

Ausarbeitung  
zum Thema

**Bienenschwarme  
(Bee Colony Optimization)  
und Anwendungen**

im Rahmen des Seminars  
Verkehr und Logistik

erstellt von  
**Tobias Möllmann**

# Inhaltsverzeichnis

1 Einleitung.....	1
2 Bienen in der Natur.....	1
3 Die künstliche Biene.....	4
4 Lösen des Traveling Salesman Problem mit Bienensystemen.....	5
4.1 Das Traveling Salesman Problem.....	5
4.2 BeeSystem als Lösungsalgorithmus.....	5
4.3 Optimierung des BeeSystem mit k-opt-Algorithmen.....	9
4.4 Testergebnisse.....	9
4.5 Bee Colony Optimization mit Lokaler Suche.....	11
4.6 Bewertung.....	12
5 Ein Routingalgorithmus: BeeHive.....	13
5.1 Idee hinter BeeHive.....	13
5.2 Der Algorithmus.....	13
5.2.1 Grundgedanken.....	13
5.2.2 Netzwerkorganisation.....	14
5.3 Simulation.....	16
5.4 Bewertung.....	18
6 Weitere Bienenschwarmalgorithmen.....	19
7 Vergleich Bienensysteme / Ameisensysteme.....	20
8 Schlusswort.....	21
Literaturverzeichnis.....	21

# 1 Einleitung

Soziale Insekten leben seit Millionen von Jahren auf der Erde und haben im Laufe der Zeit ein ausgeprägtes Verhalten beim Bau von Behausungen oder dem Beschaffen von Nahrung entwickelt. So reagieren sie sehr flexibel auf Veränderungen in ihrer Umwelt, sodass sie trotz erheblicher Störungen überleben können. Die Dynamik der Insekten ergibt sich aus verschiedenen Handlungen und Interaktionen miteinander und mit der Umgebung. Einige Beispiele solcher interaktiven Insekten sind die Ameisen, die Pheromone zur Kommunikation benutzen, oder Bienen, die durch einen Tanz mit ihren Artgenossen kommunizieren. Das Verhalten der letzteren soll im folgenden genauer untersucht werden. Auch wird die Tauglichkeit dieses Systems für die Informatik überprüft.

Zunächst wird dafür das Verhalten der Biene bei der Nahrungssuche in der Natur dargestellt. Im nächsten Kapitel wird beschrieben, welche Konzepte bei der künstlichen Biene von der Natur übernommen werden. In Kapitel vier wird ein von Teodorovic und Lucic entwickelter Algorithmus, der auf dem Verhalten von Bienen basiert, vorgestellt. Es wird beschrieben, wie er zum Lösen des Travelling Salesman Problems eingesetzt werden kann. Im nächsten Abschnitt wird der Telekommunikationsalgorithmus BeeHive erläutert. Im sechsten Kapitel sind bisher entwickelte Algorithmen dargestellt, die auf dem Bienenverhalten beruhen. Anschließend werden Bienenschwarmsysteme mit Ameisensystemen verglichen. Abschließend folgt noch ein Schlusswort über das Thema der Bienenschwarmsysteme.

## 2 Bienen in der Natur

Bienen haben in der Natur ein sehr gut entwickeltes Verhalten bei der Nahrungssuche. Diese Fähigkeit entsteht aus den einfachen Verhaltensweisen der einzelnen Bienen und wenig Kommunikationsaufwand. Dabei nutzt eine Biene meist **lokale Informationen**, um eigene Entscheidungen zu treffen. Dabei spielt der Austausch der Informationen untereinander die wichtigste Rolle. Dieser Informationsaustausch findet auf dem sogenannten **Tanzboden** im Bienenstock statt. An der Art eines Tanzes können andere Bienen erkennen, wo genau eine Nahrungsquelle liegt und wie die Qualität dieser ist. Es gibt dabei zwei verschiedene Tänze:

- **Schwänzeltanz:** Dieser Tanz wird ausgeführt, wenn ein Standort weiter entfernt, aber dennoch ergiebig ist. Je weiter die Trachtquelle dabei entfernt ist, desto langsamer ist der Tanz der Biene. Der Winkel, den die tanzende Biene dabei zur Sonne einhält, gibt die genaue Richtung an, in der gesucht werden muss. So werden Schwerstern über Entfernung – auch von einigen Kilometern – und Richtung der Nahrung bis auf wenige Meter genau informiert.
- **Rundtanz:** Eine Biene vollführt einen Rundtanz, wenn die Nahrungsquelle in der Nähe gelegen ist. Gewöhnlich liegt sie dabei näher als 100 Meter. Anhand des Temperaments der Biene ist die Qualität dieser Quelle erkennbar.



Quelle: [http://www.imkervereintempelhof.de/html/bilder/bie\\_tan\\_2.gif](http://www.imkervereintempelhof.de/html/bilder/bie_tan_2.gif)



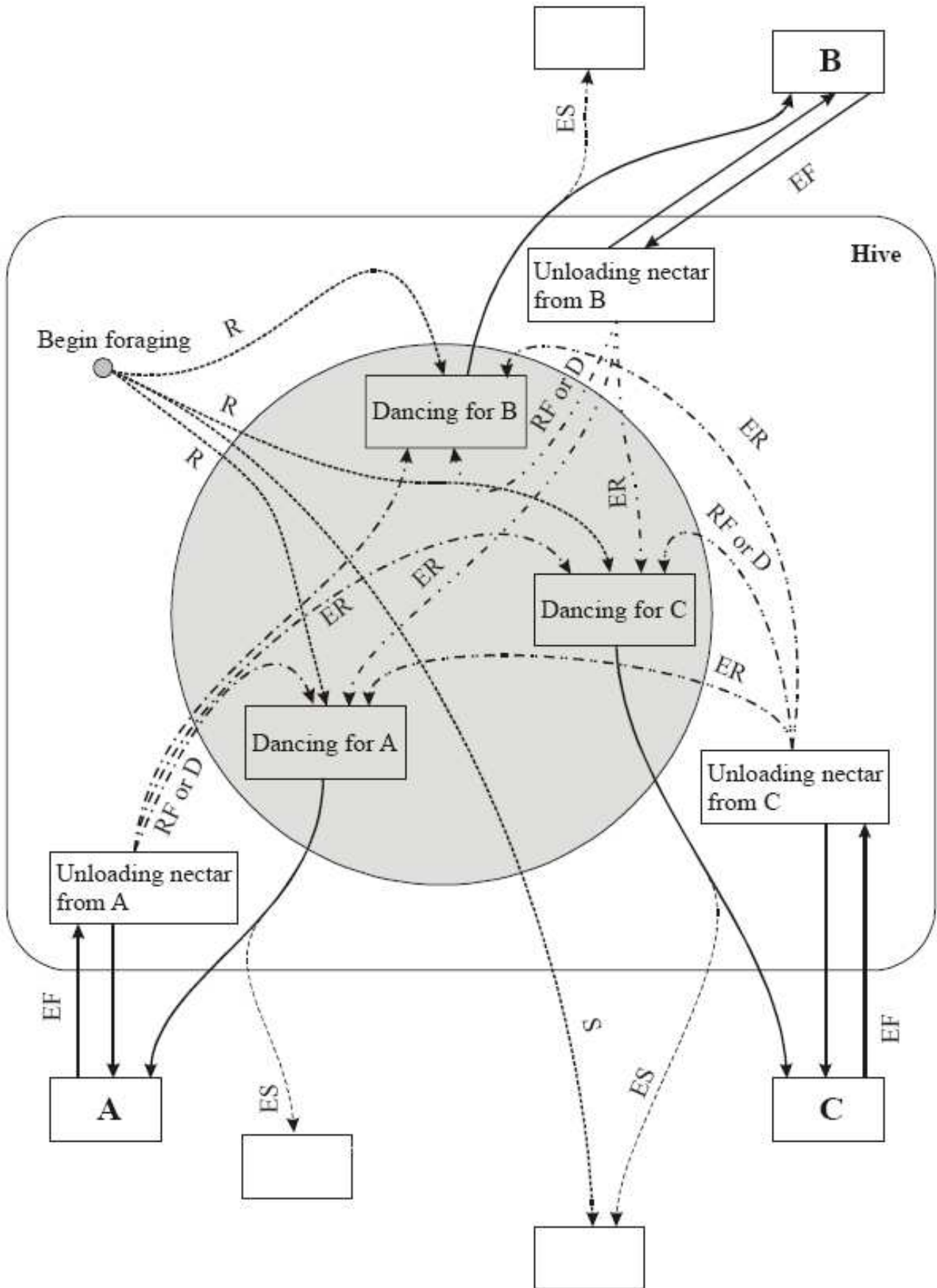
Bei der Nahrungssuche sind verschiedene Arten von Arbeiterinnen beteiligt:

- Eine **Sammlerin** (forager) nutzt bereits bestehende Nahrungsquellen.
- **Sucherinnen** (scouts) erforschen die Umgebung nach neuen Quellen.
- Eine **Futterabnehmerin** (food storer) nimmt die Nahrung der Sammlerinnen entgegen, um diese einzulagern.

In der Natur beginnen nicht alle Bienen gleichzeitig mit der Nahrungssuche. Es wurde festgestellt, dass die Wahrscheinlichkeit, mit der Nahrungssuche zu beginnen proportional zur Differenz der gesamten Anzahl der Bienen und der aktuellen Anzahl der Nahrungssammlerinnen ist. Sobald sich eine Biene entscheidet zu einer Sucherin zu werden, sucht sie entweder eigenständig nach einer Nahrungsquelle oder sie folgt dem Hinweis einer Schwester, um den beschriebenen Platz zu erkunden.

Hat diese Sucherin eine Nahrungsquelle gefunden, nutzt sie ihre eigene Begabung, um sich den genauen Ort zu merken und dort die Ressourcen abzubauen. Damit wird sie zu einer beschäftigten Sammlerin. Sobald die Biene ihre Nahrung zum Bienenstock zurückgebracht hat, muss sie entscheiden, wie sie fortfährt. Sie kann die gerade gefundene Nahrungsquelle aufgeben, womit sie wieder zu einer freien Nahrungssucherin wird. Alternativ kann sie an der Stelle den Nahrungsabbau fortsetzen oder auf dem Tanzboden tanzen, um so weitere Bienen für diese Nahrungsquelle anzuwerben. Die Biene wählt mit bestimmter Wahrscheinlichkeiten eine der Möglichkeiten, je nach subjektiv eingeschätzter Qualität der Nahrungsquelle.

Die Grundcharakteristiken des Verhaltens der Bienen in der Nahrungssuche können folgender Abbildung entnommen werden:



Quelle: Modeling Transportation Problems Using Concepts of Swarm Intelligence and Soft Computing, S. 33

In dem Bienenstock ist der Tanzboden durch den grauen Kreis dargestellt. Darin werben mehrere Bienen für verschiedene Standorte. Die bereits bekannten Standorte sind durch die Buchstaben **A**, **B**, **C** markiert. In der Abbildung werden die Bienen durch Wissen über Trachtquellen und genauer Arbeit unterschieden.

**Beschäftigte Nahrungssucherinnen** (Employed forager, **EF**) kennen und nutzen profitable Nahrungsquellen.

**Unbeschäftigte Kundschafterinnen** (Scouts, **S**) suchen ohne Vorwissen nach guten Plätzen. **Unbeschäftigte Rekruten** (Recruits, **R**) suchen nach einer im Tanz beschriebenen Stelle. Sie kennen ungefähre Position, jedoch nicht die Qualität.

Eine erfahrene Nahrungssammlerin hat das Wissen über die Lage und Profitabilität von mindestens einer Quelle. Eine **reaktivierte Nahrungssammlerin** (Reactivated forager, **RF**) oder **Tänzerin** (Dancer, **D**) ist eine erfahrene freie Biene, die auf dem Tanzboden für die ihr bekannte Trachtquelle wirbt. Eine **erfahrene Sucherin** (Experienced scout, **ES**) sucht nach einer neuen Quelle, nachdem die alte erschöpft ist oder diese ihr zu unprofitabel erscheint. Eine **erfahrene Rekrutin** (Experienced recruit, **ER**) ist unzufrieden mit der zuletzt besuchten Quelle. Deshalb sucht sie nach einer neuen, die durch einen Tanz einer ihrer Artgenossinnen beschrieben wurde.

Zusammengefasst gibt es folgende mögliche Tätigkeiten einer Sammelbiene:

- Nicht aktiv
- Abladen von Nektar
- Werben für eine Nahrungsquelle
- Ernten von Nahrung
- Einem Tänzer folgend
- Erforschen

### 3 Die künstliche Biene

Der Erfolg von Ameisensystemen in der Informatik hat einige Forscher aktiviert, andere Systeme aus der Natur in der Informatik umzusetzen. So haben Teodorovic und Lucic ein Bienenschwarmsystem entwickelt, welches auf komplexe Systeme angewandt werden kann.

Dabei werden wie in der Natur Sucher zum Finden neuer Nahrungsquellen eingesetzt. Ihre Suchkosten sind sehr gering und sie findet nur gelegentlich große unbekannte Nahrungsquellen. Die Rekrutierungsrate der Bienen repräsentiert Ausmaß, wie schnell ein Bienenvolk neu gefundene Quellen ausbeutet oder aber die Geschwindigkeit mit der gute Lösungen gefunden werden. Die Zusammenarbeit von den Bienen verringert die Kosten, neue Quellen zu finden und erhöht die Qualität der Nahrungsquellen. Die Zusammenarbeit ist also zur schnellen Entdeckung von guten Lösungen notwendig.

Künstliche Sucherinnen sollen versuchen schwierige kombinatorische Probleme zu lösen, die in der schnellen Entdeckung von einer Gruppe von nützlichen Lösungen besteht. Einige dieser Lösungen könnten sich als nahezu optimale herausstellen.

## 4 Lösen des Traveling Salesman Problem mit Bienensystemen

Das vorderste Ziel der Forschung von Lucic und Teodorovic war das Finden möglicher Anwendungen für den von ihnen entwickelten Algorithmus. Ein solches Problem ist das Traveling Salesman Problem. In diesem Abschnitt wird eine Lösung dafür erläutert .

### 4.1 Das Traveling Salesman Problem

Bei dem Traveling Salesman Problem (TSP) muss zu einem bestehenden Graphen  $G = (N, A)$ , wobei  $N$  die Menge aller Knoten und  $A$  die Menge aller Kanten ist, ein Kreis gefunden werden, der alle Knoten genau ein mal durchläuft. Dieser Kreis soll möglichst kurz sein. Dabei ist

$$\underline{C} = \begin{pmatrix} c_{1,1} & \dots & c_{1,n} \\ \dots & c_{i,j} & \dots \\ c_{n,1} & \dots & c_{n,n} \end{pmatrix} \text{ die Distanzmatrix oder Kostenmatrix.}$$

$c_{i,j}$  Kosten einer Kante

$n$  Anzahl der Knoten,  $n = |N|$

Das TSP wurde gewählt, da es wegen der NP-Vollständigkeit schwierig zu lösen, allerdings gleichzeitig auch einfach zu verstehen ist.

### 4.2 BeeSystem als Lösungsalgorithmus

Im Jahr 2001 haben Lucic und Teodorovic einen Bienenschwarm-Algorithmus entwickelt, mit welchem es möglich ist, das Traveling Salesman Problem effizient zu lösen: BeeSystem. Dieser Algorithmus wird in diesem Abschnitt genauer erläutert.

Mit den Annahmen, dass die am nächsten liegende Nahrungsquellen am attraktivsten sind und dass die Nektarmenge, die entlang einer Kante getragen werden kann, umgekehrt proportional zu den Kosten einer Kante ist, kann das Verhalten der Bienen bei der Nahrungssuche in der Natur zum Lösen des TSP umgesetzt werden. Wegen der Einfachheit wird hier nur ein Lösungsalgorithmus für das symmetrische TSP beschrieben, bei dem die Kosten einer Kante in beide Richtungen gleich groß ist:  $\forall (i, j) \in A : c_{ij} = c_{ji}$

Der Algorithmus besteht aus einer vorbestimmten Anzahl von **Iterationen**. Eine andere Abbruchbedingung kann darin bestehen, dass eine bestimmte Zeit ohne Verbesserungen der Route verstrichen ist. Eine Iteration repräsentiert dabei den Wechsel des Bienenstocks. Er wird zufällig an einem Knoten im Graphen platziert. Zu Beginn jeder Iteration befinden sich alle Bienen im Bienenstock. Danach erforschen die Bienen der Graphen in einem bestimmten Zeitintervall. Am Ende der Iteration wird die Komplettlösung aktualisiert, falls die gerade beste gefundene besser ist als die bisher best-bekannte.

Ein solches Zeitintervall besteht aus einer bestimmten Anzahl von **Stufen**. Während einer Stufe wird eine Biene  $s$  Knoten besuchen und eine Teilstrecke erstellen oder fortzusetzen. Die Anzahl  $s$  ist zu Beginn zu definieren. Dieser Teil der Stufe wird **Vorwärtsrechnung** genannt. Am Ende einer Stufe kehren alle Bienen zurück in den Bienenstock, wo jede Biene am Entscheidungsprozess mit teilnimmt. Jede Biene veröffentlicht dabei ihre eigene Teillösung und muss sich danach entscheiden, die eigene Nahrungsquelle aufzugeben und frei zu werden, die Nahrungssuche ohne Rekrutierung von anderen Bienen fortzusetzen oder aber sie wirbt für die Trachtquelle durch einen Tanz. Dieser zweite Teil der Stufe wird **Rückwärtsrechnung** genannt.

$B$  ist die gesamte Anzahl der Bienen in einem Stock.  $B(u, z)$  ist die Anzahl der

nahrungssammelnden Bienen während der Iteration  $z$  und der Stufe  $u$  mit  $u = 0, 1, \dots, \left\lceil \frac{|N| - 1}{s} \right\rceil$ . Zu Beginn jeder Iteration  $z$  befinden sich alle Bienen im Bienenstock:  $B(0, z) = 0$ .

Jede untätige Biene wird zu Beginn einer Stufe die Nahrungssuche mit einer vordefinierten Wahrscheinlichkeit starten. Diese Wahrscheinlichkeit ist konstant und für jede Biene gleich. Falls sie zu hoch gewählt wird, starten die meisten Bienen zu Beginn. Sie können damit die Routen von anderen Bienen nicht fortsetzen, außer sie geben eigene auf. Ist die Wahrscheinlichkeit zu niedrig gewählt, starten keine oder nur wenige Bienen. Es werden dann kaum Lösungsansätze gebildet.

Die Wahrscheinlichkeit, dass die  $k$ -te Biene, die sich am Knoten  $i$  befindet, als nächstes den Knoten  $j$  auswählt, berechnet sich wie folgt:

$$p_{ij}^k(u+1, z) = \begin{cases} \frac{e^{-a d_{i,j} \frac{z}{\sum_{r=\max(z-b, 1)}^{z-1} n_{i,j}(r)}}}{\sum_{l \in N_k(u, z)} e^{-a d_{i,l} \frac{z}{\sum_{r=\max(z-b, 1)}^{z-1} n_{i,l}(r)}}}, & i = g_k(u, z), j \in N_k(u, z), \forall k, u, z \\ 0, & \text{sonst} \end{cases}$$

mit:

- $i, j$  - Knotenindex ( $i, j = 1, 2, \dots, |N|$ )
- $d_{i,j}$  - Kantenlänge vom der Verbindung ( $i, j$ )
- $k$  - Bienenindex ( $k = 1, 2, \dots, B$ )
- $z$  - Iterationsindex ( $z = 1, 2, \dots, M$ );  $M$  - Anzahl der Iterationen
- $g_k(u, z)$  - letzter Knoten, den die Biene  $k$  besucht hat
- $N_k(u, z)$  - Menge von nicht besuchten Knoten für Biene  $k$ ;  $|N_k(u, z)| = |N| - u \cdot s$
- $b$  - „Speicherlänge“, Gedächtnis
- $n_{i,l}(r)$  - Anzahl der Bienen, die Knoten ( $i, l$ ) in Iteration  $r$  besucht haben
- $a$  - Inputparameter

Die Formel lässt erkennen, dass die Wahrscheinlichkeit einen bestimmten Knoten  $j$  zu besuchen mit der Distanz  $d_{i,j}$  zum Knoten abnimmt. Zu Beginn des Suchprozesses fließt die Distanz jedoch noch nicht so stark ein, wie im späteren Verlauf. So haben die Bienen zu Beginn eine größere Freiheit beim Erforschen des Suchraums. Bienen haben auch ein Gedächtnis. Sie haben Zugriff auf die Information, wie viele Bienen einen bestimmten Knoten während der letzten  $b$  Iterationen besucht haben. Je mehr Bienen einen Knoten besucht haben, desto größer ist die Wahrscheinlichkeit, diesen Knoten auch in der Zukunft zu besuchen. Dieses stellt die Interaktion untereinander da. Die Attraktivität von Nahrungsquellen wird beworben.

Während der Rückwärtsrechnung einer Stufe, muss jede Biene entscheiden, ob sie ihre Tour fortsetzen will. Im Gegensatz zur Natur kann jede Biene dabei die Information über die Qualität der Routen aller Bienen erhalten. Die Wahrscheinlichkeit, dass Biene  $k$  zu Beginn von Stufe  $u+1$  die gleiche Teilstrecke wie zuvor wählt berechnet sich so:

$$p_k(u+1, z) = e^{-\frac{L_k(u, z) - \min_{r \in w(u, z)} (L_r(u, z))}{u \cdot z}}$$

mit:

- $L_k(u, z)$  - Länge der von der Biene entdeckten partiellen Route

Es ergibt sich eine Wahrscheinlichkeit von 1, dieselbe Partillösung fortzusetzen, wenn die Biene die kürzeste TS-Tour während der Stufe  $u$  und Iteration  $z$  entdeckt hat. Je länger die entdeckte Tour ist, desto geringer ist die Wahrscheinlichkeit, die Tour fortzusetzen. Wenn sich eine Biene entscheidet, ihre Teillösung nicht aufzugeben, muss sie sich entscheiden, alleine weiter zu suchen oder um Mitglieder zu werben. Hier wird angenommen, dass Wahrscheinlichkeit  $p^*$ , die selbe Tour ohne Rekrutierung weiterzuführen sehr gering ist:  $p^* \ll 1$ . Höchstwahrscheinlich wird eine Biene also durch einen Tanz im Bienenstock für die Route werben.

Wenn eine Biene die eigene Route nicht weiterführt, folgt sie einer anderen Biene, um eine partielle TS-Tour  $\xi$  auszubauen. Jede TS-Tour hat dabei zwei Hauptattribute:

- 1) totale Länge der Tour
- 2) Anzahl der Bienen, die diese partielle Route bewerben

In dem BeeSystem-Algorithmus wird mit normalisierten Werten der beiden Attribute gearbeitet. Dabei muss sichergestellt werden, dass je kleiner die Länge von  $\xi$  und je größer Anzahl der Bienen, die diese Strecke bewerben, desto besser ist Partillösung.

Wenn  $Y(u, z)$  die Menge aller Teil-Routen, die von mindestens einer Biene besucht wurden, ist, dann ergibt sich für die normalisierte Länge der Teilstrecken:

$$\alpha_{\xi}(u, z) = \begin{cases} \frac{L_{\xi}(u, z) - \min_{r \in Y(u, z)}(L_r(u, z))}{\max_{r \in Y(u, z)}(L_r(u, z)) - \min_{r \in Y(u, z)}(L_r(u, z))} & \text{für } \max_{r \in Y(u, z)}(L_r(u, z)) \neq \min_{r \in Y(u, z)}(L_r(u, z)), \xi \in Y(u, z), \forall u, z \\ 0 & \text{sonst} \end{cases}$$

Mit  $B_{\xi}(u, z)$  als Anzahl der Bienen, die die partielle Tour  $\xi$  besuchten errechnet sich der normalisierte Wert der Bienen, die eine Strecke bewerben wie folgt:

$$\beta_{\xi}(u, z) = \begin{cases} \frac{B_{\xi}(u, z) - \min_{r \in Y(u, z)}(B_r(u, z))}{\max_{r \in Y(u, z)}(B_r(u, z)) - \min_{r \in Y(u, z)}(B_r(u, z))} & \text{für } \max_{r \in Y(u, z)}(B_r(u, z)) \neq \min_{r \in Y(u, z)}(B_r(u, z)), \xi \in Y(u, z), \forall u, z \\ 0 & \text{sonst} \end{cases}$$

Die Wahrscheinlichkeit, dass  $\xi$  von einer Biene gewählt wird, ist so definiert:

$$p_{\xi}(u, z) = \frac{e^{\rho \beta_{\xi}(u, z) - \theta \alpha_{\xi}(u, z)}}{\sum_{\tau \in Y(u, z)} e^{\rho \beta_{\tau}(u, z) - \theta \alpha_{\tau}(u, z)}} \quad \text{mit } \xi \in Y(u, z), \forall u, z$$

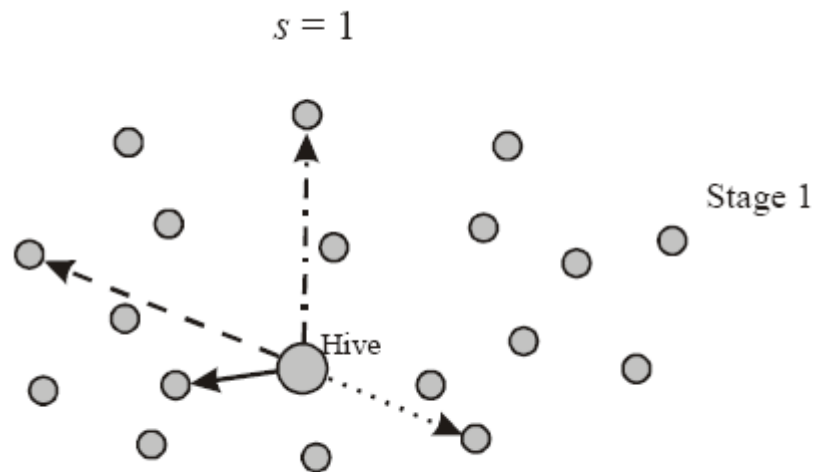
mit:

$\rho, \theta$  - vordefinierte Parameter

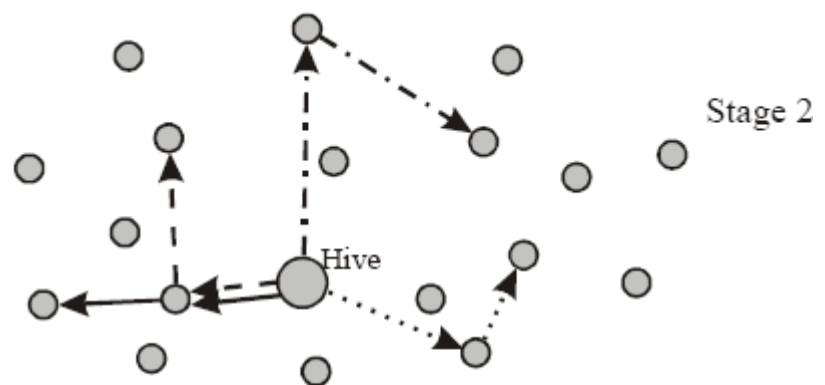
Damit ist die Grundstruktur des BeeSystem-Algorithmus beschrieben.

Im nachfolgenden wird ein Beispiel zur Erstellung einer Teilstrecke beschrieben. Jede Biene wird dabei pro Stufe einen Knoten besuchen ( $s = 1$ ):

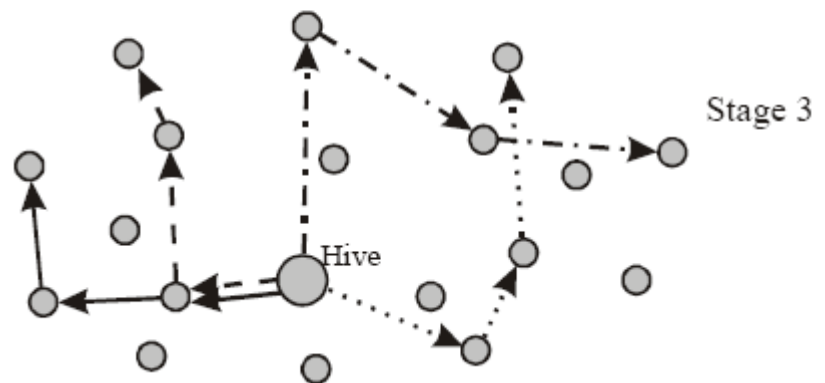
Während der ersten Stufe suchen sich alle Bienen zufällig einen Knoten aus.



In der zweiten Stufe, gibt die gestrichelt dargestellte Biene ihre Partialtour auf. Deren Strecke war relativ gesehen die längste, wodurch die Wahrscheinlichkeit zu Aufgabe der Route bei ihr am größten war. Sie schließt sich der mit einem durchgezogenen Strich dargestellten Biene an und setzt von da aus eine Route fort.



In der dritten gezeigten Stufe setzen alle Bienen ihre Teilstrecke fort, ohne dem Weg einer anderen Biene zu folgen.



Quelle: Modeling Transportation Problems Using Concepts of Swarm Intelligence and Soft Computing, S. 45

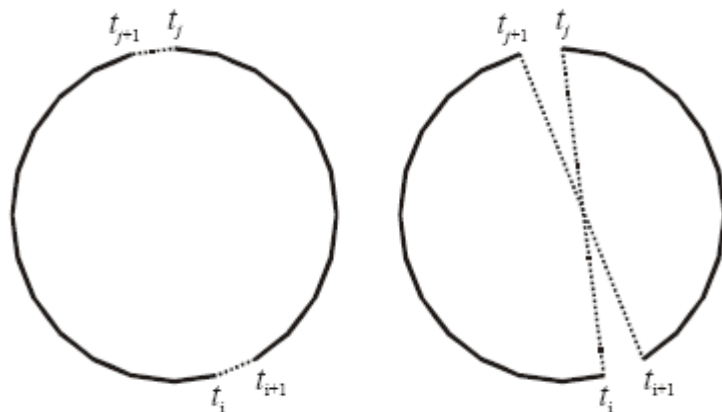
### 4.3 Optimierung des BeeSystem mit k-opt-Algorithmen

Um die Ergebnisse des BeeSystem weiter zu verbessern, haben Teodorovic und Lucic diesen Algorithmus mit k-opt-Heuristiken verbessert. Diese werden am Ende jeder Iteration durchgeführt.

Bei einem **k-opt**-Algorithmus wird eine Menge von  $k$  Kanten durch eine andere Menge der gleichen Mächtigkeit und geringer totaler Länge ersetzt, sodass eine kleinere Tour entsteht. Die Komplexität des Algorithmus ist  $O(n^k)$ , wobei  $n$  die Anzahl der Knoten ist.

Hier ein Beispiel des **2-opt**-Algorithmus:

Zwei Kanten werden aus der Traveling Salesman Tour entfernt und durch zwei andere ersetzt, sodass insgesamt weiterhin ein Rundkreis vorhanden ist. Wenn das neue Kantenpaar kürzer ist, wird die Tour entsprechend verändert.



Der **3-opt**-Algorithmus arbeitet auf die gleiche Weise. Der einzige Unterschied ist, dass gleichzeitig drei Kanten entfernt und durch drei neue Kanten ersetzt werden.

Da der 3-opt-Algorithmus wegen seiner Komplexität von  $O(n^3)$  sehr

viel Zeit in Anspruch nimmt, haben sich Teodorovic und Lucic eine Möglichkeit einfallen lassen, mit der dieser Algorithmus deutlich schneller läuft, jedoch nur geringe Einbußen in der Lösungsqualität hat. So kann der Suchraum einfach verkleinert werden. Drei Knoten sollten nah genug zusammen liegen, damit entsprechende Kanten entfernt und durch neue ersetzt werden.

Quelle: Modeling Transportation Problems Using Concepts of Swarm Intelligence and Soft Computing, S. 46

### 4.4 Testergebnisse

Der Algorithmus wurde mit den beschriebenen Optimierungen für einige bekannte Traveling Salesman Probleme von der folgenden Internetseite getestet: <http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/tsp/>

Ausgeführt wurden die Tests mit einem PIII-Prozessor (533 MHz). Die Suche wurde limitiert auf 100 Zyklen. Dabei wurden mit der Optimierung der 2-opt-Heuristik folgende Ergebnisse erzielt:

Problem	Number of Nodes	Optimal Value (O)	Best value BeeSystem (B)	(B-O)/O (%)	Time for the best solution (seconds)	Average value over 20 runs (A)	(A-O)/O (%)
Eil51	51	426	431,12	1,20%	44	433,76	1,82%
Berlin52	52	7542	7544,37	0,03%	18	7634,37	1,22%
St70	70	675	678,62	0,54%	238	684,28	1,37%
Pr76	76	108159	108790	0,58%	127	109444,6	1,19%
Kro100	100	21282	21441,5	0,75%	58	21575,7	1,38%
Eil101	101	629	642,45	2,14%	146	665,62	5,82%
Tsp225	225	3859	4065,56	5,35%	2076	4113,71	6,60%
A280	280	2579	2740,63	6,27%	1855	2784	7,95%

Die optimalen Werte sind die besten bisher gefundenen Werte. Ob es wirklich die kürzesten Strecken sind, kann nicht gesagt werden, da noch nicht alle möglichen Konstellationen überprüft werden konnten. Bei einem Graphen mit 30 Knoten bräuchte ein heutiger Rechner beispielsweise noch über 1000 Jahre, um alle Möglichkeiten zu überprüfen.

Das BeeSystem mit dem 2-opt-Algorithmus liefert dabei recht vernünftige Werte in passablen Zeiten. Erst bei Graphen mit mehr als 100 Knoten ist die mittlere Abweichung größer als 5%. Bei 180 Knoten war eine Berechnungszeit von etwa 30 min. nötig.

Noch bessere Ergebnisse werden mit der 3-opt-Optimierung erzielt:

Problem	Number of Nodes	Optimal Value (O)	Best value BeeSystem (B)	(B-O)/O (%)	Time for the best solution (seconds)	Average value over 20 runs (A)	(A-O)/O (%)
Eil51	51	426	428,87	0,67%	37	428,87	0,67%
Berlin52	52	7542	7544,37	0,03%	1	7544,37	0,03%
St70	70	675	677,11	0,31%	22	677,11	0,31%
Pr76	76	108159	108159	0,00%	11	108159	0,00%
Kro100	100	21282	21285,4	0,02%	10	21285,4	0,02%
Eil101	101	629	640,21	1,78%	1741	643,05	2,23%
Tsp225	225	3859	3876,05	0,44%	5153	3905,32	1,20%
A280	280	2579	2600,34	0,83%	13465	2627,45	1,88%

Hier ist die mittlere Abweichung zur optimalen Lösung bei allen Graphen unter 3%. Im Gegenzug wird hier allerdings eine deutlich längere Berechnungszeit benötigt. So dauert die Berechnung der Lösung bei 280 Knoten ungefähr 3:45 h statt 30 min. in der 2-opt-Variante.

Zuletzt noch der Algorithmus mit der schnelleren 3-opt-Optimierung:

Problem	Number of Nodes	Optimal Value (O)	Best value BeeSystem (B)	(B-O)/O (%)	Time for the best solution (seconds)	Average value over 20 runs (A)	(A-O)/O (%)
Eil51	51	426	428,87	0,67%	29	428,87	0,67%
Berlin52	52	7542	7544,37	0,03%	0	7544,37	0,03%
St70	70	675	677,11	0,31%	7	677,11	0,31%
Pr76	76	108159	108159	0,00%	2	108159	0,00%
Kro100	100	21282	21285,4	0,02%	10	21285,4	0,02%
Eil101	101	629	640,21	1,78%	61	643,07	2,24%
Tsp225	225	3859	3899,9	1,06%	11651	3909,69	1,31%
A280	280	2579	2608,33	1,14%	6270	2632,42	2,07%
Pcb442	442	50778	51366,04	1,16%	4384	51756,89	1,93%
Pr1002	1002	259045	267340,7	3,20%	28101	268965,6	3,83%

Die Abweichungen sind ähnlich den des normalen 3-opt-Algorithmus, zum Teil nur etwas schlechter. Der Vorteil liegt vor allem darin, dass die Zeiten hier deutlich geringer sind. So braucht der Algorithmus mit der schnelleren Variante für den Graphen mit 280 Knoten nur 1:45 h statt der vorher benötigten 3:45 h.

In neueren Arbeiten von Teodorovic werden die Ergebnisse der letzten Tabelle dargestellt. Dabei wird der Algorithmus nicht mehr BeeSystem sondern Bee Colony Optimization (BCO) genannt. Teilweise wird auch verschwiegen, dass k-opt-Optimierungen angewandt wurden.

## 4.5 Bee Colony Optimization mit Lokaler Suche

2008 haben Wong, Low, und Chong den BeeSystem-Algorithmus aufgegriffen, jedoch einige grundlegende Dinge geändert, um noch bessere Ergebnisse erzielen zu können. Dieser Algorithmus wird Bee Colony Optimization genannt.

Die Unterschiede sind:

- Bienen haben nicht die Fähigkeit, sich die Anzahl der Bienen merken zu können, die eine bestimmte Kante besucht haben.
- Bienen werben für nützliche Gesamtrouten anstatt für Teilpfade.
- Der Bienenstock wird nicht zufällig an einem Knoten platziert. Stattdessen ist die Distanz vom Bienenstock zu allen Knoten gleich.
- Bienen werden sowohl durch Kantentauglichkeit als auch der Distanz zwischen Knoten beeinflusst.

Mit den Veränderungen im Algorithmus lassen sich deutliche Steigerungen der Lösungen erkennen. So hat der Algorithmus selbst bei einer größeren Anzahl von 280 Knoten keine Abweichung vom Optimum. Die Ausführungszeiten können nicht genau verglichen werden, da die Algorithmen auf komplett unterschiedlichen Systemen getestet wurden.

Problem Instances	BCO+2-opt (%)		BS+2-opt [19] (%)		BS+Modified 3-opt [19] (%)	
	Avg.	Best	Avg.	Best	Avg.	Best
	EIL51	0.00	0.00	1.14	0.53	0.00
BERLIN52	0.00	0.00	1.19	0.00	0.00	0.00
ST70	0.00	0.00	1.06	0.22	0.00	0.00
PR76	0.00	0.00	1.19	0.58	0.00	0.00
KROA100	0.00	0.00	1.36	0.73	0.00	0.00
EIL101	0.00	0.00	3.97	0.35	0.45	0.00
TSP225	0.00	0.00	6.60	5.35	1.31	1.06
A280	0.00	0.00	7.66	5.95	1.76	0.83

Quelle: Bee Colony Optimization with Local Search for Traveling Salesman Problem

Der Algorithmus wurde auch mit einigen anderen Algorithmen zum Lösen von Traveling Salesman Problemen verglichen. Folgende Algorithmen wurden getestet:

- ACS: Ant Colony System
- MMAS: MAX-MIN Ant System
- GA: Genetic Algorithms
- HGA: Hybrid Version von GA
- Lin-Ker: Lin-Kernigham-Heuristik
- BS: BeeSystem
- BCO: Bee Colony Optimization

PERFORMANCE COMPARISON OF ACS, MMAS, GA, HGA, LIN-KER, BCO AND BCO+2-OPT

Instance	EIL51	KROA100	LIN318
ACS [17]	0.48/0.00	0.65/0.00	n/a
MMAS [18]	0.16/0.00	0.10/0.00	n/a
GA [14,15]	0.00/0.00	0.00/0.00	0.00/0.00
HGA [16]	0.59/0.47	0.01/0.01	1.37/0.73
Lin-Ker [11,28]	0.00/0.00	0.00/0.00	0.08/0.00
BS+2-opt [19]	1.14/0.53	1.36/0.73	n/a
BS+Modified 3-opt [19]	0.00/0.00	0.00/0.00	n/a
BCO+2-opt	0.00/0.00	0.00/0.00	0.09/0.00

x/y: x and y represent percentage difference of average tour length and best tour length from optimal value respectively. "n/a" stands for "not available".

Die Ergebnisse sind der obigen Abbildung zu entnehmen. Nur der Lin-Kernigham-Algorithmus und der Genetsche Algorithmus liefern bessere Ergebnisse als der dargestellte BCO-Algorithmus.

Auch hier kann wieder keine Aussage über die Ausführungszeit gemacht werden, da verschiedene Systeme zum Einsatz kamen.

## 4.6 Bewertung

Es wurde ein von Teodorovic und Lucic entwickelter Algorithmus zum Lösen des TSP-Problems dargestellt, der recht gute Ergebnisse bei passablen Ausführungszeiten lieferte. Dieser Algorithmus wurde von Wong, Low und Chong verbessert, der noch bessere Testergebnisse liefert. Nur der Lin-Kernigham-Algorithmus und der Genetische Algorithmus liefern bessere Ergebnisse als der BCO-Algorithmus. Da der BCO-Algorithmus noch relativ jung ist, wird sein volles Potential wahrscheinlich auch noch nicht ausgeschöpft, sodass weitere Verbesserungen möglich sind.

Um die Tauglichkeit der verschiedenen Algorithmen für das Traveling Salesman Problem genauer vergleichen zu können sind insgesamt noch weitere Tests notwendig. So liefern einzelne Algorithmen je nach lokalen Optimierungen oder eingesetztem System unterschiedliche Ergebnisse. Auch kann wie oben beschrieben das Zeitverhalten nicht genauer verglichen werden.

Hier kann deshalb noch keine genaue Aussage über die Zukunftsträchtigkeit des Algorithmus gemacht werden.

## 5 Ein Routingalgorithmus: BeeHive

### 5.1 Idee hinter BeeHive

Wedde und Farooq haben 2004 den BeeHive-Algorithmus als Routingalgorithmus für Telekommunikationsnetzwerke entwickelt. Das Ziel ihrer Forschung war, einen Natur inspirierten Algorithmus auf ein reales Problem anzuwenden. Dafür entwickelten sie den BeeHive-Algorithmus als Routingalgorithmus.

Zunächst wurden die Grenzen angedacht, unter denen das vorgesehene Routingprotokoll arbeiten sollte:

- Nichtverfügbarkeit von einem globalem Takt für Laufzeitberechnung
- Router und Verbindungen können abstürzen
- Router haben begrenzte Warteschlangenkapazität
- Verbindungen haben jeweils bestimmte BER (Bit Error Rate)
- Protokoll muss den Linux Kernel Routing Framework-Anforderungen gerecht werden
- Anforderungen des IP-Protokolls müssen erfüllt sein

Im Netzwerk werden Bienenagenten eingesetzt, die dieses genauer untersuchen sollen, um wichtige Parameter zu sammeln. Sie treffen Routing-Entscheidungen auf dezentrale Weise. Die Bienenagenten sollen die Qualität einer Route abschätzen und diese anderen Bienen mitteilen wie die Nahrungssammler in der Natur. Die Struktur der Routingtabelle sollte Funktionalität des Tanzbodens beinhalten, sodass die Bienen dort Informationen austauschen können.

### 5.2 Der Algorithmus

#### 5.2.1 Grundgedanken

Nachfolgend ist erklärt, wie die Konzepte aus der Natur in diesem Algorithmus umgesetzt werden.

Jeder Knoten im Netzwerk stellt ein Bienenstock dar, der seine eigenen Bienenagenten hat. Ein solcher Bienenagent wird periodisch ins Netzwerk eingeführt. Er erforscht dabei das Netzwerk, sammelt Routinginformationen und bietet den Knoten, die er besucht, Teilinformationen über den Netzwerkstatus an. Die Bienenagenten können als Sucherinnen betrachtet werden, welche die Qualität mehrerer Pfade zwischen ihrem Startknoten und denen, die sie besuchen, erforschen und auswerten.

Bienenagenten bieten den Knoten, die sie besuchen, Informationen über die Laufzeitverzögerung (Distanzinformationen) und Warteschlangen-Verzögerung (Richtungsinformationen) der Pfade an, welche sie erforscht haben. Die Bienenagenten bieten Pfadinformationen allerdings nur dann an, wenn die Qualität oberhalb einer bestimmten Grenze ist. Die Grenze wird durch die Anzahl der Hops, die ein Bienenagent machen darf, bestimmt. Die Agenten modellieren Pfadqualität als eine Funktion abhängig von der Laufzeitverzögerung und der Warteschlangenverzögerung. Geringere Verzögerungen steigern die Qualität.

Die Agenten sind aufgeteilt in **Kurz-Distanz-** und **Lang-Distanz-Agenten**. Kurz-Distanz-Agenten, welche den Großteil ausmachen, erforschen das Netzwerk nur in Nähe ihres Startknotens. Nur sehr wenige Lang-Distanz-Agenten erforschen das gesamte Netzwerk. Dadurch soll weniger Overhead entstehen und die Routingtabellen werden klein gehalten. Eine Routingtabelle wird als Tanzboden aufgefasst. Hier bieten Agenten ihre Pfadinformationen an. Eine Routingtabelle dient als Informationsaustausch zwischen Bienen, die am selben Knoten starten, aber über verschiedene Nachbarn einen bestimmten Knoten erreichen. Der Informationsaustausch soll helfen, die Qualität eines Knotens zu bewerten. Die Qualität eines Knotens, ein Ziel zu erreichen, ist eine Funktion

proportional zur Qualität von nur den Nachbarn, welche möglicherweise in Richtung des Ziels liegen.

Bienen, die am selben Knoten eingeführt wurden, bilden eine **Ähnlichkeitsgruppe**. Wenn zwei Agenten der selben Gruppe den gleichen Knoten über verschiedene Nachbarn erreichen, rufen sie die Routinginformationen ab. Sie werden sich nach Speicherung ihrer Daten entscheiden, ihre Erforschung zu unterbrechen, wenn ein Mitglied bereits den Knoten erreicht hat. In diesem Algorithmus kommunizieren die Bienen also per Blackboard-System. Ameisenalgorithmen hingegen nutzen das Stigmergieprinzip.

Datenpakete können im BeeHive-Algorithmus als Nahrungssammlerinnen aufgefasst werden. Sie rufen bei Ankunft an einem Knoten Informationen der Routingtabelle ab. Das Datenpaket wählt den nächsten Knoten auf stochastische Weise abhängig von der Güte. Daraus folgt, dass nicht alle Pakete dem besten Pfad folgen, wodurch eine Maximierung der Systemperformance versucht wird zu erreichen.

### 5.2.2 Netzwerkorganisation

Bei dem BeeHive-Algorithmus ist das Netzwerk in feste Teilbereiche organisiert, in **foraging regions**. Jede dieser foraging regions hat einen repräsentativen Knoten. Gleichzeitig hat jeder Knoten im Netzwerk eine spezifische **foraging zone**. Diese besteht aus allen Knoten, deren Kurz-Distanz-Agenten diesen Knoten erreichen können.

Jeder nicht-repräsentative Knoten sendet periodisch einen Kurz-Distanz-Agenten per Broadcast an alle Nachbarn. Wenn ein bestimmter Agent einen Knoten erreicht, werden die Routinginformationen dort aktualisiert. Der Bienenagent wird danach an alle Nachbarn bis auf den Herkunftsnachbarn geschickt. Dieser Prozess setzt sich so lange fort, bis ein Knoten bereits von einem Agenten der Ähnlichkeitsgruppe besucht wurde, oder bis das Hoplimit erreicht ist.

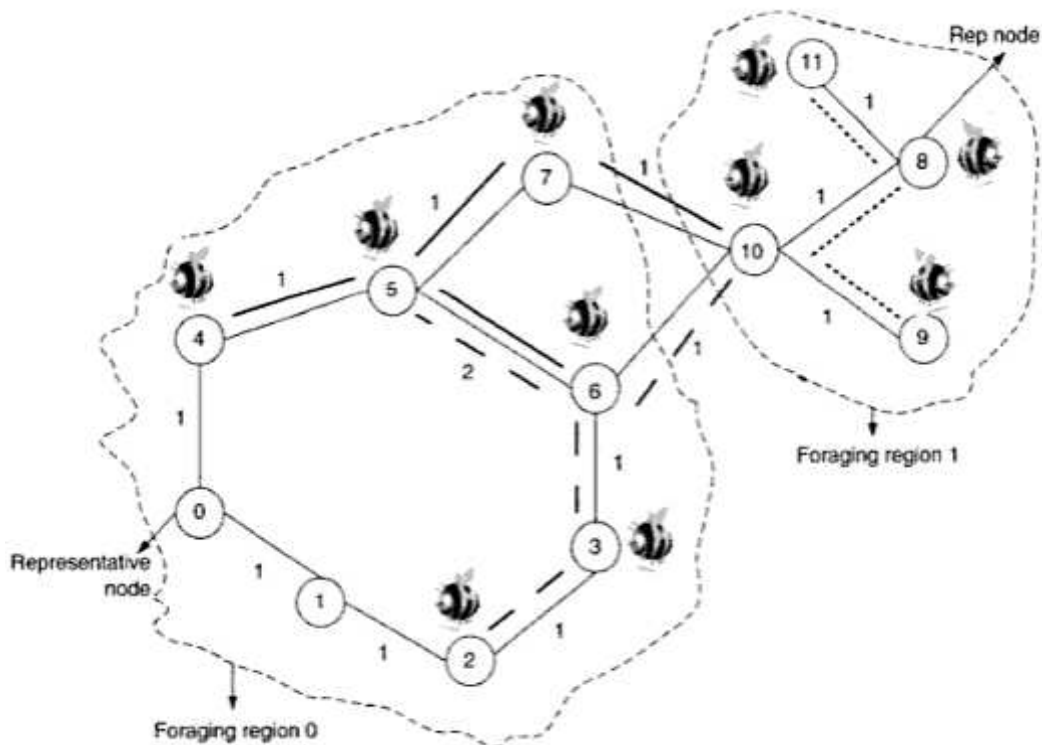
Repräsentative Knoten setzen Lang-Distanz-Agenten ein. Diese werden wie die Kurz-Distanz-Agenten verbreitet, haben allerdings ein höheres Hoplimit, da sie das gesamte Netzwerk fluten sollen.

Jeder Agent sammelt Informationen und gibt am erreichten Knoten eine Abschätzung über die Dauer zum Ausgangsknoten ab. Die Agenten benutzen Prioritätswarteschlangen für die schnelle Ausbreitung von Routinginformationen.

Jeder Knoten im Netz erhält aktuelle Informationen zum Erreichen der Knoten innerhalb der foraging zone und der repräsentativen Knoten aller foraging zones. Knoten außerhalb einer foraging zone können durch die repräsentativen Knoten angesteuert werden.

Das nächste Ziel eines Datenpakets wird auf stochastische Weise gemäß der Güte eines Nachbarn bestimmt, um so die Systemperformance zu maximieren.

Hier ein Beispiel zur Bildung von einer foraging zone:



Quelle: Handbook of Bioinspired Algorithms and Applications, S. 21-329

Der sichtbare Teil des Netzwerkes ist in zwei foraging regions aufgeteilt, wobei jeweils der Knoten mit der niedrigsten IP-Adresse der repräsentative ist. Das Hoplimit der Kurz-Distanz-Agenten in diesem Beispiel liegt bei 3. Jede Kante hat bestimmte Kosten, die dort jeweils vermerkt sind.

In dem Beispiel wird die foraging zone des Knotens 10 dargestellt. Dabei wird jeder Bienenagent, der dort startet, mit einer anderen Linienart dargestellt, um dessen Weg eindeutig bestimmen zu können. Die foraging zone des Knotens 10 beinhaltet die Knoten 2, 3, 4, 5, 6, 7, 8, 9 und 11. Es ist erkennbar, dass der Agent, der Knoten 6 durchläuft und anschließend Knoten 5 ansteuert, dort seinen Weg abbricht, da dieser Knoten bereits von einem anderen Agenten der Ähnlichkeitsgruppe besucht wurde.

Die Abschätzung, wie lange Datenpaketen vom aktuellen Knoten  $i$  zur Quelle  $s$  braucht, wird in dem Algorithmus so berechnet:

$$t_{is} \approx \frac{ql_{in}}{b_{in}} + tx_{in} + pd_{in} + t_{ns}$$

mit:

- $ql_{in}$  - Warteschlangengröße für Nachbar  $n$  am Knoten  $i$
- $b_{in}$  - Bandbreite an der Kante zwischen  $n$  und  $i$
- $tx_{in}, pd_{in}$  - Weiterleitungsverzögerung und Laufzeitverzögerung zwischen  $n$  und  $i$
- $t_{ns}$  - Sendezeit von  $n$  nach  $s$

Die Güte des Nachbarn  $j$  vom Knoten  $l$ , das Ziel  $d$  zu erreichen, ist so definiert:

$$g_{jd} = \frac{\left( \frac{1}{p_{jd}} \right) \cdot e^{-q_{jd}/p_{jd}} + \left( \frac{1}{q_{jd}} \right) \cdot \left( 1 - e^{-q_{jd}/p_{jd}} \right)}{\sum_{k=1}^n \left( \left( \frac{1}{p_{kd}} \right) \cdot e^{-q_{kd}/p_{kd}} + \left( \frac{1}{q_{kd}} \right) \cdot \left( 1 - e^{-q_{kd}/p_{kd}} \right) \right)}$$

mit:

$n$  - Anzahl der Nachbarn von  $l$

Wenn die Auslastung sehr groß ist, hat die Warteschlangenverzögerung die primäre Rolle. Mit

$$q_{jd} \gg p_{jd} \text{ gilt: } g_{jd} \approx \frac{1/q_{jd}}{\sum_{k=1}^n 1/q_{kd}}$$

Umgekehrt ist bei niedriger Auslastung der Warteschlangenverzögerung vernachlässigbar. Es fließt

$$\text{hauptsächlich die Laufzeitverzögerung mit ein. Bei } q_{jd} \ll p_{jd} \text{ gilt: } g_{jd} \approx \frac{1/p_{jd}}{\sum_{k=1}^n 1/p_{kd}}$$

Jeder Knoten führt drei Arten von Routingtabellen:

- **Intra-Foraging-Zone (IFZ):** Diese Tabelle beinhaltet die Routen zu allen Knoten in der foraging Zone über alle Nachbarn. Ein Eintrag ist Paar von Warteschlangen-Verzögerung und Laufzeitverzögerung.
- **Inter-Foraging-Region (IFR):** Die Tabelle ist aufgebaut wie die IFZ-Tabelle, allerdings mit allen repräsentativen Knoten als Ziel.
- **Foraging Region Membership (FRM):** In der Tabelle werden bekannte Zielen auf eine foraging region abgebildet.

### 5.3 Simulation

Simuliert wurde der BeeHive-Algorithmus mit dem OMNeT++-Simulator, einem open-source Simulationspaket unter anderem zur Modellierung von Telekommunikationsnetzwerken und Multiprozessorsystemen.

Verglichen wurde der Algorithmus dabei mit AntNet, Distributed Genetic Algorithm und Open Shortest Path First (OSPF). Der letztere wird hier nicht weiter erläutert.

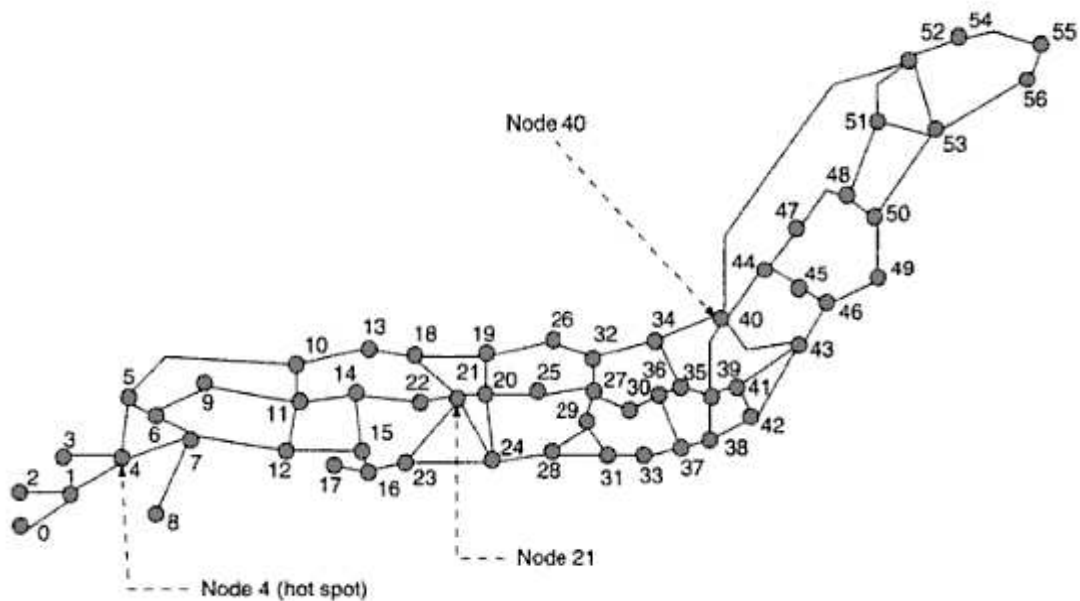
AntNet wurde von Di Caro und Dorigo entwickelt. Der Netzwerkstatus wird dabei durch **Vorwärts-Ameisen** und **Rückwärts-Ameisen** beaufsichtigt. Vorwärts-Ameisen werden dabei periodisch ins Netz eingeführt, die das Netzwerk erforschen. Sobald eine Vorwärts-Ameise ihr Ziel erreicht, wird eine Rückwärts-Ameise erzeugt, die die selben Knoten in umgekehrter Reihenfolge besucht. Dabei werden jeweils die Einträge der Routingtabellen verändert, basierend auf der Reisedauer von den Knoten zum Ziel. Es werden mittlere und beste Reisezeit sowie die Varianz abgespeichert.

Bei dem Distributed Genetic Algorithm durchlaufen die Agenten eine vordefinierte Anzahl  $n$  von Knoten in einer bestimmten Reihenfolge, ein **Chromosom**. Wenn der  $n$ -te Knoten besucht wird,

wird der Agent in einen Rückwärts-Agenten umgewandelt, der zum Ursprungsknoten zurückkehrt. Im Gegensatz zu AntNet werden dabei nur die Einträge der Routingtabelle des Ausgangsknotens verändert. Dieser Knoten schätzt auch die Tauglichkeit eines Agenten ab anhand der Reisedauer. In den Routingtabellen wird dabei die ID eines Agenten, dessen Tauglichkeit und die Reisezeit zu den besuchten Knoten gespeichert.

Im Gegensatz zu diesen beiden Algorithmen, gibt es bei BeeHive nur Vorwärts-Bienen. Sie nutzen ein Abschätzungsmodell, um die Reisezeit von dem Quellknoten zum gegebenen zu berechnen. Dabei wird keine globale Taktsynchronisierung zwischen Routern gebraucht. Für sehr große Netzwerke sollen Routinginformationen schnell und mit geringem Overhead verbreitet werden können, im Vergleich zu AntNet. Da nicht mittlere, beste Reisezeit sowie die Varianz abgespeichert, fällt die Routingtabelle hier auch deutlich kleiner aus.

Die verschiedenen Algorithmen wurden in folgendem Netzwerk verglichen:



Quelle: Handbook of Bioinspired Algorithms and Applications, S. 21-333

Die Tests lieferten folgende Ergebnisse:

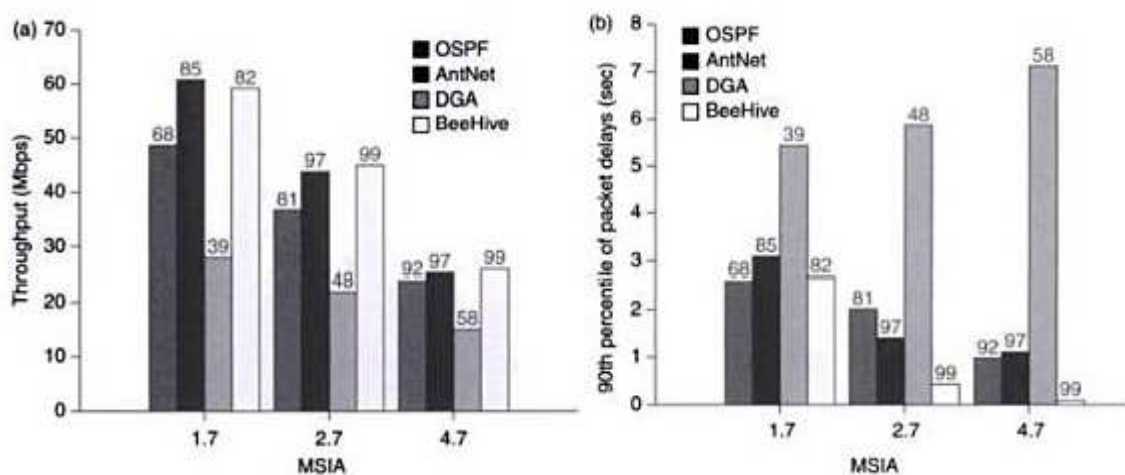


FIGURE 21.4 Behavior under saturating traffic loads.

Quelle: Handbook of Bioinspired Algorithms and Applications, S. 21-334

Der BeeHive-Algorithmus und der AntNet-Algorithmus haben bei verschiedenen mittleren session inter-arrival Zeiten den größten Durchsatz. Auffällig ist jedoch auch, dass der BeeHive-Algorithmus die geringsten Verzögerungen aufweist.

Unterschiede sind auch bei den Größen der Routingtabellen zu finden. OSPF braucht im Mittel am wenigsten Einträge, nur 57, BeeHive 94 Einträge und AntNet 162.

Insgesamt werden mit dem BeeHive-Algorithmus bessere Ergebnisse erzielt als mit den anderen beschriebenen. Auch in weiteren Tests, in denen ein Router als Hot Spot agiert oder ein Router ausfällt, weist der BeeHive-Algorithmus in der Simulation sehr gute Ergebnisse auf.

## **5.4 Bewertung**

Wie in dem Abschnitt der Simulation beschrieben, liefert der BeeHive-Algorithmus sehr gute Ergebnisse, verglichen mit anderen Routingalgorithmen.

Jedoch stellt dieser Algorithmus eher eine Weiterentwicklung von bestehenden Ameisensystemen dar. Die Umsetzungen des Verhaltens der Bienen in der Natur auf das Netzwerk sind zum großen Teil nicht stimmig. So benutzen die Bienen in dem Algorithmus den Tanzboden eines komplett anderen Bienenstocks, anstatt den eigenen und die Kommunikation darüber findet nicht wie in der Natur direkt statt. Diese Art der Kommunikation ähnelt eher der Absonderung von Pheromonen durch Ameisen. Andere Aspekte wie die Aufteilung in foraging regions wurden von OSPF übernommen. Insgesamt stellt der von Wedde und Farooq vorgestellte BeeHive-Algorithmus keine komplette Neuentwicklung, sondern eine Mischung aus OSPF und einem veränderten Ameisenalgorithmus.

Nichtsdestotrotz sollen die erhaltenen Resultate des BeeHive-Algorithmus hier nicht verschlechtert werden. Die Simulation zeigt, dass die Ergebnisse gut sind und er mit anderen gängigen Verfahren gut mithalten kann. Auch bei hoher Netzwerkdynamik liefert der dargestellte Algorithmus gute Ergebnisse. Daher ist es vorstellbar, dass der BeeHive-Algorithmus in Netzwerken Anwendung findet. Bisher wurde er allerdings nicht in einem realen Netz genutzt.

## 6 Weitere Bienenschwarmalgorithmen

Hier ein Überblick über die bisher entwickelten Bienenschwarme mit den jeweiligen Anwendungen, für die sie getestet wurden. Der Großteil der Algorithmen wurde innerhalb der letzten Dekade entwickelt:

Year	Authors	Algorithm	Problem studied
1996	Yonezawa and Kikuchi	Ecological algorithm	Description of the collective intelligence based on bees' behavior
1997	Sato and Hagiwara	Bee System (BS)	Genetic Algorithm Improvement
2001	Lučić and Teodorović	BCO	Traveling salesman problem
2001	Abbas	MBO	Propositional satisfiability problems
2002	Lučić and Teodorović	BCO	Traveling salesman problem
2003	Lučić and Teodorović	BCO	Vehicle routing problem in the case of uncertain demand
2003	Lučić and Teodorović	BCO	Traveling salesman problem
2004	Wedde, Farooq, and Zhang	BeeHive	Routing protocols
2005	Teodorović, and Dell' Oreo	BCO	Ride-matching problem
2005	Karaboga	ABC	Numerical optimization
2005	Drias, Sadeg, and Yahi	BSO	Maximum Weighted Satisfiability Problem
2005	Yang	Virtual Bee Algorithm (VBA)	Function optimizations with the application in engineering problems
2005	Benatchba, Admane, and Koudil	MBO	Max-Sat problem
2006	Teodorović, Lučić, Marković, and Dell' Oreo	BCO	Traveling salesman problem and a routing problems in networks
2006	Chong, Low, Sivakumar, and Gay	Honey Bee Colony Algorithms	Job shop scheduling problem
2006	Pham, Soroka, Ghanbarzadeh, and Koc	Bees Algorithm	Optimization of neural networks for wood defect detection
2006	Basturk and Karaboga	ABC	Numeric function optimization
2006	Navrat	Bee Hive Model	Web search
2006	Wedde, Timm, and Farooq	BeeHiveAIS	Routing protocols
2007	Yang, Chen, and Tu	MBO	Improvement of the MBO algorithm
2007	Koudil, Benatchba, Tarabetand, and El Batoul Sahraoui	MBO	Partitioning and scheduling problems
2007	Quijano and Passino	Honey Bee Social Foraging Algorithm	Solving optimal resource allocation problems
2007	Marković, Teodorović, and Aćimović-Raspopović	BCO	Routing and wavelength assignment in all-optical networks
2007	Wedde, Lehnhoff, B.van Bonn, Bay, Becker, Böttcher, Brunner, Büscher, Fürst, Lazarescu, Rotaru, Senge, Steinbach, Yilmaz, and Zimmermann	BeeHive	Highway traffic congestion mitigation
2007	Karaboga and Basturk	ABC	Testing ABC algorithm on a set of multi-dimensional numerical optimization problems

Quelle: Innovations in Swarm Intelligence, S. 41

Year	Authors	Algorithm	Problem studied
2007	Karaboga, Akay and Ozturk	ABC	Feed-forward neural networks training
2007	Afshar, Bozorg Haddada, Marin, Adams	Honey-bee mating optimization (HBMO) algorithm	Single reservoir operation optimization problems
2007	Baykasoglu, Özbakýr, and Tapkan	Artificial Bee Colony	Generalized Assignment Problem
2007	Teodorović and Šelmić	BCO	$p$ -Median Problem
2008	Karaboga and Basturk	ABC	Comparison performances of ABC algorithm with the performances of other population-based techniques
2008	Fathian, Amiri, and Maroosi	Honeybee mating optimization algorithm	Cluster analysis
2008	Teodorović	BCO	Comparison performances of BCO algorithm with the performances of other Swarm Intelligence-based techniques
2009	Pham, Haj Darwish, Eldukhr	Bees Algorithm	Tuning the parameters of a fuzzy logic controller
2009	Davidović, Šelmić and Teodorović	BCO	Static scheduling of independent tasks on homogeneous multiprocessor systems

Quelle: Innovations in Swarm Intelligence, S. 42

## 7 Vergleich Bienensysteme / Ameisensysteme

Bienensysteme und Ameisensysteme sind sich sehr ähnlich, da beide von dem Verhalten sozialer Insekten aus der Natur übernommen wurden. Doch gibt es auch einige grundlegende Unterschiede.

Bei beiden Systemen gibt es keine globale Kontrolle. Sie basieren beide auf dem Kooperationskonzept, wodurch die Agenten effizienter ihre Ziele erreichen können. Sowohl Bienen als auch Ameisen haben nur unvollständige, lokale Informationen. Die Daten beider Systeme sind dezentralisiert verfügbar. Auch sind sowohl Bienen- als auch Ameisenalgorithmen auf komplexe NP-vollständige Problemen gut anwendbar.

Der Informationsaustausch geschieht auf unterschiedliche Weise. Ameisen erhalten indirekt ihre Informationen über Pheromone. Sie kennen nicht die Qualität einer Teillösung, die von anderen Ameisen erstellt wurden. Bei den Bienen findet der Informationsaustausch unter allen Agenten direkt auf dem Tanzboden statt. Bei dem von Teodorovic und Lucic entwickeltem BeeSystem-Algorithmus unterliegt das System einem globalem Takt. Dieser Takt ist durch die einzelnen Stufen und Iterationen vorgegeben. So kehren am Ende jeder Stufe alle Bienen in den Bienenstock zurück, oder zu Beginn jeder Iteration wird der Bienenstock neu platziert. Ein solcher Takt ist bei Ameisensystemen nicht vorhanden.

## 8 Schlusswort

Bienenschwarmalgorithmen sind die bisher jüngste Technik, die auf Schwarmintelligenz basieren. Die Meta-Heuristik kann auf verschiedene Optimierungsprobleme angewandt werden, vor allem wenn es um das Lösen NP-vollständiger Probleme geht. Dabei muss der Algorithmus immer auf das jeweilige Problem zugeschnitten werden.

Bisher werden Bienenschwarme nicht eingesetzt, was damit begründet werden kann, dass diese Heuristiken erst vor kurzem entwickelt wurden. Vorläufige Ergebnisse haben gezeigt, dass neue Modelle, die auf Bienenschwarm-Heuristiken basieren, eventuell dazu beisteuern könnten, komplexere Probleme zu lösen.

## Literaturverzeichnis

- Lucic, P., Teodorovic, D.: Bee System Approach to the Modeling of Combinatorial Optimization Problems.  
In: Modeling Transportation Problems Using Concepts of Swarm Intelligence and Soft Computing (2002)
- Wedde, H. F., Farooq, M.: Beeive: New Ideas for Developing Routing Algorithms Inspired by Honey Bee Behavior.  
In: Handbook of Bioinspired Algorithms and Applications (2006)
- Teodorovic, D.: Swarm intelligence systems for transportation engineering: Principles and applications (2008)
- Wong, L., Low, M. Y. H., Chong, C. S.: Bee Colony Optimization with Local Search for Traveling Salesman Problem (2008)
- Teodorovic, D.: Bee Colony Optimization (BCO)  
In: Innovations in Swarm Intelligence (2009)

### Internetquellen:

- Forschungen über Bienen von Karl von Frisch  
[http://de.wikipedia.org/wiki/Karl\\_von\\_Frisch](http://de.wikipedia.org/wiki/Karl_von_Frisch)
- Bientänze  
<http://www.imkerverein-tempelhof.de/html/sprache.html>
- Bibliothek mit zahlreichen TSP-Problemen  
<http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/>