{max} und somit wird stets ein entsprechend großer Suchraum erkundet. Durch die Wahl von $\delta = 0$ kann der Mechanismus deaktiviert werden (Abb. 4.5(b)). Hier fällt auf, dass sich die jeweilige Lösung s^{ib} nicht so intensiv auf ein lokales Optimum konzentriert wie in der vorigen Variante. Der Suchraum wird stets stark erkundet, auch in fortschreitenden Iterationen.



(a) $p_{best} = 0.5$, $\delta = 0.3$ (b) $p_{best} = 0.00005$, $\delta = 0$

Abbildung 4.5: Zwei beispielhafte Läufe der hybriden Optimierung aus MMAS und der randomisierten Greedy-Heuristik für $\mathcal{C}_{20}(1, 2, 3, 4)$, $k = 2$.

Gegebenenfalls mag der Einsatz des PTS nicht nutzbringend sein. Diesbezüglich wurden die Parameter der MMAS Algorithmen zweimal optimiert - einmal mit und einmal ohne PTS

Mechanismus. Letzteres berücksichtigt keinen Parameter δ; dieser ist statisch auf 0 gesetzt.

Die Ergebnisse sind in Tabelle 4.4 zusammengefasst. Die Spalten mit „+“-Symbol stehen für ein aktiviertes PTS, die mit „-“ gekennzeichneten Spalten für die Deaktivierung desselbigen. Die Datenwerte repräsentieren den Mittelwert der erzielten 2-seitigen Buchkreuzungszahl über alle mit dem optimierten Designpunkt gestarteten Algorithmenläufe.
Die Differenzen sind nicht maßgeblich ausschlaggebend und ein deutlicher Vorteil des PTS-Einsatzes ist nicht zu erkennen. Das PTS wird im weiteren Verlauf dennoch mit eingesetzt, da die durchschnittliche Lösungsqualität durch den Einsatz zumeist geringfügig niedriger ist.

	MMAS+GR		MMAS+SL		MMAS	
Graph	+	-	+	-	+	-
$\mathcal{C}_{20}(1,2,3)$	18	18	19,83	20,5	18	18,1
$\mathcal{C}_{20}(1,2,3,4)$	68	67,2	76,29	75,56	67,94	68
$\mathcal{C}_{24}(1,3,5,7)$	212,35	210,67	218,3	220,25	209,56	212,3
$\mathcal{C}_{26}(1,3,5)$	73,38	73,24	82,5	83,13	71	73,94
$\mathcal{C}_{28}(1,3)$	10,5	10,67	12	12,67	10,71	12,1
$\mathcal{C}_{30}(1,3,5,8)$	241,17	244,67	279,36	280,88	239	250,88
$\mathcal{C}_{32}(1,2,4,6)$	130,72	129	177,89	175,75	130,94	126,2
$\mathcal{C}_{34}(1,3,5)$	97,71	97,7	113,5	115,91	95,47	98,25
$\mathcal{C}_{36}(1,2,4)$	36,45	36,12	58	60,81	36,6	36
$\mathcal{C}_{38}(1,4,7)$	163,46	161,36	183,91	188,77	160,83	165,12
$\mathcal{C}_{42}(1,4)$	25	26,69	30,79	34,08	25,45	26,44
$P(14,3)$	8	8,38	8,75	8,94	8,5	8,4
$P(18,3)$	8,5	8,5	10	11	9	9
$P(20,3)$	11,88	12	13,5	14,6	12	12,5
$P(23,3)$	13,17	13,35	16,5	17,5	13,05	15,33
$P(26,3)$	15,13	16,5	20,4	21	15,5	17,5
$P(28,3)$	16,82	18,18	24	24,5	19,5	19,75
$P(30,3)$	16,63	18,18	26,44	27,7	17,88	18,75
$P(32,3)$	19,5	20,22	29,79	31,08	19,1	22
$P(34,3)$	21,36	22,13	32,25	33	21,79	22,94
$P(38,3)$	24	26,25	38	41	24,36	27,44
$ug10.40$	4,24	4,15	6,53	6,67	4,38	4,79
$ug82.45$	15	15	19	19,4	14,75	15,75
$ug38.50$	7,5	7,67	11,33	11,5	7,67	8,13
$ug63.70$	12,25	11,56	19	19,27	13	13,5

Tabelle 4.4: Durchschnittlich erzielte Lösungsqualität (2-seitige Buchkreuzungszahl) pro Graph und Optimierungsvariante mit den optimierten Parametereinstellungen

4.5 Interpretation der Parametersätze

Es stellt sich die Frage, ob aus den von der SPOT optimierten Parametersätzen gewisse Gesetzmäßigkeiten erkennbar sind, die auf allgemein gute Einstellungen schließen lassen. In Anhang D sind die optimierten Parametersätze für jeden Graphen und eingesetzten ACO Algorithmus aufgelistet.

Bezüglich einer Analyse müssen eingangs die vorhandenen „einfachen“ Probleminstanzen eliminiert werden. Das sind diejenigen, für die zahlreich gute Ergebnisse erzielt werden und anzunehmen ist, dass hierbei der Parametersatz nicht in hohem Maße ausschlaggebend ist. Wie aus den Boxplots in Anhang E erkennbar ist, zählen dazu die Graphen $P(14,3)$, $\mathcal{C}_{20}(1,2,3)$, $\mathcal{C}_{36}(1,2,4)$ sowie $ug25.30$. Zusätzlich werden noch die beiden nächstgrößeren Petersen Graphen $P(18,3)$ und $P(20,3)$ hinzugenommen. Die Parametersätze dieser Graphen werden für die folgende Analyse nicht betrachtet.

Auf den ersten Blick variieren die optimierten Parametereinstellungen der Algorithmen für unterschiedliche Probleminstanzen, so dass sich hieraus keine klare Tendenz ergibt. Diese Schwankungen stehen bei tieferer Betrachtung der Daten nicht in Bezug zur Größe der zugrundeliegenden Probleminstanz. Sogar innerhalb der SPOT-Ausgabedateien mit chronologisch geordneten besten Parametersätzen sind starke Schwankungen erkennenbar.

Um die Parametersätze näher zu untersuchen, wird im nächsten Punkt ein Clustering-Verfahren angewendet. Es besteht das Ziel der Clustereinteilung, so dass sich die Parametersätze innerhalb eines Clusters ähnlich sind.

4.5.1 Anwendung eines Clustering-Verfahrens

In diesem Schritt wird versucht, die Parametersätze jeder Optimierungsvariante in Cluster mit ähnlichen Eigenschaften zu gruppieren. Zur Anwendung kommt dabei das k-means Clustering Verfahren[11]. Hierbei ist die Anzahl der Cluster, k, von vornherein festzulegen. Das Verfahren bestimmt k Clustermitten, die mit je einem Cluster assoziiert sind. Eine Beobachtung wird demjenigen Cluster zugeordnet, zu dessen Mitte sie die geringste euklidische Distanz aufweist.

Der Parameter β (und auch $\beta 2$) erhält insgesamt eine Sonderrolle: Auffällig ist die häufige Einstellung auf einen Wert größer als fünf. Aus den von der SPOT erstellten Regressionsbäumen[12] geht hervor, dass im Allgemeinen nur sehr geringe Werte von β zu schlechten Ergebnissen führen. Da β als Exponent in die Entscheidungsfindung eingeht, beeinflussen Einstellungen ab einem bestimmten Wert den Prozess nicht mehr maßgeblich. Innerhalb bestimmter Bereiche kann β nahezu beliebig gewählt werden. Daher wird β nicht in den Clusteringprozess mit einbezogen. Es lässt sich daraus weiterhin folgern, dass die in Kapitel 3.5 definierten heuristischen Informationen nutzbringend für die Qualität der Algorithmen sind.

Aufgrund der unterschiedlichen Skalierung sind die Daten vor dem Clustering normiert worden. Die Parametersätze wurden für jede Optimierungsvariante sowohl nach Graphenklassen getrennt, als auch insgesamt betrachtet. Im ersten Fall wurden zwei Cluster verwendet, im letzten drei.
Abbildung 4.6 und 4.7 verdeutlichen die Cluster aller Parametersätze der hybriden Optimierung aus MMAS bzw. ACS und der randomisierten Greedy-Heuristik als Scatterplot. In jedem Plot können unterschiedlich viele Datenpunkte abgetragen sein. Das liegt daran, dass bei mehrfachem Vorkommen gleicher Einstellungen alle diese Parametersätze auf den gleichen Datenpunkt projiziert werden und dann insgesamt weniger Punkte abgetragen sind.

Es ist ersichtlich, dass sich die Datenpunkte innerhalb eines Clusters zum Teil über ein weites Intervall streuen und keine empfehlenswerte Einstellung ersichtlich wird. Desweiteren verteilen sich die Parametersätze einer Graphenklasse bei tieferer Betrachtung der Daten auf unter-

[11] [Har75, S.84ff]
[12] [BB06, S.55f]

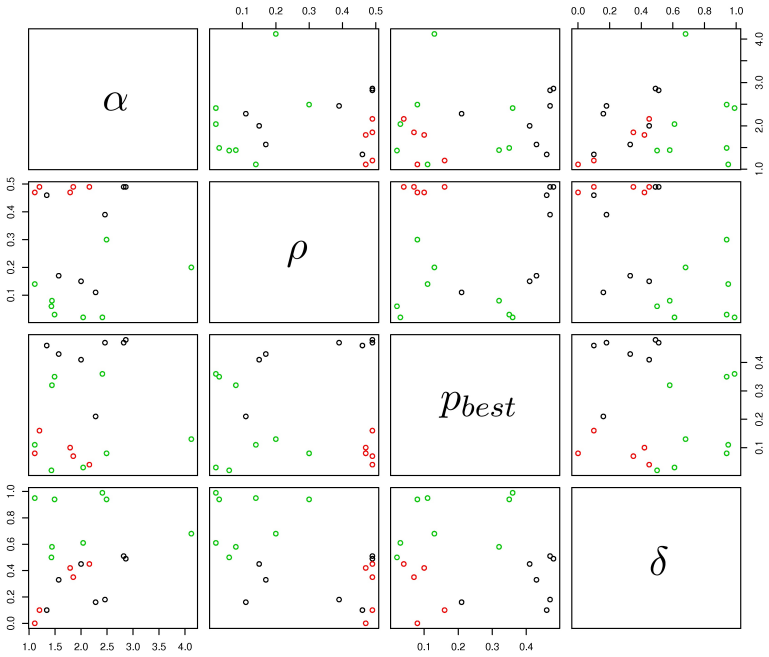


Abbildung 4.6: Clustering für die Variante MMAS+GR als Scatterplot. Die Farbe eines Datenpunktes stellt dessen Cluster dar.

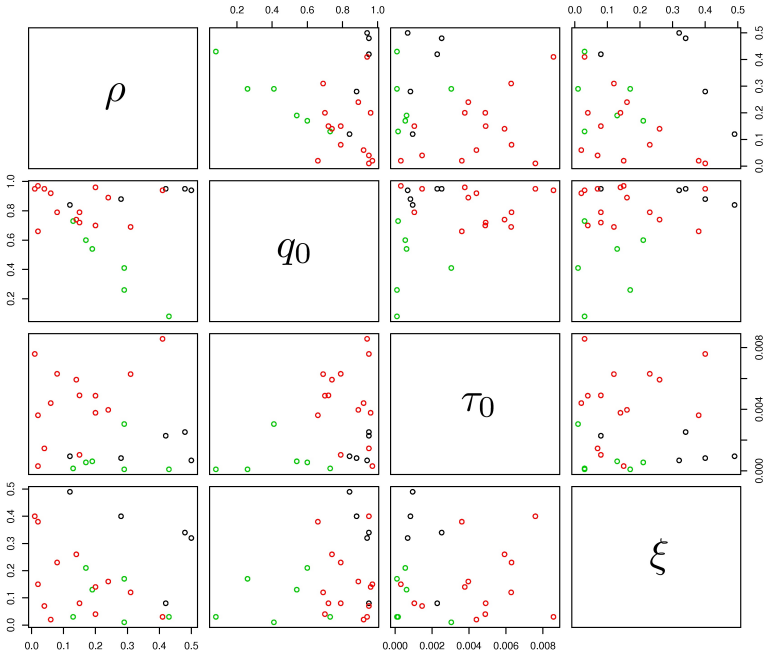


Abbildung 4.7: Clustering für die Variante ACS+GR als Scatterplot. Die Farbe eines Datenpunktes stellt dessen Cluster dar.

schiedliche Cluster. Ähnliche Resultate zeigen sich bei den anderen Optimierungsvarianten, die hier aus Platzgründen nicht aufgeführt sind.

Insgesamt ist allerdings erkennbar, dass im ACS der Parameter q_0 zahlreich auf einen Wert zwischen 0.7 und 1.0 eingestellt worden ist. Es scheint sich somit allgemein als sinnvoll herausgestellt zu haben, mit hoher Wahrscheinlichkeit die beste Alternative direkt zu wählen. Parallelen finden sich in den von Dorigo und Stützle veröffentlichten Empfehlungen für das TSP [DS04, S.71]; dort wird die Einstellung $q_0 = 0.9$ angeraten.

Das nach den einzelnen Graphenklassen getrennte Clustering lässt vereinzelt Tendenzen zu ähnlichen Parametrisierungen erkennen. Beispielsweise ist α für einige Optimierungsvarianten und Graphenklassen zahlreich im Bereich zwischen 1.0 und 1.5 eingestellt worden. Eine allgemeine Aussage lässt sich dabei jedoch nicht aufstellen, zumal sich in Abhängigkeit der Graphenklassen Veränderungen ergeben.
Demnach können insgesamt keine eindeutigen Parameterempfehlungen ausgemacht werden.

Erklärungsansätze für die Schwankungen in den optimierten Parametereinstellungen bieten zwei Hypothesen:

1. Die Parameter sind sehr sensitiv bezüglich der Probleminstanz
2. Es existieren mehrere Einstellungen, mit denen gute Ergebnisse erzielt werden können

Im ersten Fall würden Abweichungen von den optimierten Parametersätzen sehr schnell schlechtere Resultate hervorrufen und empfehlenswerte Einstellungen nur innerhalb eines kleinen Bereiches liegen. Auf verschiedenen Probleminstanzen könnten diese Bereiche voneinander abweichen und demnach auch die optimierten Parameter.
Im zweiten Fall könnte einerseits ein weiter Bereich existieren, innerhalb dessen Parameterveränderungen nur geringe Auswirkungen auf die Zielfunktion aufweisen. Dann ist es möglich, mit unterschiedlichen Einstellungen zu vergleichbaren Ergebnissen zu gelangen. Das Optimierungsverfahren könnte einen Parameter mit beliebigen Werten aus einem solchen Bereich belegen. Andererseits wäre denkbar, dass mehrere kleinere Bereiche bestehen, innerhalb derer eine ähnliche Lösungsqualität erreicht wird, die jedoch weit auseinander liegen.

Mit Hilfe einer Sensitivitätsanalyse lässt sich der Grund für die Parameterschwankungen ermitteln. Der nächste Abschnitt widmet sich dieser Thematik.

4.5.2 Sensitivitätsanalyse

Eine umfassende Einführung in die Sensitivitätsanalyse (SA) liefert die Literatur [CSST00].

Zur Durchführung einer Sensitivitätsanalyse existieren verschiedene Methodiken mit jeweils differenzierenden Eigenschaften [vgl. CSST00, S.16f].
Dabei unterscheidet man zwischen folgenden Verfahren:

- **Untersuchung der Faktoren** (*Screening Designs*)
  Das Ziel besteht in der Identifizierung der einflussreichsten Faktoren innerhalb eines Systems mit mehreren Parametern. Üblicherweise ist nicht jedem Parameter eine hohe Bedeutung beigemessen.

- **Lokale SA**
  Betont wird hierbei der lokale Parametereinfluss auf das Modell. Dieses ist eine analytische Methode und beinhaltet die Auswertung partieller Ableitungen.

- **Globale SA**
  Der Schwerpunkt dieser Methode besteht in der Assoziation von Ausgabe-Schwankungen mit den Schwankungen der Eingabefaktoren.

Geeignet in Bezug auf die vorliegende Problemstellung sind Screening Designs [siehe CSST00, S.16f; CKA00, S.65ff] und die lokale Sensitivitätsanalyse. Aufgrund der Bildung von partiellen Ableitungen ist die lokale SA jedoch aufwendig. Screening Designs betrachten ausschließlich die Haupteffekte der Parametereinstellungen. Ihre Anwendung besteht grundsätzlich in der Minimierung zu betrachtender Parameter vor einer Parameteroptimierung. Auf diese Weise wird eine Beschränkung des zu untersuchenden Parametersatzes in weiteren Schritten ermöglicht. Mit einer geringen Modifikation lassen sich die Parametereinstellungen jedoch etwas genauer und für die vorliegende Problemstellung ausreichend untersuchen.
Parameter, die sich als nicht einflussreich herausstellen, können prinzipiell mit mehreren Einstellungen belegt werden, ohne die Zielfunktion maßgeblich zu beeinflussen. Dieses würde Schwankungen in den Parametersätzen begründen. Screening Designs berücksichtigen dabei keine gegenseitigen Parameterabhängigkeiten, d.h. die simultane Veränderung mehrerer Variablen ist kein Gegenstand der Analyse.
Im nächsten Punkt wird eine solche Analyse durchgeführt.

Anwendung eines Screening Designs

Die einfachste Klasse von Screening Designs stellen die *one-at-a-time* Experimente dar. Hierbei wird der Einfluss von Veränderungen jedes Parameters der Reihe nach ermittelt. Grundlage bildet standardmäßig ein vorab definierter Parametersatz, der sich *Kontrollszenario* nennt. Ausgehend von diesem Kontrollszenario wird in jedem Experiment der Wert genau eines Parameters verändert. Die Veränderungen basieren auf zwei bestimmten Extrempunkten. Damit ergeben sich $2k + 1$ Experimente, wenn k die Anzahl der Parameter ist (ein Experiment wird zur Evaluierung des Kontrollszenarios verwendet).
Vielfach stellt das Kontrollszenario einen Parametersatz dar, dessen Werte in der Literatur als „Standardeinstellungen“ gelten. Man geht davon aus, dass sich diese für jeden Parameter in der Mitte beider Extrempunkten befinden.

In der hier durchzuführenden Sensitivitätsanalyse liegen jedoch veränderte Voraussetzungen vor: Optimierte Parametereinstellungen sind für spezifische Probleminstanzen bereits bekannt (siehe Anhang D). Diesbezüglich wird das Kontrollszenario im Folgenden aus den optimierten Parametereinstellungen bestehen, um die Auswirkungen von Veränderungen zu untersuchen. Die Extrempunkte sind durch die Parameterintervalle in Kapitel 4.4.1 bereits bestimmt. Damit die Experimente eine größere Bandbreite abgedecken, werden mehr als zwei Veränderungen für jeden Parameter realisiert. Aus den so realisierten Experimenten lassen sich Rückschlüsse darüber ziehen, inwieweit Abweichungen von den optimierten Parametereinstellungen relevant für die Qualität der ermittelten Ergebnisse sind.

Die Experimente unterliegen folgendem Testdesign:
Mit jedem Parametersatz werden fünf Algorithmenläufe durchgeführt. Das arithmetische Mittel der optimierten 2-seitigen Buchkreuzungszahl aller fünf Läufe dient als Indikator für die

Qualität. Durch die Einbeziehung mehrerer Läufe können die Auswirkungen besser eingeschätzt werden.

Damit die Unterschiede zwischen den Parametersätzen objektiv beurteilt werden können, ist eine Initialisierung des Zufallszahlen-Generators mit der gleichen Einstellung vor jedem Experiment sinnvoll. Hiermit stehen jedem Experiment die gleichen Zufallszahlen zur Verfügung und durch Randomisierung entstehende Performanceunterschiede werden vermieden. Unterschiede in der durchschnittlichen Lösungsqualität sind damit ausschließlich auf die Parameterveränderungen zurückführbar.

Die folgende Sensitivitätsanalyse wird nur für eine Probleminstanz exemplarisch durchgeführt. Nachfolgend sind die Sensitivitätsanalysen aller neun eingesetzten Optimierungsvarianten für den Graphen $\mathcal{C}_{24}(1,3,5,7)$ realisiert.
Vereinzelt können dabei mit Abweichungen von dem optimierten Parametersatz bessere Ergebnisse erzielt werden. Das liegt zum einen an den wenigen Multistarts dieser Analyse im Vergleich zu den von der SPOT durchgeführten Experimenten. Zum anderen müssen die optimierten Parametereinstellungen nicht notwendigerweise optimal sein.

Sensitivitätsanalyse für das AS

Abbildung 4.8 visualisiert die Ergebnisse der Sensitivitätsanalyse für die drei Optimierungsvarianten des AS. Die Bezeichnung „AS“ kennzeichnet die Optimierung auf dem Graphen $\mathcal{G}_{\pi,\sigma}$ und somit die Generierung von Permutation und Seitenzuordnung durch den ACO Algorithmus. „AS+GR“ steht für die hybride Optimierung aus dem AS und der randomisierten Greedy-Heuristik. Äquivalent bezeichnet „AS+SL“ die Kombination aus AS und der *SLOPE*-Heuristik.

Auf der y-Achse ist die durchschnittlich erzielte 2-seitige Buchkreuzungszahl der fünf Testläufe eines Parametersatzes abgetragen. Die rot gestrichelte horizontale Linie kennzeichnet das von dem Kontrollszenario (siehe Anhang D für die optimierten Parameter) erzielte Ergebnis. Die roten Punkte stellen für jeden Parameter dessen Wert im Kontrollszenario dar, also die optimierte Einstellung. Sie liegen somit zwangsläufig alle auf der erwähnten Linie. So werden Auswirkungen von Parameterveränderungen in Bezug zum Kontrollszenario direkt sichtbar. Die blauen Punkte stellen die weiteren untersuchten Einstellungen für den jeweiligen Parameter und die damit erzielte durchschnittliche Lösungsqualität dar.

Abbildung 4.8 verdeutlicht, dass α, α_2 und β_2 den größten Einfluss aufweisen. Durch eine veränderte Einstellung werden deutlich schlechtere Ergebnisse erzielt, wobei α_2 erst ab einem Wert von 4.0 sehr sensitiv reagiert. Besonders für ρ und τ_0 liegen jedoch die Bereiche, in denen gute Ergebnisse erzielt werden können, größtenteils sehr weit auseinander. Hierbei können durch unterschiedliche Einstellungen vergleichbare Ergebnisse erreicht werden.
Auffällig ist überdies, dass β_2 offensichtlich eine dominante Bedeutung zukommt. Bei Werten kleiner als 15.0 steigt die durchschnittliche Lösungsqualität rapide an.

In der Gesamtbetrachtung existieren für β, ρ und τ_0 weit gestreute Einstellungen, mit denen gute Ergebnisse erzielt werden.

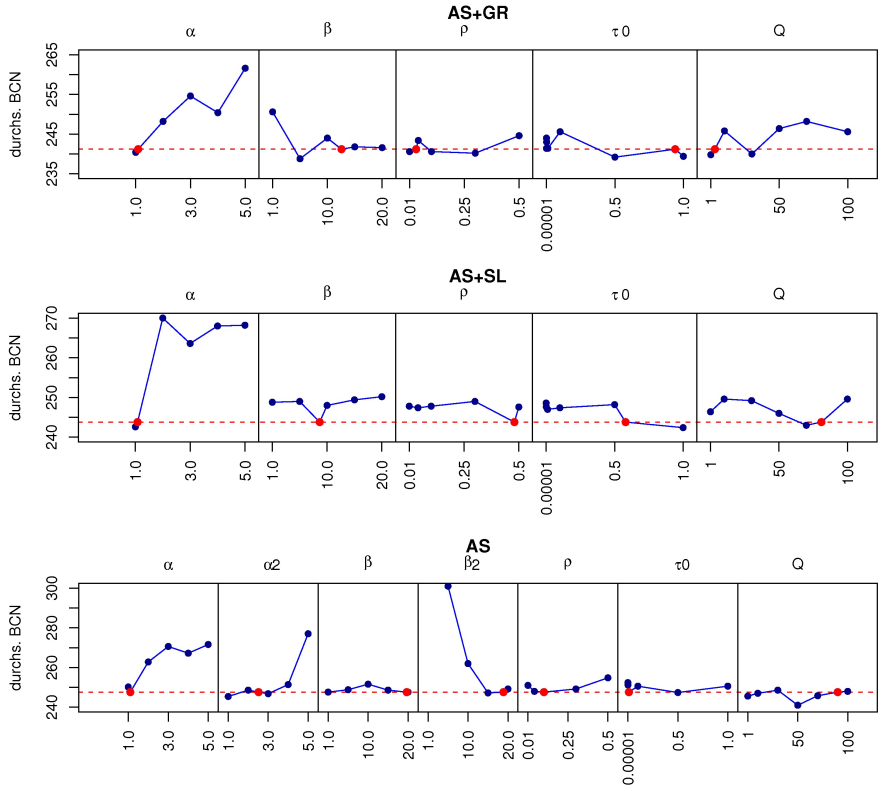


Abbildung 4.8: Sensitivitätsanalyse für das AS

Sensitivitätsanalyse für das MMAS

Äquivalent zu dem vorangegangenen Unterkapitel verdeutlicht Abbildung 4.9 die Sensitivitätsanalyse für das MMAS.

Auffällig ist erneut eine hohe Bedeutung des Parameters α. α_2 zeigt ein vergleichbares Verhalten zu den AS Varianten, genauso wie β_2.
Der Parameter β erreicht nur für geringe Werte (kleiner als 10.0) eine geringere Lösungsqualität, wenngleich die Differenzen nicht sonderlich auffällig erscheinen.

Äußerst sensitiv reagiert die Einstellung der Pheromonverdunstungsrate ρ, wohingegen p_{best} bei mehreren Einstellungen Lösungen nahezu identischer Qualität liefert. Die geringe Sensibilität von p_{best} ist durch das Zusammenspiel mit dem PTS zu begründen [siehe SH00]. Allgemein wird die Bedeutung von p_{best} bei Einsatz des PTS vermindert.

Das Einleiten des PTS ist vom jeweiligen Lauf des Algorithmus und insbesondere dem verwendeten Parametersatz abhängig. Aus diesem Grund mag auch die Bedeutung des Parameters δ in den unterschiedlichen Optimierungsvarianten und auch auf unterschiedlichen Graphen variieren. Eine Aussage über die Sensitivität von δ kann daher nicht getätigt werden. Es ist aber zu erkennen, dass das PTS hierbei in allen Varianten eingeleitet wurde; ansonsten würden die Kurven von δ eine Gerade darstellen.

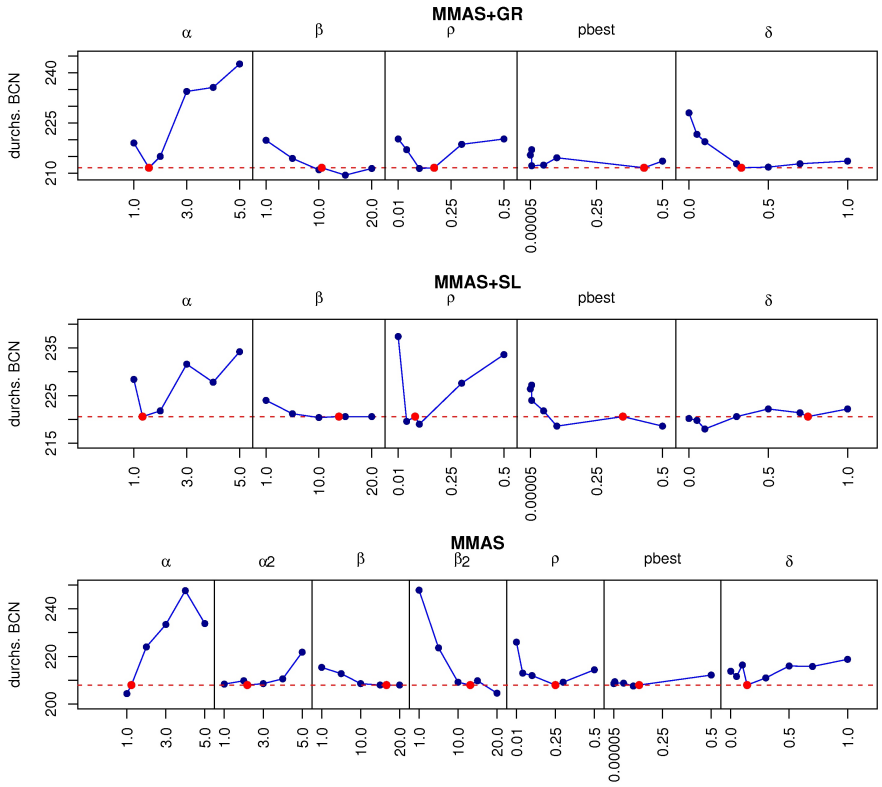


Abbildung 4.9: Sensitivitätsanalyse für das MMAS

Sensitivitätsanalyse für das ACS

Nach Abbildung 4.10 reagieren ausschließlich q_0 und τ_0 sensitiv gegenüber Veränderungen. Interessant ist zunächst, dass die restlichen Parameter allesamt einen unbedeutenden Einfluss auf die Qualität der resultierenden Lösungen aufweisen. Die Begründung hierfür ist in der Einstellung von q_0 zu finden: Die optimierten Werte von q_0 liegen geringfügig unterhalb von 1.0. Mit äußerst hoher Wahrscheinlichkeit wird daher die in jeder Entscheidung beste Alternative direkt gewählt. Seitens der Pheromonspuren scheint nur die Wahl von τ_0 relevant für den Auswahlprozess zu sein. Der Einfluss von ρ und ξ ist verschwindend gering.
In der Kombination aus ACS und $SLOPE$-Heuristik weist q_0 einen etwas geringeren Wert auf, so dass der „Standard“-Auswahlprozess eine frequentere Anwendung erfährt. Die Bedeutung von β, ρ und ξ steigt dabei geringfügig.
Schlussfolgernd scheint die Wahl von q_0 maßgeblich die Bedeutung dieser Parameter zu beeinflussen.

Aus den Parametertabellen und der Clustering-Analyse geht hervor, dass q_0 oft auf einen hohen Wert eingestellt wurde. Grundsätzlich ist zu vermuten, dass damit die Bedeutung der Parameter β, ρ und ξ herabgestuft wird und daher Parameterschwankungen auftreten.

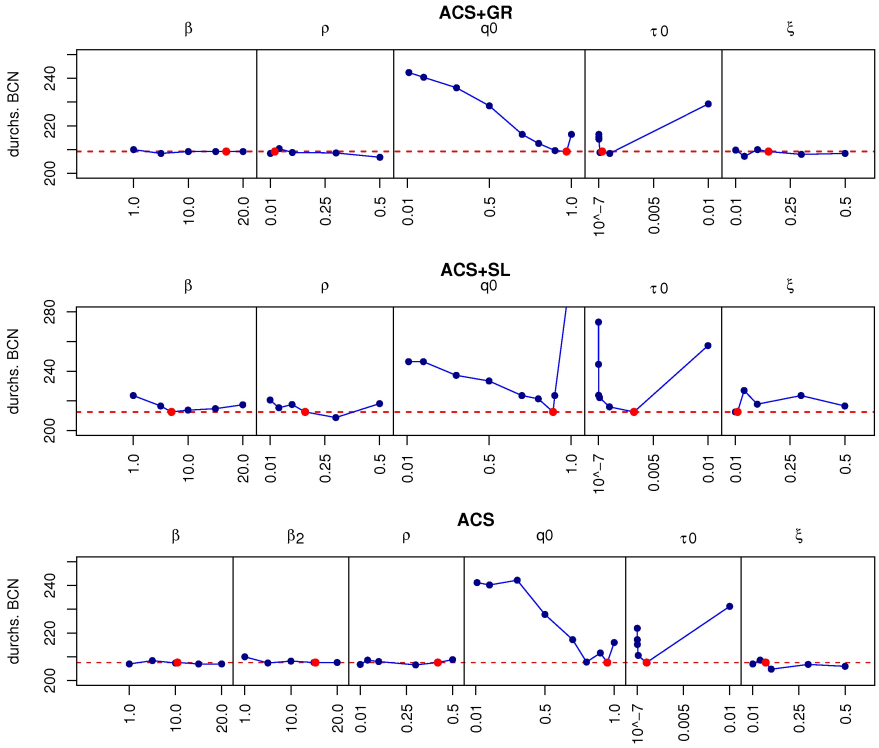


Abbildung 4.10: Sensitivitätsanalyse für das ACS

Analyse für eine Optimierungsvariante

Die vorigen Unterkapitel untersuchen die Sensitivität aller Optimierungsvarianten ausschließlich für eine bestimmte Probleminstanz. Auf anderen Probleminstanzen und deren optimierten Parametersätzen könnten sich Veränderungen ergeben und die Parameter anders reagieren. Dieses Unterkapitel untersucht nun die Sensitivität der einzelnen Parameter für je zwei Probleminstanzen der zirkulären, Petersen und Rome Graphen. Aufgrund der Datenmengen ist die Analyse hier nicht für jede Optimierungsvariante durchführbar und wird dazu auf das MMAS in Kombination mit der randomisierten Greedy-Heuristik beschränkt (Abb. 4.11).

Der Parameter β erreicht im Bereich zwischen 1.0 und 5.0 auffällig schlechtere Ergebnisse. Ab einem bestimmten Wert ändert sich das erzielte Resultat nicht mehr maßgeblich, was in Kapitel 4.5.1 begründet wurde. Die Bedeutung von p_{best} ist generell als gering anzusehen und es ist möglich, mit mehreren Einstellungen ähnliche Resultate hervorzurufen.

Der Verlauf der Parameterkurven ist nicht immer gleich und die Bedeutung einzelner Parameter ist von der Probleminstanz sowie dem dafür optimierten Parametersatz abhängig. Man vergleiche dazu die Kurven von α und ρ für verschiedene Graphen, die teilweise sensitiv reagieren und teilweise eine eher geringe Bedeutung aufweisen. Aus diesem Grund ist es nicht möglich, zu einer allgemeinen Aussage über die Parameterbedeutung zu gelangen.

Insgesamt betrachtet existieren für einige Parameter mehrere Einstellungen, mit denen sich die Ergebnisse nahezu kaum verändern. Dieses deutet darauf hin, dass für eine Probleminstanz mehrere Parametersätze existieren, mit denen gute Resultate erzielt werden. Dieses würde zusätzlich eine Erklärung für die Schwankungen der optimierten Parametersätze innerhalb der

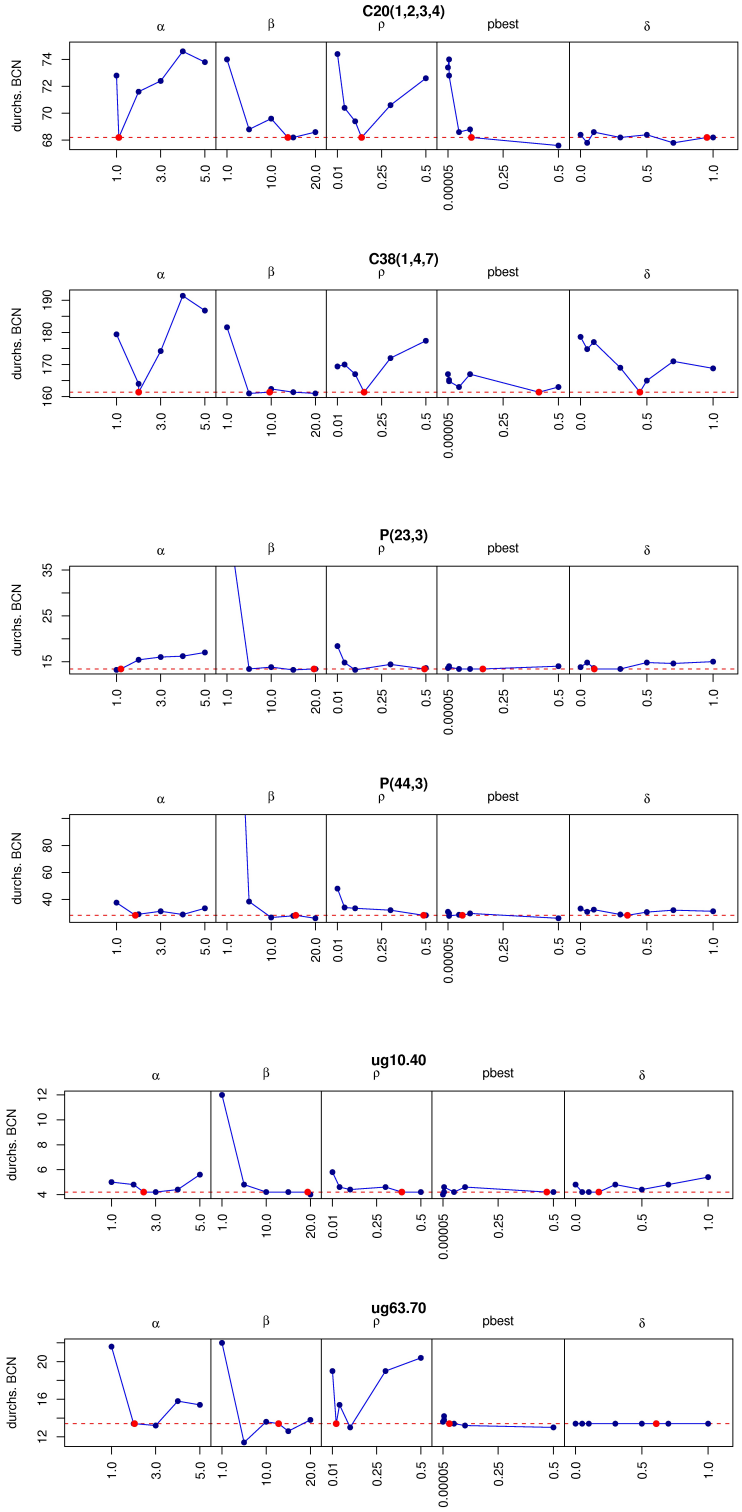


Abbildung 4.11: Sensitivitätsanalyse für das MMAS in Kombination mit der randomisierten Greedy-Heuristik auf verschiedenen Graphen

SPOT-Ausgabedateien liefern.
Ob Parametersätze bestehen, die auf verschiedenen Probleminstanzen gute Ergebnisse hervorrufen und somit allgemein empfohlen werden könnten, bleibt offen, da hier nur Veränderungen jeweils eines Parameters untersucht wurden.

4.6 Testergebnisse

ACO Algorithmen sind randomisierte Algorithmen und so müssen zur Beurteilung der Performance mehrere Läufe mit den optimierten Parametern (siehe Anhang D) realisiert werden. Die Algorithmen wurden dazu 100 mal auf einer Probleminstanz gestartet und ausgewertet. Für eine statistische Darstellung der Resultate bieten sich insbesondere die populären *Boxplots*[13] als visuelles Tool an (siehe Abb. 4.12). Diese stellen u.a. den Median sowie das untere bzw. obere Quartil dar und geben einen schnellen Einblick in die Verteilung der erzielten Ergebnisse. Als *Interquartil-Abstand* (IQR)[14] bezeichnet man die Differenz zwischen dem oberen und unteren Quartil. Innerhalb der Box befinden sich demnach 50% aller Werte. Die Zäune werden auf die Länge von $1.5 * IQR$ begrenzt und beginnen ab dem unteren bzw. oberen Quartil. Daten außerhalb der Zäune bezeichnet man als *Ausreißer*.

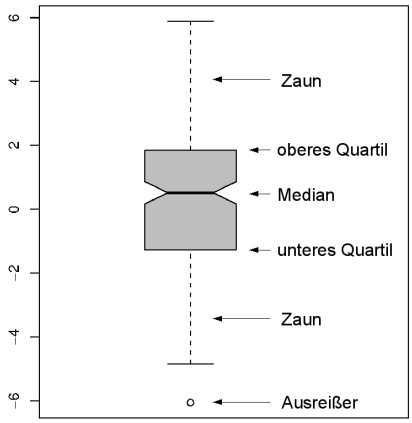


Abbildung 4.12: Beispiel für einen Boxplot

Boxplots ermöglichen jedoch keine Einsichtnahme in das Laufverhalten der Algorithmen. Es wäre weiterhin interessant zu sehen, wie schnell gute Lösungen gefunden werden können, zumal sich auch die Laufzeit unterscheidet. Um die Informationen aller Läufe zu aggregieren, wird folgend der Median der bis zur jeweiligen Iteration erzielten besten Lösungsqualität (2-seitige Buchkreuzungszahl) aller 100 Läufe abgebildet. Der Verlauf dieser Kurven ist demnach monoton fallend. Der Median aus den Boxplots entspricht dem Median der letzten Iteration.

Insgesamt sind die Läufe aller neun Optimierungsvarianten auf den 31 Graphen mit optimierten Parametern durchgeführt worden. Die Resultate sind in den folgenden Unterkapiteln

[13] [BB06, S.53]
[14] interquartile range

jeweils exemplarisch für drei Graphen der Klasse dargestellt. Anhang E listet die Resultate auf allen Graphen als Boxplot separat auf.

4.6.1 Vorexperimente und verworfene Ansätze

Wie zu Beginn dieses Kapitels erwähnt, können einige Graphenklassen die Algorithmen nicht entsprechend fordern. Dieses Unterkapitel stellt die Resultate beispielhaft für jeweils einen Graphen einer Klasse zusammen. Dazu zählen Halin Graphen sowie planare Rome Graphen.

Halin Graphen

Für jeden Halin Graph[15] existiert eine 2-seitige Buchzeichnung mit Buchkreuzungszahl 0 [vgl. He06, S.213]. So ist für diese Graphenklasse das Optimum der 2-seitigen Buchkreuzungszahl bekannt. Auf diese Weise wäre eine Messung der Abweichung vom Optimum ermöglicht worden. Eine Bibliothek mit Halin-Graphen bis zu 49 Knoten wurde freundlicherweise von He [HSM07] zur Verfügung gestellt. Die meisten Algorithmen sind hierbei jedoch in der Lage, das globale Optimum von 0 in einer frühen Iteration zu erreichen (siehe Abb. 4.13). Das gilt sogar für die größten darin enthaltenen Graphen mit 49 Knoten. Die Parametersätze sind zuvor von der SPOT optimiert worden.
Diese Graphenklasse ist somit nicht in der Lage, die Verfahren entsprechend zu fordern und vergleichen zu können.

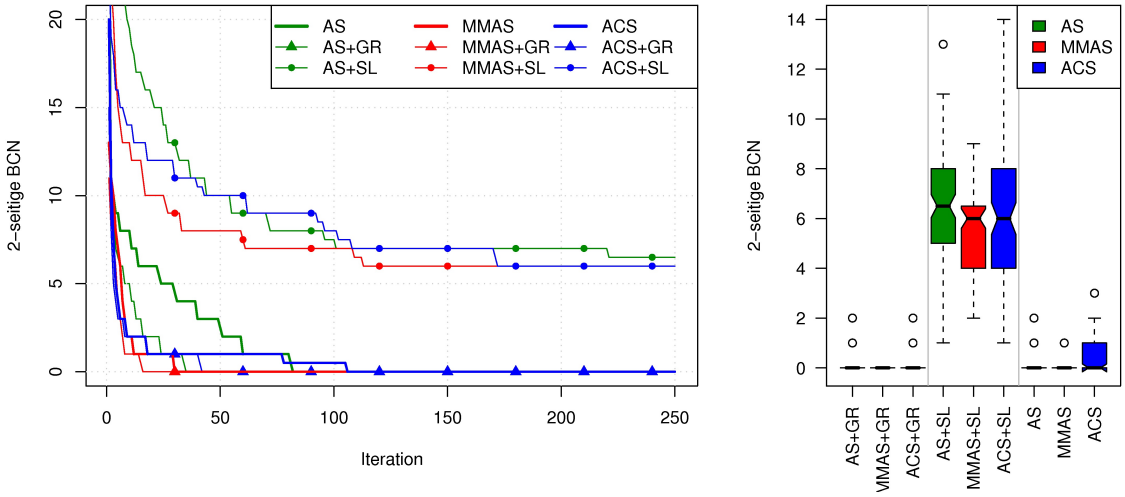


Abbildung 4.13: Ergebnisse von 100 Läufen für einen Halin Graphen mit 49 Knoten und von der SPOT optimierten Parametern

Abbildung 4.13 (links) zeigt den Median der bis zur abgetragenen Iteration erzielten besten 2-seitigen Buchkreuzungszahl aus 100 Läufen mit $n_{max} = 250$ auf. Die Bezeichnungen „AS“, „MMAS“ und „ACS“ (ohne Zusatz) stehen für die Optimierung auf dem Gesamtgraphen $\mathcal{G}_{\pi,\sigma}$. Der Zusatz „+GR“ bzw. „+SL“ kennzeichnet die hybride Optimierung mit der randomisierten Greedy-Heuristik (GR) bzw. der *SLOPE*-Heuristik (SL).

[15] [BL85]

Abbildung 4.13 (rechts) veranschaulicht die Verteilung der besten Lösungen aus 100 Läufen als Boxplot.

Lediglich die Kombination aus ACO Algorithmen und der *SLOPE*-Heuristik war dabei nie in der Lage, eine optimale Lösung zu finden. Es sind nicht einmal Ausreißer vom Wert 0 abgetragen. Die ACO Algorithmen unterscheiden sich untereinander in ihrer Leistung nicht nennenswert.

Planare Rome Graphen

Die Bibliothek *RND/BUP* der Rome Graphen besteht aus 200 zufällig generierten planaren Graphen. Nach Yannakakis [Yan89] existiert für jeden planaren Graphen eine 4-seitige Bucheinbettung. Das globale Optimum der 4-seitigen Buchkreuzungszahl dieser Graphen ist daher bekannt und entspricht dem Wert 0.
Für diese Graphenklasse waren die meisten Algorithmen sogar zahlreich in der Lage, eine 2-seitige Bucheinbettung zu finden. Das gilt insbesondere auch für die größten Graphen dieser Klasse mit 100 Knoten (siehe Abb. 4.14). Selbst in Kombination mit der *SLOPE*-Heuristik werden optimale Ergebnisse erzielt, wenn auch nicht mit gleicher Konstanz wie bei den anderen Optimierungsvarianten. Somit eignet sich auch diese Graphenklasse nicht für einen Vergleich.

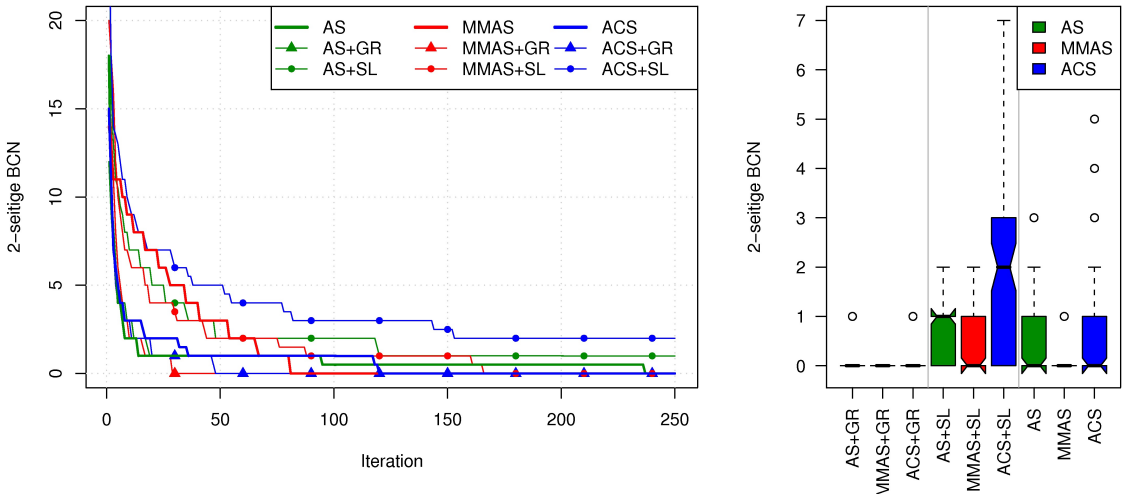


Abbildung 4.14: Ergebnisse von 100 Läufen für einen planaren Rome Graphen mit 100 Knoten und von der SPOT optimierten Parametern

4.6.2 Zirkuläre Graphen

Die ACO Algorithmen wurden auf elf zirkulären Graphen mit jeweils optimierten Parametern 100 mal gestartet. Abbildung 4.15 verdeutlicht die Verteilung der dabei erzielten 2-seitigen Buchkreuzungszahlen exemplarisch für drei Graphen als Boxplot. Die Bezeichnung „Gesamt“ kennzeichnet dabei die Optimierung auf dem Gesamtgraphen $\mathcal{G}_{\pi,\sigma}$ mit gleichzeitiger Generierung einer Seitenzuordnung durch den ACO Algorithmus.
Deutlich erkennbar ist ein Vorteil beim Einsatz des ACS und MMAS im Vergleich zum AS. Das hat sich bereits in anderen Problemen herausgestellt [siehe CHS02] und bestätigt sich

hierbei erneut. Die Differenzen sind sogar derart enorm, dass zumeist noch nicht einmal die Zäune der Boxplots überlappen. Die Verwendung des AS ist demnach nicht zu empfehlen. Desweiteren ist die Kombination aus ACO Algorithmen und der *SLOPE*-Heuristik gegenüber den anderen Optimierungsvarianten benachteiligt. Die besten Ergebnisse konnten insgesamt vom ACS und MMAS erzielt werden, wobei abhängig von der Probleminstanz teilweise eines der beiden besser geeignet scheint. In Gesamtbetrachtung gleichen sich diese Unterschiede aus, so dass nicht allgemein das ACS oder MMAS bevorzugt werden kann. Zwischen der hybriden Optimierung mit der randomisierten Greedy-Heuristik und dem alleinigen Einsatz von ACO Algorithmen auf dem Gesamtgraphen $\mathcal{G}_{\pi,\sigma}$ sind kaum Unterschiede zu erkennen. Die in Kapitel 3.1.1 erwähnte Vermutung einer pheromongesteuerten Fehlleitung der Ameisen innerhalb des Graphen $\mathcal{G}_{\pi,\sigma}$ bestätigt sich demnach nicht.

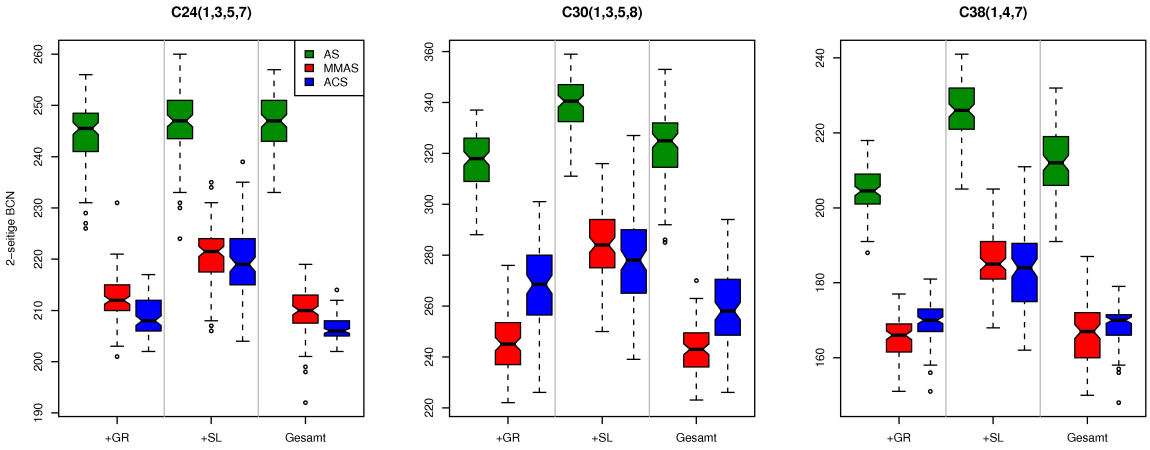


Abbildung 4.15: Boxplots von jeweils 100 Läufen mit optimierten Parametern für drei zirkuläre Graphen

Abbildung 4.16 visualisiert den Median der bis zur jeweiligen Iteration gefundenen besten Lösungsqualität aller 100 Algorithmenläufe. Beispielhaft ist dieses für zwei zirkuläre Graphen dargestellt; die Verläufe der weiteren Graphen sind ähnlich.

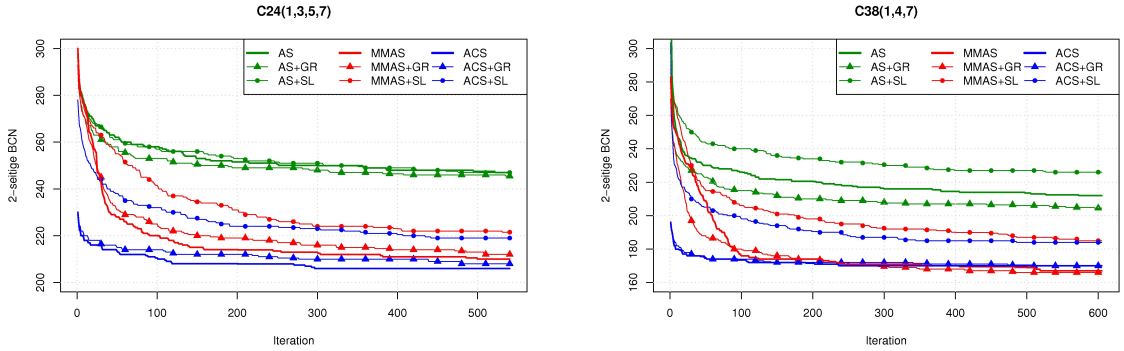


Abbildung 4.16: Median der besten Lösungsqualität von 100 Läufen

Auffällig ist, dass die ACS-Varianten besonders schon zu Beginn der Optimierung Lösungen hervorbringen, die denen der anderen ACO Algorithmen qualitativ überlegen sind (vgl. Abb. 4.16). Der Median ist bereits nach der ersten Iteration erstaunlich gering. In den folgenden Iterationen sinkt dieser zwar noch weiter, jedoch ist das Verbesserungspotential mäßig. Grund dafür ist die Einstellung von q_0 auf einen Wert nahe 1.0. Zu Beginn der Optimierung sind alle Pheromonspuren gleich. Die Bewertung einer Entscheidungsalternative während der Lösungskonstruktion beruht damit ausschließlich auf heuristischen Informationen. Die zahlreiche Wahl der heuristisch bevorzugten Alternative scheint insgesamt bereits recht gute Resultate zu liefern, wenngleich sie in den nachfolgenden Iterationen durch Einbeziehung von Pheromonspuren noch verbessert werden können.
Die hybride Optimierung aus ACO Algorithmen und der *SLOPE*-Heuristik beansprucht die geringste Rechenzeit (vgl. Kapitel 4.3). Die Grafiken lassen nun weiterhin erkennen, dass die anderen Optimierungsvarianten bereits nach sehr wenigen Iterationen schon bessere Resultate liefern. Sie sind somit auch unter Berücksichtigung der Rechenzeit zu bevorzugen.

Vergleich mit bislang erzielten besten Ergebnissen

Auf zirkulären Graphen wurden bereits diverse Verfahren zur Generierung von kreuzungsminimalen 2-seitigen Buchzeichnungen getestet [siehe Cim02; HSM07; He06, S.108]. Dabei konnte He [HSM07] mit genetischen Algorithmen die besten Resultate erzielen. In dieser Arbeit sind ACO Algorithmen für eine Auswahl der getesteten Graphen eingesetzt worden. Die Ergebnisse sind in Tabelle 4.5 auf Seite 102 zusammengetragen. Die ersten neun Spalten geben die beste erzielte 2-seitige Buchkreuzungszahl des jeweiligen ACO Algorithmus aus 100 Läufen an. Die letzten drei Spalten fassen die bislang bekannten besten Ergebnisse zusammen.
Die Notation H^*_G bezeichnet die besten von He erzielten Resultate unter Einsatz genetischer Algorithmen [siehe HSM07]. H^*_H steht für den kombinierten Einsatz von Heuristiken [siehe He06, S.108]. Dabei wurde eine Heuristik zur Generierung einer Permutation für das 1-seitige BCNP eingesetzt, auf deren Grundlage eine weitere Heuristik zur Erstellung einer Seitenzuordnung für $k = 2$ angewendet worden ist.
Die Bezeichnung C^* kennzeichnet die besten von Cimikowski erzielten Ergebnisse unter Verwendung diverser Heuristiken zur Generierung einer Seitenzuordnung [siehe Cim02]. Die zugrundeliegende Permutation ist dabei jeweils auf Basis eines Hamiltonkreises im Graphen der Probleminstanz bestimmt worden.

Die besten Resultate für einen Graphen sind jeweils unterstrichen. Die Spalte E_V (Ergebnisvergleich) fasst zusammen, ob mit den ACO Algorithmen bessere (+), schlechtere (-), oder gleich gute (=) Resultate im Vergleich zu den anderen Verfahren ermittelt wurden. Es zeigt sich, dass die mit ACO Algorithmen erzielten Ergebnisse konkurrenzfähig sind. In der Optimierung mit dem Gesamtgraphen $\mathcal{G}_{\pi,\sigma}$ sowie dem hybriden Einsatz mit der randomisierten Greedy-Heuristik können die besten bislang ermittelten 2-seitigen Buchkreuzungszahlen übertroffen werden.

Wie in Kapitel 4.1 aufgeführt, werden keine weiteren Vergleiche mit anderen Verfahren betrachtet.

In Abbildung 4.17 und 4.18 sind die ermittelten kreuzungsminimalen 2-seitigen Buchzeichnungen exemplarisch für zwei zirkuläre Graphen visualisiert.

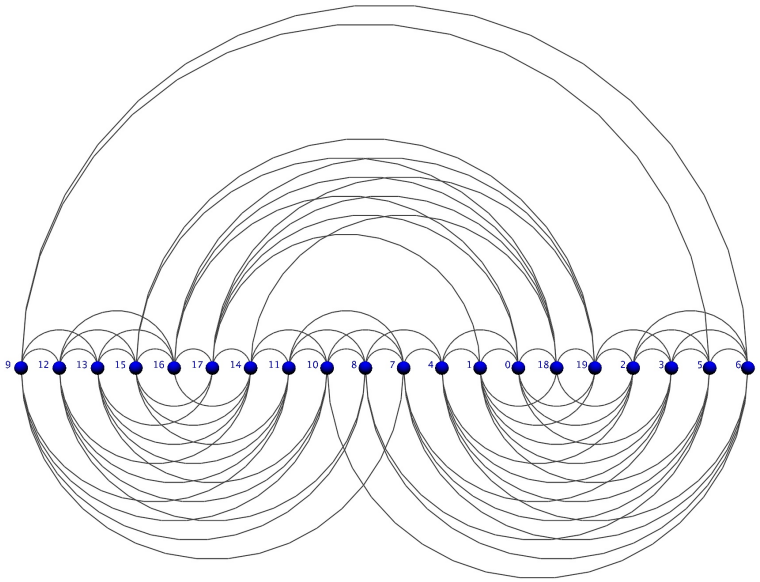


Abbildung 4.17: Optimierte 2-seitige Buchzeichnung des Graphen $\mathcal{C}_{20}(1, 2, 3, 4)$ mit Buchkreuzungszahl 62

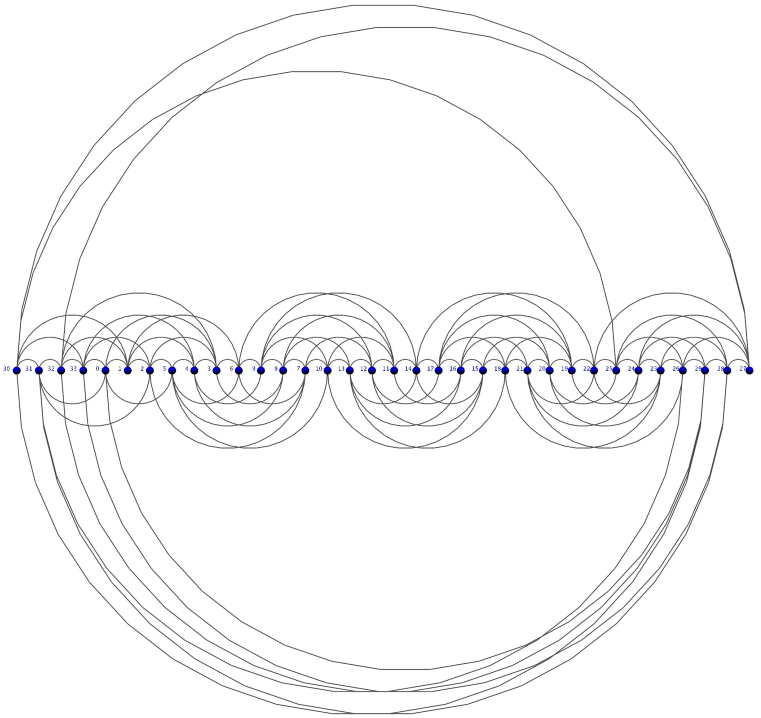


Abbildung 4.18: Optimierte 2-seitige Buchzeichnung des Graphen $\mathcal{C}_{34}(1, 3, 5)$ mit Buchkreuzungszahl 88

	+GR			+SL			Gesamtgraph						
	AS	**MMAS**	**ACS**	**AS**	**MMAS**	**ACS**	**AS**	**MMAS**	**ACS**	$\mathbf{H_G^*}$	$\mathbf{H_H^*}$	$\mathbf{C^*}$	$\mathbf{E_V}$
$\mathcal{C}_{20}(1,2,3)$	18	18	18	20	18	18	18	18	18	18	22	22	=
$\mathcal{C}_{20}(1,2,3,4)$	72	62	62	82	68	68	77	62	62	68	70	70	+
$\mathcal{C}_{24}(1,3,5,7)$	226	201	202	224	206	204	233	192	202	193	217	216	+
$\mathcal{C}_{26}(1,3,5)$	78	68	68	83	79	77	75	70	68	63	80	82	-
$\mathcal{C}_{28}(1,3)$	11	10	10	13	11	11	11	10	10	11	18	16	+
$\mathcal{C}_{30}(1,3,5,8)$	288	222	226	311	250	239	285	223	226	226	308	302	+
$\mathcal{C}_{32}(1,2,4,6)$	185	120	130	198	144	142	185	120	120	124	126	160	+
$\mathcal{C}_{34}(1,3,5)$	99	88	88	119	106	103	109	90	88	96	105	106	+
$\mathcal{C}_{36}(1,2,4)$	50	36	36	76	50	44	57	36	36	36	36	36	=
$\mathcal{C}_{38}(1,4,7)$	188	151	151	205	168	162	191	150	148	149	176	190	+
$\mathcal{C}_{42}(1,4)$	27	23	22	30	25	26	26	23	23	24	38	42	+

Tabelle 4.5: Beste ermittelte 2-seitige Buchkreuzungszahl für zirkuläre Graphen

4.6.3 Petersen Graphen

Die Schlussfolgerungen aus den mit Petersen Graphen erzielten Resultaten sind ähnlich zu denjenigen der zirkulären Graphen.
Auch hierbei ist das AS nicht in der Lage, mit der Leistung des ACS und MMAS konkurrieren zu können. Die Kombination aus ACO Algorithmen und der *SLOPE*-Heuristik bringt zudem schlechtere Ergebnisse im Vergleich zu den anderen Optimierungsvarianten hervor. Zwischen dem MMAS und ACS ist kein deutlicher Vorteil zu sehen. Die erzielten Resultate von der Optimierung auf dem Gesamtgraphen und der hybriden Variante „+GR“ unterscheiden sich nur geringfügig.

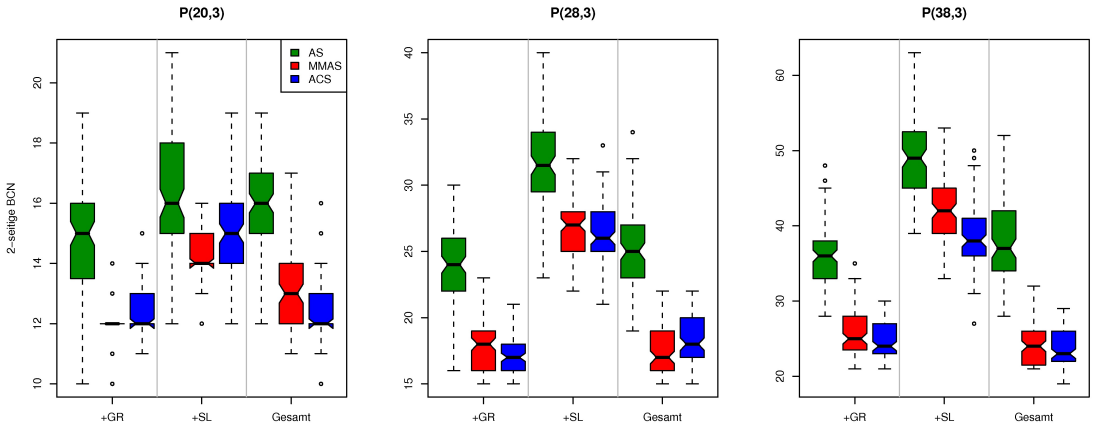


Abbildung 4.19: Boxplot der Ergebnisse für drei Petersen Graphen

Die Laufanalysen aus Abbildung 4.20 zeigen ein zu den zirkulären Graphen vergleichbares Verhalten auf. Auch hierbei generieren die ACS Algorithmen sehr schnell gute Lösungen, die

in weiteren Iterationen jedoch nicht mehr in dem Maße verbessert werden. Der Median fällt nach der 100. Iteration deutlich langsamer.

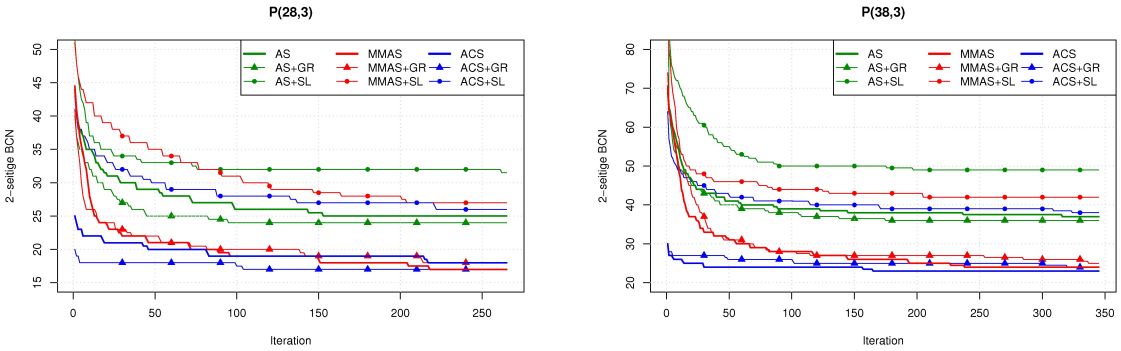


Abbildung 4.20: Median der besten Lösungsqualität von 100 Läufen

Vergleich mit der planaren Kreuzungszahl

Die planare Kreuzungszahl von Petersen Graphen $P(m, 3)$ ist nach Richter und Salazar [RS02] bekannt. Die 2-seitige Buchkreuzungszahl stellt weiterhin eine obere Schranke für die planare Kreuzungszahl eines Graphen dar. Von Interesse ist daher, inwieweit sich die besten ermittelten 2-seitigen Buchkreuzungszahlen als Approximation eignen.
Tabelle 4.6 listet die beste in 100 Läufen erzielte 2-seitige Buchkreuzungszahl für jeden ACO Algorithmus auf. Die Spalte *pl** enthält die planare Kreuzungszahl des jeweiligen Graphen. Die von ACO Algorithmen erzielte geringste Buchkreuzungszahl für einen Graphen ist unter *ACO** zusammengefasst.

	+GR			+SL			Gesamtgraph				
	AS	**MMAS**	**ACS**	**AS**	**MMAS**	**ACS**	**AS**	**MMAS**	**ACS**	**ACO***	**pl***
$P(14,3)$	7	7	7	8	8	7	7	7	7	7	6
$P(18,3)$	8	8	8	9	8	8	9	8	8	8	6
$P(20,3)$	10	10	11	12	12	12	12	11	10	10	8
$P(23,3)$	13	12	12	17	15	15	13	12	12	12	9
$P(26,3)$	15	14	13	19	18	17	15	14	14	13	10
$P(28,3)$	16	15	15	23	22	21	19	15	15	15	12
$P(30,3)$	18	14	14	27	19	22	16	15	14	14	10
$P(32,3)$	18	17	16	23	23	23	21	18	17	16	12
$P(34,3)$	24	18	18	33	26	25	25	18	18	18	14
$P(38,3)$	28	21	21	39	33	27	28	21	19	19	14
$P(44,3)$	30	25	23	49	38	40	33	24	22	22	16

Tabelle 4.6: Beste ermittelte 2-seitige Buchkreuzungszahl für Petersen Graphen, im Vergleich zur optimalen planaren Kreuzungszahl

Die absolute Differenz von *ACO** und *pl** wächst mit steigender Knotenanzahl (siehe auch Abb. 4.21). Dabei ist jedoch nicht sichergestellt, dass die ermittelte 2-seitige Buchkreuzungszahl auch tatsächlich dem Optimum entspricht.

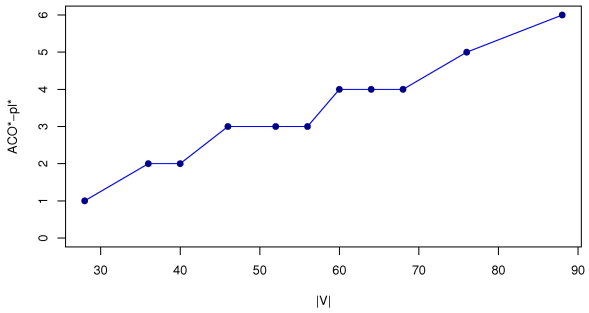


Abbildung 4.21: Differenz zwischen geringster ermittelter 2-seitiger BCN und der planaren Kreuzungszahl in Abhängigkeit von der Knotenanzahl des Graphen

4.6.4 Rome Graphen

Die Resultate der Rome Graphen sind in Abbildung 4.22 beispielhaft für drei Graphen als Boxplots visualisiert.

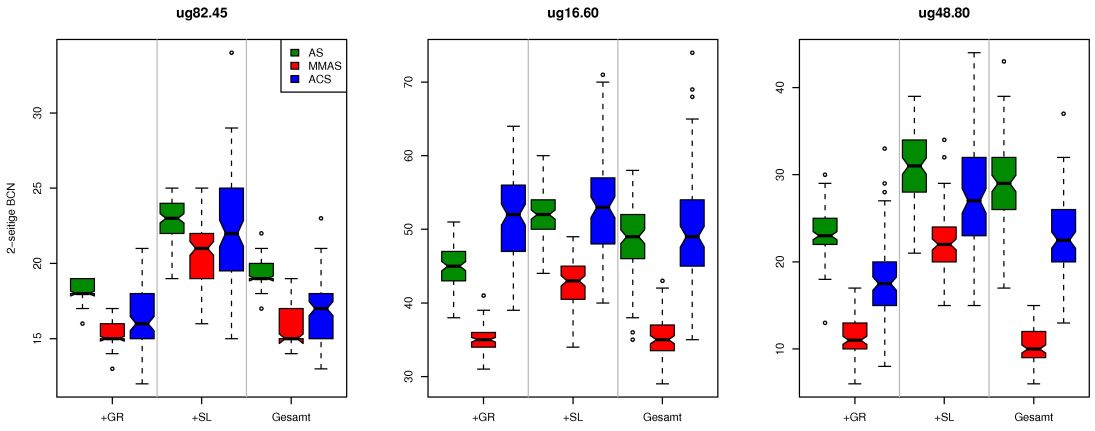


Abbildung 4.22: Boxplot der Ergebnisse für drei Rome Graphen

Zusammenfassend können die MMAS Algorithmen hierbei entgegen der Läufe auf den zirkulären und Petersen Graphen deutlich überlegene Ergebnisse im Vergleich zum ACS hervorrufen. Der Unterschied zwischen AS und ACS ist jetzt nicht mehr derart auffällig. Die Kombination mit der *SLOPE*-Strategie bringt zwar auch hierbei eine höhere Buchkreuzungszahl hervor, dennoch sind die Unterschiede zu den anderen Optimierungsvarianten nicht mehr in dem Maße offensichtlich wie bei den vorigen Graphenklassen.
Die Wahl der direkt besten Entscheidungsalternative mag für die ACS Algorithmen auf den Rome Graphen nicht so vorteilhaft sein. Mit einem Blick auf die Parametertabellen aus Anhang D hat sich die Bestimmung von q_0 auf hohen Wert anscheinend auch hierbei als sinnvoll

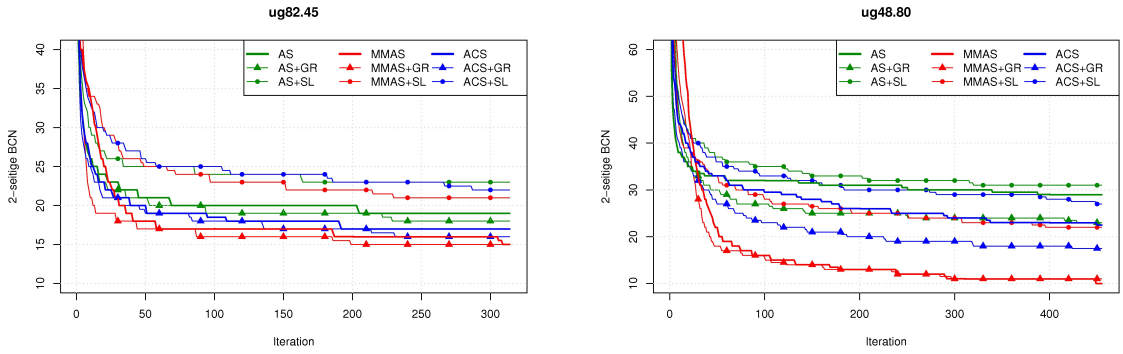


Abbildung 4.23: Median der besten Lösungsqualität von 100 Läufen

herausgestellt. Dennoch können besonders zu Anfang nicht so schnell vielversprechende Resultate ermittelt werden wie für zirkuläre Graphen und Petersen Graphen (vgl. Abb. 4.23).

4.6.5 Exemplarischer Test für $k = 3$ und $k = 4$

Aufgrund der geringen Buchkreuzungszahlen k-seitiger Buchzeichnungen mit $k > 2$ wurden diese in den bisherigen Ausführungen dieses Kapitels nicht betrachtet. Exemplarisch folgen optimierte 3- und 4-seitige Buchzeichnungen für den zirkulären Graphen $\mathcal{C}_{24}(1, 3, 5, 7)$. Dieser ist aufgrund seiner hohen 2-seitigen Buchkreuzungszahl ausgewählt worden (vgl. Tabelle 4.5). So sinkt die Lösungsqualität für steigendes k nicht direkt auf 0 und die Algorithmen werden ausreichend gefordert.

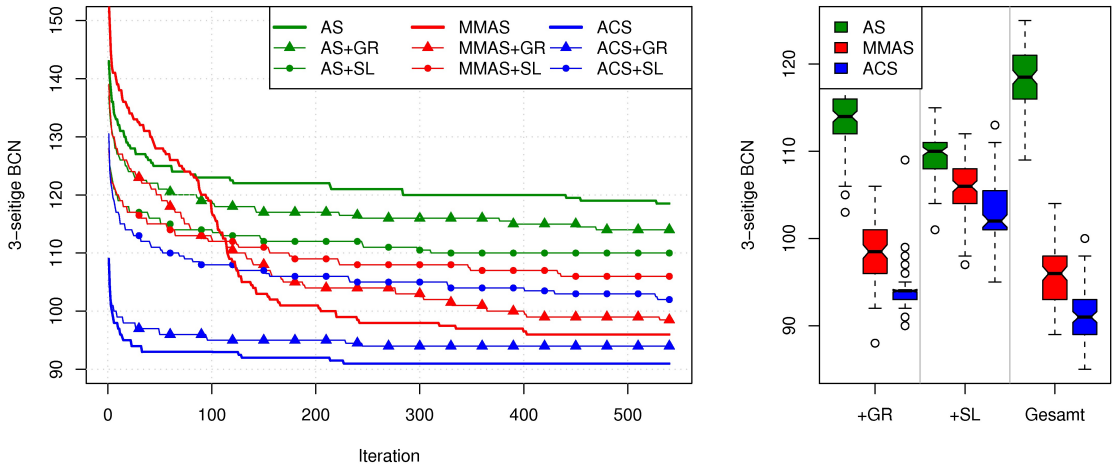


Abbildung 4.24: Optimierung der 3-seitigen Buchkreuzungszahl des Graphen $\mathcal{C}_{24}(1, 3, 5, 7)$. Die Abbildung links visualisiert den Median der besten gefundenen Lösung bis zur auf der x-Achse abgetragenen Iteration.

Interessant ist zunächst der „bauchige“ Median-Verlauf für das MMAS bei Optimierung auf dem Gesamtgraphen $\mathcal{G}_{\pi,\sigma}$. Ein Blick in die Optimierungsläufe offenbart dessen Ursache: Jeweils kurz vor der ersten Einleitung des PTS (und damit kurz vor einer Konvergenz) konnten

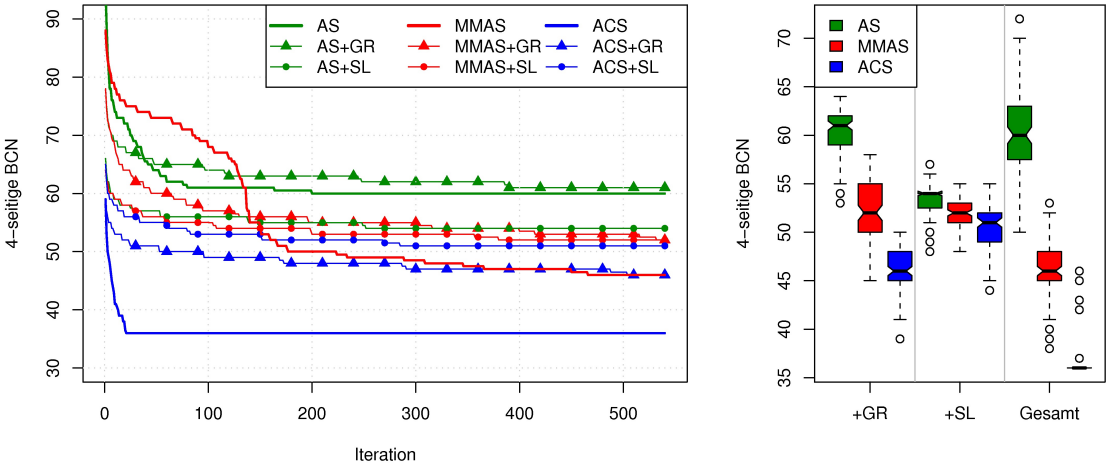


Abbildung 4.25: Optimierung der 4-seitigen Buchkreuzungszahl des Graphen $\mathcal{C}_{24}(1,3,5,7)$

die Algorithmen noch einmal deutlich bessere Ergebnisse finden. Zu diesem Zeitpunkt hat sich die Suche so weit stabilisiert, dass nur noch vielversprechende Lösungswege mit hohen Pheromonwerten belegt sind. Unter pheromongesteuerter Auswahl dieser Wege ist offensichtlich eine deutliche Verbesserung bislang gefundener Resultate möglich.
Parallelen lassen sich in der Anwendung des MMAS auf das TSP finden. Die besten Ergebnisse konnten dort kurz vor einer Stagnation gefunden werden [vgl. SH00].
Dieser Umstand ist für $k = 4$ sogar noch ausgeprägter, bei den hybriden Optimierungsvarianten jedoch weiterhin nicht zu beobachten. Der Konstruktionsgraph $\mathcal{G}_{\pi,\sigma}$ wächst mit steigendem k, was hingegen für $\mathcal{G}_{\pi}$ nicht gilt und eine Erklärung für dieses Verhalten bieten würde.

Auffällig ist überdies die Performance des ACS für die Optimierung auf dem Gesamtgraphen mit $k = 4$. Der Boxplot in Abb. 4.25 stellt das obere und untere Quartil bei dem Wert 36 dar. Anhand des Median-Verlaufs ist erkennbar, dass mindestens die Hälfte aller Algorithmenläufe diese Buchkreuzungszahl bis Iteration 20 gefunden hat und nicht mehr verbessern konnte. Das lässt den Rückschluss auf das Erreichen eines Optimums zu.
Die Frage nach der schnellen Konvergenz dieses Verfahren lässt sich mit einem Blick auf die Parametrisierung beantworten: Der Parameter q_0 erhielt den Wert 0.996, so dass in 99,6% aller Entscheidungen der beste Weg gewählt wurde. Am Anfang sind alle Pheromonspuren gleich. Alle *verbundenen*[16] und noch nicht aufgenommenen Knoten werden demnach gleich gut bewertet. Der implementierte Algorithmus wählt unter diesen jeweils die erste Variante aus, so dass insgesamt mit äußerst hoher Wahrscheinlichkeit die Permutation $(0, 1, \ldots, n-1)$ erschaffen wird. Dieses ist auch in der optimalen Zeichnung aus Abb. 4.27 sichtbar. Sie entstand aus einem Lauf dieser Optimierungsvariante. Trivialerweise mag diese Permutation hierbei gute Ergebnisse hervorrufen. Auch bei der Generierung der Seitenzuordnung wird mit großer Wahrscheinlichkeit die heuristisch höchstbewertete Seite angelaufen. In der Kombination aus ACS und der randomisierten Greedy-Heuristik wurde der Parameter q_0 ebenfalls sehr hoch eingestellt. Hierbei konnten jedoch keine ähnlich guten Ergebnisse erzielt werden, was an der separaten Generierung der Seitenzuordnung liegen könnte, die nicht unter dem Einfluss des ACO Algorithmus steht.

Das ACS bringt für beide Probleminstanzen insgesamt die besten Ergebnisse hervor. Bei der

[16] vgl. Kapitel 3.5 über die Definition der heuristischen Informationen.

hybriden Optimierung mit der *SLOPE*-Heuristik schwinden die Unterschiede zwischen den ACO Algorithmen. Die erzielten Resultate liegen nun sehr nahe beieinander.

Abbildung 4.26 und 4.27 stellen die optimierten 3- und 4-seitigen Buchzeichnungen des Graphen $\mathcal{C}_{24}(1, 3, 5, 7)$ dar.

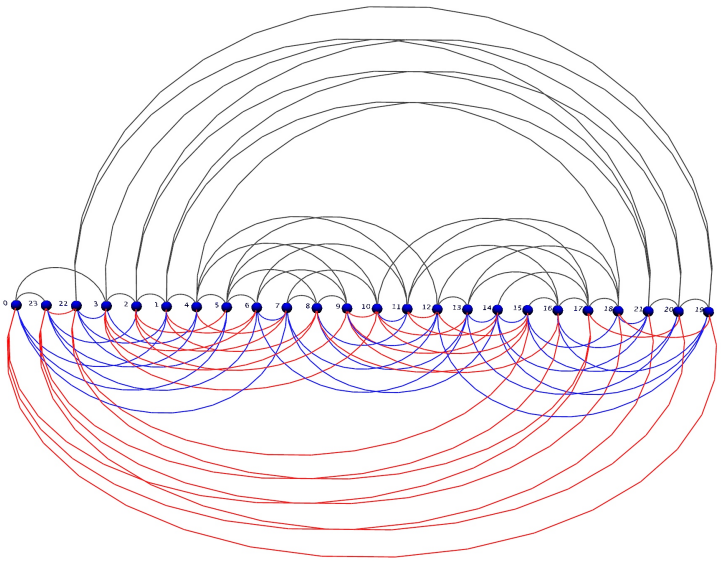


Abbildung 4.26: Optimierte 3-seitige Buchzeichnung des Graphen $\mathcal{C}_{24}(1, 3, 5, 7)$ mit 85 Kreuzungen

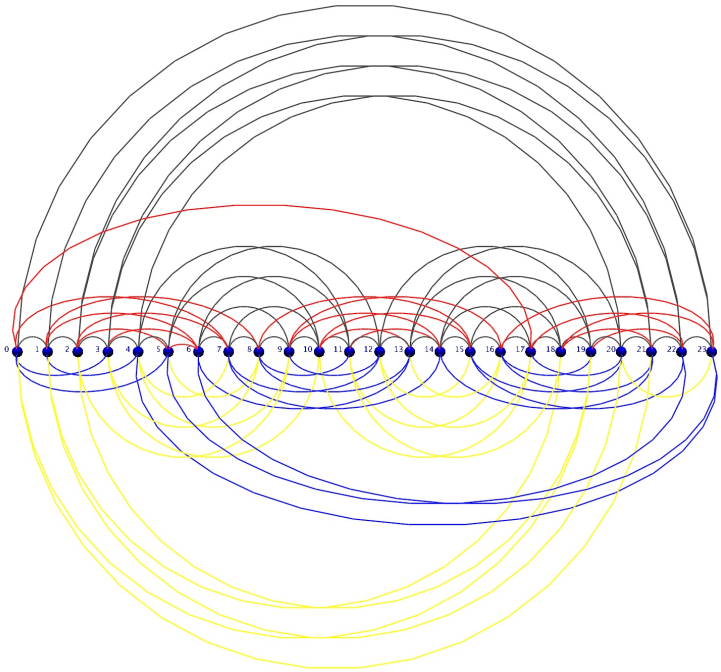


Abbildung 4.27: Optimierte 4-seitige Buchzeichnung des Graphen $\mathcal{C}_{24}(1, 3, 5, 7)$ mit 36 Kreuzungen

4.7 Vergleich mit einer randomisierten Suche

Die randomisierte Suche ohne Strategie gilt es in Bezug auf die Lösungsqualität immer zu übertreffen. Ansonsten würde sich der Aufwand zur Entwicklung und dem Einsatz von Heuristiken bzw. Metaheuristiken nicht lohnen.
Aus diesem Grund wurde zusätzlich eine randomisierte Suche implementiert. Diese generiert eine zufällige Permutation sowie eine zufällige Seitenzuordnung. Dabei wird die Seite jeder Kante zufällig gleichverteilt ausgewählt. Die zufällige Permutation entsteht durch mehrfaches „Mischen“ der Reihenfolge $0, ..., |V| - 1$. In jedem Schritt werden dabei zwei zufällig gleichverteilt ausgewählte Positionen vertauscht. Sobald Permutation und Seitenzuordnung feststehen, wird die Buchkreuzungszahl der repräsentierten 2-seitigen Buchzeichnung berechnet. Der Vorgang wiederholt sich, bis eine Terminierungsbedingung erfüllt ist.
Die randomisierte Suche wird nach 50000 Funktionsauswertungen abgebrochen, was als ausreichend genug betrachtet werden kann. Damit ist zumindest die Tendenz der mit einer randomisierten Suche erzielbaren Ergebnisse einsehbar. Abbildung 4.28 stellt die Resultate für neun beispielhafte Graphen aus den drei Klassen gegenüber.

Es zeigt sich, dass die randomisierte Suche bei jedem der getesteten Graphen schlechtere Ergebnisse hervorruft. Anzumerken ist, dass die implementierte Suche im Gegensatz zu den ACO Algorithmen kein Verbesserungsverfahren darstellt.

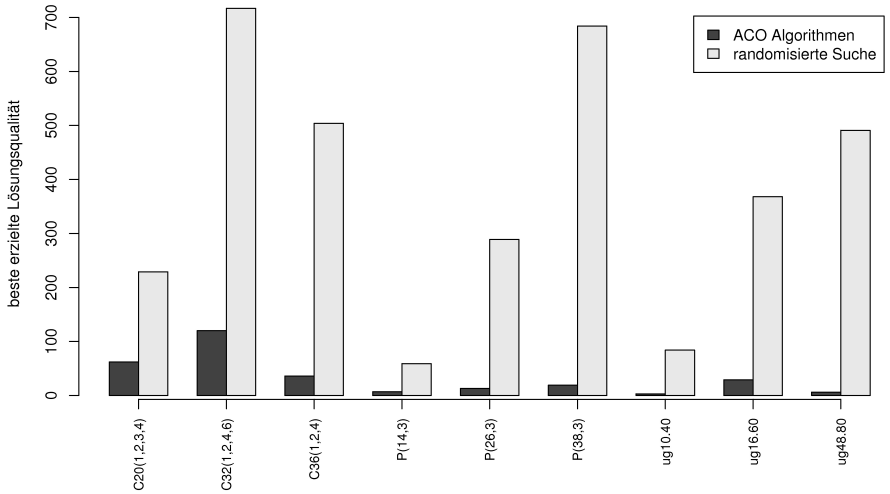


Abbildung 4.28: Vergleich der besten von einer randomisierten Suche und den ACO Algorithmen erzielten 2-seitigen Buchkreuzungszahlen

5 Zusammenfassung und Ausblick

Im Rahmen dieser Diplomarbeit wurden ACO Algorithmen zur Optimierung des k-seitigen BCNP entwickelt. Für die hybriden Optimierungsansätze sind zwei Heuristiken auf das allgemeine Problem mit k Seiten erweitert und eingesetzt worden. Zusätzlich wurde ein Konstruktionsgraph entwickelt, der die direkte Generierung von Lösungen durch einen ACO Algorithmus ermöglicht. Diese erforderte die Einführung zweier weiterer Parameter mit der Bezeichnung α_2 und β_2.

Es hat sich herausgestellt, dass die erreichbare Buchkreuzungszahl für $k > 2$ schnell auf den Wert 0 fällt und somit ein Optimum darstellt. Dann ist das Problem einfach zu lösen und nahezu alle Algorithmen bringen gute Resultate hervor, da sie nicht entsprechend gefordert werden. Ein Vergleich ist nur möglich, wenn die Probleminstanzen bereits eine hohe 2-seitige Buchkreuzungszahl aufweisen oder aber entsprechend groß sind. Der Rechenaufwand erhöht sich jedoch ebenfalls mit steigender Graphengröße. Aus diesem Grund mussten die in dieser Arbeit betrachteten Probleminstanzen auf bestimmte Größen beschränkt und eine Betrachtung für $k = 3$ bzw. $k = 4$ konnte ausschließlich exemplarisch realisiert werden.
Insgesamt können die Verfahren als rechenaufwendig betrachtet werden. Daran sind maßgeblich die Lösungskonstruktion, die Heuristiken sowie die Auswertung der Buchkreuzungszahl beteiligt. Eine Beschleunigung ließe sich durch den Einsatz von Kandidatenlisten erreichen. Diese hätten Auswirkungen auf den Prozess der Lösungskonstruktion. In welchem Maße diese zur Performancesteigerung beitragen würden, ist jedoch unklar und wurde hier nicht untersucht.

Um ungünstige Parametrisierungen auszuschließen, sind die Parametersätze für jede Optimierungsvariante auf bestimmten Probleminstanzen von der SPO-Toolbox optimiert worden.
Bei Betrachtung der optimierten Parametersätze stellte sich eine Einstellung von β bzw. β_2 auf einen Wert größer als 5 als sinnvoll heraus. Im ACS ist es unter Betrachtung der optimierten Parametersätze allgemein vorteilhaft, q_0 auf einen Wert im Bereich zwischen 0.7 und 1.0 einzustellen. Somit sollte in der Lösungskonstruktion die beste Alternative mit hoher Wahrscheinlichkeit direkt gewählt werden.
Weitere Parameterempfehlungen konnten sich nicht allgemein herauskristallisieren; die optimierten Parametersätze weisen in weiten Bereichen starke Schwankungen auf. Die durchgeführte Sensitivitätsanalyse verdeutlichte, dass einige Parameter mit mehreren Werten belegt werden können, ohne dass sich die Qualität der Lösungen maßgeblich verändert. Das deutet darauf hin, dass mehrere Parametersätze existieren mögen, mit denen gute Resultate erzielt werden. Offen bleibt, ob Parametersätze bestehen, die auf mehreren Probleminstanzen gute Ergebnisse ermöglichen und somit allgemein empfohlen werden könnten.

In der Ergebnisanalyse aus Kapitel 4.6 zeigte sich, dass das AS auf allen betrachteten Graphenklassen die nachteiligsten Resultate hervorbringt und ein Einsatz damit nicht empfohlen werden kann. Ebenso brachte die hybride Optimierung aus ACO Algorithmen und der *SLOPE*-Heuristik im Vergleich schlechtere Ergebnisse hervor als die Kombination mit der randomisierten Greedy-Heuristik bzw. der Optimierung auf dem Gesamtgraphen. Zwischen

den restlichen Optimierungsvarianten sind in Betrachtung aller Boxplots aus Anhang E keine Präferenzen ersichtlich.
Insgesamt kristallisiert sich der Einsatz des MMAS als empfehlenswert heraus, da es auf keiner Graphenklasse deutliche Schwächen erkennen lässt.

Die Kombination aus ACO Algorithmen und der randomisierten Greedy-Heuristik liefert vergleichbare Resultate zu der Optimierung auf dem Gesamtgraphen, wobei diese erstaunlicherweise mit die besten Ergebnisse ermöglicht. Die in Kapitel 3.1.1 als problematisch vermutete Codierung innerhalb eines Konstruktionsgraphen bestätigt sich demnach nicht. Das wirft die Frage nach dem Grund für dieses Verhalten auf, die jedoch in dieser Arbeit offen bleiben muss.

Im Vergleich zu bislang ermittelten besten 2-seitigen Buchkreuzungszahlen zirkulärer Graphen erweisen sich die erzielten Resultate als konkurrenzfähig. Die bislang bekannten besten Lösungen werden dabei sogar übertroffen. Demnach haben die implementierten Algorithmen das Potential zur Generierung vielversprechender Lösungen. Unberücksichtigt blieb dabei die erforderliche Rechenzeit, da Implementierungen der anderen Verfahren nicht vorliegen und so die Lösungsqualität nicht in Abhängigkeit von der benötigten Rechenzeit verglichen werden konnte.
Als Erweiterung ist der Einsatz einer lokalen Suche in Kombination mit den ACO Algorithmen denkbar. Dieses könnte, wie bereits in Kapitel 3.1 erwähnt, zu einer möglichen Steigerung der Lösungsqualität führen. Zu Bedenken wäre auch die damit einhergehende Rechenzeiterhöhung, so dass die einzusetzende lokale Suche effizient und effektiv arbeiten müsste.

A Symbole

$\mathcal{G}_{BCNP}$	Graph einer Probleminstanz, dessen Kreuzungszahl in einer k-seitigen Buchzeichnung minimiert werden soll
$cr(\mathcal{G})$	(planare) Kreuzungszahl von $\mathcal{G}$
$v_k(\mathcal{G})$	k-seitige Buchkreuzungszahl von $\mathcal{G}$
G_π	Konstruktionsgraph zur Generierung einer Permutation
G_σ	Konstruktionsgraph zur Generierung einer Seitenzuordnung
$G_{\pi,\sigma}$	Konstruktionsgraph zur Generierung einer Permutation und Seitenzuordnung
$\mathcal{C}$	Komponentenmenge eines Konstruktionsgraphen
$\mathcal{C}_E$	Menge der Kantenkomponenten (Komponenten, die eine Kante repräsentieren)
$\mathcal{C}_V$	Menge der Knotenkomponenten (Komponenten, die einen Knoten repräsentieren)
L	Menge der Transitionen zwischen den Komponenten
Ω	Nebenbedingungen für zulässige Zustände
$\mathcal{X}$	Menge aller Zustände
$\tilde{\mathcal{X}}$	Menge der zulässigen Zustände
$\mathcal{S}$	Menge der Lösungskandidaten
$\tilde{\mathcal{S}}$	Menge der zulässigen Lösungen
S^*	Menge der optimalen Lösungen
$f(s)$	Kosten einer Lösung $s \in S$
$\mathcal{N}_i^k$	Zulässige Nachbarschaft der Ameise k an der Komponente i
τ_0	initiale Pheromonkonzentration
τ_{ij}	Pheromonkonzentration auf der Kante (i,j)
$\Delta\tau_{ij}^k$	Von der Ameise k auf der Kante (i,j) abgelegte Pheromonmenge
η_{ij}	Heuristischer Wert der Kante (i,j)
M^k	Gedächtnis der Ameise k
x^k	Zustand der Ameise k
s^k	Lösungsweg der Ameise k, repräsentiert eine (zulässige) Lösung.
p_{ij}^k	Übergangswahrscheinlichkeit der Ameise k, von Komponente i aus zu Komponente j zu wandern

ρ	Verdunstungsrate des Pheromons pro Iteration
α	Parameter für die Gewichtung der Pheromonspuren
α_2	Parameter für die Gewichtung der Pheromonspuren zur Seitenzuordnung innerhalb des Graphen $G_{\pi,\sigma}$. α repräsentiert in diesem Fall die Gewichtung der Pheromonspuren zur Permutationsbestimmung.
β	Parameter für die Gewichtung heuristischer Informationen
β_2	Parameter für die Gewichtung der heuristischen Informationen zur Seitenzuordnung innerhalb des Graphen $G_{\pi,\sigma}$. β repräsentiert in diesem Fall die Gewichtung der heuristischen Informationen zur Permutationsbestimmung.
s^{gb}	beste gefundene Lösung seit Beginn der Optimierung
s^{ib}	beste in der aktuellen Iteration gefundene Lösung
s^{best}	beste Lösung, die für das Update von Pheromonspuren verwendet wird (nur ACS und (HC-)MMAS)
$\Delta\tau_{ij}^{best}$	Auf der Kante (i,j) von der Ameise, welche die Lösung s^{best} erzeugt hat, abgelegte Pheromonmenge
τ_{min}	untere Schranke der Pheromonspuren (nur MMAS)
τ_{max}	obere Schranke der Pheromonspuren (nur MMAS)
p_{best}	Wahrscheinlichkeit, die beste Lösung s^{best} in einem Durchlauf zu konstruieren
avg	durchschnittliche Anzahl von Entscheidungsmöglichkeiten an jeder Komponente
τ_{ij}^{*}	Pheromonkonzentration auf der Kante (i,j) nach Einleiten des PTS
δ	Parameter, der den Einfluss voriger Pheromonwerte im PTS bestimmt
q_0	Wahrscheinlichkeit, mit der in einer Entscheidung direkt die beste Komponente gewählt wird (nur ACS)
ξ	Pheromonverdunstungsrate des *lokalen* Pheromonupdates im ACS
m	Anzahl der Ameisen

B ACO Algorithmen für das BEP

B.1 Vorüberlegungen

Eine zulässige Lösung für das BEP unterscheidet sich von einer (zulässigen) Lösung für das BCNP ausschließlich in der Einführung einer Nebenbedingung bzw. Restriktion: In der generierten k seitigen Buchzeichnung dürfen sich keine zwei Kanten, die auf derselben Seite gezeichnet sind, schneiden. Das ist dann eine k seitige Bucheinbettung.
Zudem ändert sich die Kostenfunktion, die einer zulässigen Lösung nicht mehr Buchkreuzungszahl sondern die Anzahl der benötigten Seiten der repräsentierten Bucheinbettung zuordnet.

Die verschiedenen Optimierungsansätze aus Kapitel 3.1.1 sind nach wie vor anwendbar, wie auch die in Kapitel 3.2 vorgestellten Graphencodierungen. Zur Generierung einer Permutation für das BEP bedarf es keinerlei Änderungen, allerdings sind durch die zusätzliche Nebenbedingung weitere Gedanken zur Generierung der Seitenzuordnung nötig, die jetzt immer eine Bucheinbettung sicherstellen muss.

B.2 Generierung der Seitenzuordnung mit einem ACO Algorithmus

Es ist möglich, die Seitenzuordnung von einem ACO Algorithmus generieren zu lassen. Dazu können die bereits entwickelten Konstruktionsgraphen $\mathcal{G}_\sigma$ und $\mathcal{G}_{\pi,\sigma}$ eingesetzt werden. Die Ameise darf ausschließlich zulässige Lösungen erschaffen. Unter dieser Voraussetzung wird die Lösungskonstruktion dahingehend verändert, dass die Ameise nur Wege auswählt, in deren repräsentierter Buchzeichnung sich keine zwei Kanten in der gleichen Seite überschneiden.
Um solch eine Funktion sicherstellen zu können, muss vorab die Permutation der Knoten bekannt sein. Das ist bei allen hier vorgestellten Optimierungsansätzen der Fall (vgl. Kapitel 3.5).
Die Lösungskonstruktion muss sicherstellen, dass die aktuell zuzuordnende Kante keiner Seite zugeteilt wird, auf der sie (unter der gegebenen Permutation) in der (lokalen) Buchzeichnung eine Kreuzung hervorruft. Der problemabhängige Test definiert also die zulässige Nachbarschaft zu allen Knoten, die sich in der Nachbarschaft des aktuellen Knotens befinden und unter deren Hinzunahme die repräsentierte (Teil-)Buchzeichnung kreuzungsfrei bleibt.

Jedoch kann der Konstruktionsgraph allein nicht sicherstellen, dass die von der Ameise generierte Lösung auch einen vielversprechenden Zielfunktionswert hervorruft. Die Lösungskonstruktion allein kann einen geringen Seitenverbrauch nicht garantieren. Wenn der Ameise allerdings heuristische Informationen zur Verfügung stehen, könnte sie von vornherein zu einer seitensparenden Zuordnung verleitet werden. Das ist möglich, indem etwa Seiten mit geringerer Seitennummer eine höhere heuristische Präferenz zugewiesen wird. Auf diese Weise wird eine Wahl der ersten kreuzungsfreien Seiten mit höherer Wahrscheinlichkeit realisiert, so dass

sich auf diesen mehrere Kanten ansammeln. Die bereits belegten Seiten werden so besser ausgenutzt.

B.3 Hybride Optimierungsansätze

Um einen hybriden Optimierungsansatz zu verfolgen, bei dem die Seitenzuordnung von einer Heuristik generiert wird, muss diese nicht nur für die Erstellung einer Bucheinbettung Sorge tragen. Wie erwähnt ist es zudem von hoher Relevanz, dass die Bucheinbettung eine geringe Seitenanzahl $p(\mathcal{G})$ aufweist. Die Heuristik muss sozusagen „seitensparend“ arbeiten. Dazu eignet sich die in Kapitel 3.4.2 entwickelte k-seitige randomisierte Greedy-Heuristik.

B.3.1 Adaption der randomisierten Greedy-Heuristik auf das BEP

Wie bereits in Kapitel 3.4 dargestellt, arbeitet die randomisierte Greedy-Heuristik „seitensparend“. Sie ist stets in der Lage, eine zulässige Seitenzuordnung für das BEP unter einer gegebenen Knotenpermutation zu generieren. Dazu müssen jedoch genügend noch nicht belegte Seiten zur Verfügung stehen. Jede Kante muss stets einer Seite zugeordnet werden können, auf der sie keine Kreuzung hervorruft. Diese Forderung ist unabhängig von der randomisierten Kantenreihenfolge einzuhalten.

Die trivialste obere Schranke der Seitenanzahl $p(\mathcal{G})$ eines Graphen $\mathcal{G} = (V, E)$ ist durch die Ungleichung

$$p(\mathcal{G}) \leq |E| \tag{B.1}$$

gegeben. Wenn jede Kante allein auf einer Seite gezeichnet wird, können in der Buchzeichnung niemals Kreuzungen entstehen. Die Generierung einer Bucheinbettung ist damit auf triviale Weise möglich.

Die Heuristik wird nach den Ausführungen aus Kapitel 3.4 eine Bucheinbettung erschaffen, sofern ihr genügend freie Seiten zur Verfügung stehen. Sie braucht demnach nicht für das BEP angepasst zu werden. Es muss lediglich gestattet sein, eine ausreichend große Anzahl von Seiten zu belegen. Setzt man den Parameter k auf den Wert $|E|$, so generiert diese stets eine Bucheinbettung mit möglichst wenigen Seiten.

Auch hierbei kann die Heuristik mehrfach gestartet werden, um verschiedene Reihenfolgen zu betrachten und das beste Ergebnis zur Auswertung heranzuziehen.

C Implementierungsaspekte

Die ACO Algorithmen sind in der Programmiersprache Java[1] implementiert worden. Generell ist in hohem Maße auf effiziente Datenstrukturen zu achten, damit der Einsatz auf größeren Graphen praktisch ermöglicht werden kann.
In ersten Stadien der Implementierung wurden die Algorithmen in ein Programm mit einer Java3D Umgebung integriert (siehe Anhang C.2). Damit konnten die Konstruktionsgraphen, Lösungswege von Ameisen sowie Buchzeichnungen grafisch dargestellt werden. Es zeigte sich, dass die Implementierung auf größeren Probleminstanzen eine hohe Rechenleistung erforderte und nicht am heimischen Rechner einsetzbar war. Dieser Grund erforderte den Einsatz in einer vom Lehrstuhl zur Verfügung gestellten Testplattform. In diesen Implementierungen wurde die grafische Oberfläche entfernt und diverse Veränderungen zur Beschleunigung realisiert. Dazu zählte u.a. die Verwaltung der Instanzgraphen als Kantenliste und eine veränderte Berechnung der k-seitigen Buchkreuzungszahl entsprechend Algorithmus 3.1.2, die vor allem bei Instanzgraphen mit geringem Durchschnittsgrad Vorteile brachte.
Die beste k-seitige Buchzeichnung einer Optimierung wird jeweils in einer Datei gespeichert. Diese kann mit der grafischen Oberfläche eingelesen und dargestellt werden.

C.1 Interne Repräsentation der Graphen

Graphen der Probleminstanz

Jeder Graph ist in einer Datei abgespeichert, die dessen Struktur beinhaltet. Der Graph einer Probleminstanz $G_{BCNP} = (V, E)$ wird aus der entsprechenden Datei eingelesen und in einer Kantenliste gespeichert. Dieses ermöglicht eine effiziente Anwendung von Algorithmus 3.1.2 und 3.15. Anhand dieser Struktur werden die Konstruktionsgraphen für diese Probleminstanz aufgestellt. Es ist sinnvoll, innerhalb dieser Datenstruktur gleichzeitig die Repräsentation der Buchzeichnung zu ermöglichen. Damit können die eingesetzten Verfahren die Zeichnung des Graphen entsprechend ihrer Lösung verändern und die Buchkreuzungszahl ermitteln. Dazu muss zusätzlich ein Integer-Array angelegt werden, das die Permutation der Knoten $v \in V$ angibt. Ein Knoten wird dabei anhand einer Knotennummer $\in \{0..|V|-1\}$ identifiziert.
Ein Objekt in der Kantenliste speichert die zwei verbundenen Knoten und die Seite, auf der die Kante in der k-seitigen Buchzeichnung eingezeichnet ist. Die Seitenzuordnung σ wird also direkt in den Kantenobjekten gespeichert.
Die Berechnung der Buchkreuzungszahl basiert auf der gespeicherten Permutation und der repräsentierten Seitenzuordnung.

Während des Einlesens aus einer Datei werden alle Kanten zunächst der ersten Seite zugeordnet und die Permutation mit der Reihenfolge $(0, 1, ..., |V|-1)$ initialisiert. Die Aufstellung des Konstruktionsgraphen nutzt diese Informationen nicht. Die Zeichnung wird von den

[1] http://java.sun.com

Optimierungsverfahren entsprechend der generierten Lösung verändert. Die Berechnung der k-seitigen Buchkreuzungszahl greift auf diese Daten zu.

Konstruktionsgraphen

Konstruktionsgraphen werden als *Adjazenzliste* [Tur04, S.26f] verwaltet. Vorteile bietet diese Datenstruktur für die Effizienz der Lösungskonstruktion einer Ameise; sie muss während des Laufs mehrfach die zulässige Nachbarschaft einer Komponente ermitteln. Dazu ist es erforderlich, die mit der aktuellen Komponente verbundenen Elemente effizient bestimmen zu können. Eine Adjazenzliste ist hierzu die geeignetste Repräsentation. Für jede Komponente wird eine Liste mit verbundenen Komponenten verwaltet. Zu Beginn der Optimierung wird die Struktur (basierend auf dem Graphen einer Probleminstanz) aufgebaut und während der Lösungskonstruktion nicht mehr verändert, sondern ausschließlich ausgelesen. Eine Komponente ist als Objekt implementiert, das alle in dieser Arbeit beschriebenen Komponenteneigenschaften realisiert.

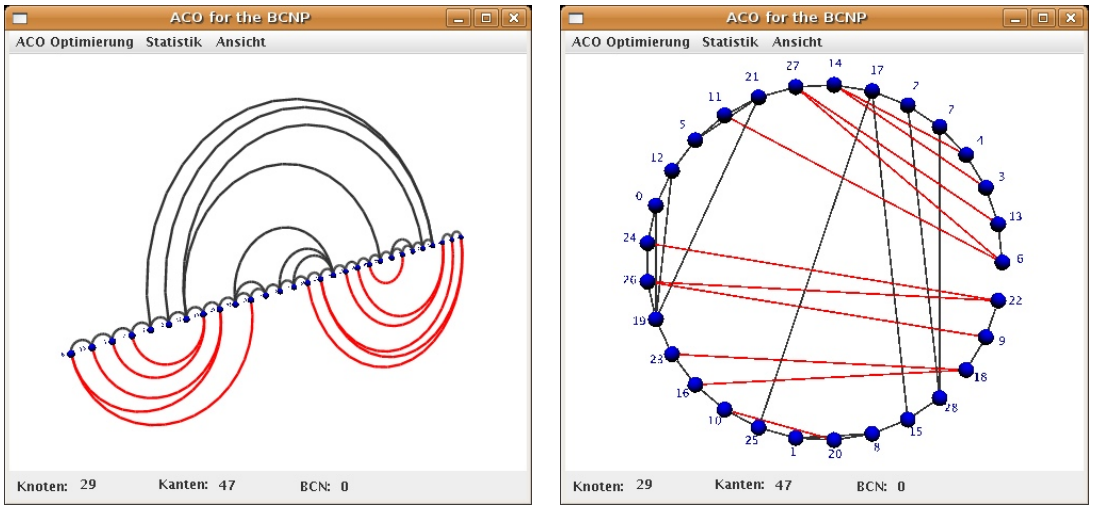
C.2 Java3D Umgebung zur Visualisierung

Zur Visualisierung der erzielten Resultate ist eine *Java3D*[2]-Umgebung entwickelt worden. Sie ist in der Lage, die Instanzgraphen als k-seitige Buch- und Kreiszeichnung grafisch darzustellen. Zudem können die Konstruktionsgraphen $\mathcal{G}_\pi$, $\mathcal{G}_\sigma$ und $\mathcal{G}_{\pi,\sigma}$ gezeichnet werden, wahlweise vollständig oder unter Angabe des Lösungsweges einer Ameise.
Nahezu alle Graphenzeichnungen in dieser Arbeit sind Visualisierungen der implementierten *Java3D*-Umgebung.

Während der Optimierungsläufe werden die k-seitigen Buchzeichnungen der besten Lösungen in jeweils einer Datei gespeichert. Das Programm ist in der Lage, diese (gespeicherten) Buchzeichnungen nachträglich auszulesen und grafisch darzustellen.

Abbildung C.1 zeigt die Visualsierung von Optimierungsergebnissen, wahlweise als Buch- oder als äquivalente Kreiszeichnung. Beispiele für die Darstellung von Lösungswegen, die eine Permutation bzw. eine Seitenzuordnung codieren, liefert Abbildung C.2.

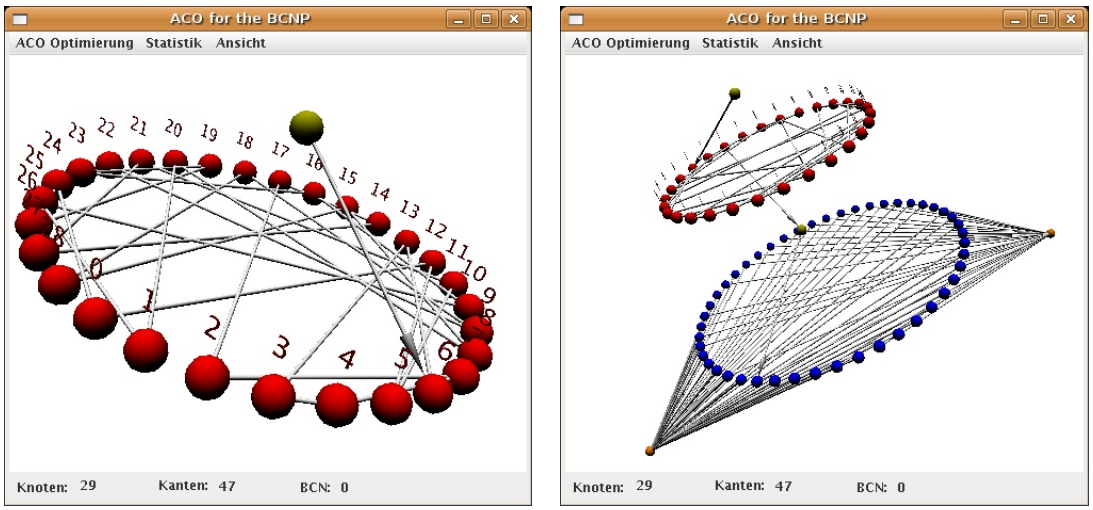
[2] https://java3d.dev.java.net



(a) 2-seitige Buchzeichnung eines Graphen

(b) Äquivalente Kreiszeichnung für die Buchzeichnung aus C.1(a)

Abbildung C.1: Darstellung von Lösungen für das 2-seitige BCNP



(a) Lösungsweg auf einem Graphen $\mathcal{G}_\pi$

(b) Lösungsweg auf einem Graphen $\mathcal{G}_{\pi,\sigma}$

Abbildung C.2: Darstellung von Lösungswegen auf bestimmten Konstruktionsgraphen

D Optimierte Parametereinstellungen

In diesem Anhang sind die von der SPOT optimierten Parametereinstellungen (Kapitel 4.4) tabellarisch für jeden ACO Algorithmus zusammengefasst. Dabei bedeutet der Zusatz „+GR“ die hybride Optimierung aus ACO Algorithmus zur Generierung einer Permutation auf dem Konstruktionsgraphen $\mathcal{G}_\pi$ in Kombination mit der randomisierten Greedy-Heuristik für eine Seitenzuordnung. „+SL“ kennzeichnet die Kombination mit der *SLOPE*-Heuristik.

Die Parameter wurden für $k = 2$ optimiert. Die letzten beiden Zeilen stellen optimierte Parametersätze für den Graphen $\mathcal{C}_{24}(1, 3, 5, 7)$ mit $k = 3$ bzw. $k = 4$ dar und sind daher getrennt aufgeführt.

Graph	α	β	ρ	τ_0	Q
$\mathcal{C}_{20}(1,2,3)$	1,47	2,82	0,49	0,76	4
$\mathcal{C}_{20}(1,2,3,4)$	1,2	19,69	0,49	0,32	11
$\mathcal{C}_{24}(1,3,5,7)$	1,09	12,58	0,03	0,94	3
$\mathcal{C}_{26}(1,3,5)$	1,1	14,42	0,09	0,98	75
$\mathcal{C}_{28}(1,3)$	1,2	19,69	0,49	0,32	11
$\mathcal{C}_{30}(1,3,5,8)$	1,3	12,28	0,49	0,95	81
$\mathcal{C}_{32}(1,2,4,6)$	1,08	17,19	0,04	0,08	6
$\mathcal{C}_{34}(1,3,5)$	1,34	19,45	0,43	0,001	96
$\mathcal{C}_{36}(1,2,4)$	1	12,03	0,02	0,37	78
$\mathcal{C}_{38}(1,4,7)$	1,04	19,08	0,5	0,48	81
$\mathcal{C}_{42}(1,4)$	1,18	6,33	0,5	0,89	21
$P(14,3)$	1,34	2,78	0,43	0,98	48
$P(18,3)$	1,91	16,63	0,04	0,11	94
$P(20,3)$	2,05	7,51	0,31	0,26	71
$P(23,3)$	2,46	19,39	0,39	0,94	18
$P(26,3)$	2,84	17,99	0,02	0,67	72
$P(28,3)$	1,8	16,85	0,28	0,85	82
$P(30,3)$	1,51	4,94	0,44	0,89	2
$P(32,3)$	1,8	16,85	0,28	0,85	82
$P(34,3)$	2,86	15,53	0,34	0,77	80
$P(38,3)$	2,41	7,34	0,02	0,73	99
$P(44,3)$	1,88	11,09	0,1	0,98	87
$ug25.30$	1,34	2,78	0,43	0,98	48
$ug10.40$	1,55	5,83	0,43	0,53	55
$ug82.45$	1,14	9,1	0,48	0,76	4
$ug38.50$	1,33	15,07	0,48	0,57	88
$ug23.55$	1,2	19,69	0,49	0,32	11
$ug16.60$	1,18	11,41	0,49	1	56
$ug76.65$	1,8	16,85	0,28	0,85	82
$ug63.70$	2,01	9,94	0,47	0,25	3
$ug48.80$	1,34	2,78	0,43	0,98	48
$\mathcal{C}_{24}(1,3,5,7)$ (k=3)	1,14	16,75	0,48	0,31	81
$\mathcal{C}_{24}(1,3,5,7)$ (k=4)	1,01	15,55	0,12	0,41	60

Tabelle D.1: Optimierte Parameter für AS+GR

Graph	α	β	ρ	τ_0	Q
$\mathcal{C}_{20}(1,2,3)$	1,48	7,56	0,07	0,65	98
$\mathcal{C}_{20}(1,2,3,4)$	1,07	17,95	0,3	0,69	1
$\mathcal{C}_{24}(1,3,5,7)$	1,08	8,65	0,48	0,58	81
$\mathcal{C}_{26}(1,3,5)$	1,08	14,11	0,2	0,75	85
$\mathcal{C}_{28}(1,3)$	1,02	7,3	0,02	0,23	91
$\mathcal{C}_{30}(1,3,5,8)$	1,03	15,29	0,23	0,001	14
$\mathcal{C}_{32}(1,2,4,6)$	1,04	14,05	0,47	0,65	64
$\mathcal{C}_{34}(1,3,5)$	1,11	17,38	0,49	0,01	22
$\mathcal{C}_{36}(1,2,4)$	4,38	5,05	0,05	0,13	66
$\mathcal{C}_{38}(1,4,7)$	1,2	19,69	0,49	0,32	11
$\mathcal{C}_{42}(1,4)$	1,44	14,77	0,08	0,64	58
$P(14,3)$	1,44	14,77	0,08	0,64	58
$P(18,3)$	1,72	3,72	0,03	0,88	72
$P(20,3)$	1,36	15,53	0,48	0,9	86
$P(23,3)$	3,61	18,64	0,5	0,57	4
$P(26,3)$	1,72	3,72	0,03	0,88	72
$P(28,3)$	3,13	11,19	0,06	0,55	93
$P(30,3)$	1,85	8,94	0,02	0,66	93
$P(32,3)$	1,62	7,28	0,49	0,56	4
$P(34,3)$	1,53	12,43	0,49	0,91	76
$P(38,3)$	2	9,66	0,15	0,82	45
$P(44,3)$	1,95	12,19	0,05	0,95	84
$ug25.30$	1,34	2,78	0,43	0,98	48
$ug10.40$	1,44	14,77	0,08	0,64	58
$ug82.45$	1,55	5,83	0,43	0,53	55
$ug38.50$	1,17	5,82	0,47	0,71	2
$ug23.55$	1,46	12,97	0,5	0,61	3
$ug16.60$	1,2	19,69	0,49	0,32	11
$ug76.65$	2,02	12,5	0,05	0,03	50
$ug63.70$	2,7	14,26	0,02	0,89	96
$ug48.80$	1,57	10,54	0,17	0,87	33
$\mathcal{C}_{24}(1,3,5,7)$ (k=3)	1,02	11,36	0,43	0,25	35
$\mathcal{C}_{24}(1,3,5,7)$ (k=4)	1,14	5,14	0,02	0,41	97

Tabelle D.2: Optimierte Parameter für AS+SL

Graph	α	α_2	β	β_2	ρ	τ_0	Q
$\mathcal{C}_{20}(1,2,3)$	1,36	1,95	4,77	18	0,12	0,03	89
$\mathcal{C}_{20}(1,2,3,4)$	1,01	2,65	12,33	18,25	0,12	0,1	40
$\mathcal{C}_{24}(1,3,5,7)$	1,1	2,52	19,67	18,86	0,1	0,01	90
$\mathcal{C}_{26}(1,3,5)$	1,36	1,19	17,48	9,21	0,39	0,36	30
$\mathcal{C}_{28}(1,3)$	1,3	2,63	2,8	18,31	0,25	0,02	91
$\mathcal{C}_{30}(1,3,5,8)$	1,34	1,04	18,86	15,6	0,47	0,39	94
$\mathcal{C}_{32}(1,2,4,6)$	2,63	2,39	12,57	10,54	0,03	0,15	57
$\mathcal{C}_{34}(1,3,5)$	1,03	1,25	12,01	13,14	0,14	0,09	71
$\mathcal{C}_{36}(1,2,4)$	2,63	2,39	12,57	10,54	0,03	0,15	57
$\mathcal{C}_{38}(1,4,7)$	1,39	1,23	2,76	14,35	0,02	0,02	70
$\mathcal{C}_{42}(1,4)$	1,36	1,19	17,48	9,21	0,39	0,36	30
$P(14,3)$	1,45	1,04	13,2	12,5	0,42	0,7	2
$P(18,3)$	1,06	1,1	5,02	7,3	0,2	0,06	21
$P(20,3)$	1,06	1,1	5,02	7,3	0,2	0,06	21
$P(23,3)$	3,13	1,28	4,06	18,59	0,05	0,91	18
$P(26,3)$	2,65	1,41	8,3	15,93	0,06	0,91	34
$P(28,3)$	2,19	1,53	14,91	17,92	0,04	0,97	92
$P(30,3)$	3,19	4,01	13,87	19,95	0,4	0,29	36
$P(32,3)$	2,29	2,6	15,27	18,47	0,02	0,97	56
$P(34,3)$	1,71	1,09	18,14	3,14	0,31	0,98	14
$P(38,3)$	2,59	1,31	17,88	13,91	0,16	0,63	47
$P(44,3)$	1,74	2,84	8,73	17,29	0,12	0,77	88
$ug25.30$	1,39	1,32	10,26	10,26	0,23	0,91	14
$ug10.40$	1,24	2,84	5,9	17,37	0,19	0,87	5
$ug82.45$	1,36	1,19	17,48	9,21	0,39	0,36	30
$ug38.50$	1,94	3,93	7,23	18,13	0,08	0,95	82
$ug23.55$	1,19	1,39	11,78	10,23	0,46	0,46	7
$ug16.60$	1,24	2,84	5,9	17,37	0,19	0,87	5
$ug76.65$	2,02	4,08	14,72	17,94	0,02	0,96	55
$ug63.70$	2,41	1,58	11,24	19,4	0,24	0,73	12
$ug48.80$	3,04	1,78	15,62	14,15	0,1	0,2	98
$\mathcal{C}_{24}(1,3,5,7)$ (k=3)	1,12	2,09	17,97	16,29	0,21	0,12	99
$\mathcal{C}_{24}(1,3,5,7)$ (k=4)	2,09	1,57	1,1	15,16	0,42	0,28	31

Tabelle D.3: Optimierte Parameter für das AS auf dem Gesamtgraphen $\mathcal{G}_{\pi,\sigma}$

Graph	α	β	ρ	p_{best}	δ
$\mathcal{C}_{20}(1,2,3)$	1,44	14,77	0,08	0,32	0,58
$\mathcal{C}_{20}(1,2,3,4)$	1,11	13,77	0,14	0,11	0,95
$\mathcal{C}_{24}(1,3,5,7)$	1,57	10,54	0,17	0,43	0,33
$\mathcal{C}_{26}(1,3,5)$	1,34	19,5	0,46	0,46	0,1
$\mathcal{C}_{28}(1,3)$	1,11	11,54	0,47	0,08	0
$\mathcal{C}_{30}(1,3,5,8)$	1,43	16,77	0,06	0,02	0,5
$\mathcal{C}_{32}(1,2,4,6)$	1,49	7,08	0,03	0,35	0,94
$\mathcal{C}_{34}(1,3,5)$	1,57	10,54	0,17	0,43	0,33
$\mathcal{C}_{36}(1,2,4)$	1,57	10,54	0,17	0,43	0,33
$\mathcal{C}_{38}(1,4,7)$	2	9,66	0,15	0,41	0,45
$\mathcal{C}_{42}(1,4)$	1,2	19,69	0,49	0,16	0,1
$P(14,3)$	2,49	9,93	0,3	0,08	0,94
$P(18,3)$	1,44	14,77	0,08	0,32	0,58
$P(20,3)$	1,74	14,01	0,28	0,06	0,54
$P(23,3)$	1,2	19,69	0,49	0,16	0,1
$P(26,3)$	1,79	11,17	0,47	0,1	0,42
$P(28,3)$	2,86	12,65	0,49	0,48	0,49
$P(30,3)$	2,16	18,79	0,49	0,04	0,45
$P(32,3)$	2,82	12,46	0,49	0,47	0,51
$P(34,3)$	1,79	11,17	0,47	0,1	0,42
$P(38,3)$	1,57	10,54	0,17	0,43	0,33
$P(44,3)$	1,85	15,56	0,49	0,07	0,35
$ug25.30$	1,79	11,17	0,47	0,1	0,42
$ug10.40$	2,46	19,39	0,39	0,47	0,18
$ug82.45$	2,49	9,93	0,3	0,08	0,94
$ug38.50$	4,12	12,98	0,2	0,13	0,68
$ug23.55$	2,41	7,34	0,02	0,36	0,99
$ug16.60$	1,44	14,77	0,08	0,32	0,58
$ug76.65$	2	9,66	0,15	0,41	0,45
$ug63.70$	2,04	12,81	0,02	0,03	0,61
$ug48.80$	2,28	11,47	0,11	0,21	0,16
$\mathcal{C}_{24}(1,3,5,7)$ (k=3)	1,44	14,77	0,08	0,32	0,58
$\mathcal{C}_{24}(1,3,5,7)$ (k=4)	2,67	7,97	0,27	0,03	0,25

Tabelle D.4: Optimierte Parameter für MMAS+GR

Graph	α	β	ρ	p_{best}	δ
$\mathcal{C}_{20}(1,2,3)$	3,51	19,77	0,49	0,47	0,43
$\mathcal{C}_{20}(1,2,3,4)$	1,14	9,1	0,48	0,38	0,03
$\mathcal{C}_{24}(1,3,5,7)$	1,33	13,79	0,08	0,35	0,75
$\mathcal{C}_{26}(1,3,5)$	1,55	14,55	0,37	0,24	0,51
$\mathcal{C}_{28}(1,3)$	2,05	7,51	0,31	0,13	0,71
$\mathcal{C}_{30}(1,3,5,8)$	1,44	14,77	0,08	0,32	0,58
$\mathcal{C}_{32}(1,2,4,6)$	2	9,66	0,15	0,41	0,45
$\mathcal{C}_{34}(1,3,5)$	1,36	10,62	0,03	0,45	0,56
$\mathcal{C}_{36}(1,2,4)$	3,95	17,92	0,48	0,004	0,41
$\mathcal{C}_{38}(1,4,7)$	2	9,66	0,15	0,41	0,45
$\mathcal{C}_{42}(1,4)$	1,2	19,69	0,49	0,16	0,1
$P(14,3)$	2,19	14,56	0,41	0,01	0,99
$P(18,3)$	2,92	14,62	0,35	0,28	0,75
$P(20,3)$	1,66	10,71	0,37	0,43	0,96
$P(23,3)$	3,2	8,77	0,25	0,22	0,87
$P(26,3)$	1,44	14,77	0,08	0,32	0,58
$P(28,3)$	1,11	13,77	0,14	0,11	0,95
$P(30,3)$	2,86	12,65	0,49	0,48	0,49
$P(32,3)$	1,57	10,54	0,17	0,43	0,33
$P(34,3)$	2,82	12,46	0,49	0,47	0,51
$P(38,3)$	1,2	19,69	0,49	0,16	0,1
$P(44,3)$	1,2	19,69	0,49	0,16	0,1
$ug25.30$	4,87	16,03	0,23	0,36	0,79
$ug10.40$	2,67	7,97	0,27	0,03	0,25
$ug82.45$	3,88	5,98	0,37	0,3	0,5
$ug38.50$	2,97	12,07	0,02	0,43	0,99
$ug23.55$	2,49	10,75	0,02	0,27	0,95
$ug16.60$	1,44	14,77	0,08	0,32	0,58
$ug76.65$	4,03	13,28	0,02	0,27	0,98
$ug63.70$	4,12	12,98	0,2	0,13	0,68
$ug48.80$	4,02	12,64	0,35	0,31	0,23
$\mathcal{C}_{24}(1,3,5,7)$ (k=3)	1,44	14,77	0,08	0,32	0,58
$\mathcal{C}_{24}(1,3,5,7)$ (k=4)	1,57	10,54	0,17	0,43	0,33

Tabelle D.5: Optimierte Parameter für MMAS+SL

Graph	α	α_2	β	β_2	ρ	p_{best}	δ
$\mathcal{C}_{20}(1,2,3)$	1,06	1,1	5,02	7,3	0,2	0,03	0,2
$\mathcal{C}_{20}(1,2,3,4)$	1,96	1,89	17,48	10,22	0,42	0,01	0,16
$\mathcal{C}_{24}(1,3,5,7)$	1,23	2,18	16,63	13,14	0,25	0,13	0,14
$\mathcal{C}_{26}(1,3,5)$	3,19	4,01	13,87	19,95	0,4	0,15	0,35
$\mathcal{C}_{28}(1,3)$	1,06	1,1	5,02	7,3	0,2	0,03	0,2
$\mathcal{C}_{30}(1,3,5,8)$	1,16	3,21	4,82	19,81	0,13	0,04	0,52
$\mathcal{C}_{32}(1,2,4,6)$	1,24	2,84	5,9	17,37	0,19	0,44	0,04
$\mathcal{C}_{34}(1,3,5)$	1,51	2,96	7,93	19,68	0,04	0,23	0,5
$\mathcal{C}_{36}(1,2,4)$	1,06	1,1	5,02	7,3	0,2	0,03	0,2
$\mathcal{C}_{38}(1,4,7)$	2,35	2,91	7,75	15,41	0,06	0,17	0,65
$\mathcal{C}_{42}(1,4)$	1,36	1,19	17,48	9,21	0,39	0,18	0,29
$P(14,3)$	1,06	1,1	5,02	7,3	0,2	0,03	0,2
$P(18,3)$	1,06	1,1	5,02	7,3	0,2	0,03	0,2
$P(20,3)$	2,86	1,48	4,3	12,89	0,28	0,29	0,68
$P(23,3)$	1,26	2	15,44	18,21	0,39	0,29	0,36
$P(26,3)$	4,94	1,09	19,96	4,95	0,42	0,45	0,74
$P(28,3)$	1,13	1,43	15,73	14,66	0,5	0,47	0,23
$P(30,3)$	1,06	1,1	5,02	7,3	0,2	0,03	0,2
$P(32,3)$	1,03	1,63	15,28	18,76	0,26	0,2	0,44
$P(34,3)$	1,36	1,19	17,48	9,21	0,39	0,18	0,29
$P(38,3)$	1,36	1,19	17,48	9,21	0,39	0,18	0,29
$P(44,3)$	1,72	1,93	16,47	12,7	0,39	0,47	0,49
$ug25.30$	4,86	1,91	19,75	14,51	0,01	0,42	0,78
$ug10.40$	3,53	1,64	12,62	11,91	0,11	0,41	0,93
$ug82.45$	1,74	2,84	8,73	17,29	0,12	0,38	0,88
$ug38.50$	1,94	3,93	7,23	18,13	0,08	0,48	0,82
$ug23.55$	3,95	4,48	17,13	16,49	0,34	0,05	0,39
$ug16.60$	2,1	1,65	12,32	13,55	0,36	0,4	0,25
$ug76.65$	1,41	1,31	9,28	15,23	0,23	0,49	0,22
$ug63.70$	1,74	2,84	8,73	17,29	0,12	0,38	0,88
$ug48.80$	1,24	2,84	5,9	17,37	0,19	0,44	0,04
$\mathcal{C}_{24}(1,3,5,7)$ (k=3)	1,06	1,1	5,02	7,3	0,2	0,03	0,2
$\mathcal{C}_{24}(1,3,5,7)$ (k=4)	1,06	1,1	5,02	7,3	0,2	0,03	0,2

Tabelle D.6: Optimierte Parameter für das MMAS auf dem Gesamtgraphen $\mathcal{G}_{\pi,\sigma}$

Graph	β	ρ	q_0	τ_0	ξ
$\mathcal{C}_{20}(1,2,3)$	12,73	0,44	0,47	0,00032	0,18
$\mathcal{C}_{20}(1,2,3,4)$	8,93	0,19	0,54	0,00062	0,13
$\mathcal{C}_{24}(1,3,5,7)$	16,93	0,02	0,97	0,00031	0,15
$\mathcal{C}_{26}(1,3,5)$	7,1	0,24	0,89	0,00396	0,16
$\mathcal{C}_{28}(1,3)$	16,52	0,17	0,6	0,00055	0,21
$\mathcal{C}_{30}(1,3,5,8)$	4,28	0,13	0,73	0,00016	0,03
$\mathcal{C}_{32}(1,2,4,6)$	10,26	0,2	0,7	0,00488	0,04
$\mathcal{C}_{34}(1,3,5)$	15,01	0,01	0,95	0,00759	0,4
$\mathcal{C}_{36}(1,2,4)$	11,27	0,08	0,97	0,00029	0,02
$\mathcal{C}_{38}(1,4,7)$	14,51	0,2	0,96	0,00377	0,14
$\mathcal{C}_{42}(1,4)$	11,69	0,5	0,94	0,00068	0,32
$P(14,3)$	7,08	0,28	0,21	0,00413	0,09
$P(18,3)$	10,48	0,47	0,65	0,00913	0,05
$P(20,3)$	18,79	0,02	0,66	0,00361	0,38
$P(23,3)$	9,32	0,08	0,79	0,00630	0,23
$P(26,3)$	14,67	0,14	0,74	0,00592	0,26
$P(28,3)$	9,56	0,48	0,95	0,00252	0,34
$P(30,3)$	14,49	0,15	0,79	0,00104	0,08
$P(32,3)$	16,94	0,12	0,84	0,00095	0,49
$P(34,3)$	12,85	0,28	0,88	0,00083	0,4
$P(38,3)$	6	0,42	0,95	0,00228	0,08
$P(44,3)$	13,5	0,41	0,94	0,00857	0,03
$ug25.30$	18,79	0,02	0,66	0,00361	0,38
$ug10.40$	19,92	0,29	0,26	0,00010	0,17
$ug82.45$	19,95	0,15	0,72	0,00490	0,08
$ug38.50$	15,37	0,31	0,69	0,00628	0,12
$ug23.55$	13,45	0,29	0,41	0,00304	0,01
$ug16.60$	18,79	0,02	0,66	0,00361	0,38
$ug76.65$	12,3	0,04	0,95	0,00146	0,07
$ug63.70$	7,25	0,06	0,92	0,00440	0,02
$ug48.80$	19,52	0,43	0,08	0,00010	0,03
$\mathcal{C}_{24}(1,3,5,7)$ (k=3)	11,27	0,08	0,97	0,00029	0,02
$\mathcal{C}_{24}(1,3,5,7)$ (k=4)	14,86	0,21	0,99	0,00014	0,19

Tabelle D.7: Optimierte Parameter für ACS+GR

Graph	β	ρ	q_0	τ_0	ξ
$\mathcal{C}_{20}(1,2,3)$	7,92	0,48	0,78	0,00940	0,10
$\mathcal{C}_{20}(1,2,3,4)$	7,92	0,48	0,78	0,00940	0,10
$\mathcal{C}_{24}(1,3,5,7)$	6,98	0,16	0,89	0,00324	0,01
$\mathcal{C}_{26}(1,3,5)$	1,95	0,49	0,98	0,00316	0,06
$\mathcal{C}_{28}(1,3)$	19,98	0,39	0,1	0,00266	0,09
$\mathcal{C}_{30}(1,3,5,8)$	6,59	0,21	0,83	0,00014	0,24
$\mathcal{C}_{32}(1,2,4,6)$	16,63	0,05	0,83	0,00019	0,36
$\mathcal{C}_{34}(1,3,5)$	11,35	0,34	0,83	0,00232	0,47
$\mathcal{C}_{36}(1,2,4)$	13,19	0,07	0,77	0,00007	0,23
$\mathcal{C}_{38}(1,4,7)$	13,4	0,09	0,9	0,00072	0,05
$\mathcal{C}_{42}(1,4)$	4,77	0,27	0,93	0,00198	0,22
$P(14,3)$	7,08	0,28	0,21	0,00413	0,09
$P(18,3)$	9,41	0,09	0,9	0,00568	0,45
$P(20,3)$	18,79	0,02	0,66	0,00361	0,38
$P(23,3)$	19,9	0,28	0,46	0,00031	0,15
$P(26,3)$	13,45	0,29	0,41	0,00304	0,01
$P(28,3)$	19,95	0,15	0,72	0,00490	0,08
$P(30,3)$	15,37	0,31	0,69	0,00628	0,12
$P(32,3)$	10,64	0,39	0,79	0,00622	0,36
$P(34,3)$	13,65	0,11	0,95	0,00969	0,48
$P(38,3)$	9,41	0,09	0,9	0,00568	0,45
$P(44,3)$	9,41	0,09	0,9	0,00568	0,45
$ug25.30$	14,14	0,22	0,53	0,00399	0,17
$ug10.40$	4,77	0,27	0,93	0,00198	0,22
$ug82.45$	7,92	0,48	0,78	0,00940	0,10
$ug38.50$	13,45	0,29	0,41	0,00304	0,01
$ug23.55$	8,73	0,08	0,93	0,00997	0,04
$ug16.60$	10,41	0,06	0,9	0,00580	0,20
$ug76.65$	13,29	0,39	0,88	0,00170	0,05
$ug63.70$	14,51	0,2	0,96	0,00377	0,14
$ug48.80$	13,41	0,23	0,95	0,00090	0,34
$\mathcal{C}_{24}(1,3,5,7)$ (k=3)	5,58	0,03	0,86	0,00041	0,04
$\mathcal{C}_{24}(1,3,5,7)$ (k=4)	12,48	0,16	0,76	0,00012	0,30

Tabelle D.8: Optimierte Parameter für ACS+SL

Graph	β	β_2	ρ	q_0	τ_0	ξ
$\mathcal{C}_{20}(1,2,3)$	14,99	17,54	0,43	0,82	0,00680	0,06
$\mathcal{C}_{20}(1,2,3,4)$	9,51	11,43	0,31	0,8	0,00735	0,08
$\mathcal{C}_{24}(1,3,5,7)$	10,43	15,31	0,42	0,95	0,00101	0,07
$\mathcal{C}_{26}(1,3,5)$	12,25	13,08	0,14	0,89	0,00849	0,11
$\mathcal{C}_{28}(1,3)$	14,37	19,18	0,47	0,34	0,00342	0,02
$\mathcal{C}_{30}(1,3,5,8)$	13,79	17,29	0,42	0,55	0,00006	0,07
$\mathcal{C}_{32}(1,2,4,6)$	8,14	13,52	0,45	0,69	0,00023	0,01
$\mathcal{C}_{34}(1,3,5)$	12,51	14,29	0,35	0,97	0,00243	0,49
$\mathcal{C}_{36}(1,2,4)$	9,26	19,5	0,23	0,92	0,00414	0,23
$\mathcal{C}_{38}(1,4,7)$	6,39	7,91	0,24	0,97	0,00016	0,5
$\mathcal{C}_{42}(1,4)$	6,79	9,84	0,04	0,91	0,00828	0,5
$P(14,3)$	14,99	17,54	0,43	0,82	0,00680	0,06
$P(18,3)$	10,68	4,7	0,39	0,69	0,00179	0,11
$P(20,3)$	14,37	19,18	0,47	0,34	0,00342	0,02
$P(23,3)$	6,56	15,75	0,5	0,97	0,00006	0,38
$P(26,3)$	10,48	18,4	0,28	0,95	0,00729	0,2
$P(28,3)$	16,14	18	0,1	0,8	0,00920	0,04
$P(30,3)$	10,95	18,44	0,17	0,93	0,00988	0,47
$P(32,3)$	12,25	13,08	0,14	0,89	0,00849	0,11
$P(34,3)$	15,89	4,26	0,36	0,89	0,00995	0,24
$P(38,3)$	19,48	7,72	0,28	0,92	0,00930	0,46
$P(44,3)$	10,48	18,4	0,28	0,95	0,00729	0,2
$ug25.30$	19,2	16,6	0,31	0,29	0,00076	0,33
$ug10.40$	3,03	17,4	0,06	0,97	0,00380	0,4
$ug82.45$	12,25	13,08	0,14	0,89	0,00849	0,11
$ug38.50$	18,67	14,94	0,46	0,23	0,00012	0,03
$ug23.55$	7,69	3,74	0,27	0,97	0,00474	0,37
$ug16.60$	10,22	4,87	0,15	0,98	0,00195	0,48
$ug76.65$	8,51	14,43	0,27	0,89	0,00354	0,01
$ug63.70$	15,91	17,94	0,48	0,72	0,00106	0,08
$ug48.80$	16,14	18	0,1	0,8	0,00920	0,04
$\mathcal{C}_{24}(1,3,5,7)$ (k=3)	6,16	7	0,02	0,91	0,00067	0,43
$\mathcal{C}_{24}(1,3,5,7)$ (k=4)	15,17	11,44	0,09	0,99	0,00002	0,28

Tabelle D.9: Optimierte Parameter für das ACS auf dem Gesamtgraphen $\mathcal{G}_{\pi,\sigma}$

E Vollständige Ergebnisse der Algorithmen als Boxplot

In Kapitel 4.6 konnten die Ergebnisse der Optimierungsläufe nur ausschnitthaft dargestellt werden. Dieser Anhang enthält nun die vollständigen Ergebnisse aller eingesetzten Graphen, für die Parametersätze von der SPOT optimiert worden sind.

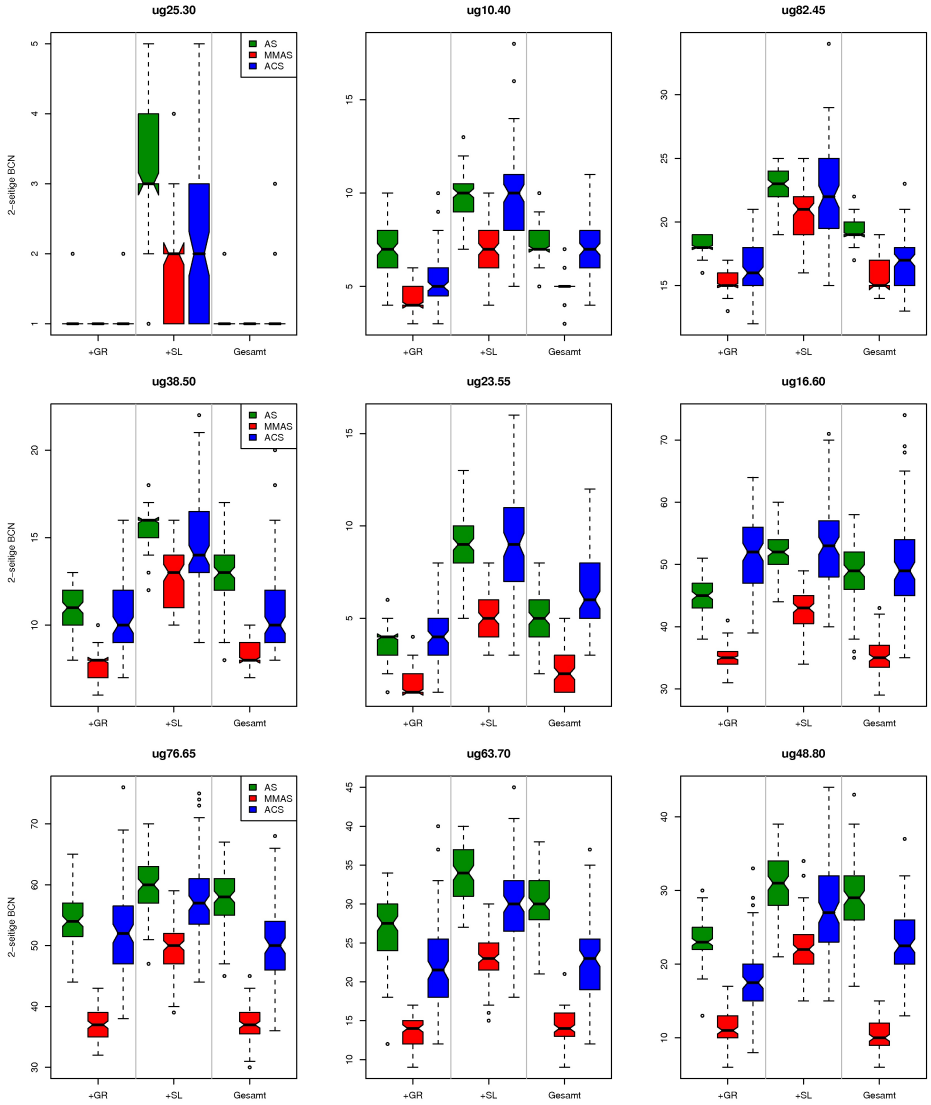


Abbildung E.1: Boxplots von jeweils 100 Läufen mit optimierten Parametern für Rome Graphen

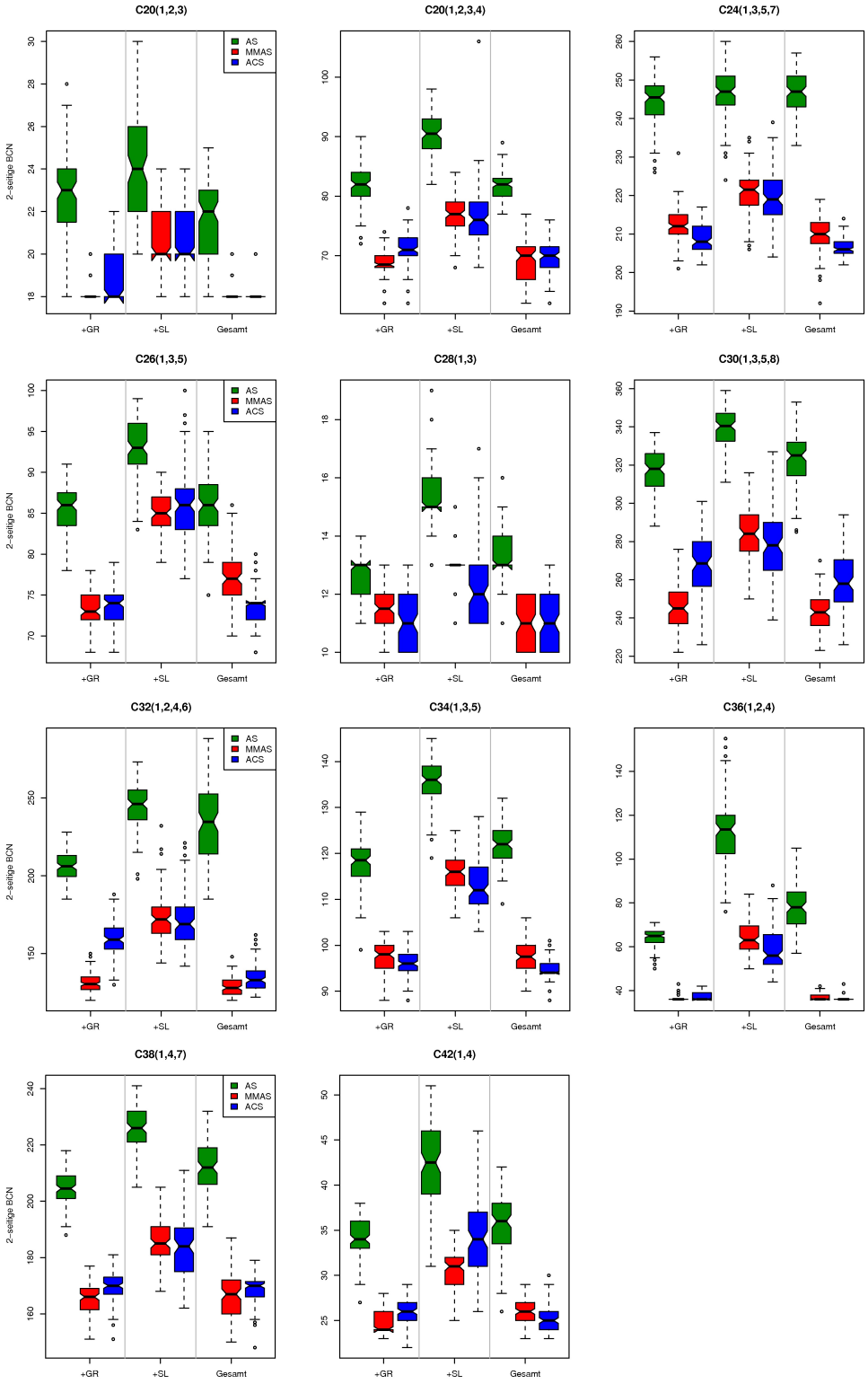


Abbildung E.2: Boxplots von jeweils 100 Läufen mit optimierten Parametern für zirkuläre Graphen

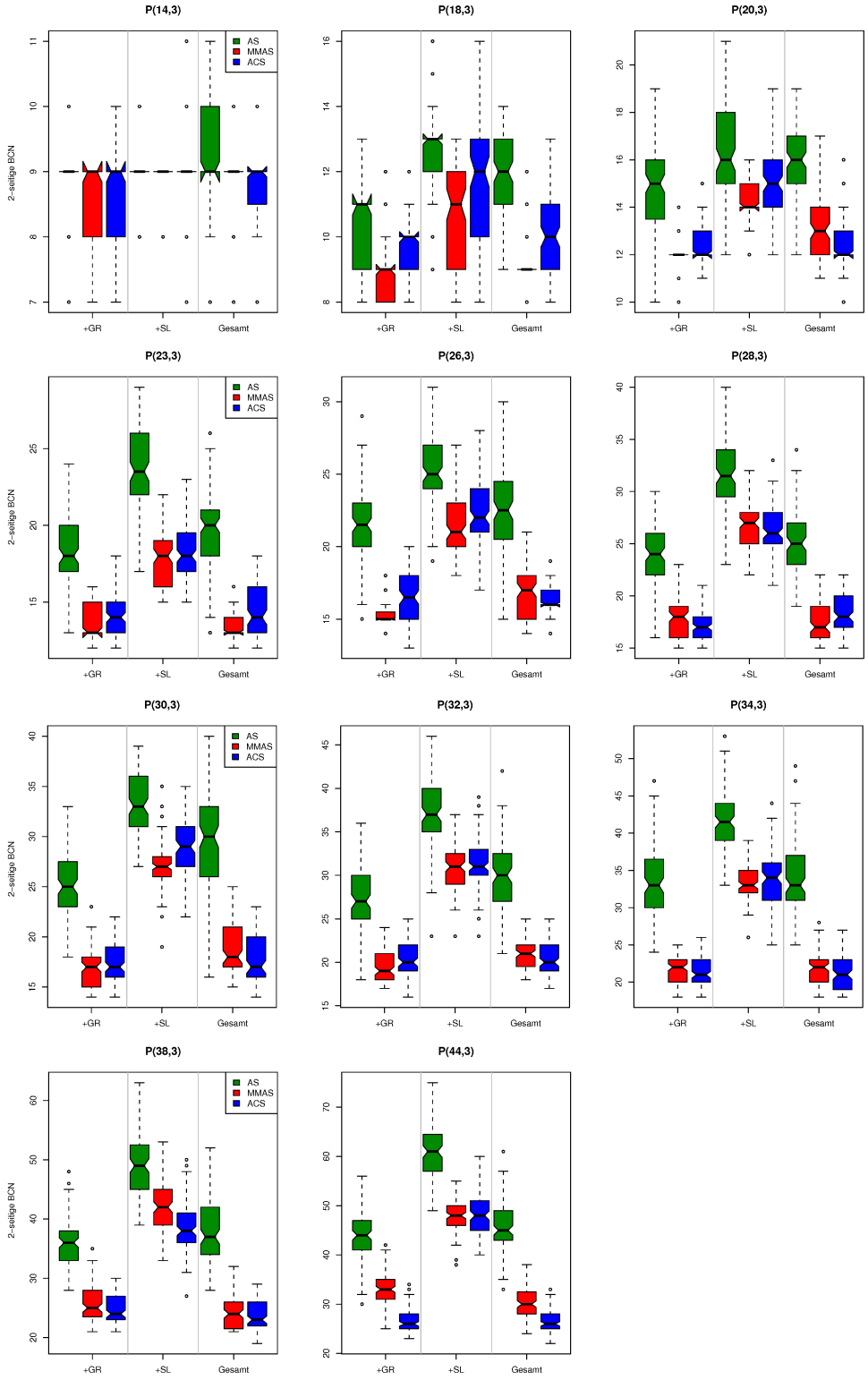


Abbildung E.3: Boxplots von jeweils 100 Läufen mit optimierten Parametern für Petersen Graphen

Literaturverzeichnis

[AL97] Aarts, E.H.L. ; Lenstra, J.K.: Introduction. In: Aarts, E.H.L. (Hrsg.) ; Lenstra, J.K. (Hrsg.): *Local Search in Combinatorial Optimization.* Chicester : Wiley, 1997, S. 1–17

[Arc96] Archdeacon, D.: Topological graph theory: A survey. In: *Congressus Numerantium* 115 (1996), S. 5–54

[BB98] Botee, H.M. ; Bonabeau, E.: Evolving ant colony optimization. In: *Advanced Complex Systems* 1 (1998), S. 149–159

[BB06] Bartz-Beielstein, T.: *Experimental Research in Evolutionary Computation.* Berlin : Springer, 2006

[BBLP06] Bartz-Beielstein, T. ; Lasarczyk, C. ; Preuss, M.: Sequential Parameter Optimization Toolbox / Universität Dortmund. 2006 (CI–15x/06). – Technical Report

[BDT99] Bonabeau, E. ; Dorigo, M. ; Theraulaz, G.: *Swarm Intelligence - From Natural to Artificial Systems.* New York : Oxford University Press, 1999

[BL84] Bhatt, S.N. ; Leighton, F.T.: A framework for solving VLSI graph layout problems. In: *Journal of Computer and System Sciences* 28 (1984), Nr. 2, S. 300–343

[BL85] Bondy, J.A. ; L.Lovàsz: Lengths of cycles in halin graphs. In: *Journal of Graph Theory* 9 (1985), Nr. 1, S. 397–410

[Blu02] Blum, C.: Ant colony optimization for the edge-weighted k-cardinality tree problem. In: Langdon, W. B. (Hrsg.) u.a.: *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2002).* New York : Morgan Kaufman, 2002, S. 27–34

[BRD01] Blum, C. ; Roli, A. ; Dorigo, M.: HC–ACO: The hyper-cube framework for ant colony optimization. In: *Proceedings of the 4th Metaheuristics International Conference (MIC'2001).* Porto, Portugal, 2001, S. 399–403

[BSMM05] Bronstein, I.N. ; Semendjajew, K.A. ; Musiol, G. ; Mühling, H.: *Taschbuch der Mathematik.* 6. Frankfurt a.M. : Harri, 2005

[BT84] Boesch, F. ; Tindell, R.: Circulants and their connectivities. In: *Journal of Graph Theory* 8 (1984), Nr. 4, S. 487–499

[CDM92] Colorni, A. ; Dorigo, M. ; Maniezzo, V.: Distributed optimization by ant colonies. In: Varela, F. (Hrsg.) ; Bourgine, P. (Hrsg.): *Proceedings of the 1st European Conference on Artificial Life (ECAL'91).* Cambridge : MIT Press, 1992, S. 134–142

[CHS02] Cordon, O. ; Herrera, F. ; Stützle, T.: A review on the ant colony optimization metaheuristic: Basis, models and new trends. In: *Mathware and Soft Computing* 9 (2002), Nr. (2-3), S. 141–175

[Cim02] Cimikowski, R.J.: Algorithms for the fixed linear crossing number problem. In: *Discrete Applied Mathematics* 122 (2002), Nr. 1-3, S. 93–115

[CKA00] Campolongo, F. ; Kleijnen, J. ; Andres, T.: Screening methods. In: Saltelli, A. (Hrsg.) ; Chan, K. (Hrsg.) ; Scott, E.M. (Hrsg.): *Sensitivity Analysis.* Chichester : Wiley, 2000, S. 65–80

[CLR87] Chung, F.R.K. ; Leighton, F.T: ; Rosenberg, A.L.: Embedding graphs in books: A layout problem with applications to VLSI design. In: *SIAM Journal on Algebraic and Discrete Methods* 8 (1987), Nr. 1, S. 33–58

[CSST00] Campolongo, F. ; Saltelli, A. ; Sorensen, T. ; Tarantola, S.: Hitchhiker's guide to sensitivity analysis. In: Saltelli, A. (Hrsg.) ; Chan, K. (Hrsg.) ; Scott, E.M. (Hrsg.): *Sensitivity Analysis.* Chichester : Wiley, 2000, S. 15–45

[DAGP90] Deneubourg, J.L. ; Aron, S. ; Goss, S. ; Pasteels, J.M.: The self-organizing exploratory pattern of the argentine ant. In: *Journal of Insect Behavior* 3 (1990), Nr. 2, S. 159–168

[DD99] Dorigo, M. ; Di Caro, G.: The ant colony optimization meta-heuristic. In: Corne, D. (Hrsg.) ; Dorigo, M. (Hrsg.) ; Glover, F. (Hrsg.): *New Ideas in Optimization.* London : McGraw Hill, 1999, S. 11–32

[DG97] Dorigo, M. ; Gambardella, L.M.: Ant colony system: A cooperative learning approach to the traveling salesman problem. In: *IEEE Transactions on Evolutionary Computation* 1 (1997), Nr. 1, S. 53–66

[Die00] Diestel, R.: *Graphentheorie.* Berlin : Springer, 2000

[DMC96] Dorigo, M. ; Maniezzo, V. ; Colorni, A.: The ant system: optimization by a colony of cooperating agents. In: *IEEE Transactions on Systems, Man and Cybernetics–Part B* 26 (1996), Nr. 1, S. 29–41

[DP06] Dréo, J. ; Pétrowski, A.: *Metaheuristics for Hard Optimization.* Berlin : Springer, 2006

[DS02] Dorigo, M. ; Stützle, T.: The ant colony optimization metaheuristic: Algorithms, applications, and advances. In: Glover, F. (Hrsg.) ; Kochenberger, G. (Hrsg.): *Handbook of Metaheuristics* Bd. 57. Boston : Kluwer, 2002, S. 251–285

[DS04] Dorigo, M. ; Stützle, T.: *Ant Colony Optimization.* Cambridge : MIT Press, 2004

[Eng05] Engelbrecht, A.P.: *Fundamentals of Computational Swarm Intelligence.* Chichester : Wiley, 2005

[GADP89] Goss, S. ; Aron, S. ; Deneubourg, J.L. ; Pasteels, J. M.: Self-organized shortcuts in the argentine ant. In: *Naturwissenschaften* 76 (1989), Nr. 12, S. 579–581

[Gam86] Games, R.A.: Optimal book embeddings of the FFT, benes, and barrel shifter networks. In: *Algorithmica* 1 (1986), Nr. 1, S. 233–250

[GJ83] Garey, M.R. ; Johnson, D.S.: Crossing number is NP-complete. In: *SIAM Journal on Algebraic and Discrete Methods* 4 (1983), Nr. 3, S. 312–316

[GT94] Goldschmidt, O. ; Takvorian, A.: An efficient graph planarization two-phase heuristic. In: *Networks* 24 (1994), Nr. 2, S. 69–73

[GWE03] Giunta, A. ; Wojtkiewicz, S. ; Eldred, M.: Overview of modern design of experiments methods for computational simulations. In: *Proceedings of the 41st AIAA Aerospace Sciences Meeting and Exhibit, American Institute of Aeronautics and Astronautics*, 2003. – Artikel AIAA-2003-0649

[Har74] Harary, F.: *Graphentheorie.* München : Oldenbourg, 1974

[Har75] Hartigan, J.A.: *Clustering Algorithms.* New York : Wiley, 1975

[He06] He, H.: *Algorithms for Book Crossing Number Problems*, Loughborough University, United Kingdom, Dissertation, 2006

[Hof97] Hoffmeyer, J.: The swarming body. In: Rauch, I. (Hrsg.) ; Carr, G.F. (Hrsg.): *Proceedings of the 5th Congress of the International Association for Semiotic Studies*, Mouton de Gruyter, 1997 (Semiotics Around the World), S. 937–940

[HS04] He, H. ; Sýkora, O.: New circular drawing algorithms. In: *Proceedings of the Workshop on Information Technologies - Applications and Theory (ITAT 2004).* Slovakia, September 15-19 2004

[HSM07] He, H. ; Sýkora, O. ; Mäkinen, E.: Genetic algorithms for the 2-page book drawing problem of graphs. In: *Journal of Heuristics* 13 (2007), Nr. 1, S. 77–93

[HSV05] He, H. ; Sýkora, O. ; Vrt'o, I.: Crossing Minimisation Heuristics for 2-page Drawings. In: *Electronic Notes in Discrete Mathematics* 22 (2005), S. 527–534

[KE01] Kennedy, J. ; Eberhart, R.C.: *Swarm Intelligence.* San Francisco : Morgan Kaufmann, 2001

[LD04] Levine, J. ; Ducatelle, F.: Ant colony optimization and local search for bin packing and cutting stock problems. In: *Journal of the Operational Research Society* 55 (2004), Nr. 7, S. 705–716

[LS98] Lourenco, H.R. ; Serra, D.: Adaptive Approach Heuristics for the Generalized Assignment Problem / Universitat Pompeu Fabra, Spain. 1998 (Economic Working Papers No. 288). – Technical Report

[Man99] Maniezzo, V.: Exact and approximate nondeterministic tree-search procedures for the quadratic assignment problem. In: *INFORMS Journal on Computing* 11 (1999), Nr. 4, S. 358–369

[MC01] Maniezzo, V. ; Carbonaro, A.: Ant colony optimization: An overview. In: Ribeiro, C.C. (Hrsg.) ; Hansen, P. (Hrsg.): *Essays and Surveys in Metaheuristics.* Boston : Kluwer, 2001, S. 21–44

[MKNF87] Masuda, S. ; Kashiwabara, T. ; Nakajima, K. ; Fujisawa, T.: On the NP-completeness of a computer network layout problem. In: *Proceedings of the IEEE International Symposium on Circuits and Systems*, IEEE Press, 1987, S. 292–295

[MKNF90] Masuda, S. ; Kashiwabara, T. ; Nakajima, K. ; Fujisawa, T.: Crossing minimization in linear embeddings of graphs. In: *IEEE Transactions on Computers* 39 (1990), Nr. 1, S. 124–127

[MM05] Merkle, D. ; Middendorf, M.: Swarm intelligence. In: Burke, E.K. (Hrsg.): *Search Methodologies - Introductory Tutorials in Optimization and Decision Support Techniques.* New York : Springer, 2005, S. 401–429

[OK96] Osman, I.H. ; Kelly, J.P.: Meta-Heuristics: An overview. In: Osman, I.H. (Hrsg.) ; Kelly, J.P. (Hrsg.): *Meta-Heuristics: Theory & applications.* Norwell : Kluwer, 1996, S. 1–21

[PS88] Papadimitriou, C.H. ; Steiglitz, K.: *Combinatorial Optimization: Algorithms and Complexity.* New York : Dover Publications, 1988

[Pur97] Purchase, H.C.: Which aesthetic has the greatest effect on human understanding. In: *Proceedings of the 5th International Symposium on Graph Drawing* Bd. 1353. Berlin : Springer, 1997 (Lecture Notes in Computer Science), S. 248–261

[Ree95] Reeves, C.R.: *Modern Heuristic Techniques for Combinatorial Problems.* Berkshire : McGraw-Hill, 1995

[RR01] Resende, M.G.C. ; Ribeiro, C.C.: Graph planarization. In: *Encyclopedia of Optimization* Bd. 2. Dordrecht : Kluwer, 2001, S. 368–373

[RS02] Richter, R.B. ; Salazar, G.: The crossing number of P(N, 3). In: *Graphs and Combinatorics* 18 (2002), Nr. 2, S. 381–394

[SD99] Stützle, T. ; Dorigo, M.: ACO algorithms for the traveling salesman problem. In: Miettinen, K. (Hrsg.) u. a.: *Evolutionary Algorithms in Engineering and Computer Science.* Chichester : Wiley, 1999, S. 163–183

[SD02] Stützle, T. ; Dorigo, M.: A short convergence proof for a class of ant colony optimization algorithms. In: *IEEE Transactions on Evolutionary Computation* 6 (2002), Nr. 4, S. 358–365

[SH96] Stützle, T. ; Hoos, H.H.: Improving the Ant System: A Detailed Report on the MAX-MIN Ant System / FG Intellektik, TU Darmstadt, Germany. 1996 (AIDA-96-12). – Technical Report

[SH00] Stützle, T. ; Hoos, H.H.: MAX MIN Ant system. In: *Journal of Future Generation Computer Systems* 16 (2000), Nr. 8, S. 889–914

[SKS02] Socha, K. ; Knowles, J.D. ; Sampels, M.: A MAX-MIN ant system for the university course timetabling problem. In: Dorigo, M. (Hrsg.) ; Di Caro, G. (Hrsg.) ; Sampels, M. (Hrsg.): *Ant Algorithms, Third International Workshop (ANTS 2002)* Bd. 2463. Berlin : Springer, 2002 (Lecture Notes in Computer Science), S. 1–13

[SSSV95] Shahrokhi, F. ; Székely, L.A. ; Sýkora, O. ; Vrt'o, I.: Book embeddings and crossing numbers. In: Mayr, E.W. (Hrsg.) ; Schmidt, G. (Hrsg.) ; Tinhofer, G. (Hrsg.): *Proceedings of the 20th International Workshop on Graph-Theoretic Concepts in Computer Science (WG'94)* Bd. 903. Berlin : Springer, 1995 (Lecture Notes in Computer Science), S. 256–268

[SSSV96] Shahrokhi, F. ; Székely, L. ; Sýkora, O. ; Vrt'o, I.: Drawings of graphs on surfaces with few crossings. In: *Algorithmica* 16 (1996), Nr. 1, S. 118–131

[SSSV97] Shahrokhi, F. ; Sýkora, O. ; Székely, L. ; Vrt'o, I.: Crossing numbers: Bounds and applications. In: Bárány, I. (Hrsg.) ; Böröczky, K. (Hrsg.): *Intuitive Geometry*. Budapest : Akadémiai Kiadó, 1997 (Bolyai Society Mathematical Studies 6), S. 179–206

[ST99] Six, J.M. ; Tollis, I.G.: Circular drawings of biconnected graphs. In: Goodrich, M.T. (Hrsg.) ; McGeoch, C.C. (Hrsg.): *Proceedings of the 1st Workshop on Algorithm Engineering and Experimentation (ALENEX '99)* Bd. 1619. Berlin : Springer, 1999 (Lecture Notes in Computer Science), S. 57–73

[Stü98] Stützle, T.: *Local Search Algorithms for Combinatorial Problems - Analysis, Algorithms, and New Applications*, TU Darmstadt, Germany, Dissertation, 1998

[Tur77] Turán, Paul: A note of welcome. In: *Journal of Graph Theory* 1 (1977), Nr. 1, S. 7–9

[Tur04] Turau, V.: *Algorithmische Graphentheorie*. München : Oldenbourg, 2004

[Win05] Winterbach, W.: *The Crossing Number of a Graph in the Plane*, University of Stellenbosch, South Africa, Dissertation, 2005

[Woo02] Wood, David R.: Degree constrained book embeddings. In: *Journal of Algorithms* 45 (2002), Nr. 2, S. 144–154

[Yan89] Yannakakis, M.: Embedding planar graphs in four pages. In: *Journal of Computing and System Sciences* 38 (1989), Nr. 1, S. 36–67

[YI96] Yagiura, M. ; Ibaraki, T.: Genetic and local search algorithms as robust and simple optimization tools. In: Osman, I.H. (Hrsg.) ; Kelly, J.P. (Hrsg.): *Meta-Heuristics: Theory & applications*. Norwell : Kluwer, 1996, S. 63–82