HOCHSCHULE
HANNOVER
UNIVERSITY OF
APPLIED SCIENCES
AND ARTS
–
Fakultät IV
Wirtschaft und
Informatik

Ein Bat-Algorithmus zum Lernen von Complex Event Processing Regeln aus Ereignisdaten

René Knop

Masterarbeit im Studiengang „Angewandte Informatik“

21. August 2017

Autor: René Knop
Am Neuen Krug 50
31582 Nienburg
Matrikelnummer: 1380961
rene.knop@gmx.net

Erstprüfer: Prof. Dr. Ralf Bruns
Abteilung Informatik, Fakultät IV
Hochschule Hannover
ralf.bruns@hs-hannover.de

Zweitprüfer: Prof. Dr. Jürgen Dunkel
Abteilung Informatik, Fakultät IV
Hochschule Hannover
juergen.dunkel@hs-hannover.de

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Masterarbeit selbstständig und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe. Die aus fremden Quellen direkt oder indirekt übernommenen Stellen sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Ort, Datum Unterschrift

Inhaltsverzeichnis

1 Einführung

Heutzutage müssen viele Unternehmen, Organisationen oder Institutionen Informationen aus riesigen Datenmengen beziehen. Eine gescheite Extraktion von tatsächlich relevanten Informationen aus dieser Datenflut erfordert spezielle Technologien. Das sogenannte „Complex Event Processing“ (CEP) ist eine solche Technologie. Mit CEP lassen sich gewünschte Datenkonstellationen modellieren und Datenströme auf solche Konstellationen untersuchen. Tritt tatsächlich eine der gewünschten Konstellationen im Datenstrom auf, so wird eine domänenspezifische Reaktion ausgeführt. Dies lässt sich gut an einem Beispiel verdeutlichen:

Ein Unternehmen möchte gewinnbringend an der Börse handeln. Im Unternehmen arbeiten Fachexperten, die gewisse Zusammenhänge zwischen den Ereignissen am Markt kennen und gut abschätzen können, wann Aktien zu handeln sind und wann nicht. Doch die riesige Menge sich stetig ändernder Börsendaten in Form neuer Ereignisse ist für sie einfach nicht überschaubar – also setzten sie CEP ein. Mit CEP modellieren sie ihr Marktwissen und lassen den Marktdatenstrom computergestützt analysieren. Immer dann wenn eine von ihnen definierte Ereigniskonstellation im Datenstrom vorkommt, dann wird mit „kaufen“ oder „verkaufen“ reagiert. Eine mögliche Konstellation mit entsprechender Reaktion ist beispielsweise: „Wenn die IBM-Aktie im Wert gesunken ist und die Kosten für seltene Erden ansteigen, dann kaufe IBM-Aktien.“

CEP ermöglicht es also höheres Wissen aus einer großen Menge einfacher Informationen abzuleiten und zu formulieren. Damit ist CEP besonders heutzutage und auch zukünftig eine essentielle Technologie zum analysieren massiver Datenströme.

1.1 Motivation

Mit CEP können Fachexperten ihr vorhandenes Wissen modellieren und auf Datenströme ihrer Domäne anwenden. CEP erkennt Ereigniskonstellation und reagiert entsprechend darauf. Der umgekehrte Weg ist jedoch nicht möglich: es ist einem Fachexperten nicht möglich eine Reaktion zu definieren, und daraus eine Ereigniskonstellation durch CEP herleiten zu lassen (Abbildung 1). CEP kann also nicht dabei helfen, wenn die Fachexperten Ursachen für bestimmte Ereignisse herausfinden möchten. Bezogen auf das obige

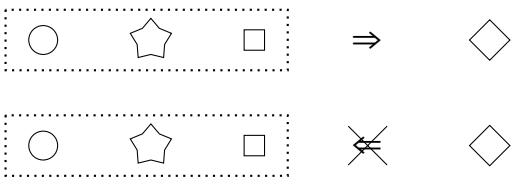


Abbildung 1: CEP kann Ereigniskonstellationen erkennen aber keine Rückschlüsse auf Ursachen ziehen.

Beispiel hieße das, dass die Fachexperten CEP das Ereignis „IBM-Aktie kaufen“ übergeben und CEP findet die Konstellation „IBM-Aktie ist im Wert gesunken und der Wert für seltene Erden ist gestiegen“ – *so etwas kann CEP nicht*.

Im Zuge dieser Masterarbeit wird ein Verfahren entwickelt, das genau diesen umgekehrten Ansatz realisiert. Das Verfahren analysiert Muster in einem Ereignisdatenstrom, mit dem Ziel die Ursachen für ein gegebenes, spezielles Ereignis (beispielsweise „IBM-Aktie kaufen“) zu finden (Abbildung 2). Dafür wird ein Algorithmus aus dem Bereich der so-

Abbildung 2: Das Verfahren in dieser Masterarbeit findet die Ursachen für ein vorgegebenes Ereignis in einem Ereignisdatenstrom

genannten Schwarmintelligenz eingesetzt. Schwarmintelligenz umfasst von der Natur inspirierte Algorithmen, mit denen sich komplexe Probleme lösen lassen. Beispielsweise hat man das Verhalten von Ameisen bei der Futtersuche studiert und festgestellt, dass sie über die Zeit zumeist einen sehr kurzen Pfad zwischen ihrem Bau und dem Futter erschließen, was einer Optimierung entspricht. Dabei folgt jede einzelne Ameise nur wenigen, festen Regeln. Die Regeln und das Verhalten hat man in einen Algorithmus übertragen, mit dem sich anstelle des kürzesten Pfades zum Futter die Lösung für ein anderes Problem finden lässt – wie beispielsweise dem Finden von Ursachen in Ereignisdatenströmen.

Das in dieser Masterarbeit entwickelte Verfahren arbeitet mit mitgeschnittenen Datenströmen, also mit abgeschlossenen Datensätzen die aufgezeichnete Ereignisse beinhalten. Weiterhin muss dem Verfahren das spezielle Ereignis dessen Ursachen man wissen möchte bekannt gegeben werden und zudem muss dieses selbst auch im Datensatz vorkommen. Das Verfahren lernt also Muster aus historischen Ereignisdaten. Die Fachexperten geben also das Ereignis „IBM-Aktien kaufen“ in das Verfahren ein und das Verfahren liefert ihnen die Ereigniskonstellation „IBM-Aktie ist im Wert gesunken und der Wert für seltene Erden ist gestiegen“.

Um das Verfahren möglichst wirklichkeitsgetreu zu evaluieren, werden mithilfe definierter Ereigniskonstellationen in Form sogenannter CEP-Regeln Trainingsdatensätze hergestellt. Dazu werden zufällige Ereignisse in chronologischer Reihenfolge in einem Strom erzeugt und eine solche CEP-Regel auf den Strom angewendet. Die CEP-Regel besteht aus zwei Teilen: einer Bedingung und einer definierten Aktion als Reaktion auf die Erfüllung der Bedingung. Im Kontext dieser Masterarbeit repräsentiert die Ereigniskonstellation im Bedingungsteil die wahre Ursache für das Auftreten des speziellen Ereignisses, das seinerseits im Aktionsteil steht. Immer dann wenn die Konstellation der zufällig erzeugten Ereignisse im Strom die Regelbedingung erfüllt, dann fügt die CEP-Regel das spezielle Ereignis in den Datenstrom ein. Anschließend wird der gesamte Strom in eine Datei gelenkt, und als abgeschlossener Trainingsdatensatz abgelegt. Wenn das Lernverfahren nun auf den Trainingsdatensatz angewendet wird, kann seine Ausgabe, bei der es sich ebenfalls um eine CEP-Regel handelt, mit der zur Erzeugung des Trainingsdatensatzes verwendeten CEP-Regel abgeglichen werden. Durch diesen Vergleich der CEP-Regeln lässt sich feststellen wie gut das Verfahren die Ursachen für das spezielle Ereignis beschreibt.

1.2 Aufbau dieser Masterarbeit

Diese Masterarbeit beginnt im nachfolgenden Abschnitt 2 mit der Beschreibung der grundlegenden Themen: dem Complex Event Processing und den Optimierungsalgorithem aus dem Gebiet der Schwarmintelligenz. Von besonderem Interesse ist der sogenannte Bat-Algorithmus im Abschnitt 2.3, dem die Echo-Ortung kleiner Fledermaus-Arten als Vorbild dient. Danach werden in Abschnitt 3 einige bereits entwickelte Verfahren zum Lernen von Regeln vorgestellt, insbesondere solche Verfahren aus dem Bereich der Schwarmintelligenz. Der darauf folgende Abschnitt 4 zeigt Konzepte und Möglichkeiten auf, mit denen der Bat-Algorithmus an das Lernen von CEP-Regeln angepasst werden kann. Direkt danach wird *BatCEP* im Abschnitt 5 konzeptionell vorgestellt. Es handelt sich dabei um das in dieser Masterarbeit entwickelte Verfahren zum Finden von Ursachen in Ereignisdatenströmen. Seine Implementirung wird im darauf folgenden Abschnitt 6 erläutert. Abschnitt 7 zeigt eine ausführliche Analyse von BatCEP mit verschiedenen Evaluationsergebnissen. Abschließend folgt in Abschnitt 8 eine Zusammenfassung und Bewertung dieser Masterarbeit. In einem Ausblick im selben Abschnitt werden mögliche Aufgaben für zukünftige Projekte vorgeschlagen.

2 Grundlagen

2.1 Complex Event Processing (CEP)

Das sogenannte *Complex Event processing* kurz CEP ist eine Technologie, mit der sich in chronologischer Reihenfolge auftretende und zugleich voneinander abhängige Ereignisse in kontinuierlichen Strömen erkennen und analysieren lassen. CEP kann Zusammenhänge und Muster im Auftreten der Ereignisse finden und daraus höheres Wissen im Form von *komplexen* Ereignissen ableiten. Besonders vorteilhaft ist dabei die Echtzeitverarbeitung, denn CEP analysiert kontinuierliche, massive Datenströme aus verschiedenen Datenquellen ohne Zeitversatz. Eine solche Technologie ist besonders heutzutage von wachsender Bedeutsamkeit, weil immer größere Datenmengen produziert werden. Zugleich sind darunter häufig Daten mit einem sehr geringen Informationsgehalt und um wirklich relevante Informationen daraus zu gewinnen, müssen die Daten miteinander in Verbindung gebracht werden – beispielsweise durch aggregieren oder korrelieren.

CEP hat gewisse Charakteristika und umfasst einige Konzepte, die im Zuge der nachfolgenden Abschnitte tiefer gehend erläutert werden. Beginnend mit der Definition von Ereignissen, über Operationen auf den Datenströmen bis hin zu möglichen Reaktionen auf Mustererkennungen. Dabei sei darauf hingewiesen, dass ein Großteil der Inhalte in den nachfolgenden Abschnitten auf „Complex Event Processing – Komplexe Analyse von massiven Datenströmen mit CEP“ von Bruns und Dunkel [10] und „Event Processing Technical Society: Event Processing Glossary – Version 2.0“ [44] basiert.

2.1.1 Ereignis

Ein Ereignis repräsentiert ein Geschehnis und ist damit all das, was passiert oder von dem man erwartet, dass es passiert. Beispielsweise eine Sensorausgabe, eine Banktransaktion oder eine Veränderung der Temperatur. Damit entspricht ein Ereignis grundsätzlich einem Zustandswechsel jeglicher Art. Überdies abstrahiert CEP Ereignisse, sodass ein physisch auftretendes (rohes) Geschehnis als virtuelles Ereignis mit individuellem Bezeichner repräsentiert wird. Somit ist es beispielsweise auch möglich Ereignisse desselben Typs zu unterscheiden.

Ereignistypen Der Ereignis*typ* bezeichnet die Klasse entsprechender Ereignisinstanzen, so gehören beispielsweise alle aufgezeichneten, tatsächlich durchgeführten Banktransaktionen zur selben Klasse „Banktransaktion“. Jede Ereignisinstanz stammt von einem Typ ab und dieser spezifiziert Eigenschaften für die Instanzen. Daher kann der Typ auch als Definition oder Schema bezeichnet werden. So gibt der Typ Banktransaktion beispielsweise vor, dass jede entsprechende Instanz die Attribut-Wert-Paare `Betrag=x`, `Sender=a` und `Empfänger=b` haben muss. Hinzu kommen Metadaten wie ein eindeutiger Bezeichner, Informationen über die Datenquelle und ein Zeitstempel.

Ereignisinstanzen Eine Ereignisinstanz beschreibt ein konkretes Vorkommnis mit seinen spezifischen Daten. Die aufgezeichnete Banktransaktion von Konto A auf Konto B über

einen Betrag von 200 Euro zum Zeitpunkt 22.6.2017, 10:32 Uhr beispielsweise ist eine Ereignisinstanz. CEP arbeitet wie bereits erwähnt mit Ereignisströmen und daher folgen viele solcher Ereignisinstanzen aufeinander, was als Ereignis*folge* bezeichnet wird.

Ereignisfolgen Eine Sequenz aus aufeinander folgenden Ereignisinstanzen wird als Ereignisfolge bezeichnet. Dabei spielt es keine Rolle, von welchem Ereignistyp die Instanzen abstammen und so kann eine Ereignisfolge folgendermaßen beschrieben werden: $a_1 b_1 a_2 a_3 b_2 c_1 c_2 c_3$. Dabei bezeichnet a_i eine Ereignisinstanz vom Typ A und der Index i steht für das i-te Vorkommen eines Ereignisses dieses Typs. Es fehlen jedoch noch konkrete Bedingungen an die Folge wie beispielsweise die Vorgabe einer besonderen Reihenfolge von ausgewählten Ereignisinstanzen; oder Invertierungen, also dass ausgewählte Ereignisinstanzen explizit nicht in der Folge auftauchen dürfen. Überdies geht die Folge in dieser Form auch noch nicht auf die Attribute der Ereignisinstanzen ein.

2.1.2 Ereignismuster

Mithilfe von *Ereignismustern* lassen sich Bedingungen an die Ereignisinstanzen in Ereignisfolgen knüpfen. Sie ermöglichen es Beziehungen und Einschränkungen zu definieren. Formuliert werden Ereignismuster mit den nachfolgend beschriebenen Operatoren der Ereignisalgebra.

Sequenzen ($\rightarrow$) Der Sequenzoperator ermöglicht es die zeitliche Reihenfolge für zwei Ereignisinstanzen festzulegen. Ein Beispiel hierzu: A $\rightarrow$ B definiert die Sequenz, in der zuerst eine Instanz des Typs A und dann eine Instanz des Typs B im Datenstrom auftauchen muss. Die Folge: $a_1 c_1 b_1 a_2 c_2$ erfüllt demnach die Sequenzbedingung.

Boolsche Operatoren ($\wedge, \vee$) Boolsche Operatoren definieren, welche Typen im Datenstrom vorkommen müssen. Sie haben im Gegensatz zum Sequenzoperator jedoch keinen Anspruch an die Reihenfolge. Ein Beispiel hierzu: A $\wedge$ B bedeutet, dass im Datenstrom sowohl eine Instanz des Typs A als auch eine Instanz des Typs B auftauchen muss. Die Folge: $b_1 c_1 a_1 a_2 a_3$ erfüllt demnach die Bedingung, weil mit b_1 und $a_1 a_2 a_3$ sowohl eine Instanz des Typs B als auch Instanzen des Typs A enthalten sind.

Negation ($\neg$) Der Negationsoperator definiert, dass für einen bestimmten Ereignistypen keine Instanz vorkommen darf. Ein Beispiel hierzu: $\neg$A besagt, dass im Datenstrom keine Instanz des Typs A auftauchen darf. Die Folge: $b_1 c_1 c_2 a_1 b_2$ erfüllt die Bedingung folglich nicht, weil mit a_1 eine Instanz des Typs A enthalten ist. Der Negationsoperator kann jedoch nicht auf den gesamten Datensatz, sondern nur innerhalb eines Ausschnittes dessen (Siehe Abschnitt „Fenster“) sinnvoll eingesetzt werden. Würde er auf den ganzen Datenstrom angewendet werden, so käme das dem kompletten Ausschluss aller Instanzen des entsprechenden Typs gleich.

All diese Operatoren lassen sich auch miteinander kombinieren um komplexe Muster zu bilden. Hierzu seien ein paar Beispiele gegeben:

`A → (B ∧ C)`

Dieses Muster sucht nach einer Ereignisfolge, in der nach einem Ereignis des Typs A jeweils ein Ereignis des Typs B und des Typs C folgen. Die Ereignisfolge: $c_1 b_1 b_2 a_1 b_3 b_4 c_2$ erfüllt das Muster wegen $a_1 b_3 b_4 c_2$.

`¬A → (B ∨ C)`

Dieses Muster erfordert eine Instanz des Typs B oder des Typs C, die aber beide nicht auf eine Instanz vom Typ A folgen dürfen. Anders formuliert: ein Ereignis des Typs A darf nicht vor einem Ereignis des Typs B oder einem des Typs C auftreten. Die Ereignisfolge: $b_1 a_1 c_1 b_2$ erfüllt das Muster wegen b_1; diese jedoch nicht: $a_1 b_1 c_1$ weil sie bereits mit einem Ereignis des Typs A beginnt.

`(A ∧ B) → (B ∧ C)`

Dieses Muster sucht nach einer Ereignisfolge, in der zuerst Ereignisse der Typen A und B vorkommen. Darauf müssen Ereignisse der Typen B und C folgen. Die Ereignisfolge: $a_1 b_1 a_2 b_2 c_1$ erfüllt das Muster. Dieses hingegen erfüllt es nicht: $a_1 b_1 c_1 c_2$, weil hier nur ein Ereignis des Typs B vorkommt, es aber zwei sein müssen.

Kontextbedingungen Kontextbedingungen definieren Anforderungen an die Attribut-Werte der Ereignisinstanzen. Folgende Sprachkonstrukte sind notwendig, um Ereignisinstanzen in Mustern explizit auszeichnen zu können und um auf ihre Attribute zuzugreifen:

- *Aliasnamen* ermöglichen es in Bedingungen Ereignisinstanzen zu benennen (Schlüsselwort `AS`) und dann mithilfe des Namens auf das Ereignis zuzugreifen. Somit können auch Ereignisse desselben Typs unterscheiden werden.
- Der „.“-Operator wird verwendet, um auf Attribute eines Ereignisses zuzugreifen.
- Durch Einbringung *domänenspezifischer Daten* mithilfe von Methodenaufrufen in den Regeln, lassen sich weitere Kontextbedingungen formulieren. So kann beispielsweise der Wert des Ereignis-Attributs `Betrag` mit der Rückgabe der Methode `checkGuthaben()` aus der Anwendungsklasse `Konto` verglichen werden.
- Wertvergleiche sind mit den üblichen Vergleichsoperatoren möglich:

  $+, -, *, / <, >, \leq, \geq, =, \neq$.

Auch hierzu seien ein paar Beispiele gegeben:

```
(A AS a) → (B AS b) ∧ (a.temperatur = b.temperatur)
```

Gesucht wird eine Ereignisfolge, in der ein Ereignis des Typs B auf ein Ereignis des Typs A folgt. Als weitere Bedingung ist gefordert, dass beide Ereignisse das gleichwertige Attribut `temperatur` besitzen.

```
(Banktransaktion AS t) ∧ (t.betrag > Konto.checkGuthaben(t.sender))
```

Hier wird im Datenstrom nach Banktransaktionen gesucht, deren Beträge größer sind als das Guthaben auf dem Sender-Konto. Zur Überprüfung des Kontos wird die Methode `checkGuthaben()` aus der Klasse `Konto` aufgerufen.

```
((DruckEreignis AS d1) → (DruckEreignis AS d2) → (WasserEreignis)) ∧
(d1.ventil = d2.ventil ∧ d1.wert > DB.getMax(d1.vendor) ∧ d2.wert = 0)
```

Dieses Beispiel bezieht sich auf Ausgaben von Druck- und Feuchtigkeitssensoren in einer Industriekaffeemaschine. Drucksensoren sind an Ventilen angebracht und Feuchtigkeitssensoren im Bereich elektronischer Schaltkreise. Gesucht wird nun eine Ereignisfolge, in der zuerst ein Druckereignis mit einem kritisch großen Wert auftritt. Um den Wert zu überprüfen wird die Datenbank-Methode `getMax()` mit dem Argument `d1.vendor` aufgerufen, um den maximal zulässigen Druckwert eines Ventils des Herstellers `vendor` zu beziehen. Dieses Druckereignis wird gefolgt von einem weiteren Druckereignis am selben Ventil, dessen Wert bei Null liegt. Darauf folgt letztlich noch ein Ereignis eines Feuchtigkeitssensors (WasserEreignis). Damit realisiert dieses Muster den realen Fall, in dem ein Ventil überlastet und dadurch beschädigt wird. Als Konsequenz tritt Wasser aus, das in den Bereich elektronischer Schaltkreise fließt.

Solche Bedingungen lassen sich noch präziser definieren, wenn die relevanten Ereignisse in zeitliche Rahmen gelegt werden. Bezogen auf das letzte Beispiel könnte eine weitere Bedingung vorgeben, dass die beiden Druckereignisse sehr schnell aufeinander folgen müssen, um ihre kausale Abhängigkeit zu unterstreichen. CEP verwendet solche Rahmen, sie werden als *Fenster* beziehungsweise als „Sliding Windows“ bezeichnet und im nachfolgenden Abschnitt beschrieben.

Fenster Fenster grenzen bestimmte Mengen von Ereignissen in Ereignisströmen ab, indem sie die Anzahl der betrachteten Ereignisse beschränken. Sie unterscheiden sich in *Längenfenster* und *Zeitfenster*. Ein Längenfenster der Länge *n* bezieht *n* Ereignisse des Datenstroms ein und wird syntaktisch mit `win:len:n` angegeben. Ein Zeitfenster der Größe *n* macht sich hingegen das zeitlich fortlaufende Auftreten der Ereignisse zunutze und beschränkt den Datenstrom für eine gegebene Bedingung auf einen Zeitraum der Größe *n*, wobei sich *n* in einer definierbaren Zeiteinheit bemisst. So betrachtet eine Bedingung beispielsweise alle Ereignisse in einem Zeitraum von zwei Minuten. Syntaktisch ist ein Zeitfenster mit `win:time:n` anzugeben.

Um die Anwendung von Fenstern zu demonstrieren, werden zwei der zuvor aufgeführten Beispiele erneut aufgegriffen und mit Fenstern versehen:

```
A → (B ∧ C)[win:len:10]
```

Hier bezieht sich das Längenfenster auf die Teilbedingung (B ∧ C). Zu Beginn muss ein Ereignis von Typ A im Datenstrom vorkommen. Dieses hat keinen Bezug zum Fenster, es muss einfach auftreten. Wenn dann im späteren Verlauf ein Ereignis des Typs B auftritt, dann muss sich unter den darauf folgenden zehn Ereignissen auch eines des Typs C befinden, damit das Muster erfüllt ist.

```
(¬A → (B ∨ C))[win:time:2min]
```

In diesem Fall bezieht sich das Fenster auf die gesamte Bedingung. In einem Zeitraum von zwei Minuten muss ein Ereignis des Typs B oder C (oder beide) vorkommen. Zugleich darf aber kein Ereignis des Typs A in diesem Zeitraum vorangegangen sein, damit das Muster erfüllt ist.

Aggregationen Mithilfe von *Aggregationen* lassen sich die Werte einer definierten Menge von Ereignissen aus einem Datenstrom zusammenfassen. Weil Datenströme wie bereits erwähnt, aus kontinuierlichen Daten bestehen und damit keine abgeschlossene Menge bilden, sind Aggregationen nur in Kombination mit Fenstern sinnvoll einzusetzen. Typische Aggregationsfunktionen sind:

- `sum()`: bildet die Summe der Attribut-Werte über alle Ereignisse der definierten Menge
- `avg()`: bildet den Durchschnittswert eines Attributes über alle Ereignisse der definierten Menge
- `min()` und `max()`: liefern den minimalen oder maximalen Wert eines Attributes aus allen Ereignissen der definierten Menge

Auch hierzu sei ein Beispiel gegeben. Es berechnet den maximalen Wert der Temperatur über einen Zeitraum von zwanzig Minuten und benennt das Ergebnis als `maxTemp`.

```
TemperaturEreignis.max(temperaturWert)[win:time:20min] AS maxTemp
```

Jedes neue Temperaturereignis bedingt dabei eine neue Berechnung. Soll die Berechnung jedoch erst nach Ablauf des Fensters, also in diesem Fall nach zwanzig Minuten durchgeführt werden, so müssen alle Ereignisse bis dahin gesammelt beziehungsweise „gestapelt“ werden. Dies ist mit einem sogenannten *Batch-Window* (frei übersetzt: Stapel-Fenster) möglich:

```
TemperaturEreignis.max(temperaturWert)[win:time:batch:20min] AS maxTemp
```

2.1.3 Ereignisregel

Ereignisregeln bestehen aus zwei Komponenten: einer *Bedingung* und einer *Aktion* und sie lassen sich wie folgt beschreiben:

`Bedingung P(`$e_1, e_2, ..., e_n$`) Aktion A(`$e_1, e_2, ...e_n$`)`

Die Bedingung beinhaltet ein Ereignismuster in der Form, die in den letzten Abschnitten beschrieben wurde. Die Sequenz $e_1, e_2, ..., e_n$ ist eine zeitlich fortlaufende Ereignisfolge. Die Aktion wird bei der Erfüllung des Musters (dem sogenannten „Feuern“ der Regel) durchgeführt und bekommt die Ereignisfolge und damit den Aktionskontext als Argument. Damit hat die Aktion Zugriff auf die Ereignisse aus der Folge.

Aktion Die *Aktion* stellt eine Reaktion auf das erfüllte Muster dar und kann auf eine der zwei nachfolgenden Arten realisiert werden:

Neue Ereignisse erzeugen (Transaktion) Es wird ein neues, komplexes Ereignis erzeugt (Schlüsselwort: `create`), das die gewonnene Information aus dem erkannten Muster beschreibt oder die darin enthaltenen Werte aufbereitet. Anschließend wird das neu erzeugte Ereignis zurück in einen Datenstrom gegeben und somit von einer CEP-Komponente weiterverarbeitet. Als Beispiel sei die folgende Aktion gegeben:

```
create BauteilschadenEreignis(ventil=a, druck=kritisch)
```

Dienste anstoßen Die Aktion stößt einen Dienst in einem nachgelagerten Anwendungssystem an, der seinerseits domänenspezifische Aktionen durchführt.

Transaktionen werden weiterhin in zwei verschiedene Klassen eingeordnet. Zum einen in solche, die Ereignisse nur umwandeln und keine neuen Informationen hinzufügen. Zum anderen in solche, die mehr Informationen beinhalten als die ursprüngliche Ereignisreihenfolge. Für weitere Details dazu sei auf [10], Seite 27 verwiesen.

Regelsprache Für die Formulierung von Ereignisregeln werden sogenannte „Event Processing Languages“ (EPL) verwendet. Dabei handelt es sich um höher abstrahierte, deklarative Regelsprachen, mit denen sich Regeln auf eine für Menschen verständliche Weise beschreiben lassen. Ein Beispiel hierzu ist die an SQL angelehnte EPL von *Esper* [1], einem quelloffenen und in Java implementierten CEP-Produkt:

```
select avg(temperaturWert)
from TemperaturEreignis.win:time(30 min)
where raum='Aula'
```

Gesucht ist der durchschnittliche Temperaturwert über alle Ereignisse des Typs `TemperaturEreignis`, die innerhalb der letzten halben Stunde empfangen wurden. Eine weitere Bedingung fordert, dass nur solche Ereignisse betrachtet werden, die aus der „Aula“ stammen.

Diese Sprache ist nur ein Beispiel für eine EPL, denn es gibt keinen etablierten Standard. So bringen unterschiedliche CEP-Produkte ihre eigenen Sprachen zum formulieren von CEP-Regeln mit.

2.1.4 „Event Processing Agents“ und „Event Processing Networks“

Der sogenannte „Event Processing Agent“ (EPA) ist ein Softwaremodul, das Ereignisdatenströme empfängt und Ereignisse daraus verarbeitet. Damit ist es ein Kernmodul in einem ereignisgesteuerten System, das drei wesentliche Komponenten beinhaltet:

- Ereignismodell: spezifiziert die erlaubten Ereignistypen und die zugehörigen Attribute, sowie die Beziehungen und Abhängigkeiten zwischen den Typen.
- Ereignisregeln: werden auf der Basis des Ereignismodells zur Verarbeitung der Ereignisse mithilfe einer EPL definiert.
- *Event Processing Engine*: ist ein spezieller Regelinterpreter und realisiert die eigentliche Mustererkennung. Dazu gleicht sie den Datenstrom kontinuierlich mit den Bedingungsteilen der hinterlegten Ereignisregeln ab und sobald ein Ereignismuster im Datenstrom passt, wird die Aktion der entsprechen Regel ausgeführt. Da oftmals auch Ereignisse aus der Vergangenheit relevant sind, werden relevante Ausschnitte

[1] `http://www.espertech.com/esper/release-5.3.0/esper-reference/html/epl_clauses.html` (zuletzt zugegriffen am 21. August 2017)

(vorgegeben von den Fenstern in den Regeln) des Datenstroms im Arbeitsspeicher vorgehalten.

Ein EPA wendet also Ereignisregeln auf einen Ereignisstrom an und führt damit verschiedene Berechnungen durch wie beispielsweise Aggregationen, Filterungen oder das Erkennen von Ereignismustern. Weiterhin kann sie entweder eine Ereignissenke oder eine Ereignisquelle sein. Sie empfängt also einerseits Ereignisse, die sie auswertet und andererseits kann sie auch selbst Ereignisse produzieren und diese zurück in einen Datenstrom leiten. Ein EPA ist eine „kleine“ Einheit, die einen klar umrissenen, domänenspezifischen Aufgabenbereich hat. Mehrere EPA realisieren in Zusammenarbeit eine größere Aufgabe im System.

Das „Event Processing Network“ (EPN) ist ein Netzwerk aus mehreren, untereinander kommunizierenden EPA. Damit verknüpft es mehrere Teilfunktionalitäten im System zu einer größeren Funktionalität. Weiterhin kann ein einzelner EPA auch als EPN aufgebaut sein, so kann er seine Teilfunktionalität im System auch mithilfe weiterer, untergeordneter EPA realisieren, die noch kleinere Teilaufgaben haben.

2.2 Biologisch inspirierte Optimierungsalgorithmen (BIO)

Dieser Abschnitt macht einen Sprung zu den von der Natur inspirierten Optimierungsalgorithmen. Optimierung verfolgt das Ziel die bestmögliche Lösung für ein Problem zu finden. Die Realisierung mittels deterministischer Verfahren gelingt dabei jedoch nicht immer. Besonders dann nicht, wenn das Problem komplex und sein Suchraum sehr groß ist. In solchen Fällen stoßen deterministische Verfahren an die Grenzen der Berechenbarkeit. Um dennoch komplexe Probleme optimieren zu können, stehen stochastische, metaheuristische Verfahren zur Verfügung und einige dieser Verfahren sind von Abläufen und Verhaltensweisen aus der Natur motiviert – das sind die sogenannten biologisch inspirierten Optimierungsalgorithmen (BIO). Warum ist die Natur ein gutes Vorbild zum Lösen von Problemen? Die Natur findet zumeist sehr gute Lösungen für Probleme, oftmals sogar mit wenig Vorwissen über den Suchraum. Sie ist dynamisch, komplex, robust und Entscheidungen basieren häufig auf dem Zufall. Das sind gute Eigenschaften für nicht-deterministische Lösungsansätze. Zudem können viele Prozesse in der Natur als Prozesse der bedingten Optimierung angesehen werden [7]. Beispielsweise unterliegt die Futtersuche einer Ameise bekannten, fest vorgegebenen Verhaltensweisen: die Ameise läuft mit einer gleichbleibenden Geschwindigkeit v einen zufällig gewählten Weg der Länge l ab und hinterlässt dabei Pheromon-Spuren der Stärke τ. Mit diesen drei Attributen und einer Zufallsvariable lässt sich die *Futtersuche* algorithmisch abbilden. Dieses Beispiel ist ein Teilschritt des in den nachfolgenden Abschnitten etwas genauer beschriebenen „Ameisen-Algorithmus“, bei dem es sich um einen *schwarmbasierten* Algorithmus handelt. Grundsätzlich teilt sich die Menge aller BIO auf drei Paradigmen auf: evolutionäre Algorithmen, ökologische Algorithmen und schwarmbasierte Algorithmen. Binitha und Siva haben dies in [7] übersichtlich dargestellt. In dieser Masterarbeit wird ein schwarmbasierter Algorithmus eingesetzt und aus diesem Grund wird das schwarmbasierte Paradigma im nachfolgenden Abschnitt genauer erläutert.

Schwarmbasierte Algorithmen (SA) SA sind vom gemeinschaftlichen Sozialverhalten einfacher Individuen wie Ameisen, Bienen oder Fischen inspiriert. Diese können ein großes Ziel wie beispielsweise die Erschließung einer neuen Futterquelle nur erreichen, wenn sie strukturiert zusammenarbeiten. Doch wie können Individuen, die nur über eine äußerst begrenzte Intelligenz verfügen, strukturiert zusammen? Die Antwort ist einfach: *Selbst-Organisation* [8]. Der ganze Schwarm kann große (globale) Ziele nur erreichen, weil jeder einzelne Organismus mit seiner beschränkten, lokalen Sichtweise nach immer gleichen Verhaltensmuster agiert – sich also selbst organisiert. Selbst-Organisation basiert auf multiplen Interaktionen und steht für die nachfolgenden Eigenschaften:

Positive Reaktion ist die Fähigkeit anderen Individuen sein Wissen über etwas nützliches mitzuteilen und zwar auf eine Weise, die von den anderen verstanden wird. Der Tanz einer Biene ist ein Beispiel hierfür. Wenn eine Biene eine neue Futterquelle gefunden hat, dann fliegt sie zum Bienenstock zurück und teilt den Bienen dort durch einen Tanz die Richtung und Distanz zur Futterquelle mit. Ein weiteres Beispiel für eine positive Reaktion ist das Markieren eines Pfades. Ameisen markieren ihre zurückgelegten Pfade mit einem Pheromon, das von anderen Ameisen wahrgenommen wird. Somit zeigen sie sich gegenseitig Pfade auf.

Negative Reaktion gleicht die positive Reaktion aus und stellt sich in der Realität beispielsweise durch Verdunstung dar. Wenn eine Ameise einen *uninteressanten Pfad* abwandert und auf diesem wie üblich eine Pheromonspur hinterlässt, dann wird die Spur im Laufe der Zeit verdunsten. Grund hierfür ist Tatsache, dass nur wenige Ameisen oder gegebenenfalls sogar keine weitere Ameise diesem Pfad folgt und die Spur demnach nicht erneuert wird. Negative Reaktion ist also in der Regel das Nicht-Vorhandensein von Informationen.

Schwankungen im Verhalten sind wichtig, weil sie für Zufallshandlungen sorgen und Zufallshandlungen sind wichtig, um neue Gebiete zu erschließen. Wenn Ameisen beispielsweise immer derselben Pheromonspur folgen würden, dann würden sie in irgendeiner Form im Kreis wandern. Um dies zu verhindern entscheiden sie manchmal einen neuen Pfad zu wählen und erschließen damit gegebenenfalls neue Futterquellen.

Unbeeinflusste Individuen Ein Schwarm profitiert davon, wenn er eine kleine Anzahl von Individuen hat, die sich dem Schwarm-üblichen Verhalten widersetzen. Sie lassen sich nicht oder nur geringfügig von den Spuren der anderen Schwarm-Mitglieder beeinflussen oder sind sogar in der Lage ihre eigenen Spuren von fremden zu unterscheiden. Diese Individuen erschließen häufig komplett neue Pfade oder folgen beliebten Pfaden und erweitern diese in unbekannte Richtungen.

Um einen SA zu implementieren bedarf es einer Population, bestehend aus Gruppen primitiver Individuen sowie der Beachtung der fünf nachfolgenden „Prinzipien der Schwarm-Intelligenz“, die von Binitha und Siva in [7] und von Kennedy und Eberhart in [35] beschrieben sind als:

1. *Näherungsprinzip*: Die Population sollte einfache Raum- und Zeitberechnungen durchführen können.

2. *Qualitätsprinzip*: Die Population sollte auf Qualitätsfaktoren (Ergebnisse von *Qualitätsmessungen*, Lösungen von Nachbarn, o.ä.) reagieren können.
3. *Kommunikationsprinzip*: Die Population sollte ihre Aktivitäten mitteilen. Beispielsweise sollte das lokal beste Individuum seine Lösung der ganzen Population mitteilen, sodass sich andere Individuen daran orientieren können.
4. *Stabilitätsprinzip*: Veränderungen in der Umgebung sollten die Population nicht in ihrem grundsätzlichen Verhalten beeinflussen.
5. *Anpassungsprinzip*: Die Population sollte ihr Verhalten anpassen, wenn dies den Berechnungsaufwand verringert.

Zu bekannten und teilweise schon seit geraumer Zeit in vielen Bereichen eingesetzten SA zählen: die „Partikelschwarmoptimierung“, die am allgemeinen Verhalten von Schwarmtieren orientiert ist; sowie der inhaltlich bereits beschriebene Ameisen-Algorithmus („Ant Colony Optimization“), der sich das Verhalten der Ameisen zum Erschließen kürzester Pfade zwischen ihrem Nest und dem Futter zunutze macht; der „Bat-Algorithmus“, der von der Echo-Ortung kleiner Fledermaus-Arten inspiriert ist; sowie der sogenannte „Artificial Bee Colony Algorithm“, der „Firefly Algorithm“, der „Artificial Immune System Algorithm“ und viele mehr. Für eine grobe Übersicht über weitere BIO sei als Einstieg auf den entsprechenden Wikipedia-Eintrag [2] verwiesen. Eine detailliertere Übersicht ist in [7] dargelegt und Bonabeau, Dorigo und Theraulaz klassifizieren einige SI-Algorithmen in [8]. Kennedy und Eberhart gehen noch tiefer in die Grundlagen und thematisieren in [36] „Intelligenz“ zunächst recht umfangreich. Sie beschäftigen sich unter anderem damit, wie sich Intelligenz mithilfe der Partikelschwarmoptimierung auf einen Computer übertragen lässt.

In den nachfolgenden Abschnitten wird auf die Partikelschwarmoptimierung, den Ameisen-Algorithmus und den Bat-Algorithmus tiefgründiger eingegangen. Die Partikelschwarmoptimierung und der Ameisen-Algorithmus wurden gewählt, weil sie schon für eine Vielzahl von Anwendungsfällen eingesetzt worden sind und dabei gute Ergebnisse erzielten. Der Bat-Algorithmus ist im Vergleich dazu recht neu und weniger erprobt. Zugleich wird an einigen Stellen in der Literatur gezeigt, dass der Bat-Algorithmus gute Ergebnisse erzielt. Besonders im direkten Vergleich mit anderen BIO [69, 21, 23], was ihn zu einem interessanten Kandidaten für diese Masterarbeit macht.

2.2.1 Partikelschwarmoptimierung

Die Partikelschwarmoptimierung (PSO) ist eine Populations-basierte und globale Optimierungstechnik, die von James Kennedy und Russel Eberhart entwickelt und im Jahr 1995 erstmals in [35] veröffentlicht wurde. Das Verfahren sucht nach dem Vorbild des Schwarmverhaltens von Tieren (beispielsweise Vogel- oder Fischschwärme) eine Lösung für ein Optimierungsproblem F_{opt} im n-dimensionalen Raum und liefert dabei schnell akzeptable Ergebnisse. Vorteilhaft an dem Verfahren sind zudem die wenigen Parameter, die vom Benutzer voreingestellt werden müssen.

[2] `https://en.wikipedia.org/wiki/Bio-inspired_computing` (zuletzt zugegriffen am 21. August 2017)

Zu Beginn wird eine Initialisierungsphase durchlaufen, in der eine Gruppe aus zufällig im Suchraum der zu optimierenden Funktion positionierten Partikeln erzeugt wird. Jeder Partikel stellt mit seiner Position im Suchraum einen möglichen Lösungskandidaten dar. Im Anschluss an die Initialisierung *sucht* jeder Partikel nach einer optimalen Lösung, indem er seine Position im Raum verändert. Masse und Volumen werden dabei vernachlässigt, sodass die Partikel weder eine Masse noch ein Volumen haben. Aus diesem Grund kann es keine Kollisionen geben.

Jedes Partikel trägt folgende Informationen mit sich: die eigene aktuelle Postion x, die eigene beste Position p, die eigene Geschwindigkeit v sowie die beste Position im ganzen Schwarm Y. Die eigene Position wird in jeder Iteration verändert. Dies geschieht in Abhängigkeit von der eigenen Geschwindigkeit und Bewegungsrichtung, sowie der eigenen besten Position und der Schwarm-besten Position und hängt damit also auch von nachbarschaftlichen Einflüssen ab. Doch wie setzt sich eine Nachbarschaft zusammen? Dafür gibt es verschiedene Möglichkeiten, die von den sogenannten *Topologien* vorgegeben werden. Topologien sind Beziehungsmodelle, mit denen festgelegt wird welche Partikel für den Zeitraum eines ganzen PSO-Durchlaufes Nachbarn sind. Die Wahl der Topologie beeinflusst die Laufzeit in großem Maße und muss problemspezifisch gewählt werden. Für weitere Details sei auf [34] verwiesen.

Die erste Version des Verfahrens ist als Pseudocode in Algorithmus 1 dargestellt. Dabei bezeichnet x_i die aktuelle Position eines Partikels; p_i die beste je erreichte Position desselben Partikels und v_i seine aktuelle Geschwindigkeit. Die Geschwindigkeit v_i und die Position x_i werden in jeder Iteration mit den Gleichungen 1 und 2 aktualisiert. Diese erste

$$v_i^{t+1} = v_i^t + c_1 r_1 [y_i^t - x_i^t] + c_2 r_2 [Y^t - x_i^t] \tag{1}$$

$$x_i^{t+1} = x_i^t + v_i^{t+1} \tag{2}$$

Version des Algorithmus wurde im Laufe der Zeit mehrfach angepasst und verbessert weil sie Schwächen aufwies. Die sogenannte „Schwarmexplosion“ ist eine dieser Schwächen und sie beschreibt den Fall, in dem sich alle Partikel von einander entfernen, sodass keine Konvergenz an einem lösungsnahen Punkt erfolgt. Für weitere Informationen über und Maßnahmen gegen die Schwarmexplosion sei auf [12, 17] verwiesen.

PSO wurde als eine der ersten Schwarm-basierten Metaheuristiken, schon auf verschiedene Probleme abgebildet. Das Lernen von Regeln gehört auch dazu und eine entsprechende Implementierung ist in [43] beschrieben.

Algorithmus 1 Partikelschwarmoptimierung

```
1:  procedure PSO
2:      Definiere die zu optimierende Funktion F_opt;
3:      Initialisiere den n-dimensionalen Partikelschwarm;        ▷ Population erzeugen
4:      Evaluiere alle Partikel i des Schwarmes mit F_opt(x_i) um Y herauszufinden;
5:      while Stopp-Kriterium nicht erfüllt do
6:          for all Partikel i aus dem Schwarm do
7:              Aktualisiere die Geschwindigkeit v_i (Gleichung 1);
8:              Aktualisiere die Position x_i (Gleichung 2);
9:              if F_opt(x_i) < F_opt(p_i) then
10:                 p_i ← x_i;                          ▷ lokal beste Position p_i aktualisieren
11:             end if
12:             if F_opt(p_i) < F_opt(Y) then
13:                 Y ← y_i;                            ▷ global beste Position Y aktualisieren
14:             end if
15:         end for
16:     end while
17:     Gebe die global beste Position Y als Lösung für F_opt zurück;
18: end procedure
```

2.2.2 Ameisen-Algorithmus

Der Ameisen-Algorithms (*Ant Colony Optimization* [ACO]) ist mittlerweile das wohl bekannteste schwarmbasierte Optimierungsverfahren. Es wurde erstmals im Jahr 1999 von Dorigo und Caro in [16] vorgestellt. Dabei handelt es sich um ein Verfahren, das vom Verhalten von Ameisen insbesondere bei der Futtersuche inspiriert ist. Auch hier geht es wieder darum ein spezielles Problem F_{opt} durch das Finden korrekter Parameter dafür zu lösen.

Ameisen erschließen über die Zeit zumeist den kürzesten Pfad zwischen ihrem Nest und dem gefundenen Futter. Dies ist möglich, weil alle Ameisen (bestimmter Arten) auf eine ganz spezielle Weise – die sogenannte *Stigmergie* [3] – miteinander interagieren. Der Stigmergie liegt die Tatsache zugrunde, dass eine große Anzahl von Individuen ihre Umgebung verändert, um sich zu koordinieren und miteinander zu kommunizieren. Dazu verwenden Ameisen sogenanntes „Pheromon“, ein Botenstoff der bei anderen Ameisen wiederum zu einer Verhaltensreaktion führt. Je mehr Ameisen denselben Pfad abwandern, desto stärker ist die Pheromonspur, was den Pfad wiederum noch attraktiver macht und noch mehr Ameisen anlockt. Die Wahrscheinlichkeit dafür, dass eine Ameise einen Pfad auswählt, steht folglich proportional zur Anzahl der Ameisen, die den Pfad bereits markiert haben.

[3] Stigmergie: https://de.wikipedia.org/wiki/Stigmergie (zuletzt zugegriffen am 21. August 2017)

Algorithmus 2 Ameisen-Algorithmus

```
1:  procedure ACO
2:      Definiere die zu optimierende Funktion F_opt;
3:      Initialisiere die Ameisen-Kolonie;                ▷ Population erzeugen
4:      Initialisiere den Pheromonwert;
5:      while Stopp-Kriterium nicht erfüllt do
6:          for all Ameisen do
7:              Erzeuge eine einen Pfad;                  ▷ Lösungskandidat erzeugen
8:              Platziere eine Pheromonspur auf dem Pfad;
9:          end for
10:         Verflüchtige die Pheromonspuren ungenutzter Pfade;
11:     end while
12:     Liefere den besten Pfad als Lösung für F_opt zurück;
13: end procedure
```

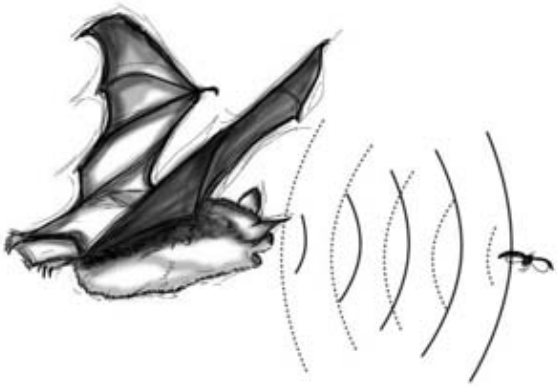
Befindet sich ein Hindernis auf einem zunächst unmarkierten Pfad, so wandert die Ameise mit einer fünfzig-prozentigen Wahrscheinlich links oder rechts daran vorbei und markiert ihren Pfad. Eine nachfolgende Ameise, die von dieser schwachen Pheromonspur noch nicht so stark beeinflusst wird, wandert gegebenenfalls an der anderen Seite am Hindernis vorbei. Selbiges Verhalten setzt sich bei allen nachfolgenden Ameisen solange fort, bis einer der Pfade, links oder rechts sehr stark markiert ist. Dann wird dieser so gut wie immer und von jeder nachfolgenden Ameise ausgewählt. Doch wieso sollte gerade der kürzeste Pfad am stärksten markiert werden? Das liegt in der Tatsache begründet, dass sich alle Ameisen mit derselben Geschwindigkeit bewegen. Kurze Pfade werden folglich öfter abgewandert und dadurch häufiger markiert. Der Algorithmus 2 zeigt den generellen Ablauf und damit die ursprüngliche Idee zu ACO. Auch hier wird zunächst das zu optimierende Problem als Funktion F_{opt} definiert. Dann beginnt das Optimierungsverfahren durch die Erzeugung der initialen Population in Form einer Ameisen-Kolonie und durch Festlegung des Pheromonwertes. Dieser Wert bestimmt wie stark hervorgebrachte Lösungsansätze für F_{opt} als „gut“ markiert werden. Im Hauptteil des Algorithmus (Zeile 5 bis 11) erzeugt jede Ameise nun einen Lösungsansatz in Form eines Pfades, der sich aus mehreren Teilpfaden zusammensetzt und den sie mit Pheromon markiert. Gute Pfade, also solche die häufig und von verschiedenen Ameisen erzeugt werden, haben folglich einen hohen Pheromonwert. Schlechte Pfade hingegen nicht, deren Pheromon wird im Laufe der Zeit verflüchtigt. Je größer der Pheromonwert eines Pfades ist, desto mehr Ameisen werden Teilpfade daraus in den Erzeugungsprozess eigener Pfade übernehmen. Insgesamt bemisst sie die Qualität eines Pfades zum einen anhand seines Pheromonwertes und zum anderen durch das Ergebnis beim Einsetzen des Pfades in F_{opt}.

2.3 Bat-Algorithmus (BA)

Der Bat-Algorithmus ist ein metaheuristisches Verfahren für globale Funktionsoptimierung, das in seiner originalen Version *kontinuierliche* Funktionen optimiert. Das im Zuge dieser Masterarbeit entwickelte Verfahren setzt den Bat-Algorithmus (zukünftig nur noch

als BA bezeichnet) als Optimierungsalgorithmus ein. Seine guten Ergebnisse, verglichen mit anderen SA in der einschlägigen Literatur sind die Gründe hierfür. Deshalb wird der BA nachfolgend sehr ausführlich beschrieben.

Biologisches Vorbild Der BA ist von der Echo-Ortung kleiner Fledermaus-Arten inspiriert und wurde im Jahr 2010 von Yang in [69] vorgestellt. Fledermäuse stoßen während ihrer Flüge Rufe im Frequenzbereich von etwa 20kHz bis 150kHz (Ultraschall) aus und orientieren sich räumlich anhand der Echos. Damit sind sie in der Lage Hindernisse wahrzunehmen. Selbst solche mit den Maßen eines menschlichen Haares können sie erkennen. Ebenso spüren sie durch ihre Rufe Beute auf und erkennen dabei deren Größe und Position. Die Ruflänge bemisst sich im Normalfall auf etwa 5ms bis 20ms. Dabei werden pro Sekunde etwa 10 bis 20 Rufe mit einer gewissen Wellenlänge λ ausgestoßen. Befindet sich eine Fledermaus auf der Jagd und nähert sich der Beute an, dann kann die Rufrate auf bis zu 200 Rufe pro Minute ansteigen. Gleichzeitig nimmt die Lautstärke ab, die im Normalfall bei etwa 110dB liegt. Studien haben gezeigt, dass Fledermäuse ihre Beute sogar anhand von Variationen des Doppler-Effekts unterscheiden können, die durch unterschiedliche Flügelschlagraten verschiedener Beutetiere entstehen [6]. Obwohl viele Fledermaus-Arten noch weitere Sinne haben und diese auch zur Ortung und auf der Jagd verwenden, beschränkt sich der von Yang vorgestellte Algorithmus konzeptionell auf die Funktionsweise der Echo-Ortung. Dazu werden charakteristische Eigenschaften dieses Sonar-Typs wie beispielsweise die Frequenz, die Pulsrate, die Geschwindigkeit und die Lautstärke als Einflussfaktoren für den Algorithmus eingesetzt. Das Ziel einer Fledermaus ist es, sich an die Position der Beute zu begeben um diese zu fangen.



Generelles Prinzip Das Ziel des BA ist die Optimierung der Parameter für eine gegebene Funktion F_{opt}. F_{opt} hat irgendwo einen Nullpunkt, den es durch Einsetzen der korrekten Parameter zu finden gilt. Die *Sphere*-Funktion, definiert als

$$F_{sphere}(z) = \sum_{i=1}^{n} z_i^2$$

ist beispielsweise eine solche Funktion. Ihr Wertebereich ist definiert als $-\infty \leq z \leq \infty$. Der Wertebereich bildet zugleich den Suchraum ab, in dem sich der korrekte Wert (auch Beute genannt) für den Parameter z befindet. Es geht also darum ein z zu finden, für das gilt: $F_{sphere}(z) = 0$. Folgende Eigenschaften sind für den BA definiert:

- Ein Fledermaus-Schwarm besteht aus n Fledermäusen: $S = \{1, 2, ..., n\}$.

- Die Position x_i einer Fledermaus i stellt einen Lösungskandidaten, also einen Wert aus dem Funktions-spezifischen Suchraum dar. Für die Sphere-Funktion ist die Position x_i einer Fledermaus i also irgendein Wert im Bereich $[-\infty, \infty]$.
- Jede Fledermaus i bewegt sich im Laufe des Algorithmus durch den Suchraum. Sie verändert also ihre Position x_i mehrfach, die folglich verschiedene Werte aus dem Suchraum annimmt.
- Das Bewegungsmuster sieht vor, dass sich alle Fledermäuse des Schwarmes zeitgleich bewegen. Um dies zu realisieren werden sogenannte Zeitschritte t eingesetzt. Jede Fledermaus bewegt sich in jedem Zeitschritt.
- Dabei bestimmen die Fledermaus-Attribute: Frequenz f_i und Geschwindigkeit v_i ihre Flugreichweite und wirken sich damit auf ihre neue Position x_i aus.
- Die beste Position im Schwarm, also die Position der Fledermaus mit der geringsten Entfernung zur Beute, ist die sogenannte *global beste Position* x^*. Sie dient anderen (schlechter positionierten) Fledermäusen als Referenz, um sich zu verbessern.
- Die Pulsrate r_i, also die Rate mit der die Fledermaus i ihre Rufe ausstößt, verändert sich je nachdem wie erfolgreich die Fledermaus den Schwarm in Richtung der Beute führt. So ist r_i klein, wenn sich die Fledermaus weit entfernt von der Beute befindet und sie ist groß, wenn sie sich nah an der Beute befindet – noch näher als irgendeine Fledermaus aus dem Schwarm zuvor.
- Für die Lautstärke a_i gilt das gleiche Verhältnis, jedoch umgekehrt: ihre Rufe sind laut, wenn sie sich weit entfernt von der Beute befindet (wenn sie erfolglos ist) und sie ist klein wenn sich die Fledermaus der Beute annähert und dieser noch näher kommt als jede andere Fledermaus im Schwarm zuvor.
- Die Nähe zur Beute wird durch Einsetzen ihrer Position x_i in die zu optimierende Funktion bestimmt. Für die Sphere-Funktion misst sich die Nähe zur Beute also durch berechnen von $F_{sphere}(x_i)$. Ist das Ergebnis daraus groß, so befindet sich die Fledermaus i weit entfernt von der Beute. Ist es hingegen klein (nahe 0), so ist sie der Beute nah und ist es gleich 0, dann hat die Fledermaus die Beute gefangen und damit einen Wert für z gefunden, der zum Funktionsnullpunkt führt – dann konvergiert das Verfahren.

Abbildung 3 illustriert sechs Fledermäuse {1,2,3,4,5,6}, die sich im Zuge von zwei Zeitschritten *zufällig* im Suchraum der Sphere-Funktion bewegen. Alle Fledermäuse haben verschiedene Distanzen zum Nullpunkt, also zur Beute. Die Sphere-Funktion hat ihren Nullpunkt bei $z = 0$, das sei zum besseren Verständnis der Abbildung erwähnt. In einem realen Beispiel wäre z wie bereits erwähnt nicht bekannt, sondern soll gerade mit einem Optimierungsalgorithmus wie dem BA gefunden werden.

Im ersten Zeitschritt ($t = 1$) ist zu sehen, dass die Fledermäuse 1, 2 und 3 recht weit vom Nullpunkt entfernt sind. Das bedeutet $F_{sphere}(x_1), F_{sphere}(x_2)$ und $F_{sphere}(x_3)$ sind wesentlich größer als Null. Die Fledermäuse 4, 5 und 6 befinden sich näher an der Beute und so sind auch $F_{sphere}(x_4), F_{sphere}(x_4)$ und $F_{sphere}(x_6)$ näher an Null. $F_{sphere}(x_6)$ ist sogar ziemlich nah am Nullpunkt (x_6 geht gegen 0) und damit hat die Fledermaus 6 mit x_6 aktuell die global

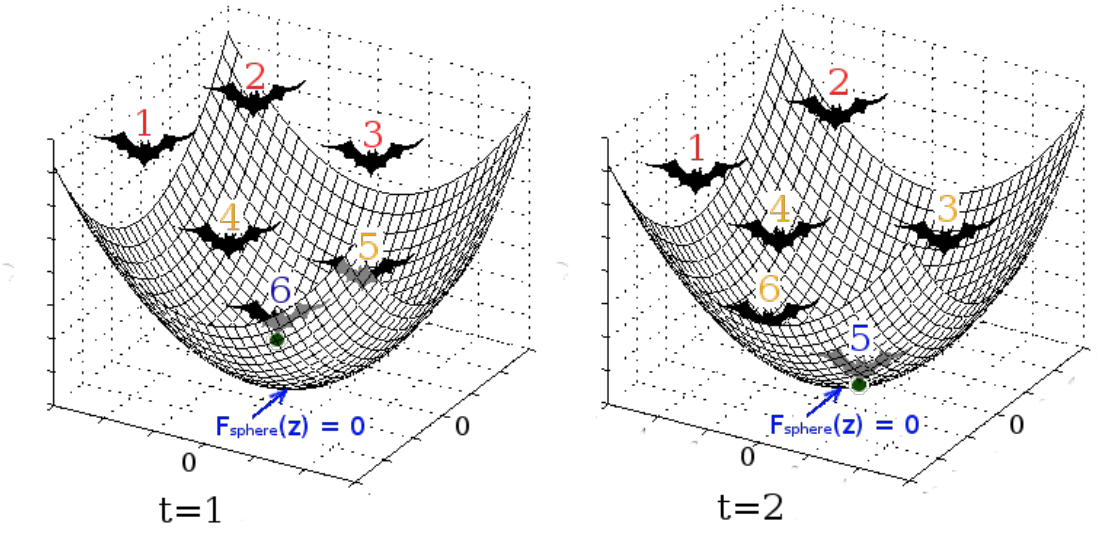


Abbildung 3: Beispielhaft positionierte Fledermäuse im Suchraum der Sphere-Funktion

beste Position: $x^* = x_6$ (in der Abbildung mit dem grünen Punkt markiert). Aus diesem Grund wird die Pulsrate von 6 erhöht und ihre Lautstärke verringert – sie ist schließlich eine erfolgreiche Fledermaus.

Im zweiten Zeitschritt ($t = 2$) hat sich der ganze Schwarm bewegt. Die Fledermäuse 1 und 2 sind noch immer recht weit von der Beute entfernt, während sich 3 ein wenig annähern konnte. Dennoch ist 3 nicht näher an der Beute als 6 im letzten Zeitschritt. Dies ist bei 5 anders, denn ihr ist es gelungen der Beute noch näher zu kommen als 6. Damit hat 5 mit x_5 nun die neue global beste Position: $x^* = x_5$. Genau wie bei 6 zuvor, erhöht auch 5 jetzt ihre Pulsrate und verringert ihre Lautstärke (für 6 ändert sich hier nichts).

Im Zuge weiterer Zeitschritte wird irgendeine Fledermaus aus dem Schwarm sehr wahrscheinlich die Beute erreichen, womit der Bat-Algorithmus die Lösung für die Sphere-Funktion gefunden hätte und konvergiert. Es kann aber auch sein, dass keine Fledermaus jemals die Beute erreicht, womit der BA die Lösung nicht finden würde. Dies ist möglich, weil die Bewegungen aller Fledermäuse des Schwarmes auf ihren Attribut-Werten basieren, die wiederum großteils vom Zufall abhängig sind.

Fledermaus-Attribute Die Fledermaus-Attribute sind als Stellschrauben des BA zu betrachten. Sie werden eingangs vom Benutzer für alle Fledermäuse des Schwarmes vorgegeben und verändern sich im Laufe des Optimierungsverfahrens für jede Fledermaus individuell und sie prägen den Algorithmus in seiner Funktionsweise. Es folgt eine Beschreibung aller Fledermaus-Attribute:

- Frequenz f_i^t: wird als erstes für jede Fledermaus i in jedem Zeitschritt t mit der Gleichung 3 neu berechnet. Sie ist ein Einflussfaktor für die Bestimmung der Geschwindigkeit v_i^t im selben Zeitschritt t.

$$f_i^t = f_{min} + (f_{max} - f_{min})\beta \tag{3}$$

Ihr Initialwert ist problemspezifisch und muss aus einem gültigen Wertebereich der zu optimierenden Funktion gewählt werden. Vernünftige Werte für f_{min} und f_{max} sind oftmals jedoch nur schwer abzuschätzen. Besonders dann wenn der Wertebereich riesig ist. Dies wäre beispielsweise bei der Sphere-Funktion so. Unter der Annahme, dass $z = 0$ für den Nullpunkt der Sphere-Funktion *nicht bekannt* wäre, müsste man f_{min} und f_{max} im Bereich $[-\infty, \infty]$ festlegen.

- Geschwindigkeit v_i^t: wird in jedem Zeitschritt verändert und ist ein Einflussfaktor für die Bestimmung der neuen Position x_i^t. Sie errechnet sich aus der Geschwindigkeit und der Position des vorherigen Zeitschrittes sowie der global besten Position x^* und der zuvor berechneten Frequenz.

$$v_i^t = v_i^{t-1} + (x_i^{t-1} - x^*) f_i^t \tag{4}$$

  Initial ist sie mit 0 anzugeben.

  Im späteren Abschnitt „Vereinfachungen“ wird der exakte Zusammenhang von Frequenz, Geschwindigkeit und Position beschrieben. Vorerst ist wichtig zu verstehen, dass in jedem Zeitschritt t für jede Fledermaus i zunächst die Frequenz f_i^t, danach die Geschwindigkeit v_i^t und zuletzt die neue Position x_i^t berechnet wird.

- Pulsrate r_i^t und Lautstärke a_i^t: dienen als Indikatoren für die Güte der Position x_i^t und finden Anwendung in der lokalen Suche (Pulsrate) oder in der Prüfung auf Akzeptanz der Position als neue global beste Position (Lautstärke). Damit sind beide zugleich Einflussfaktoren für das Verhältnis zwischen lokalen Werten $x_{1...n}$ und dem globalen Extremwert x^*. Der Initialwert der Lautstärke ist grundsätzlich frei wählbar, während r_i^0 im Wertebereich $[0,1]$ anzugeben ist. Die Gründe hierfür werden durch Betrachtung der Aktualisierungsgleichungen 5 und 6 ersichtlich.

$$r_i^{t+1} = r_i^0[1 - exp(-\gamma t)] \tag{5}$$

$$a_i^{t+1} = \alpha a_i^t \tag{6}$$

  Aktualisiert werden beide Attribute immer nur dann, wenn x_i^t als neue global beste Lösung x^* akzeptiert wurde (Algorithmus 3, Zeile 19). Die Aktualisierung von a_i^t besteht immer darin, ihren Wert um einen vom Benutzer vorgegebenen, konstanten Faktor α zu verringern. Zudem nimmt ein mit ihr verglichener Zufallswert während der zuvor erwähnten Prüfung (Algorithmus 3, Zeile 17) immer einen Wert zwischen 0 und a^0 an, sodass es keine Rolle spielt, wie groß der initiale Wert a^0 ist. Die Lautstärke beantwortet dadurch, dass sie sich verändert oder eben nicht, letztlich immer nur die Frage: „Hat sich die Fledermaus der Beute angenähert?“. Die Antwort darauf ist „ja“, wenn die Lautstärke a_i^{t+1} kleiner ist als a^t und „nein“ sonst. Das ist wichtig für die Behandlung der Fledermaus in zukünftigen Zeitschritten, denn je kleiner ihre Lautstärke wird, desto kleiner wird auch die Wahrscheinlichkeit dafür, dass die Position dieser Fledermaus als neue global beste Lösung akzeptiert wird, wodurch andere Fledermäuse länger die Chance haben eine neue global beste Lösung zu liefern. Mit diesem Vorgehen bleibt eine ausgewogene Verteilung der Fledermäuse im

Suchraum erhalten, weil gute Fledermäuse mit ihren Positionen schlechten Fledermäusen ein Beispiel sind. Schlechte Fledermäuse orientieren sich an den guten (im Zuge der lokalen Suche, die später beschrieben wird) und erkunden ebenfalls „das gute Gebiet nahe der Beute“. Letztlich gelangen auf diese Weise einige Fledermäuse in das gute Gebiet, um dieses zu erschließen und im besten Fall „die Beute zu fangen“.

Die Pulsrate r_i^t startet hingegen mit einem Wert nahe Null, der sich durch die Aktualisierungen mit der Gleichung 5 Schritt für Schritt r_i^0 annähert. Wie schnell diese Annäherung vonstatten geht, hängt von γ ab, denn t startet bei 0 und inkrementiert stets um 1 bis zum Erreichen des letzten, vorgegebenen Zeitschrittes.

Die beiden Parameter α und γ sind zunächst frei wählbar und ideale Werte hängen vom jeweiligen Problem ab. Sie müssen experimentell ermittelt werden. Yang schlägt in [69] zur Evaluierung des BA mithilfe der *Rosenbrock*-Funktion [4] folgende Werte vor: $\alpha = \gamma = 0,9$.

Eine Fledermaus die den Schwarm *niemals* näher an die Beute führt, wird ihre initiale Pulsrate nicht erhöhen. Daher steht eine geringe Pulsrate für eine eher schlechte Position. Dasselbe gilt im umgekehrten Verhältnis für die Lautstärke, so hat eine schlechte Fledermaus eine große Lautstärke. Mit den gegebenen Gleichungen haben die Werte beider Attribute folgende Verläufe:

$$a_i^t \to 0;\ r_i^t \to r_i^0;\ t \to \infty.$$

Umgang mit lokalen und globalen Extremwerten Abbildung 4 zeigt ein Beispiel für lokale und globale Extremwerte. Zu sehen ist der Kurvenverlauf einer Funktion $F_{Beispiel}(z)$

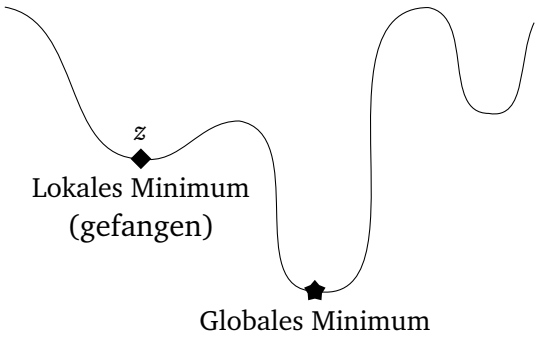


Abbildung 4: Beispiel für lokale und globale Extremwerte

mit eingezeichneten Extremwerten und gesucht wird auch hier wieder das Funktions*minimum* $F_{beispiel}(z) = 0$. Man stelle sich vor, dass als Lösungskandidat nur eine einzige Variable z existiert, die zum Optimieren der Funktion in einer Schleife aktualisiert wird. Jeder neue Wert der Variable basiert auf dem vorherigen und die Veränderung folgt einer bestimmtem Heuristik, die nicht allein auf dem Zufall basiert. Irgendwann nähert sich die Variable

[4] `https://www.sfu.ca/~ssurjano/rosen.html` (zuletzt zugegriffen am 21. August 2017)

einem Funktionsminimum an und der heuristische Ansatz gibt vor, nur solche Werte der Variable zu akzeptieren, die näher an das Funktionsminimum führen. Schlussendlich hat die Variable einen Wert angenommen, der zu einem Funktionsminimum führt – allerdings handelt es sich dabei nicht das globale Minimum. Stattdessen ist es ein zufällig gefundenes Minimum, von denen es je nach Funktion mehrere geben kann und das durch sukzessives Aktualisieren der Lösung gemäß der Heuristik erreicht wurde. Aufgrund des heuristischen Ansatzes „nicht größer zu werden“, kann die Variable „ihr Tal nicht verlassen“ und somit kann sie das globale Funktionsminimum niemals erreichen. Wenn sie in einem falschen Minimum „gefangen“ ist, dann kommt sie dort *allein* nicht mehr heraus.

Maßnahmen des Bat-Algorithmus zum Vermeiden frühzeitiger Konvergenzen mit falschen Extremwerten Eine besondere Stärke des BA ist sein ausgewogenes und zugleich steuerbares Verhältnis zwischen den *lokalen Werten* und dem *globalen Extremwert*. Dieses Verhältnis wird durch den *Zufallsflug* und die *lokale Suche* durchgesetzt. Beide werden nachfolgend beschrieben. Dabei bezeichnet ein lokaler Wert die Postion x_i einer Fledermaus i. Der globale Extremwert bezeichnet die beste Position x^* über alle Fledermäuse im Schwarm. Einfach formuliert: immer dann wenn eine Fledermaus „in einem Tal gefangen ist“ beziehungsweise sich nicht mehr verbessert und ihre Pulsrate deswegen nicht mehr erhöht, wird sie recht wahrscheinlich eine lokale Suche durchführen in der sie sich an einer besser positionierten Fledermaus orientiert. Als Beispiel sei folgende Situation gegeben: eine Fledermaus i hat eine Pulsrate von $r_i = 0,3$. Im Zuge von zwei Zeitschritten erreicht sie ein Tal aus dem sie nicht mehr heraus kommt. Das Tal ist nicht besonders weit vom Funktionsminimum entfernt, sodass ihre Lösung im Vorfeld ein Mal als global beste Lösung akzeptiert wurde. Dadurch stieg ihre Pulsrate beispielsweise auf $0,45$ an. Mit jedem nachfolgenden Zeitschritt bewegt sie sich weiter, jedoch immer nur minimal innerhalb ihres Tals und kann es niemals verlassen. Dadurch trägt sie nicht noch ein mal zur Verbesserung der global besten Lösung bei und erhöht ihre Pulsrate fortan nicht noch ein mal. Ihre Pulsrate wird in jedem Zeitschritt mit einem Zufallswert aus dem Wertebereich [0,1] verglichen. Wenn der Zufallswert in diesem Vergleich größer ist als ihre Pulsrate, dann führt sie eine lokale Suche durch, mit dem Ziel sich zu verbessern und „ihr Tal zu verlassen“. Die lokale Suche ist detailliert im entsprechenden, nachfolgenden Abschnitt beschrieben.

Letztendlich lässt sich die Maßnahme des BA gegen das verfrühte Konvergieren mit „falschen“ Werten folgendermaßen zusammenfassen: alle Fledermäuse eines Schwarmes kommunizieren miteinander, sodass jede Kenntnis von der global besten Position hat und sich selbst damit vergleichen kann. Wenn sich eine Fledermaus in einem falschen Minimum befindet, dann fliegt sie in die Nähe einer guten Fledermaus und erkundet das dortige Gebiet.

Die Position und ihre Aktualisierungsoperatoren Die Positionsaktualisierung beziehungsweise der Positionswechsel spielt eine wichtige Rolle beim BA, weil die Position letztlich einem Lösungskandidaten entspricht. Initial wird die Position für jede Fledermaus des Schwarms mit einem unterschiedlichen Wert aus dem Wertebereich der zu optimieren-

den Funktion belegt. Üblicherweise also mit einem Zufallswert aus $[f_{min}, f_{max}]$. Wie zuvor bereits erwähnt, hängt der Positionswechsel von mehreren Faktoren ab, die nachfolgend beschrieben sind. Dabei geht es zum einen um die konkreten Gleichungen, mit denen der Positionswert berechnet wird und zum anderen um die Frage: „Wann, warum und wie wird die neue Postion berechnet?“.

Zufallsflug Mit einem Zufallsflug verfolgt eine Fledermaus das Ziel, eine zufällig neu gewählte Position im Raum einzunehmen. Dazu kommt der Zufallsflug auf die kausale Abhängigkeit zwischen der Frequenz, der Geschwindigkeit und der Position zurück. Gemeint ist die Aktualisierung der Position x_i^t der Fledermaus i im aktuellen Zeitschritt t mit der Gleichung 7.

$$x_i^t = x_i^{t-1} + v_i^t \tag{7}$$

Doch zuvor muss im selben Zeitschritt die Frequenz mit der Gleichung 3 und dann die Geschwindigkeit mit der Gleichung 4 aktualisiert worden sein. Anschließend wird die zu optimierende Funktion, beispielsweise wieder die Sphere-Funktion F_{sphere}, mit der neuen Position x_i^t als Parameter ausgeführt und das Ergebnis in einer temporären Variable $eval$ gespeichert $eval = F_{sphere}(x_i^t)$. Abbildung 5 visualisiert den Zufallsflug für alle Fledermäuse des Schwarms im selben Zeitschritt.

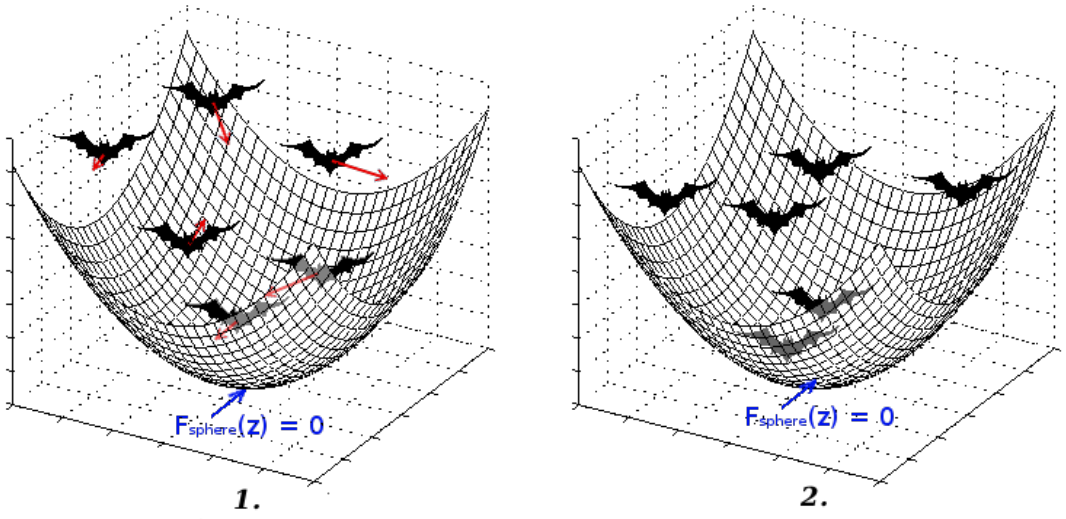


Abbildung 5: Beispiel eines Zufallsfluges.

Die linke Grafik (1.) zeigt den wesentlichen Schritt, in dem die Positionen mit der Gleichung 7 aktualisiert werden. Die rechte Grafik (2.) zeigt die darauf folgende Evaluation aller Positionen.

Lokale Suche Die Lokale Suche wird immer dann von einer Fledermaus i durchgeführt, wenn diese schlecht positioniert ist. Ob ihre Position schlecht ist, kann anhand ihrer aktuellen Pulsrate r_i^t abgeschätzt werden. Eine niedrige Pulsrate steht für eine schlechte

und eine hohe Pulsrate für eine gute Position. Wobei „schlecht“ und „gut“ relativ zum Wertebereich $[0,1]$ zu betrachten ist. Deutlich wird dies in Zeile 12 des Algorithmus 3. Die dort eingesetzte Zufallsvariable liegt im Bereich [0,1]. Eine Pulsrate kleiner als 0,5 führt wahrscheinlich zur Durchführung einer lokalen Suche und eine Pulsrate größer als 0,5 tut dies nicht.

Im Zuge der lokalen Suche nimmt sich die schlecht positionierte Fledermaus die im Schwarm am besten positionierte als Vorbild und kopiert dessen Positionswert – im übertragenen Sinne fliegt sie an die aktuell beste Position. Damit die beste Position jedoch nicht doppelt vorkommt, entfernt sie sich wieder ein kleines Stück in eine zufällige Richtung, was durch aufaddieren der durchschnittlichen Lautstärke a^t multipliziert mit einem Zufallswert ϵ realisiert wird. Die gesamte Berechnung dieser neuen Position ist in Gleichung 8 formuliert.

$$x_i^t = x^* + \epsilon a^t \tag{8}$$

Dabei ist x^* nach wie vor die aktuelle, global beste Position; ϵ ist ein Zufallswert aus dem Bereich [-1, 1] und a^t ist die durchschnittliche Lautstärke über alle Fledermäuse im aktuellen Zeitschritt t. Das Einbringen der Durchschnittslautstärke steht für die durchschnittliche Nähe zur Beute. Je kleiner a^t, desto mehr Fledermäuse haben sich erfolgreich der Beute angenähert und als umso kleiner wird die generelle Distanz zur Beute angenommen. Im Ergebnis steht mit x_i^t die Kopie der global besten Position mit einer geringen, zufälligen Abweichung als Ersatz für die bis dahin schlechte Position x_i^t. Zusammengefasst: schlecht positionierte Fledermäuse beziehungsweise solche, die ihre Position über einen längeren Zeitraum nicht verbessern konnten, führen eine lokale Suche durch, um in der Nähe der Schwarm-besten Fledermaus nach der Beute zu suchen. Abbildung 6 zeigt ein Beispiel für die lokale Suche einer Fledermaus. Wie stark oder schwach die generelle Tendenz zur Durchführung lokaler Suchen ist, steuert letztlich der Benutzer durch die initiale Angabe von r^0 und γ.

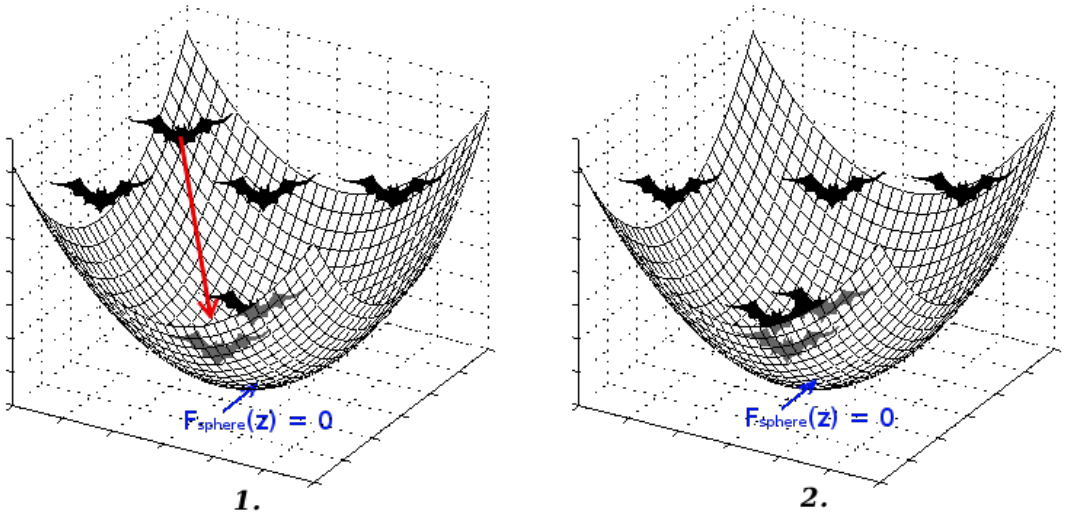


Abbildung 6: Beispiel einer lokalen Suche.

Der eigentliche Positionswechsel mit der Gleichung 8 geschieht dabei in der linken Grafik (1.). In der rechten Grafik (2.) ist die evaluierte Position zu sehen, wie schon zuvor beim Zufallsflug.

Die Durchführung von Zufallsflug und lokaler Suche im Algorithmus gestaltet sich also folgendermaßen: ein Zufallsflug wird in *jedem* Zeitschritt von *jeder* Fledermaus durchgeführt. Erfolglose Fledermäuse führen zumeist *zusätzlich* eine lokale Suche durch. Abbildung 7 zeigt dies als Ablaufdiagramm.

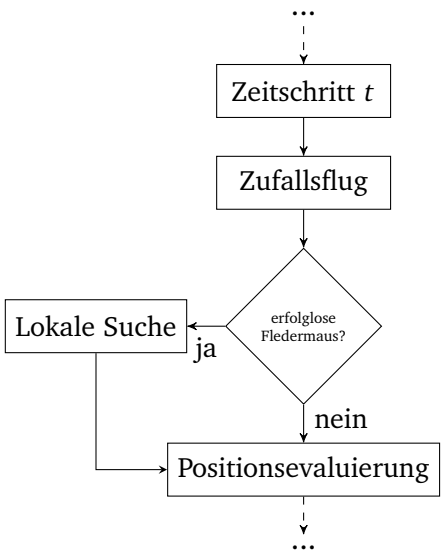


Abbildung 7: Durchführung des Zufallsfluges und der lokalen Suche.

Vereinfachungen Aufgrund der Tatsache, dass der BA nicht alle Sinne, Eigenschaften und Verhaltensmuster der realen Fledermäuse interpretiert und die die er interpretiert auch nicht in Gänze auf die Optimierung mathematischer Probleme abbilden kann, werden einige Vereinfachungen angenommen: Fledermäuse können Beute von Hindernissen unterscheiden und gezielt nur nach der Beute suchen. Hindernisse werden als nicht existent angenommen und alle Fledermäuse bewegen sich ohne zeitlichen Versatz durch den Raum. Beim Positionswechsel innerhalb eines Zeitschrittes befinden sie sich demnach sofort an der neuen Position und sie können auch nicht miteinander kollidieren. Eine weitere Vereinfachung betrifft die Wellenlänge (λ), doch diese und das Zusammenspiel zwischen ihr, der Frequenz und der Geschwindigkeit muss zuerst erklärt werden: so wie wir Menschen Lichtwellen wahrnehmen und damit visuell sehen können, orientieren sich Fledermäuse anhand auditiver Signale. Dazu stoßen sie Rufe aus und interpretieren die Echos. Die Rufe sind auf physikalischer Ebene wellenförmig (analog zu Lichtwellen) und die Wellenlänge ist das räumliche Analogon zur zeitlichen Periodendauer. Sie bezeichnet den Abstand zweier Messpunkte derselben Phase [5] einer Welle. Ein Fledermausruf entspricht einer

[5] `https://de.wikipedia.org/wiki/Wellenl%C3%A4nge` (zuletzt zugegriffen am 21. August 2017)

fortlaufenden Wellenform mit einer Phase und zwei zusammengehörige Messpunkte entsprechen zwei aufeinander folgenden Spitzenwerten einer Welle, wobei ein Spitzenwert positiv und der andere negativ ist (vereinfacht formuliert). Je kleiner die Wellenlänge ist, desto kleiner sind die Abstände zwischen den Messpunkten und umso hochauflösender kann die Fledermaus ihre Umwelt wahrnehmen. Damit geht einher, dass Beutetiere sehr klein sein können und dennoch von der Fledermaus erfasst werden. Die Wellenlänge ist definiert als

$$\lambda = \frac{v}{f} m = \frac{\frac{m}{s}}{\frac{1}{s}} m$$

und die Schallgeschwindigkeit in der Luft bemisst sich auf $v = 343 \frac{m}{s}$. Somit korrespondiert der Frequenzbereich der Fledermäuse [20kHz, 150kHz] mit einem Wellenlängenbereich von 17,1mm bis 2,3mm und es gilt die für den BA wichtige Aussage: *je kürzer die Wellenlänge, desto kürzer die zurückgelegte Distanz*. Abbildung 8 illustriert den Unterschied zweier Wellenlängen. Die einzelnen Wellen sind bei beiden Rufen jeweils als gestrichelte

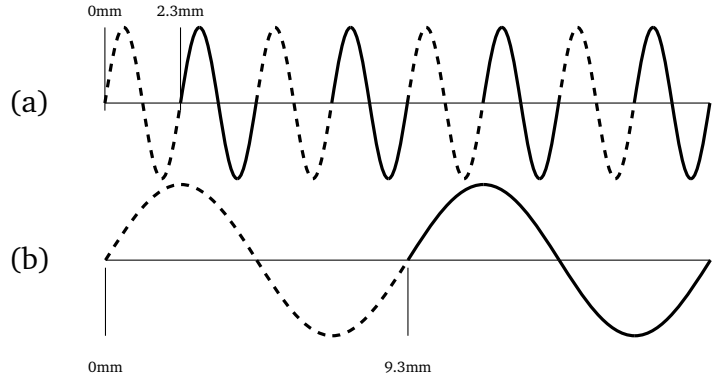


Abbildung 8: Vergleich zweier Fledermausrufe mit unterschiedlichen Wellenlängen ihrer Rufe.

und durchgezogene Kurven gekennzeichnet und somit unterscheidbar. Dabei repräsentiert (a) einen Ruf mit einer Wellenlänge von etwa 2,3mm, was mit einer Frequenz von etwa 150kHz korrespondiert. Der zweite Ruf in (b) hat eine Wellenlänge von etwa 9,3mm, was einer Frequenz von 37kHz entspricht. Jedes Beutetier, das mit seiner Größe in eine Welle passt, kann von der Fledermaus erkannt werden. Folglich kann der Ruf aus (a) Beutetiere schon ab einer Größe von etwa 2,3mm erfassen, während der Ruf aus (b) erst solche erfassen kann, die größer als etwa 9,3mm sind.

Für den BA in seiner originalen Form ist die Wellenlänge selbst jedoch nicht von Interesse. Stattdessen interessiert die Veränderung der Geschwindigkeit und damit einhergehend das Zusammenspiel von Wellenlänge, Frequenz und Geschwindigkeit und die Tatsache, dass $\lambda * f$ eine Konstante ergibt. Es ist folglich egal, ob die Wellenlänge oder die Frequenz justiert wird. Letztendlich muss mit einer der beiden Größen die Geschwindigkeit verändert werden, denn sie bestimmt über die zurückzulegende Distanz und damit über eine neue Position einer Fledermaus. Damit geht einher, dass große Veränderungen an der Geschwindigkeit mit großen Veränderungen an einer Lösung korrespondieren. Der originale BA aus [69] justiert die Geschwindigkeit mithilfe der Frequenz, deren Wertebereich zur

Algorithmus 3 Bat-Algorithmus

```
1:  procedure BA
2:      Definiere die Fitness-Funktion F_opt(x);
3:      for all Fledermaus i do
4:          Initialisiere Frequenz f_i^0, Geschwindigkeit v_i^0, Position x_i^0,
5:          Pulsrate r_i^0 und Lautstärke a_i^0;
6:      end for
7:      for all Zeitschritt t & Stopp-Kriterium nicht erfüllt do
8:          for all Fledermaus i do
9:              Führe einen Zufallsflug durch:
10:             aktualisiere hierzu f_i^t und v_i^t (Gleichungen 3 und 4);
11:             aktualisiere dann die Position x_i^t (Gleichung 7);
12:             if Zufall > r_i^t then                          ▷ Zufall ∈ [0,1]
13:                 Führe eine lokale Suche durch:
14:                 aktualisiere hierzu x_i^t erneut, jedoch mit der Gleichung 8;
15:             end if
16:             Evaluiere x_i^t mit der Fitness-Funktion: eval = F_opt(x_i^t);
17:             if (Zufall < a_i^t) & (eval < F_opt(x*)) then   ▷ Zufall ∈ [0, a^0]
18:                 Akzeptiere x_i^t als neue global beste Lösung: x* = x_i^t;
19:                 Erhöhe r_i^t und verringere a_i^t (Gleichungen 5 und 6);
20:             end if
21:         end for
22:     end for
23:     Liefere die beste Position als Lösung für F_opt aus;
24: end procedure
```

Vereinfachung auf $[0, f_{max}]$ begrenzt ist. Algorithmus 3 beschreibt den originalen BA in Form von Pseudocode.

- ***Zeile 1 bis 6 – Initialisierung:*** Definition der problemspezifischen, zu optimierenden Funktion (beispielsweise der Sphere-Funktion). Erzeugen und positionieren des Fledermaus-Schwarms. Die Attribute aller Fledermäuse werden mit Werten aus den entsprechenden Wertebereichen initialisiert.
- ***Zeile 7 bis 22 – Hauptalgorithmus:*** Die äußere Schleife läuft über alle vom Benutzer vorgegebenen Zeitschritte t, sodass der ganze Schwarm t-mal aktualisiert wird. Falls das Funktionsminimum noch vor der Abarbeitung des letzten Zeitschrittes gefunden wird, dann wird die Schleife abgebrochen (Stopp-Kriterium). Eine innere Schleife über alle Fledermäuse im Schwarm führt dazu, dass jede Fledermaus im Zuge eines Zeitschrittes einzeln aktualisiert werden kann.
- ***Zeile 9 und 11 – Zufallsflug:*** Die Fledermaus i startet einen *Zufallsflug*, durch Aktualisierung der Frequenz f_i^t und der Geschwindigkeit v_i^t. Darauf folgt eine Aktualisierung der Position x_i^t.
- ***Zeilen 12 bis 15 – Lokale Suche:*** Wenn eine Zufallsvariable aus dem Wertebereich

[0,1] größer ist als die Pulsrate r_i^t, dann vollzieht die Fledermaus i eine *lokale Suche*, um ihre Position zu verbessern.

- ***Zeile 16 – Evaluierung der neuen Position:*** Die Position x_i^t wird mit der zu optimierenden Funktion evaluiert. Das Ergebnis wird für den nachfolgenden Vergleich in Zeile 17 zwischengespeichert (Beispiel: $eval = F_{Sphere}(x_i^t)$).
- ***Zeilen 17 und 20 – Akzeptanzprüfung für x_i^t aus neue global beste Position:*** Wenn die Lautstärke a_i^t der Fledermaus i größer ist als ein Zufallswert aus dem Bereich $[0,\ a^0]$ und wenn das Ergebnis des der Evaluierung von x_i^t besser ist als das Ergebnis der Evaluierung von x^*, dann akzeptiere x_i^t als neue global beste Lösung: $x^* = x_i^t$ und aktualisiere anschließend die Pulsrate r_i und die Lautstärke a_i.
- ***Zeile 23:*** Nach Beendigung des Algorithmus wird die global beste Position x^* als Lösung für F_{opt} zurückgegeben.

Abbildung des BA auf andere Probleme Es ist möglich unter Einhaltung der besonderen Eigenschaften des BA, diesen auch auf andere Probleme abzubilden. Dafür müssen im wesentlichen die folgenden Schritte durchgeführt werden:

Zunächst muss die Form eines Lösungskandidaten klar sein, um diesen auf die Fledermaus-Position x abzubilden. Beim originalen BA ist dies ein (Parameter-)Wert aus dem problemspezifischen Wertebereich, also eine Zahl. Bei einem Graphen-basierten Problem, könnte dies beispielsweise ein Teilgraph sein. Als nächstes gilt es Aktualisierungsoperatoren für einen Lösungskandidaten einzubringen. Der originale BA verwendet hierfür die Gleichungen 7 und 8. Ein Graphen-basiertes Problem, um bei dem Beispiel zu bleiben, muss dementsprechend Graphen-spezifische Operatoren einsetzen, die unter anderem Teilgraphen berechnen können. Wichtig ist dabei, dass die Operatoren zum einen zufällige Positionsaktualisierungen durchführen können (Zufallsflug) und dass sie zum anderen auch die global beste Position einbeziehen können, um eine lokale Suche zu ermöglichen. Die Funktionsweise der Pulsrate als Einflussfaktor für das Verhältnis zwischen lokalen Werten und dem global besten Wert, sollte nicht abgewandelt werden, um diese Charakteristik des BA beizubehalten. Dies gilt in abgeschwächter Form auch für die Lautstärke. Sie kann zwar anders eingesetzt werden, sollte aber immer als Akzeptanzkriterium für die neue global beste Position dienen und auch weiterhin für erfolgreiche Fledermäuse mit voranschreitender Zeit kleiner werden. Zudem macht es Sinn die Durchschnittslautstärke auch weiterhin in die lokale Suche einzubringen, um nach wie vor eine Aussage über die gesamtheitliche Entfernung zur Beute treffen zu können.

2.4 Anwendungsfall: BA für das „Traveling-Salesman“-Problem

Der BA wurde bereits auf einige komplexe Probleme abgebildet. So auch auf das NP-vollständige Traveling-Salesman-Problem (TSP), das sich folgendermaßen beschreiben lässt: ein Handlungsreisender möchte verschiedene Städte bereisen. Jede Stadt genau ein Mal und am Ende möchte er zurück in die Stadt, von der aus er seine Reise angetreten hat. Insgesamt möchte er dabei die kostengünstigste Route auswählen und genau darin liegt

die Schwierigkeit dieses Problems. Unter der Annahme, dass ein Verbindungsweg in beide Richtungen dieselben Kosten verursacht, stehen dem Handlungsreisenden in jedem Schritt $\frac{(n-1)!}{2}$ mögliche Wege zur Verfügung, wobei n die Anzahl aller Städte ist, die er bereist. Dies ist die symmetrische Ausführung des TSP. Verursacht ein Verbindungsweg jedoch unterschiedliche Kosten, je nachdem in welcher Richtung er abgewandert wird, dann stehen dem Handlungsreisenden sogar $(n-1)!$ mögliche Wege zur Auswahl. Dies ist die asymmetrische Ausführung des TSP.

Generell ist das TSP ein kombinatorische Problem, während der BA in seiner oben beschriebenen Form kontinuierliche Optimierungsprobleme lösen kann. Um das TSP mit dem BA lösen zu können, müssen also ein paar Veränderungen am BA vorgenommen werden. Osaba, Yang, Diaz, Lopez-Garcia und Carballedo haben sich genau dies zur Aufgabe gemacht und eine Abbildung des BA auf das TSP entwickelt [52]. Diese Abbildung ist nachfolgend beschrieben.

Abbildung des BA auf das TSP Die Reise ist in Form eines Graphen $G = (V, E)$ abgebildet. Dabei ist der Graph G zumeist ein gerichteter, wenn es sich um das asymmetrische

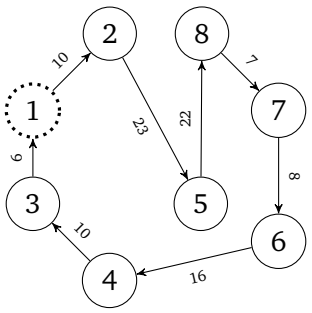


Abbildung 9: Darstellung einer Reise beim TSP

TSP handelt und ein ungerichteter beim symmetrischen TSP. V steht für die Menge aller Städte/Knoten und E für die Menge aller Verbindungswege/Kanten dazwischen. Jeder Verbindungsweg hat ein Attribut, das seine Reisekosten repräsentiert. Eine Route startet bei einem Knoten und führt über alle weiteren Knoten bis zum Startknoten zurück. Beim Start einer Reise (dem Eröffnungsverfahren) werden zunächst *zufällige Routen* festgelegt. Abbildung 9 illustriert eine beispielhafte Reise. Startpunkt ist der Knoten „1“ im gepunkteten Kreis. Von dort aus wird jede Stadt einmal besucht und am Ende kommt der Reisende wieder an seinem Startpunkt an. Dabei hat er die an den Verbindungswegen angegebenen Kosten zu tragen.

Pulsrate, Lautstärke und Frequenz Die Pulsrate r_i^t und die Lautstärke a_i^t behalten ihre zuvor beschriebenen Funktionen bei und werden mit den originalen Gleichungen 5 und 6 aktualisiert. Sie sind also Indikatoren für die Qualität einer Lösung und lenken zugleich das Verhältnis zwischen Erkundung und Ausbeutung. Die Durchschnittslautstärke a^t spielt in der lokalen Suche keine Rolle und die Frequenz f_i^t wird nicht benötigt und fällt weg.

Geschwindigkeit und Position Die Position x_i^t repräsentiert nach wie vor einen Lösungskandidaten des spezifischen Problems und muss daher einem Lösungskandidaten des TSP entsprechen. Somit ist x_i^t eine Route und damit ein Teilgraph aus G, der als $x_i^t = \{1,2,5,8,7,6,4,3\}$ kodiert wird. Jede Ziffer steht für eine Stadt und die geschriebene Reihenfolge ist die Reihenfolge in der die Städte besucht werden, wobei die letzte Ziffer dieser Folge die letzte Stadt vor der Start-Stadt repräsentiert. Die Geschwindigkeit v_i^t bekommt ebenfalls eine andere Rolle. Sie bezieht sich jetzt auf den *Hamming*-Abstand zwischen den Positionen der aktuellen Fledermaus x_i^t und der global besten Fledermaus x^*. Dazu ist sie als

$$v_i^t = Zufall[1, HammingDistanz(x_i^{t-1}, x^*)].$$

definiert und ersetzt die Gleichung 4. Damit ist v_i^t eine Zufallszahl zwischen 1 und dem Hamming-Abstand zwischen x_i^t und x^*. Der Hamming-Abstand ist als Anzahl unterschiedlicher Stellen zweier Routen definiert und je größer dieser Abstand ist, desto größer wird v_i^t. Als Beispiel seien die folgenden Routen x_1 und x_2 gegeben:

$$x_1 = \{1,2,3,4,5,6,7,8\}$$
$$x_2 = \{1,2,4,3,6,5,7,8\}$$

Der Hamming-Abstand zwischen ihnen ist gleich 4, denn die Routen unterscheiden sich an vier Stellen. Damit ist die Geschwindigkeit v_i^t hier eine Zufallszahl zwischen 1 und 4. Der Positionswechsel wird im originalen BA mit den beiden Gleichungen (Operatoren) 7 und 8 durchgeführt, die aufgrund der Abwandlung von x_i^t als Teilgraph nicht mehr anwendbar sind. Stattdessen werden hier für beide Gleichungen die beiden Graphen-Heuristiken *2-Opt* und *3-Opt* eingesetzt. Beide erzeugen aus einem gegebenen Graphen einen neuen, indem sie Teilgraphen darin verändern. Ein Teilgraph besteht entweder aus zwei (2-Opt) oder aus drei (3-Opt) Verbindungswegen, die heuristisch durch andere, günstigere Verbindungswege ersetzt werden. Ziel ist es, die Gesamtkosten der ganzen Route zu verringern. Welche der beiden Heuristiken im konkreten Fall ausgewählt wird, hängt von den folgenden Bedingungen ab:

- Wenn gilt: $v_i^t > \frac{n}{2}$, wobei n die Anzahl der Städte ist, dann scheint der aktuelle Lösungskandidat x_i^t *weit* von der besten Lösung entfernt zu sein. Folglich muss eine weite Distanz zurückgelegt werden, was einer größeren Veränderung der Verbindungswege entspricht, also wird *3-Opt* eingesetzt.
- Wenn gilt: $v_i^t < \frac{n}{2}$, dann wird eine kurze Distanz zur besten Lösung angenommen, die einer kleinen Veränderung der Verbindungswege entspricht und damit genügt der Einsatz von *2-Opt*.

Abbildung 10 illustriert die Anwendung der 3-Opt-Heuristik. Links ist die aktuelle Lösung x_i^t der Fledermaus i vor dem Positionswechsel abgebildet. In der Mitte sind drei zufällige Kanten ausgewählt, die durch andere Kanten mit geringeren Kosten ersetzt werden, was rechts zu sehen ist.

Dieser Ansatz unterscheidet sich in Bezug auf lokale Suche und Zufallsflug von der originalen Auslegung des BA: der BA aktualisiert zunächst immer die Frequenz, dann die

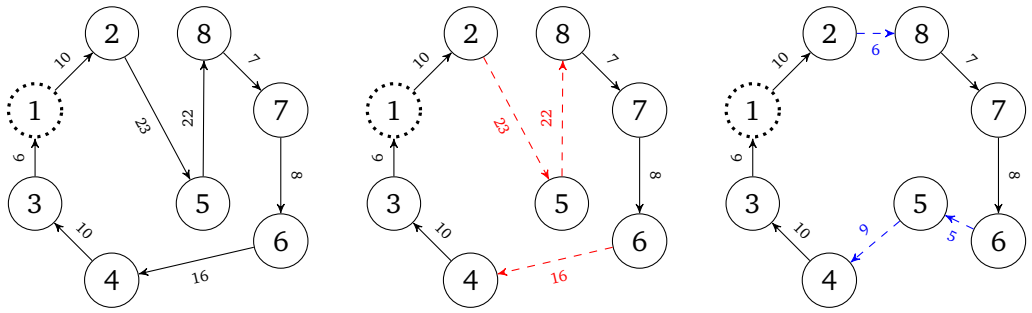


Abbildung 10: Erzeugung einer neuen Lösung für das TSP mithilfe der 3-Opt-Heuristik

Geschwindigkeit, dann die Position (Gleichung 7) und dann gegebenenfalls ein weiteres Mal die Position im Zuge einer lokalen Suche (Gleichung 8). Die Abbildung auf das TSP trennt diesen Ansatz nicht so scharf: es wird immer erst eine Geschwindigkeit berechnet (jedoch keine Frequenz, denn die fällt komplett raus). Darauf basierend wird immer die 2-Opt oder 3-Opt Heuristik gemäß der gezeigten Fälle durchgeführt, sowohl für die zufällige Positionsaktualisierung als auch während der lokalen Suche. Während der lokalen Suche wird nicht aus x_i^{t-1} sondern aus x^* eine neue Lösung berechnet. Das Resultat daraus wird dann für x_i^t eingesetzt und dieses Vorgehen ersetzt die Gleichung 8.

Fitness-Funktion Die zu optimierende Funktion $F_{TSP}(x)$ nimmt als Parameter eine Route entgegen und bemisst ihre Kosten durch Aufaddieren aller Teilkosten. Sie repräsentiert also die *Summe über die Kosten* aller Verbindungswege auf dieser Route. Es gilt folglich das Funktions*minimum* zu finden, also die kostengünstigste Route.

Pseudocode: BA für TSP Der Ablauf des ganzen Verfahrens ist in Form von Pseudocode in Algorithmus 4 dargestellt. Die Fitness-Funktion zur Kostenberechnung einer Route ist als F_{TSP} gekennzeichnet. Das Stopp-Kriterium in der äußeren Schleife über die Zeitschritte ist genau wie beim originalen BA dann erfüllt, wenn das Funktionsminimum gefunden wurde. In Zeile 20 wird für die lokale Suche zufällig eine der besten Lösungen ausgewählt, diese sind in absteigender Reihenfolge als $x_1^*, x_2^*, ..., x_n^*$ bezeichnet. Somit ist x_1^* die global beste Lösung, x_2^* die global zweitbeste Lösung und so weiter. Wie viele der besten Lösungen einbezogen werden, ist von der Schwarmgröße abhängig. Osaba, Yang, Diaz, Lopez-Garcia und Carballedo verwenden für ihre Experimente 50 Fledermäuse und beziehen immer die 10 besten ein ($x_{auswahl} = zufall(x_1^*, x_2^*, ..., x_{10}^*)$).

Algorithmus 4 BA für TSP

1: **procedure** BA
2: Definiere die Fitness-Funktion $F_{TSP}(x)$;
3: **for all** Fledermaus i **do**
4: Initialisiere Geschwindigkeit v_i, Position x_i,
5: Pulsrate r_i und Lautstärke a_i ;
6: **end for**
7: **for all** Zeitschritt t & Stopp-Kriterium nicht erfüllt **do**
8: **for all** Fledermaus i **do**
9: Führe einen Zufallsflug durch:
10: aktualisiere hierfür die Geschwindigkeit mit:
11: $v_i^t = Zufall[1, HammingDistanz(x_i^{t-1}, x^*)]$;
12: berechne dann x_i^t mit 2-Opt oder 3-Opt:
13: **if** $v_i < \frac{n}{2}$ **then**
14: $x_i^t \leftarrow$ 2-Opt(x_i^{t-1}, v_i^t);
15: **else**
16: $x_i^t \leftarrow$ 3-Opt(x_i^{t-1}, v_i^t);
17: **end if**
18: **if** Zufall $> r_i^t$ **then** ▷ Zufall $\in [0,1]$
19: Führe eine lokale Suche durch:
20: wähle hierzu eine der besten Lösungen aus:
21: $x_{auswahl} = zufall(x_1^*, x_2^*, ..., x_n^*)$ und
22: 1. berechne v_i^t erneut mit:
23: $v_i^t = Zufall[1, HammingDistanz(x_i^t, x_{auswahl})]$
24: 2. berechne die Position x_i^t erneut durch wiederholen der Schritte aus
25: Zeile 13 bis 17;
26: **end if**
27: **if** (Zufall $< a_i^t$) & ($F_{TSP}(x_i^t) < F_{TSP}(x^*)$) **then** ▷ Zufall $\in [0, a^0]$
28: Akzeptiere x_i^t als neue global beste Position: $x^* = x_i^t$;
29: Erhöhe r_i^t und verringere a_i^t (Gleichungen 5 und 6);
30: **end if**
31: **end for**
32: **end for**
33: Liefere die beste Position als Lösung aus;
34: **end procedure**

3 Stand der Technik

Die Assoziationsanalyse und damit das Lernen von Assoziationsregeln ist ein breit gefächertes Gebiet in der Informatik. Es geht darum Korrelationen zwischen Termen in einer Menge von Transaktionen herzustellen. Eine Transaktion umfasst beispielsweise alle Artikel, die eine Person in einem Supermarkt gekauft hat. Durch eine Analyse dieser Kundentransaktionen kann festgestellt werden, welche Artikel häufig zusammen gekauft werden. „Jemand der Milch und Eier kauft, kauft zumeist auch Brot“. Eine Assoziationsregel beschreibt genau diese Implikation: `Milch, Eier` $\rightarrow$ `Brot`. Terme entsprechen hier folglich Artikeln. Es sollte klar sein, dass eine solche Implikation nicht immer vollkommen korrekt ist, denn nicht jeder der Milch und Eier kauft, kauft auch Brot. Aus diesem Grund lässt sich für eine solche Regel mit einem gegebenen Datensatz messen, in wie vielen Fällen sie stimmt.

Beim Lernen von CEP-Regeln aus einem Ereignisdatensatz, mit dem Ziel die Ursachen für ein spezielles Ereignis zu finden, entspricht eine CEP-Regel einer Assoziationsregel. Das Prinzip dahinter ist die sogenannte *Klassifizierung*, die von Domingos in [15] beschrieben wird und nachfolgend auf CEP-Regeln im Kontext dieser Masterarbeit abgebildet ist: das Lernverfahren lernt Muster und Konstellationen aus einem Trainingsdatensatz, der diskrete und/oder kontinuierliche, domänenspezifische Daten beinhaltet. Als Ausgabe liefert das Lernverfahren einen sogenannten Klassifizierer: die CEP-Regel. Der Klassifizierer ordnet Terme in eine gegebene Klasse ein, was im Falle einer CEP-Regel bedeutet, dass das Ereignismuster dem vordefinierten Ereignis in der Regelkonklusion zugeordnet wird. Das vordefinierte Ereignis dessen Ursachen man sucht wird folglich auch als *Klasse* bezeichnet.

```
(IBMAktie AS i1 → IBMAktie AS i2 ∧
SelteneErden AS s1 → SelteneErden AS s2) ∧
(i1.wert > i2.wert ∧ s1.wert < s2.wert)[win:time:10min] ⇒ IBM-Aktie kaufen
```

Assoziationsregeln unterscheiden sich von CEP-Regeln jedoch in ihrer Komplexität, womit Verfahren zum Lernen von CEP-Regeln prinzipiell eine komplexere Logik realisieren müssen.

Nachfolgend werden einige Verfahren zum Lernen sowohl von Assoziationsregeln, als auch von CEP-Regeln vorgestellt. Die Verfahren für Assoziationsregeln basieren größtenteils auf Algorithmen der Schwarm-Intelligenz. Für CEP-Regeln gibt es auf dem Gebiet jedoch noch nicht viele Lösungsansätze, sodass die dortigen Verfahren spezielle Operatoren verwenden, mit denen die CEP-Konzepte getrennt voneinander analysiert und optimiert werden. Der letzte Abschnitt dieses Kapitels widmet sich Anwendungsbereichen des BA. Der BA nimmt aufgrund seiner Relevanz für diese Masterarbeit eine gesonderte Position ein. Der Abschnitt wird ebenfalls Ansätze zum Lernen von Assoziationsregeln beinhalten.

3.1 Computergestütztes Lernen von Assoziationsregeln

Die Fähigkeit zu lernen ist eine der wichtigsten Eigenschaften der menschlichen Intelligenz. Zugleich können Menschen große und komplexe Datenmengen nur schlecht bis gar nicht überschauen, geschweige denn Muster darin erkennen. Computer hingegen sind in

der Lage solche Datenmengen zu erfassen, sie besitzen aber nicht die Fähigkeit zu lernen. Wenn man es also schafft Computern eine Form des Lernens beizubringen, dann kann man sie in die Lage versetzen gezielt Ereignismuster und Konstellationen in großen Ereignisströmen zu finden.

Erste Ansätze des maschinellen Lernens gehen zurück in die 1950er Jahre zu Arthur Lee Samuel [59, 58] der auch den Begriff „maschinelles Lernen“ geprägt hat. Er hat die Frage gestellt: „Können Computer denken?“ und eröffnete damit ein neues Gebiet in der Informatik: *die künstliche Intelligenz*. Seitdem wurde in diesem Bereich viel geforscht. Einige Wissenschaftler haben neue Ansätze entwickelt, die Computern das Lernen beibringen. Um den Rahmen dieser Masterarbeit nicht zu sprengen, werden nachfolgend nur die Ansätze aufgeführt, die sich im Speziellem mit dem Lernen von Assoziationsregeln beschäftigen.

3.1.1 Lernen von Assoziationsregeln mit der Partikelschwarmoptimierung

Liu, Qin, Shi und Chen stellen in [43] ein Verfahren vor, das Assoziationsregeln mithilfe der Partikelschwarmoptimierung realisiert. Dazu wird jede Regel als Partikel abgebildet. Die Form eines Partikels ist beschrieben als eine Konkatenation von Attribut- und Operator-Feldern:

`[Attribut-Existenz][Operator][Attribut]`

„Attribut“ bezeichnet dabei den Wert eines Terms, beispielsweise `Milch` im Falle kategorischer Daten oder `2,29` im Falle kontinuierlicher Daten. „Operator“ meint den Operator, der ein Attribut mit seinem Wert verknüpft. Das Attribut-Existenz-Feld gibt an, ob ein Attribut i generell in der Regelprämisse auftaucht oder nicht. Ein Wert größer als 0 bedeutet dass es auftaucht und bei 0 taucht es nicht auf. Hat der Wert i im Operator-Feld einen Wert größer als 0, so steht dies im Falle kategorischer Daten für den „=“-Operator und bei kontinuierlichen Daten für den „≥“-Operator. Ist der Wert stattdessen gleich 0, dann repräsentiert dies den „!=“-Operator im Falle kategorischer Daten und den „<“-Operator im Falle kontinuierlicher Daten.

Die Regel-Klassifizierung ist das Ziel des Verfahrens und wird somit als die zu optimierende Funktion formuliert. Dabei wird die Güte der Regeln mit einer Kombination aus Recall, Precision und der Prägnanz gemessen. Recall und Precision werden im Abschnitt 5.2.5 beschrieben und die Prägnanz trifft eine Aussage über die Kürze einer Regel. Letztlich wird die Partikelschwarmoptimierung mit ihrem typischen Algorithmus dazu verwendet, die Funktion zu optimieren, also eine möglichst gute Assoziationsregel zu lernen. In der Tat erzeugt der Algorithmus nur eine Regel pro Durchlauf, die Erstellung eines Regelsatzes erfordert folglich das mehrfache Ausführen des Algorithmus.

Chen und Ludwig präsentieren in [11] ebenfalls einen Ansatz zum Lernen von Assoziationsregeln mit der Partikelschwarmoptimierung. Auch in ihrem Ansatz werden Partikel zur Lösungsrepräsentation eingesetzt, jedoch in einer anderen Kodierung als im Ansatz zuvor. Hier steht jedes Partikel für eine Menge von Regeln, der sogenannten „Rule base“ in Form einer Matrix. Die Geschwindigkeit für jedes Partikel wird in einer weiteren Matrix mit derselben Dimension verwaltet. Diese Geschwindigkeitsmatrix hat an jeder Stelle i den Wert 1,2 oder 3.

- 1: Partikel i bewegt sich von der aktuellen Position zur *lokal* besten Position.
- 2: Partikel i bewegt sich von der aktuellen Position zur *global* besten Position.
- 3: Partikel i bewegt sich von der aktuellen Position zu einer zufällig ausgewählten Position.

Bewegungen werden durch austauschen von Termen realisiert. Die Bewegung hin zur global besten Position bedeutet folglich, dass das Partikel i einen Term des global besten Partikels übernimmt.

Gupta und Ram stellen in [24] einen weiteren Ansatz zum Lernen von Assoziationsregeln mit der Partikelschwarmoptimierung vor. Ziel ist die Minimierung von beiden, Recall und Precision der erzeugten Regeln. Sie setzen in ihrem Optimierungsverfahren einen Vorverarbeitungsschritt ein, der alle Transaktionen in der Datenbank in einen Binär-Vektor überführt. Dafür werden alle Terme (beispielsweise alle Supermarkt-Artikel) durchnummeriert. Angenommen der Supermarkt verkauft lediglich vier verschiedene Produkte, so sähe die Abbildung folgendermaßen aus:

```
T1=[1,2,4] ⇒ B1=[1,1,0,1]
T2=[3]     ⇒ B1=[0,0,1,0]
T3=[1,3,4] ⇒ B1=[1,0,1,1]
```

Sinn und Zweck dieser Kodierung ist das Vermeiden von häufigem Einlesen des Datensatzes. Dadurch lässt sich die Qualität der Regeln wesentlich schneller bestimmen.

Hassani und Lee zerteilen den Eingabedatensatz in mehrere sogenannte „Chunks“ [26]. Dann setzen sie eine parallelisierte Variante der Partikelschwarmoptimierung ein (inkrementelle parallelisierte PSO (IPPSO)), um aus jedem „Chunk“ einzeln Assoziationsregeln zu extrahieren. Als Fitness-Funktion dient die sogenannte F_βScore, eine gewichtete Variante der „F-Score“, die im Abschnitt 5.2.5 genauer beschrieben wird. Ziel der Autoren Hassani und Lee ist ebenfalls die Optimierung von beidem, Recall und Precision.

In [5] stellen Almeida, Toracio und Pozo einen sogenannten „multi-objektiven“ Ansatz vor, um mit der Partikelschwarmoptimierung Assoziationsregeln zu lernen. Sie nennen ihren Ansatz *MOPSO: Multi-Objective PSO*. Auch hier wird eine Regel wieder durch ein Partikel repräsentiert, aber ihr Ansatz setzt nicht nur eine Fitness-Funktion ein. Die Veränderung eines Partikels, mit dem Ziel ihn hinsichtlich einer Fitness-Funktion A zu optimieren, verschlechtert ihn gegebenenfalls hinsichtlich einer anderen Fitness-Funktion B. Aus diesem Grund gibt es keine global beste Lösung. Dieser Zustand wird als sogenanntes „Pareto-Optimum“ [6] bezeichnet. Das Verfahren erzeugt Regeln gemäß des „Pareto Dominanz Konzept“, wodurch im Ergebnis ein Regelsatz steht, dessen Regeln den bestmöglichen Kompromiss zwischen allen Fitness-Funktionen eingehen. Dieser Regelsatz wird als *Pareto Optimal Front* bezeichnet.

3.1.2 Lernen von Assoziationsregeln mit dem Ameisen-Algorithmus

Der Ameisen-Algorithmus als eine der populärsten schwarmbasierten Metaheuristiken wurde bereits auf einige Anwendungsgebiete übertragen. So liegt es nahe, dass auch ein ent-

[6] `https://de.wikipedia.org/wiki/Pareto-Optimum` (zuletzt zugegriffen am 21. August 2017)

sprechender Ansatz zum Lernen von Assoziationsregeln existiert. Der wohl bekannteste ist *AntMiner*, vorgestellt in [53] von Parpinelli, Lopes und Freitas. AntMiner lässt sich vereinfacht folgendermaßen beschreiben:

1. Jeder abgewanderte Pfad einer Ameise stellt einen erzeugten Lösungskandidaten dar und dieser entspricht wiederum einer *Assoziationsregel*. Die Erzeugung eines Pfades ist von probabilistischer Natur und zugleich der erste und auch ein wiederkehrender Schritt des Algorithmus. Sie geschieht durch Auswählen von Termen für die Regelprämisse und einer anschließenden Klassifizierung, also der Zuordnung eines Terms in der Konklusion. Die Stärke einer Pfad-zugehörigen Pheromonspur repräsentiert seine Qualität, die sich beispielsweise mithilfe der sogenannten „Confusion Matrix“ [7] berechnen lassen kann.
2. Jeder Pfad besteht weiterhin aus kleineren Teilpfaden – den einzelnen *Termen*. Auch diese müssen hinsichtlich ihrer Qualität bewertet werden. Dazu gibt es eine Problemabhängige *Fitness-Funktion* (η), in die der Informationsgehalt jedes Terms eingeht und die so gestaltet sein muss, dass sie relevante Terme als solche erkennt und ihnen eine hohe Qualität zuordnet, während weniger relevante Terme eine niedrigere Qualität bekommen.
3. Wenn derselbe Pfad erneut abgewandert wird, dann verstärkt sich seine Pheromonspur durch weitere Markierungen, wodurch auch seine Qualität ansteigt. Pfade die lange Zeit nicht abgewandert wurden verlieren an Relevanz, was zu einer Absenkung des Pheromonwertes und der Qualität führt.
4. Solange sich keine akzeptable Lösung ergeben hat, wird das Verfahren mit neuinitialisierten Ameisen wiederholt.

AntMiner wurde von einigen Wissenschaftlern weiterentwickelt. Liu, Abbass und McKay ersetzen die Heuristik-Funktion, durch eine weniger genaue [42]. Sie argumentieren damit, dass es nicht notwendig sei den Informationsgehalt (Entropie) für jeden Term mit in die Heuristik-Funktion einfließen zu lassen. Stattdessen lässt sich der Informationsgehalt mit dem Pheromon kompensieren. Sie zeigen dass ihr Ansatz die gleiche Leistung erbringt und beweisen damit, dass der Ameisen-Algorithmus tolerant gegenüber vereinfachten beziehungsweise weniger genauen Heuristik-Funktionen ist.

Dieselben Autoren stellen in [41] auch eine neue Form der Pheromon-Aktualisierung vor. Grund hierfür sei das zu schnelle Ausschließen von Termen, also eine zu geringe Erkundung mit der originalen Pheromon-Aktualisierung.

Smaldon und Freitas präsentieren in [64] eine Abwandlung von AntMiner, mit der Regeln auf eine andere Weise erzeugt werden: entgegen dem originalen AntMiner, wird eine Regel beginnend mit dem Term in ihrer Konklusion erzeugt. Dieser Term wird festgelegt und dann versuchen die Ameisen passende Terme in der Regelprämisse zusammenzustellen. Damit entspricht dieser Ansatz seinem Vorgehen gemäß der Ursachenerkennung, wie sie auch für diese Masterarbeit vorgesehen ist.

[7] `https://en.wikipedia.org/wiki/Confusion_matrix` (zuletzt zugegriffen am 21. August 2017)

Manju und Kant stellen in [46] einen neuen Ansatz zum Lernen von Assoziationsregeln vor. Dieser ist ebenfalls von AntMiner inspiriert. Hier liegt der Fokus jedoch auf dem Finden von Assoziationsregeln *direkt* in einer großen Datendank beziehungsweise einem großen Datensatz, ohne wiederholtes Einlesen zum messen der Regelqualität. Dazu implementieren sie zwei Konzepte: zunächst wird die Datenbank in einen Graphen konvertiert. Mithilfe einer Bit-Vektor-Abbildung für jeden Term lässt sich ohne erneutes Einlesen der Datenbank die Qualität einer Regel berechnen. Im zweiten Schritt werden Regeln durch auswählen zufälliger Knoten aus dem Graphen erzeugt.

3.1.3 Lernen von Assoziationsregeln mit hybriden und anderen Natur-analogen Algorithmen

Einige Ansätze kombinieren zwei oder mehrere Natur-analoge Algorithmen miteinander, um sich die jeweiligen Stärken zunutze zu machen. Solche gemischten Algorithmen werden als *hybrid* bezeichnet. Einige dieser hybriden Algorithmen finden Anwendung zum Lernen von Assoziationsregeln und sind nachfolgend beschrieben.

In [32] erläutern Ji, Zhang, Liu und Zhong eine erweiterte Version des zuvor beschriebenen *AntMiner*. Die Erweiterung betrifft den sogenannten „Mutation Operator“, der die beste Regel aus jeder Iteration mithilfe eines genetischen Algorithmus mutiert. Sofern die Mutation eine Verbesserung an der Regel bewirkt hat, wird sie akzeptiert, ansonsten wird die Mutation rückgängig gemacht. Hierbei handelt es sich also um einen hybriden Ansatz aus dem Ameisen-Algorithmus und eines genetischen Algorithmus.

Gao, Hu und Tang stellen in [20] einen hybriden Algorithmus bestehend aus dem Ameisen-Algorithmus und der Partikelschwarmoptimierung vor. Ein initialer Durchlauf der Partikelschwarmoptimierung wird dazu verwendet, einen besonders guten initialen Pheromon-Wert des Ameisen-Algorithmus zu berechnen.

Freitas und weitere Wissenschaftler stellen in einer Reihe von Berichten [30, 29, 60] ebenfalls einen hybriden Ansatz zum Lernen von Assoziationsregeln mit dem Ameisen-Algorithmus und der Partikelschwarmoptimierung vor. Das Ganze geschieht vor einem biologischen Hintergrund: Proteine sollen in hierarchisch strukturierte Klassen eingeordnet werden. Es handelt sich hierbei um ein Klassifizierungsverfahren mittels Assoziationsregeln. Auch hier werden wieder die Stärken zweier Algorithmen kombiniert, mitunter weil die Autoren keinen bereits vorhandenen, geeigneten Algorithmus für ihre hierarchische Datenstruktur gefunden haben. „Der Ameisen-Algorithmus erwies sich als effektiv im Umgang mit Problemen basierend auf kategorischen Daten und die Partikelschwarmoptimierung hingegen erwies sich als effektiv im Umgang mit kontinuierlichen Daten.“ heißt es. Das hier entwickelte Verfahren erzeugt letztlich einen „Schwarm von Ameisen-Kolonien“, wobei jedes Partikel eine Regelprämisse zu einer gegebenen, festen Klasse repräsentiert. Weiterhin besitzt jedes Partikel für jeden im Datensatz vorkommenden Term eine Pheromon-Matrix und eine Information darüber, ob ein Term in seiner Regel vorkommt oder nicht.

Mangat bezieht diesen Ansatz auf einen medizinischen Bereich [45]. Ziel ist das Finden beispielsweise von wiederkehrenden Mustern in Genen oder Krankheiten, die Interaktion

verschiedener Proteine oder um Risikofaktoren für verschiedene Krankheiten zu erkennen.

Agarwal und Nanavati verknüpfen die Partikelschwarmoptimierung mit einem genetischen Algorithmus und verfolgen das Ziel multi-objektiv Assoziationsregeln zu erzeugen. Damit arbeitet dieser Ansatz genau wie der Ansatz von Almeida, Toracio und Pozo [5] mit mehreren Fitness-Funktionen womit es auch hierbei keine einzelne global beste Lösung geben kann – stattdessen die Pareto Optimal Front. Agarwal und Nanavati repräsentieren einen Lösungskandidaten, also eine Assoziationsregel als Partikel beziehungsweise als Chromosom in einer binären Kodierung. Das Verfahren teilt einen initial erzeugten Regelsatz in eine gute und eine schlechte Hälfte. Die gute wird mithilfe des genetischen Algorithmus optimiert, während die schlechte Hälfte mit der Partikelschwarmoptimierung verbessert wird.

In einem früheren Bericht [2] haben Agrawal, Mishra und Kushwah bereits einen ähnlichen Ansatz vorgestellt. Der dortige Algorithmus basiert im wesentlichen auf der Partikelschwarmoptimierung, die um einen „genetischen Operator“ erweitert wurde. Jedes Partikel repräsentiert eine Assoziationsregel und der genetische Operator verbessert zufällig ausgewählte Partikel.

Zhang hat den Fisch-Schwarm Algorithmus zum Lernen von Assoziationsregeln verwendet und stellt diesen Ansatz in [74] vor. Sehrawat, Manju und Rohil setzen dafür den Schmetterling-Algorithmus ein [61]. Auch der Bienen-Algorithmus wurde bereits zum Lernen von Assoziationsregeln verwendet. So beispielsweise von Djenouri, Drias, Habbas und Mosteghanemi in [14], von Sharma, Tiwari und Gupta in [62] und von Shukran, Chung, Yeh, Wahid und Zaidi in [63]. Farhana und Heber verwenden den sogenannten „Biogeographie-basierten“ Algorithmus und damit einen *evolutionären* Algorithmus zum Lernen von Assoziationsregeln und beschreiben ihren Ansatz in [18]. Der Biogeographie-basierte Algorithmus ist inspiriert von der räumlichen Verteilung von Spezies über die Zeit.

3.1.4 Lernen von Assoziationsregeln mit *nicht* Natur-analogen Algorithmen

Weiterhin wurden auch Algorithmen zum Lernen von Assoziationsregeln entwickelt, die sich nicht an Verfahren und Verhaltensweisen aus der Natur orientieren. Drei dieser Algorithmen werden nachfolgend erwähnt:

Zunächst sei *Apriori* genannt. Er wurde von Agrawal und Srikant entwickelt und ist in [3] beschrieben. Ein klassisches Einsatzgebiet des Apriori-Algorithmus sind die „Warenkorb-Transaktionen“ in einem Supermarkt, die zu Beginn des Kapitels bereits genannt wurden. Apriori misst die Qualität seiner erzeugten Assoziationsregeln mit Support und Konfidenz. Support beschreibt die Wahrscheinlichkeit dafür, dass eine Term-Menge in einer Transaktion vorkommt; die Wahrscheinlichkeit dafür, dass gewisse Produkte in einer Warenkorb-Transaktion vorkommen. Konfidenz ist die Wahrscheinlichkeit dafür, dass die Regelkonklusion unter der Bedingung in der Regelprämisse vorkommt. Wie oft also Brot gekauft wird, wenn Milch und Eier gekauft werden. Apriori arbeitet mit zwei Hauptschritten: zuerst werden Mengen häufiger Terme gefunden und danach Assoziationsregeln erzeugt.

FP-Growth ist ein weiterer Algorithmus zum Lernen von Assoziationsregeln, entwickelt von Han, Pei und Yin und vorgestellt in [25]. Er basiert auf einer speziellen Baumstruk-

tur, die durch zwei Suchläufe über den Eingabedatensatz erzeugt wird. Ein Vorteil von FP-Growth gegenüber Apriori ist, dass er keine häufigen Term-Mengen finden muss. Stattdessen werden nicht relevante Terme direkt aus der Baumstruktur entfernt.

Ein dritter populärer Algorithmus ist der sogenannte *C4.5*. Er wurde von Quinlan entwickelt und in [55] vorgestellt. Er verwendet einen Entscheidungsbaum, der aus einem Trainingsdatensatz erzeugt wird, um die darin enthaltenen Daten zu klassifizieren.

3.2 Computergestütztes Lernen von CEP-Regeln

Dieser Abschnitt macht einen Sprung in die Richtung des Verfahrens, das in dieser Masterarbeit entwickelt wird. Es geht nun nicht mehr um das Lernen von Assoziationsregeln, sondern speziell um das Lernen von CEP-Regeln, mit seinen spezifischen Schwierigkeiten. Das Lernen von CEP-Regeln erfordert nicht nur dass relevante Term-Mengen gefunden werden. Es müssen auch kausale Abhängigkeiten innerhalb der Regelprämisse, also zwischen den Termen gemäß der CEP-Konzepte aus Abschnitt 2.1 beachtet werden: „Jemand der zwei Tüten Milch in seinen Warenkorb legt und danach eine Schachtel Eier aus dem Regal nimmt, der wird auch zu Brot greifen.“ Des Weiteren arbeitet ein solches Lernverfahren nicht mehr mit Termen, sondern entsprechend mit Ereignissen. Der Eingabedatensatz beinhaltet Ereignisse in einer zeitlich fortlaufenden Sortierung – beispielsweise handelt es sich dabei um Mitschnitte der Daten eines Sensornetzwerkes.

Zum gegenwärtigen Zeitpunkt sind zwei populäre Lösungsansätze dafür bekannt: *iCEP* und *autoCEP*. Hinzu kam mit der Masterarbeit von Norman Offel ein dritter Ansatz: *CepGP*. Alle Ansätze werden nachfolgend vorgestellt.

iCEP wurde von Margara, Cugola und Tamburrelli entwickelt und in [47] vorgestellt. Es teilt den Eingabedatensatz in sogenannte *Traces* also Spuren ein. Es gibt positive Spuren, die das komplexe Ereignis (dort als `CE` bezeichnet) enthalten, dessen Ursachen es herauszufinden gilt und negative Spuren, die dies nicht enthalten:

```
A@10 B@11 C@14 CE@14 <- positiv
A@2 C@6 CE@6         <- positiv
B@1 C@4              <- negativ
```

Darauf basierend werden sieben wesentliche Module angewendet, die sich an den CEP-Konzepten orientieren und die das Herzstück des Ansatzes bilden. Mit ihnen lernt iCEP Muster und Konstellationen im Datensatz:

- *Event Learner:* betrachtet jede positive Spur und lernt daraus, welche Ereignisse zusammen auftreten.
- *Window Learner:* betrachtet jede positive Spur und lernt daraus die Fenstergrößen. Der Window Learner und der Event Learner stehen dabei in gegenseitiger Wechselwirkung, denn der Window Learner muss die relevanten Ereignisse kennen, um Fenster daran anzupassen und der Event Learner muss Fenster kennen, um zu entscheiden welche Ereignisse relevant sind und welche nicht.
- *Constraint Lerner:* findet Gleichheiten und Ungleichheiten der Attribute von Ereignissen desselben Typs in positiven Spuren.

- *Aggregates Learner:* lernt aggregierte Attribut-Werte in positiven Spuren.
- *Parameters Learner:* funktioniert im Wesentlichen wie der Constraint Learner. Bezieht sich aber nicht nur auf Ereignisse desselben Typs, sondern kombiniert Ereignisse verschiedener Typen in positiven Spuren miteinander.
- *Sequence Learner:* findet Ereignissequenzen in den Spuren. Betrachtet dabei zunächst jede positive Spur isoliert und bildet anschließend die Schnittmenge aus allen Spuren.
- *Negation Learner:* wird immer erst zuletzt angewendet. Der *Negation Learner* bezieht die Ergebnisse von allen anderen Learnern auf negative Spuren. Dadurch lassen sich nicht relevante Ereignisse explizit ausschließen, also negieren.

Nachdem alle Learner den Datensatz analysiert haben, stehen die kausalen Abhängigkeiten im Datensatz fest. Daraus lassen sich dann CEP-Regeln generieren, deren Bedingungen zur Konklusion passen.

Als zweiter Ansatz sei *autoCEP* genannt. Es wurde von Mousheimish, Taher und Zeitouni entwickelt und in [49] vorgestellt. Hierbei steht eine frühe Klassifizierung von zeitlich korrekten Sequenzen im Vordergrund. Dazu wird mit sogenannten *Shapelets* gearbeitet. Das sind Muster mit einer minimalen Länge, die die Ereignistypen im Datensatz von einander abgrenzen. Shapelets wurden von Ye und Keogh in [71] vorgestellt.

An dritter Stelle sei CepGP erwähnt, das von Norman Offel im Zuge seiner Masterarbeit entwickelt und in [51] vorgestellt wurde. CepGP lernt CEP-Regeln aus dem Eingabedatensatz mithilfe der *Genetischen Programmierung*. Dabei werden Lösungen, also CEP-Regeln als Bäume bestehend aus Operatoren und Operanden dargestellt. Bei den Operatoren handelt es folglich um die, die in CEP-Regeln eingesetzt werden können (Abschnitt 2.1.2). Operanden sind Ereignistypen mit Attributen, die im Datensatz vorkommen. Letztlich werden die Regelbäume von den evolutionären Operatoren wie dem *Crossover* und der *Mutation* verändert, um neue Lösungen zu erzeugen. Norman Offel hat damit gezeigt, dass das Lernen von CEP-Regeln auch mit Natur-analogen Optimierungsverfahren möglich ist.

3.3 Bisherige Anwendungsgebiete des Bat-Algorithmus

Obwohl der BA verglichen mit anderen SI-Algorithmen noch relativ neu ist, wurde er bereits erfolgreich auf einige Anwendungsgebiete übertragen. Yang und He führen in [70] eine Übersicht über diverse Probleme auf, die erfolgreich mit dem BA gelöst werden konnten. Dabei handelt es sich um kontinuierliche und kombinatorische Probleme; um inverse Probleme wie Parameterbestimmungen sowie um *Klassifizierungen*, *Clustering* und *Data Mining*. Einige davon sind nachfolgend aufgeführt. Zum Ende dieses Abschnitts werden noch Ansätze für das Lernen von Assoziationsregeln mit dem BA präsentiert.

Kontinuierliche Probleme Lin, Chou, Yang und Tsai stellen in [40] eine Version des BA vor, die chaotische Zufallsvariablen mit einem Lévy-Flug zum Erzeugen neuer Lösungen einsetzt. Die Autoren verfolgen damit das Ziel eines verbesserten Konvergenzverhaltens,

indem die Wahrscheinlichkeit für ein verfrühtes Konvergieren in einem falschen globalen Extremwert minimiert wird. Soliman und Elhamd verwenden diesen Ansatz in [65] für einen Klassifizierungsalgorithmus, mit dem sich frühe Anzeichen von Diabetes erkennen lassen.

Bora, S. Coelho und Lebensztajn wenden den BA an, um das „Brushless DC Wheel Motor Problem“ zu lösen. Ein Problem, das sowohl mono-objektiv als auch multi-objektiv ausgelegt sein kann. Es geht darum einen bürstenlosen Gleichstrom-Motor zu entwerfen und dessen Effizienz zu maximieren. Dafür gilt es fünf Parameter zu optimieren, was in [9] von den obigen Autoren mithilfe des BA realisiert wird.

Wang, Chang und Zhang führen ein Modell ein, in dem mehrere Fledermaus-Schwärme parallel eingesetzt werden. Ziel der Autoren ist ein verbessertes Verhältnis zwischen Erkundung und Ausbeutung [68].

Yilmaz, Ugur Kucuksille und Cengiz erweitern den BA um weitere Dimensionen und verändern im gleichen Schritt die Aktualisierungsgleichungen 5 und 6 für die Pulsrate und für die Lautstärke. Mehrere Dimensionen bedeutet, dass jede Fledermaus zeitgleich mehrere Position und damit Lösungen besitzt [72]

Li und Zhou führen eine neue Kodierung mit komplexen Zahlen ein, bei der der reale und der imaginäre Teil einer Zahl getrennt voneinander aktualisiert wird. Die Autoren verfolgen dabei das Ziel die Fledermäuse besser auf den Suchraum aufzuteilen. Sie wollen also die sogenannte *Diversity*, die Unterschiedlichkeit der Lösungen erhöhen.

Kombinatorische Probleme Ramesh, Mohan und Reddy verwenden den BA in [56] um ein Problem im Bereich eines Stromversorgungssystems zu lösen. Sie versuchen die optimale Leistung aus einem System zu holen und dabei den Schaden für die Umwelt so gering wie möglich zu halten. Dabei handelt es sich um ein multi-objektives Problem, mit einigen Bedingungen.

Musikapun und Pongcharoen haben mit dem BA ein Planungswerkzeug entwickelt (sie nennen es *BAST*), mit dem sich Abläufe für verschiedene Maschinen im produktiven Einsatz planen lassen. Die Autoren berichten, dass die Leistungsstärke des BA mit den von Ihnen angegebenen Parametern um 8,4% verbessert werden kann.

Bildverarbeitungsprobleme Akhtar, Ahmad und Abdel-Rahman optimieren Bildverarbeitungsprozesse mit dem BA. Sie stellen in [4] ein Verfahren vor, mit dem sich menschliche Bewegungen erkennen und verfolgen lassen.

Zhang und Wang erweitern den BA um einen Mutationsoperator und verwenden ihre Version des BA für das sogenannte *Image matching*. Image matching umfasst das Erkennen von Ähnlichkeiten zwischen zwei oder mehreren verschiedenen Bildern. In [73] stellen sie ihr Verfahren vor.

Klassifizierungsprobleme und das Lernen von Assoziationsregeln Khan und Sahai stellen in [37] einen Ansatz zum *clustern* von Arbeitsplätzen vor. Dazu setzen sie eine angepasste Version des BA und *Fuzzy Logic* ein. Überdies vergleichen sie den BA in ihrem Kontext mit der Partikelschwarmoptimierung, einem genetischen Algorithmus sowie weiteren Algorithmen.

Marichelvam und Prabaharam verwenden den BA in [48] um verschiedene, voneinander abhängige Vorgänge hinsichtlich ihrer durchschnittlichen Ablaufzeiten zu optimieren. Ihr Ziel ist weiterhin die Minimierung von Produktionszeiten.

Faritha Banu und Chandrasekar setzen den BA zum Entfernen von Datenduplikaten ein. Ihr Ziel ist die Kompression von Daten.

Damodaram und Valarmathi präsentieren in [13] einen Ansatz, mit dem sich sogenannte *Phishing*-Webseiten als solche klassifizieren lassen. Dazu verwenden sie unter anderem Assoziationsregeln.

Song, Ding, Chen, Li, Cao und Pu präsentieren in [66] einen Ansatz zum Lernen von Assoziationsregeln. Die Autoren beschreiben dies als ein multi-objektives Optimierungsproblem basierend auf Pareto. Die Positionen, also Assoziationsregeln sind als binäre Zeichenketten kodiert und die Aktualisierungsoperationen entsprechend daran angepasst. Als Fitness-Funktion wird die sogenannte *Compreheness*, eine Mischung aus Support und Konfidenz eingesetzt. Die Schwierigkeit besteht darin Support und Konfidenz in Balance zu halten, was das „multi-objektive“ Problem darstellt.

Heraguemi, Kamel und Drias verwenden den BA zum Lernen von Assoziationsregeln und ihren Ansatz stellen sie in [28, 27] vor. Aufgrund der Tatsache, dass dieser Ansatz mit einigen seiner Anpassungsweisen für das in dieser Masterarbeit entwickelte Verfahren relevant ist, wird es etwas detaillierter beschrieben als alle anderen zuvor.

Eine Assoziationsregel wird folgendermaßen kodiert:

$$[j, I_1, ..., I_j, I_{j+1}, ..., , I_k].$$

Dabei stellen I_1 bis I_j die Terme aus dem Datensatz dar, die in der Regelprämisse vorkommen; I_{j+1} bis I_n sind die Terme in der Regelkonklusion und j bezeichnet den Übergang, also die Position die die Prämisse von der Konklusion trennt. Insgesamt beinhaltet der Datensatz k Terme. Um aus einer derart kodierten Assoziationsregel eine neue zu erzeugen, werden also einfach beliebige Terme durch andere ersetzt: sowohl in der Prämisse als auch in der Konklusion.

Überdies passen sie die Gleichungen 3 und 4 folgendermaßen an:

$$f_i^t = 1 + (f_{max}\beta)$$
$$v_i^t = f_{max} - f_i^t - v_i^{t-1}.$$

Dabei ist β eine Zufallsvariable aus dem Wertebereich [0,1], womit die Frequenz folglich irgendeinen Wert zwischen 1 und $1+f_{max}$ annimmt und f_{max} ist die Anzahl aller Terme im Eingangsdatensatz. Die Geschwindigkeit v ist immer kleiner als die Frequenz und das hat

folgenden Grund: die Anzahl der zu tauschenden Terme in der Position, also der Assoziationsregel der Fledermaus bestimmt sich aus der Differenz zwischen ihrer Geschwindigkeit und ihrer Frequenz und auch die Lautstärke geht mit ein. Die Geschwindigkeit gibt den Index der ersten Änderung an und als Pseudocode sieht der Positionswechsel folgendermaßen aus:

while $v < f$ **do**
  if Zufall $> a$ **then**
    Term an Stelle $v \leftarrow$ Term an Stelle $v + 1$;
  else
    Term an Stelle $v \leftarrow$ Term an Stelle $v - 1$;
  end if
  Wenn v gleich 0 oder größer als die Anzahl der Terme im Datensatz ist,
  dann setzte den Term an der Stelle v gleich 0;
  Wenn Terme doppelt in der Regel vorkommen,
  dann behalte einen Term zufällig und entferne die anderen;
  Erhöhe v um 1;
end while

Ein Beispiel dazu:
Der Eingabedatensatz enthält vier Terme: `{Milch, Eier, Brot, Kaffee}`, die in der angegebenen Reihenfolge von eins bis vier durchnummeriert sind. Die Frequenz f hat den Wert 4, die Geschwindigkeit v den Wert 2. Damit ist die Differenz Δ_v^f gleich 2. Die Zufallsvariable im Vergleich mit der Lautstärke a hat den Wert 6 und die Lautstärke selbst hat den Wert 3. Die Position und damit die Assoziationsregel der Fledermaus ist: `[2, Milch, Eier, Brot]`. Die Regelprämisse enthält `Milch` und `Eier` und die Konklusion enthält `Brot`. Nun wird die Regel gemäß des Pseudocodes verändert, was zu der folgenden neuen Regel führt: `[2, Milch, Brot, Kaffee]`. In der ersten Schleifen-Iteration wird die Stelle 2 durch 3 ersetzt (Eier durch Brot) und in der zweiten die Stelle 3 durch 4 (Brot durch Kaffee).

Die Pulsrate und die Lautstärke behalten ihre üblichen Funktionen bei und berechnen sich nach wie vor mit den originalen Gleichungen 5 und 6. Die lokale Suche entspricht dem Austausch genau *eines* zufällig ausgewählten Terms in der global besten Assoziationsregel. Als Fitness-Funktion wird der Support und die Konfidenz eingesetzt.

4 Genereller Ansatz – Lernen von CEP-Regeln mit dem Bat-Algorithmus

Wie bereits in Abschnitt 2.3 beschrieben, ist der originale BA ein Optimierungsverfahren für kontinuierliche Funktionen. Doch das Lernen von CEP-Regeln ist ein kombinatorisches Optimierungsproblem und dies erfordert einige Anpassungen am BA. Dazu wird nachfolgend zunächst das Einsatzgebiet des BA zum Lernen von CEP-Regeln in einem Szenario beschrieben. Darauf folgen konkrete Abbildungsansätze der BA-typischen Attribute auf den Umgang mit CEP-Regeln.

4.1 Höheres Wissen aus aufgezeichneten Ereignisdaten gewinnen

Zur Erinnerung: mithilfe von CEP-Regeln können spezielle Ereigniskonstellationen in Datenströmen erkannt werden. Zeichnet sich in einem Datenstrom ein Muster ab, das dem Muster in der Regel-Bedingung entspricht, dann führt die Regel als Reaktion eine spezifische Aktion durch. Sowohl die Aktion als auch die CEP-Regel selbst mussten initial von einem Fachexperten definiert worden sein. Dieser kennt die Ereignistypen im Datenstrom und weiß auf welche Konstellation er wie reagieren möchte.

Dazu ein kurzes Beispiel (angelehnt an den Abschnitt 2.1.2):
Gegeben sei eine Industriekaffeemaschine, in der einige Sensoren verbaut sind. Alle Sensoren liefern immerzu unzählige Daten, woraus ein kontinuierlicher Datenstrom entsteht. Nach einer Weile des normalen Betriebs fällt die Kaffeemaschine schlagartig aus, worauf hin der Experte direkt an der Maschine nach den Ursachen für den Ausfall sucht und diese nach einiger Zeit findet. Das Problem entstand durch Überdruck an einem Ventil, wodurch ein Leck an einer Wasserleitung hervorgerufen wurde. Dadurch entwich Wasser, das einen Schaltkreis beschädigte. All diese Geschehnisse finden sich anschließend auch im Sensor-Datenstrom wieder. Die Maschine wird repariert und der Experte definiert eine CEP-Regel zur Überwachung des Ventildrucks. Mithilfe der CEP-Regel wird zukünftig der Sensor-Datenstrom dieser Maschine analysiert und immer dann, wenn der Druck in eine kritische Höhe ansteigt, bedingt die Regel als Reaktion eine präventive Abschaltung der Maschine. Anschließend muss der Defekt von einem Experten behoben und die Maschine wieder manuell eingeschaltet werden.

In diesem Beispiel gelang es dem Experten die Ursachen für den Ausfall der Maschine zu finden und eine präventive Maßnahme einzuleiten. Problematisch wird es jedoch, wenn der untersuchte Gegenstand keine Kaffeemaschine mit einer gerade noch überschaubaren Menge an Sensoren und Bauteilen ist, sondern ein komplexes System bestehend aus hunderten Geräten die sich gegenseitig beeinflussen. Alle Geräte haben verschiedene Sensoren, sodass sich ein massiver Datenstrom ergibt. Ursachenquellen können vom Experten in einem solchen System kaum mehr gefunden werden und so kann er auch keine CEP-Regeln als präventive Maßnahmen definieren. Ein solches System muss computergestützt auf Ursachenquellen für Probleme untersucht werden und dabei kann ein Lernverfahren helfen. Ein Lernverfahren wie das in dieser Masterarbeit entwickelte. Es erkennt Muster

und Konstellationen in Datenströmen und stellt diese als Ursachen für ein besonderes Ereignis heraus. Dieses besondere Ereignis ist selbst im Datenstrom vorhanden und wird in den nachfolgenden Abschnitten als Z bezeichnet. Bezogen auf das Beispiel zuvor entspricht Z dem Ausfall der Kaffeemaschine, dessen Ursachen in irgendeiner Kombination der Sensordaten zu finden sind. Der Fachexperte muss das System nun nicht mehr komplett überschauen. Stattdessen leitet er lediglich einen Ausschnitt des Datenstroms unter Angabe von Z an das Lernverfahren weiter. Dann startet er das Lernverfahren und wartet auf das Ergebnis: die Ursachenquellen für Z in Form einer CEP-Regel. Abbildung 11 illustriert die Anwendung des Lernverfahrens ebenfalls am Beispiel der Kaffeemaschine. Die Datenquellen liefern Informationen über aktuelle Geschehnisse und sind in der realen

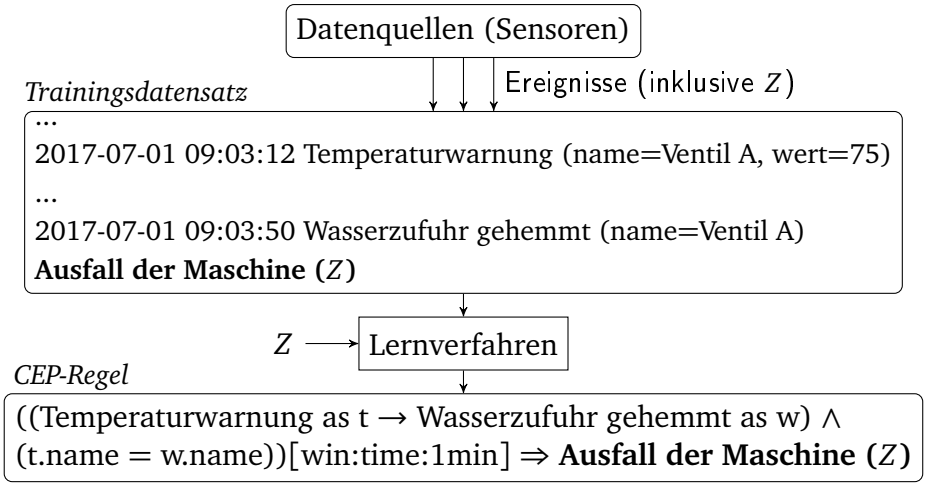


Abbildung 11: Abstrakter Ablauf des Lernverfahrens für CEP-Regeln

Welt unter anderem Sensoren, die ihre Daten an ein System weiterreichen. Das System sammelt eine gewisse Menge dieser Daten, bereitet sie auf, transformiert sie in ein einheitliches Format und legt sie anschließend als Trainingsdatensatz in einer Datei ab. So repräsentiert jeder Eintrag (jede Zeile) in der Datei ein Sensor-Ereignis das die folgenden Eigenschaften besitzt:

- einen Bezeichner
- einen Zeitstempel
- einen Typ
- (optional) Attribute

Die Datei ist folglich ein zeitlich fortlaufender Mitschnitt der Sensordaten, ein abgeschlossener Datensatz, der anschließend zusammen mit der Angabe von Z an das Lernverfahren weitergereicht wird. Die gesonderte Angabe von Z entspricht dessen individuellem Bezeichner und ist eine Information des Fachexperten an das Lernverfahren. Dieses analysiert den Datensatz gezielt um Ursachen für Z zu finden. Dazu setzt es CEP-spezifische Muster aus den Ereignissen zusammen und prüft den Zusammenhang dieser Muster mit

Z. Währenddessen lernt es beispielsweise welche Ereignisse häufig mit Z in Verbindung stehen und welche nicht. Als Ausgabe produziert es CEP-Regeln, deren Bedingungen Ereignismuster enthalten, die als Ursachen für Z festgestellt wurden. In der Konklusion, also im Aktionsteil einer solchen Regel steht immer Z. Der Fachexperte profitiert insofern von einem solchen Lernverfahren, als er die erzeugten Regeln nicht selbst hätte definieren können und jetzt als präventive Maßnahme für den zukünftigen Betrieb des Systems einsetzen kann.

Abstrakt betrachtet kann ein solches Vorgehen als *Information Retrieval* (deutsch: Informationsrückgewinnung) betrachtet werden. Dieses Themengebiet widmet sich der Analyse komplexer Inhalte beispielsweise in Texten oder Bildern und verfolgt das Ziel darin bestehende Muster zu finden, aus denen sich höheres Wissen ableiten lässt [57]. Der Datensatz als Mitschnitt des Sensor-Datenstroms ist ebenso ein „komplexer Inhalt“ und kann als eine Art Trainingsdatensatz angesehen werden, der den realen Datenstrom in einer abgeschlossenen Form repräsentiert. Die erzeugten Regeln repräsentieren das höhere Wissen, von dem der Fachexperte zukünftig profitiert.

Die wesentliche Frage ist nun: „Wie kann ein solches Lernverfahren mit dem BA realisiert werden?“ Als einer der ersten Arbeitsschritte muss der Trainingsdatensatz in seine einzelnen Komponenten zerlegt werden. Jedes vorkommende Ereignis mit seinen spezifischen Attributen muss bekannt sein, damit es überhaupt möglich ist Datensatz-spezifische CEP-Regeln zu erzeugen. Überdies müssen auch alle CEP-Operatoren (aus Abschnitt 2.1.2) auf die Ereignisse angewendet werden können, um Ereignismuster und Kontextbedingungen zu formulieren. Das komplexe Ereignis Z muss gesondert angegeben werden können, damit der BA zielgerichtet nach dazu passenden Ereignismustern suchen kann. Generell gilt es dabei unbedingt den spezifischen Ablauf des BA beizubehalten. Das bedeutet, dass charakteristische Operationen wie der Zufallsflug und die lokale Suche auch weiterhin vorhanden sein müssen. Die Lösungsrepräsentation muss auf CEP-Regeln ausgelegt sein. Dazu müssen CEP-Regeln gegebenenfalls als Zahlen kodiert werden, damit sie mit den BA-typischen Gleichungen aktualisiert werden können. Andernfalls müsse man den BA um weitere Logik zum Verarbeiten von CEP-Regeln beispielsweise als Zeichenketten erweitern und dann auch die BA-typischen Gleichungen anpassen. Außerdem muss eine auf CEP-Regeln zugeschnittene Fitness-Funktion eingeführt werden, mit der sich CEP-Regeln evaluieren lassen. Hierfür können gegebenenfalls Messmethoden aus dem Bereich der *Information Retrieval* herangezogen werden. In den nachfolgenden Abschnitten werden konkrete Realisierungsmöglichkeiten vorgestellt.

4.2 Maßnahmen zur Anpassung des Bat-Algorithmus an das Lernen von CEP-Regeln

CEP-Regeln mit dem BA zu lernen bedeutet sie in gewisser Weise als Fledermäuse darzustellen und ihnen die BA-typischen Attribute zu verleihen. Damit gehen gewisse Anpassungen des BA einher, denn genau wie beim TSP (Abschnitt 2.4) handelt es sich beim Lernen von CEP-Regeln um ein kombinatorisches Optimierungsproblem. Von den Anpassungen sind sowohl die Fledermaus-Attribute als auch alle Gleichungen und Operationen betrof-

fen. Allen voran die Repräsentation eines Lösungskandidaten als CEP-Regel. Nachfolgend wird zunächst eine mögliche Fitness-Funktion vorgestellt, mit der sich CEP-Regel hinsichtlich ihrer Qualität bezogen auf einen gegebenen Trainingsdatensatz messen lassen. Darauf folgen konkrete Abbildungsansätze sowohl für die Fledermaus-Attribute als auch für die Operatoren im Zufallsflug und in der lokalen Suche.

4.2.1 Eine Fitness-Funktion für CEP-Regeln

Jede Position, also jede CEP-Regel muss hinsichtlich ihrer Qualität messbar und auf einen korrespondierenden, skalaren Wert abbildbar sein. Dazu wird sie dem BA gemäß als Lösungskandidat in eine entsprechende Funktion F_{opt} eingesetzt – die Fitness-Funktion. Für ein Beispiel einer solchen Funktion sei noch einmal zurück auf den Abschnitt 2.4 verwiesen, in dem die Fitness-Funktion des BA für das TSP beschrieben ist. Die Qualität einer CEP-Regel im Kontext eines Lernverfahrens für CEP-Regeln, lässt sich im Wesentlichen darauf zurückführen, wie präzise ihre Regel-Bedingung zu Z führt. Eine sehr gute CEP-Regel $R*$ lässt sich demnach folgendermaßen beschreiben: gegeben sei ein Datensatz bestehend aus einigen primitiven Ereignissen und auch das komplexe Ereignis Z kommt ein paar mal darin vor. Die CEP-Regel wird auf diesen Datensatz angewendet. Genau dann wenn die primitiven Ereignisse die Regel-Bedingung erfüllen, ist Z das nächste Ereignis.

Um die Fitness zu messen, muss also jede Regel auf den Datensatz angewendet werden und daraus lassen sich dann die folgenden Informationen gewinnen:

- True Positives (TP): ist die Anzahl der Fälle, in denen die Regel korrekt gefeuert hat. Also alle Fälle, in denen die Regel feuert und Z das nächste Ereignis ist.
- False Positives (FP): ist die Anzahl der Fälle, in denen die Regel fälschlich gefeuert hat. Dies ist immer dann der Fall, wenn die Regel-Bedingung erfüllt aber Z *nicht* das nächste Ereignis ist.
- True Negatives (TN): ist die Anzahl der Fälle, in denen die Regel korrekt nicht gefeuert hat. Also alle Fälle in denen die Regel-Bedingung nicht erfüllt und Z auch nicht das nächste Ereignis ist.
- False Negatives (FN): ist die Anzahl der Fälle, in denen die Regel fälschlicherweise nicht gefeuert hat. Also alle Fälle, in denen die Regel-Bedingung nicht erfüllt ist, obwohl Z das nächste Ereignis ist.

Darauf basierend lässt sich die Qualität einer Regel in Form verschiedener Verhältnisse wie beispielsweise *Recall* und *Precision* evaluieren. Einige interessante Messungen und Verhältnisse, die hier als Fitness-Funktion eingesetzt werden können, wurden detailliert von Powers in [54] zusammengetragen und beschrieben. Das folgende Beispiel demonstriert die Anwendung einer CEP-Regel auf einen Datensatz und die Informationsgewinnung daraus:

CEP-Regel:

```
(A AS a → C AS c) ∧
(a.wert < c.wert)[win:len:4] ⇒ Z
```

Datensatz:

```
2017-07-01 09:03:20 A; wert=1
2017-07-01 09:03:23 C; wert=2
Z
2017-07-01 09:03:25 A; wert=1
2017-07-01 09:03:45 B; wert=1
2017-07-01 09:03:48 B; wert=7
2017-07-01 09:03:53 C; wert=4
Z
```

Informationen:

- TP: 2 / TN: 4
- FP: 0 / FN: 0

Mögliche Fitnessbestimmung:

$TP + TN = 6$ (positive Messungen) und $FP + FN = 0$ (negative Messungen). Damit feuert die CEP-Regel immer nur an genau den richtigen Stellen und erreicht eine Fitness von 100%.

Mit diesen einfachen Messungen und Berechnungen ist es grundsätzlich möglich, einen repräsentativen und vergleichbaren (Prozent-)Wert für eine CEP-Regel zu berechnen.

4.2.2 Die CEP-Regel als Position

Dem BA entsprechend repräsentiert die Position x_i einer Fledermaus i einen Lösungskandidaten. Daraus folgt, dass sie eine komplette CEP-Regel abbilden muss und in etwa folgende Form hat:

$$x_i = (A \text{ AS } a \rightarrow B \text{ AS } b) \wedge (\text{a.temp} = \text{b.temp})[\text{win:time:5min}] \Rightarrow Z$$

Die Position ist jetzt also *kein* skalarer Wert mehr, sondern eine kodierte CEP-Regel. A und B sind Ereignisse aus dem Trainingsdatensatz und Z steht immer in der Konklusion. Des Weiteren muss der Algorithmus die Regel im Zuge eines Positionswechsels verändern können, jedoch können die originalen Gleichungen 7 und 8 aus Abschnitt 2.4 hierfür nicht angewendet werden, eben weil x_i jetzt kein skalarer Wert mehr ist. Stattdessen gilt es spezifische Operationen dafür zu entwerfen, die CEP-Regeln manipulieren können. Diese Operationen müssen Ereignistypen, Vergleichsoperatoren und Attribut-Werte gegen andere austauschen, sie entfernen oder neue einfügen können und auch neue Fenstergrößen bestimmen – die CEP-Regel also im Rahmen der Möglichkeiten gezielt manipulieren. Dabei sind jedoch Bedingungen einzuhalten. Beispielsweise können nur solche Ereignistypen hinzukommen, die auch im Trainingsdatensatz enthalten sind. Längenfenster dürfen nicht größer werden als die Anzahl aller Ereignisse im Trainingsdatensatz und Zeitfenster dürfen den zeitlichen Raum zwischen dem ersten und dem letzten Ereignis im Trainingsdatensatz nicht verlassen. Kurzum: die Operationen müssen sich also an das in einem Vorverarbeitungsschritt erzeugte Ereignismodell halten. Des Weiteren wird hier auf Methodenaufrufe innerhalb einer CEP-Regel verzichtet, weil sich das Lernen von Regeln zunächst allein auf Ereigniskonstellationen im Trainingsdatensatz stützt.

Damit die Operationen für den Positionswechsel implementiert werden können, müssen die CEP-Regeln zunächst auf eine Weise kodiert sein, die lesenden und schreibenden Zugriff auf einzelne Elemente der Regel erlaubt und die nach Möglichkeit effizient zu bearbeiten ist. Nachfolgend werden diesbezüglich einige Kodierungen vorgestellt.

Mögliche Kodierungen für CEP-Regeln Eine naheliegende Kodierung wäre das einfache Abbilden der CEP-Regel als ***Zeichenkette***, in der durch gesonderte Trennzeichen Ereignismuster, Kontextbedingung, Regelfenster und Aktion abgegrenzt werden können. Auch die einzelnen Zeichen innerhalb dieser drei Elemente müssen individuell zugreifbar sein und sind ebenfalls durch gesonderte Zeichen angegrenzt, sofern dies erforderlich ist. Nachfolgend ist ein beispielhafter Ansatz dieser Kodierung zu sehen:

`(A;AS;a;→;B;AS;b)|∧|(a.temp;=;b.temp)|[win:time:5min]⇒Z`

Die drei Elemente Ereignismuster, Kontextbedingung und Regelfenster sind durch den geraden Strich „|“ getrennt. Die Zeichen innerhalb des Ereignismusters sind durch das Semikolon „;“ getrennt. Die Attribute lassen sich anhand des ohnehin verwendeten Punktes „.“ erkennen und das gleiche gilt für das Regelfenster, hierbei jedoch mit dem Doppelpunkt „:“. *Z* steht immer am Ende der Regel und folgt auf den Pfeil „⇒“.

Um einen Positionswechsel zu vollziehen, die Regel also gezielt zu Aktualisieren, muss die Zeichenkette zunächst Zeichen für Zeichen eingelesen werden. Darauf folgt eine Extraktion der Zeichen unter Zuhilfenahme üblicher *String*-Operationen. Dann werden die Zeichen interpretiert und aktualisiert. Am Ende muss die neue Regel wieder als Zeichenkette mit Trennzeichen abgelegt werden. Sie sieht dann beispielsweise so aus:

`(A;AS;a;→;C;AS;c)|∧|(a.temp;=;c.temp)|[win:time:2min]⇒Z`

Eine artverwandte Alternative ist die ***Zahlenkodierung***, die jedes Zeichen mit einer Ziffer repräsentiert. Die Idee dazu ist von der sogenannten „Gödelisierung“ [8] inspiriert, mit der sich Worte einer formalen Sprache als natürliche Zahlen repräsentieren lassen.

Beispielsweise steht die 1 für das Ereignis *A* und die 2 für *B*. Soll in einer Regel nun *A* durch *B* ersetzt werden, so geschieht dies einfach durch Erhöhen der 1 auf die 2. Ebenso existieren für alle Operatoren, Attribute, Fenstertypen und Fenstergrößen entsprechende Ziffern und auch hier gibt es eine spezielle Ziffer oder ein Zahlenmuster, mit dem Ereignismuster, Kontextbedingung, Regelfenster und Aktion abgegrenzt werden. Nachfolgend ist ein Beispiel angegeben, das dieselbe Regel wie zuvor kodiert. Dieses Mal jedoch als Zahl:

`1220101112101503`

Die Null (0) ist das Trennzeichen zwischen Ereignismuster, Kontextbedingung, Regelfenster und Aktion. Im Ereignismuster steht die 1 für das Ereignis *A* und die 2 für *B*. Getrennt sind diese durch den Sequenzoperator (→), der innerhalb der Menge der Operatoren für das Ereignismuster ebenfalls durch die 2 repräsentiert wird. Das Ereignismuster und die Kontextbedingung sind durch den UND-Operator (∧) miteinander verbunden. Dieser wird durch die 1 repräsentiert. Innerhalb der Kontextbedingung steht zuerst `A.temp`. Dies wird abgebildet durch 11, wobei die erste 1 für das *A* steht und die zweite für dessen Attribut `temp`. Darauf folgt das Gleichzeichen (=), das innerhalb der Menge der boolschen Operatoren durch die 1 repräsentiert wird. Als letztes wird `B.temp` analog zu `A.temp` abgebildet und weil *B* durch die 2 repräsentiert wird, ergibt sich daraus 21. Das Regelfenster ist vom Typ Zeit, für den die 1 steht (ein Längenfenster würde mit 2 angegeben werden). Für die Fenstergröße muss eine Einheit vorausgesetzt werden. In diesem Fall sind es fünf Minuten,

[8] `https://de.wikipedia.org/wiki/G%C3%B6delnummer` (zuletzt zugegriffen am 21. August 2017)

repräsentiert durch die Ziffer 5. An letzter Stelle steht das Z, was durch die 3 abgebildet ist.

Mit dieser Kodierung lassen sich CEP-Regeln einfach als lange Zahlen ausdrücken aber dennoch müssen für Manipulationen die einzelnen Ziffern zugegriffen werden – wie bei der Zeichenkette. Dafür ist der Austausch einzelner Zeichen durch einfache Additionen oder Subtraktionen der entsprechenden Ziffern möglich. Nach einer beispielhaften Aktualisierung, in der im Ereignismuster der Sequenzoperator durch ein logisches ODER ($\vee$) ersetzt wurde, sieht die Regel wie folgt aus:

```
1320101112101503
```

Das logische ODER wurde in der Menge der Operatoren mit der 3 kodiert.

Ein wenig komplizierter ist beispielsweise das Entfernen eines Ereignisses. Denn dann muss auch der nachfolgende Operator und wenn vorhanden auch die Ereignis-Attribute entfernt werden. Selbiges gilt für das Einfügen von Ereignissen. Problematisch wird es auch, wenn die Regel Aliasnamen für zwei oder mehrere Ereignisse desselben Typs hat (beispielsweise A AS $a_1 \rightarrow$ A AS a_2). Dieser Fall müsste explizit mittels einer weiteren Kodierung innerhalb dieser Kodierung abgebildet werden, damit nicht beiden Ereignissen dieselbe Ziffer zugeordnet wird. Ein weiteres Problem tritt auf, wenn es mehr als neun verschiedene Ereignistypen oder Attribute gibt, dann müssen diese durch mehrstellige Zahlen abgebildet werden und dadurch ist beim Traversieren der Regel nicht mehr klar, wie die nachfolgende Ziffer zu interpretieren ist. Auch hier müssten dann wieder Trennzeichen innerhalb des Ereignismusters und der Kontextbedingung eingesetzt werden.

Als dritte Kodierung sei die Repräsentation der ***CEP-Regel als Baum*** vorgeschlagen. Dabei würde die Regel selbst den Wurzelknoten darstellen, der jeweils für Ereignismuster, Kontextbedingung, Regelfenster und Aktion Kindknoten hat. Alle einzelnen Elemente wie Ereignisse, Attribute, Attribut-Werte, Fensterwerte und Operatoren sind Knoten im Baum. Nachfolgend ist auch hierfür ein Beispiel angegeben. Es zeigt dieselbe Regel, die auch zuvor kodiert wurde. Jetzt jedoch in einer Baumdarstellung (Abbildung 12). Besonders vor-

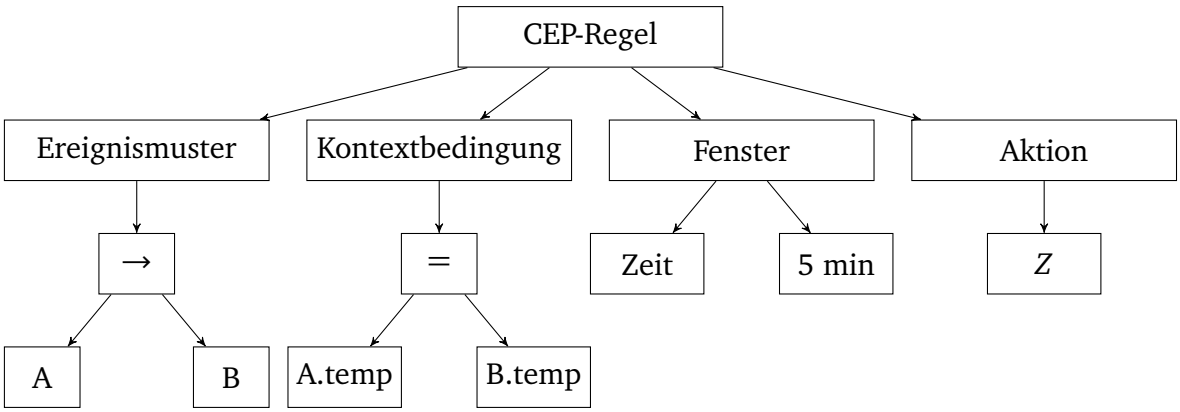


Abbildung 12: CEP-Regel in Baumdarstellung

teilhaft an dieser Kodierung ist die Tatsache, dass Bäume bekanntermaßen effiziente Datenstrukturen sind. Jeder Knoten kann durch traversieren des Baumes mit einem geringen

Aufwand zugegriffen und ausgetauscht oder entfernt werden. Auch das Einfügen neuer Knoten ist mit geringem Aufwand möglich. Überdies hat ein Baum nicht die Nachteile der zuvor aufgeführten Kodierungen. Besonders weil keine besonderen Trennzeichen benötigt werden und auch Aliasnamen für Attribute desselben Typs können einfach als Knoten abgelegt werden. Die für den Positionswechsel verwendete Operation muss folglich einfach die Struktur des Baumes kennen, also Ereignismuster, Kontextbedingung, Regelfenster und Aktion unterscheiden können und kann dann einzelne Knoten bearbeiten. Hierbei muss jedoch unbedingt die Zusammengehörigkeit von Ereignissen und ihren Attributen beachtet werden. Folglich wirken sich Manipulationen am Ereignismuster-Teilbaum auch auf den Teilbaum der Kontextbedingung aus. Wird ein Ereignis entfernt, so müssen auch dessen Attribute entfernt werden. Wird ein Ereignis hinzugefügt, so können auch seine Attribute hinzugefügt werden. Was dabei konkret geschieht ist von der Aktualisierungsmethode im Positionswechsel abhängig. Diese kann beispielsweise einfach zufällige Operationen auf zufälligen Knoten durchführen und so würden manche Ereignisse und/oder Attribute gegebenenfalls manipuliert und andere eben nicht.

Generell kann bei allen hier vorgestellten Kodierungen der Aktionsteil jeder Regel prinzipiell ausgelassen werden. Dieser beinhaltet immer dieselbe Information (Z), die ebenso an einer zentralen Stelle im Algorithmus hinterlegt werden kann.

4.2.3 Operationen für den Positionswechsel

Wie bereits zuvor erwähnt, muss eine CEP-Regel im Zuge des Positionswechsels mittels verschiedener Operationen aktualisiert beziehungsweise manipuliert werden. Diese Operationen lassen sich gemäß ihrer Funktionsweise wie folgt zusammenfassen:

- Einfügen neuer Ereignisse: ermöglicht es auch neue Attribute hinzuzufügen
- Entfernen vorhandener Ereignisse: erfordert auch das Entfernen verknüpfender Operatoren und wenn vorhanden auch das Entfernen zugehöriger Attribute und gegebenenfalls deren verknüpfende Operatoren. Das letzte Ereignis darf nicht entfernt werden, weil die Regel-Bedingung sonst leer wäre.
- Ersetzen vorhandener Ereignisse: erfordert auch das Ersetzen der Attribute, falls diese vorhanden sind. Eröffnet die Möglichkeit, einen Attribut-Vergleich hinzuzufügen.
- Einfügen neuer Attribute: ist nur möglich wenn die Ereignisse im Ereignismuster Attribute besitzen.
- Entfernen vorhandener Attribute: erfordert das Entfernen verknüpfender Operatoren und Operanden.
- Ersetzen vorhandener Attribute: ist nur dann möglich, wenn das zugehörige Ereignis noch weitere Attribute hat. Zudem ist es möglich den zweiten Operand eines Vergleichsoperators durch eine Konstante anstelle eines anderen Attributs zu ersetzen (beispielsweise `a.temp < 2` anstelle von `a.temp < b.temp`).
- Einfügen neuer Operatoren: erfordert gleichzeitig das Einfügen neuer Ereignisse beziehungsweise neuer Attribute oder Konstanten für einen Attribut-Vergleich.

- Entfernen vorhandener Operatoren: erfordert zugleich das Entfernen der Operanden.
- Ersetzen vorhandener Operatoren
- Den Fenstertyp wechseln: wechselt die Einheit von Zeit zu Länge oder andersherum und erfordert die Angabe eines gültigen Wertes für die Einheit.
- Fenstergröße neu festlegen

Zudem sei noch einmal darauf hingewiesen, dass der Aktionsteil *nicht* verändert wird. Dieser beinhaltet immer nur Z.

An einigen Stellen im BA spielen Zufallsvariablen eine Rolle. So verändert sich die Frequenz in jeder Iteration zufallsbedingt, was wiederum zu einer zufallsbedingten Veränderung sowohl der Geschwindigkeit als auch der Position führt. Diese Zufälligkeit und die kausale Abhängigkeit von Frequenz, Geschwindigkeit und Position muss auf das Anwenden der beschriebenen Operationen übertragen werden. Der nachfolgende Abschnitt beschreibt wie dies zu realisieren ist.

4.2.4 Kausale Kette bestehend aus Frequenz, Geschwindigkeit und Position

Beide, die Frequenz f_i^t und die Geschwindigkeit v_i^t behalten ihre bisherigen Funktionen bei, um dem Algorithmus so treu wie möglich zu bleiben. Gleichwohl kann die originale Gleichung 4 der Geschwindigkeit nicht bestehen bleiben. Sie wird für einen besseren Überblick noch einmal aufgegriffen:

$$v_i^t = v_i^{t-1} + (x_i^{t-1} - x^*) f_i^t.$$

Die Gleichung verlangt dass die Positionen x_i^{t-1} und x^* skalare Werte sind, doch das ist bei CEP-Regeln nicht der Fall. Also muss für die Abbildung des BA auf die Optimierung von CEP-Regeln entweder eine andere Berechnung der Geschwindigkeit stattfinden oder ein Konvertierungsschritt eingeführt werden, der CEP-Regeln auf skalare Werte abbildet.

Die Frequenz soll sich auf die Geschwindigkeit auswirken und diese wiederum soll die Position beeinflussen. So gibt es der BA vor und auch die Abbildung des BA auf die Optimierung von CEP-Regeln soll diese kausale Kette nicht brechen. Was sich jedoch verändern kann, ist die konkrete Anwendung beider Attribute: die Differenz zwischen Frequenz und Geschwindigkeit wird als Anzahl durchzuführender Manipulationsoperationen interpretiert. Hat die Geschwindigkeit v_i^t beispielsweise einen Wert von 4 und die Frequenz f_i^t einen Wert von 7, so sind 3 zufällig auszuwählende Operationen (aus dem vorherigen Abschnitt) durchzuführen, die die CEP-Regel zufällig an irgendeiner Stelle innerhalb des Ereignismusters, der Kontextbedingung oder des Regelfensters manipulieren. Je größer die Differenz zwischen v_i^t und f_i^t, desto stärker wird die Regel manipuliert, weil mehr Operationen durchgeführt werden. Somit wird die neue Position x_i^t nach wie vor mithilfe der Frequenz und der Geschwindigkeit bestimmt, womit sich die kausale Kette $f \rightarrow v \rightarrow x$ vervollständigt. Eine ähnliche Abbildung haben Heraguemi, Kamel und Drias in [28] geschaffen, um mit dem BA Assoziationsregeln zu lernen. Bei ihrem Ansatz wirkt die Geschwindigkeit nur indirekt auf die neue Position ein, wodurch die Gleichungen 4 und 3

prinzipiell vereinfacht werden können. Beispielsweise können sie ersetzt werden durch: $f_i^t = f_{max} * \beta$ und $v_i^t = f_i^t * \beta$, wobei β nach wie vor eine Zufallsvariable aus dem Wertebereich [0,1] ist. Somit stellt die Frequenz stets die obere Grenze für einen Bereich dar, aus dem auch die Geschwindigkeit ausgewählt wird. Damit ist die Geschwindigkeit stets kleiner als die Frequenz, womit sich immer eine zufällige Differenz ergibt. Das oben beschriebene Typen-Problem mit der originalen Gleichung der Geschwindigkeit wäre somit also nicht mehr relevant.

Alternativ und um dem BA möglichst treu zu bleiben, kann nach wie vor die originale Gleichung 3 verwendet werden. Die Gleichung 4 muss in jedem Fall angepasst werden, weil sie in ihrer originalen Definition skalare Werte für x_i^{t-1} und x^* erwartet. Dazu sei die folgende Anpassung vorgeschlagen:

$$v_i^t = v_i^{t-1} + (F_{opt}(x_i^{t-1}) - F_{opt}(x^*))f_i^t.$$

Durch das Einsetzen der Fitness-Funktion F_{opt} sowohl mit der Position x_i^{t-1} als auch mit der global besten Position x^* als Parameter, lassen sich die CEP-Regeln auf repräsentative skalare Werte abbilden und getreu dem BA in die Aktualisierung von v_i^t einbringen.

4.2.5 Pulsrate und Lautstärke

Damit das Verhältnis zwischen Erkundung und Ausbeutung nach wie vor erhalten bleibt, macht es Sinn auf jeden Fall die Pulsrate r_i und gegebenenfalls auch die Lautstärke a_i einer Fledermaus i nicht neu zu interpretieren. Somit behält die Abbildung des BA auf das Lernen von CEP-Regeln wesentliche Charakterzüge des originalen BA bei. Alternativ kann die Lautstärke beziehungsweise die durchschnittliche Lautstärke a^t über alle Fledermäuse in einem Zeitschritt dennoch angepasst werden, denn sie dient im Gegensatz zur Pulsrate nicht nur als Vergleichswert, sondern ist auch ein Einflussfaktor auf den Positionswechsel in der lokalen Suche (Gleichung 8). Diese Gleichung kann in ihrer originalen Form jedoch nicht angewendet werden, weil die Position kein skalarer Wert sondern eine CEP-Regel ist. Um die Position dennoch mithilfe der Lautstärke zu manipulieren, kann mit dieser ähnlich verfahren werden, wie schon zuvor mit der Geschwindigkeit und der Frequenz: der Wert der Durchschnittslautstärke a^t multipliziert mit ϵ als Zufallsvariable aus dem Wertebereich [0,1] repräsentiert die Anzahl zufällig durchzuführender Manipulationsoperationen *in der lokalen Suche*. Daraus folgt: je größer der Wert der Durchschnittslautstärke ist, desto ausgeprägter wird die Position in der lokalen Suche verändert. Hat die Durchschnittslautstärke beispielsweise den Wert 3, so sind $\lfloor 3\epsilon \rfloor$ zufällige Operationen auf x^* (beziehungsweise eine Kopie von davon) anzuwenden um eine neue Position x_i^t daraus zu erzeugen.

4.2.6 Zufallsflug und lokale Suche

Der Zufallsflug und die lokale Suche verknüpfen nun die zuvor aufgeführten Operationen mit den neu interpretierten Attributen Frequenz, Geschwindigkeit und Lautstärke. Dabei behalten beide, der Zufallsflug und die lokale Suche ihre charakteristischen Funktionsweisen bei. So verändert sich eine CEP-Regel im Zufallsflug zufällig und in der lokalen Suche in Abhängigkeit von der besten CEP-Regel, also von der am besten positionierten

Fledermaus im Schwarm. Die konkrete Anwendung der Operationen hängt dabei von der Kodierung der CEP-Regeln ab. Nachfolgend sind Beispiele für den Zufallsflug mit den drei vorgestellten Kodierungen angegeben.

Zufallsflug Es gelten die folgenden beispielhaften Attribut-Werte, deren konkrete Berechnungen für die Beispiele erst einmal keine Rolle spielen: Frequenz $f_i^t = 5,2$ und Geschwindigkeit $v_i^t = 2,1$. Daraus ergibt sich eine Differenz $\Delta_v^f = 3,1$ und diese wird auf eine ganze Zahl, also 3 abgerundet. Folglich sind drei zufällige Operationen an der CEP-Regel x_i^{t-1} durchzuführen, um sie für den aktuellen Zeitschritt zu aktualisieren (x_i^t):

- ***CEP-Regel als Zeichenkette:***

  x_i^{t-1} = `(A;AS;a;→;B;AS;b)|∧|(a.temp;=;b.temp)|[win:time:5min]⇒Z`

  x_i^t = `(A;AS;a;∨;B;AS;b)|∧|(a.temp;<;b.temp)|[win:time:10sec]⇒Z`

  Zufällig wurde der Sequenzoperator durch das logische ODER ersetzt; der „=“-Operator durch den „<“-Operator in der Kontextbedingung und das Fenster von 5 Minuten auf 10 Sekunden reduziert.

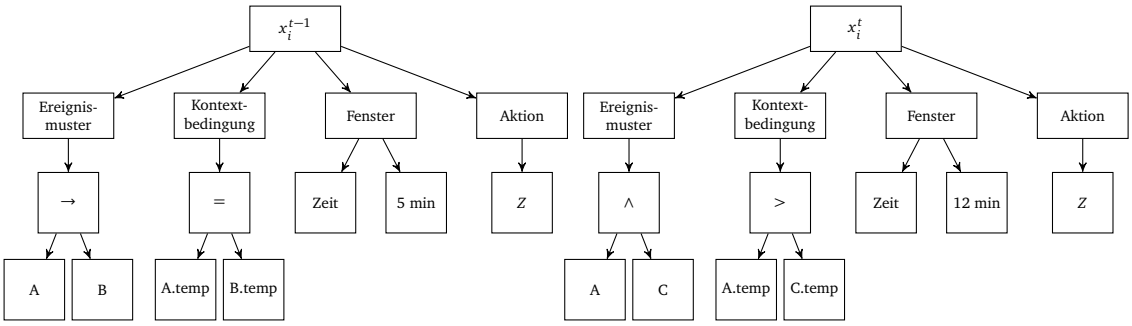
- ***CEP-Regel als Zahl kodiert:***

  x_i^{t-1} = `1220101112101503`

  x_i^t = `1210101111201703`

  Durch dreimaliges Addieren oder Subtrahieren zufälliger Stellen (ausgenommen der Trennzeichen und des komplexen Ereignisses Z) wurden sowohl das Ereignismuster als auch die Kontextbedingung und das Fenster verändert.

- ***CEP-Regel als Baum kodiert:*** Eine Baum-traversierende Operation hat zunächst den



Ereignistyp B durch C im Ereignismuster ausgetauscht. Dadurch musste auch die Kontextbedingung von B an C angepasst werden. Im zweiten Schritt wurde der Vergleichsoperator in der Kontextbedingung verändert und zuletzt auch das Fenster.

Lokale Suche Diese für den Zufallsflug angegebenen Beispiele gelten in gleichem Maße auch für die lokale Suche; jedoch mit zwei Unterschieden: erstens leitet sich die aktualisierte Position x_i^t in der lokalen Suche nicht von x_i^{t-1} ab, sondern immer von der aktuellen, global besten Position x^*. Zweitens ergibt sich die Anzahl der Operationen nicht durch Δ_v^f sondern durch den auf eine ganze Zahl abgerundeten Wert der Durchschnittslautstärke multipliziert mit einem Zufallswert: $\lfloor a^t \epsilon \rfloor$.

5 BatCEP – Ein Bat-Algorithmus zum Lernen von CEP-Regeln

BatCEP ist ein auf dem Bat-Algorithmus basierendes und damit von der Natur inspiriertes Verfahren zum Lernen von CEP-Regeln. Es kann Muster in Ereignisströmen erkennen und daraus Ursachen-Quellen für bestimmte Ereignisse herleiten. Seine Kernkomponente ist eine an die Optimierung von CEP-Regeln angepasste Variante des *Bat-Algorithmus* und damit handelt es sich um ein maschinelles Lernverfahren aus dem Bereich der Schwarmintelligenz. Ähnlich wie *cepGP* [51], *iCEP* [47] und *autoCEP* [49] nimmt es als Eingabe einen Ereignisdatenstrom (beziehungsweise Ausschnitte daraus) sowie einen vom Benutzer definierten Ziel-Ereignistyp Z entgegen und liefert im Ergebnis CEP-Regeln, deren Bedingungen gelernte Muster repräsentieren, die zu Z führen.

In den nachfolgenden Abschnitten wird BatCEP tiefer gehend erläutert, womit die konkrete Umsetzung von einigen Ansätzen aus dem vorherigen Kapitel einhergeht. Dies beginnt mit der Repräsentation der CEP-Regeln als Bäume und führt weiter über die Realisierung des Optimierungsalgorithmus als Kern des Verfahrens mit seinen spezifischen Aktualisierungsoperationen.

Zuletzt realisiert BatCEP den Einsatz mehrerer, parallel eingesetzter Schwärme. Die Idee dazu stammt aus dem Dokument [28] von Heraguemi, Kamel und Drias, einem der bereits in Abschnitt 3.3 erläuterten Ansätze, um Assoziationsregeln mithilfe des BA zu lernen. Im Zuge dessen suchen mehrere Schwärme zeitgleich nach einer Lösung und kommunizieren ihre besten Ergebnisse untereinander.

5.1 Repräsentation von CEP-Regeln als Lösungskandidaten in BatCEP

BatCEP verwendet die im Rahmen von *cepGP* entwickelte CEP-Engine, für die bereits gewisse Entwurfsentscheidungen wie beispielsweise die Repräsentation der CEP-Regeln getroffen wurden. Diese Engine formuliert die in Abschnitt 2.1 behandelten Konzepte in einer einfachen *CEP-Pseudosprache*, die wiederum von Bruns und Dunkel [10] übernommen wurde. Aus Gründen der Komplexität wurde zunächst auf ein populäres CEP-Produkt wie beispielsweise *Esper* und damit auf eine komplexe Regelsprache verzichtet. Viele populäre CEP-Produkte haben einen großen Funktionsumfang, womit zumeist eine vielfältige Regelsprache und Syntax einhergeht. BatCEP verfolgt jedoch den theoretischen Ansatz des *Lernens von CEP-Regeln zur Ursachen-Erkennung* unter Einhaltung grundlegender CEP-Konzepte und bezieht sich damit nicht auf ein spezielles Produkt. Aufgrund dessen ist eine zukünftige Adaptierung des Verfahrens auf eine populäre CEP-Engine durchaus möglich. Die nachfolgenden Abschnitte sind der Repräsentation von CEP-Regeln gewidmet, die inhaltlich im Wesentlichen auf der Masterarbeit von Norman Offel [51] basiert.

CEP-Regeln bestehen – wie bereits in Abschnitt 2.1.3 beschrieben – aus zwei wesentlichen Komponenten: *Regelbedingung* und *Aktion*. BatCEP führt an dieser Stelle eine Trennung durch und teilt die Regelbedingung in zwei Komponenten auf – namentlich in *Ereignisbedingung*, bestehend aus Ereignismuster und Kontextbedingung; sowie *Fenster*, ge-

messen in Länge oder Zeit. Effizienz und Granularität sind die Gründe für diese Trennung. So betreffen Veränderungsoperationen nicht stets die komplette Regelbedingung, sondern gezielt das Ereignismuster, die Kontextbedingung oder das Regelfenster.

Strukturell ist die Regel als Baum aufgebaut, ähnlich dem, der bereits im letzten Kapitel als mögliche Kodierung vorgeschlagen wurde. Die Regel selbst ist der Wurzelknoten, der immer drei Kindknoten hat: die Bedingung, das Fenster und die Aktion. Alle drei sind jeweils auch Wurzelknoten entsprechender Teilbäume, mit konkreten Ereignistypen, Attributen, Fensterwerten und dem Ereignis Z als Kindknoten. Abbildung 13 zeigt den grundlegenden Aufbau einer solchen Regel. Im gepunkteten Rechteck zusammengefasst ist die

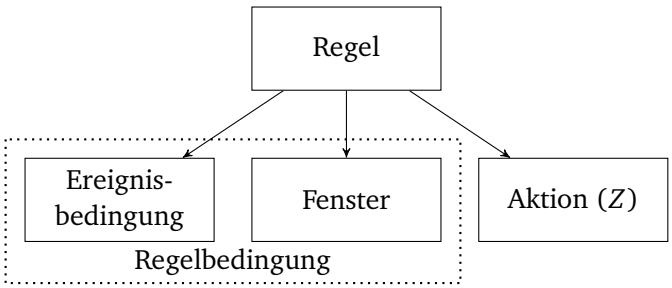


Abbildung 13: Abstrakt dargestellte Repräsentation einer CEP-Regel als Lösungskandidat in BatCEP.

Regelbedingung, bestehend aus Ereignismuster, Kontextbedingung und Regelfenster. Die Aktion repräsentiert die beim Feuern der Regel durchgeführte Aktion, die immer nur Z enthält.
Für BatCEP ist die Regelbedingung Hauptarbeitsgegenstand und damit von besonderem Interesse. Da die Aktion beim Feuern der Regel immer nur zu Z führt, trägt sie lediglich zur Vollständigkeit einer wohlgeformten CEP-Regel bei.

5.1.1 Ereignisbedingung

Die Ereignisbedingung ist ein essentieller Teil der Regel und besteht zum einen aus dem sogenannten Ereignismusterteilbaum *(Eventconditiontree ECT)*, der das Ereignismuster mit all seinen Bestandteilen abbildet. Zum anderen aus dem Kontextbedingungsteilbaum *(Attributeconditiontree ACT)*. Letzterer nimmt Bezug auf den Ereignismusterteilbaum, indem er den Ereignissen zugehörige Attribute und Bedingungen abbildet. Abbildung 14 illustriert die Position im Regelbaum. Die Bezeichnungen ECT und ACT sind für eine einheitliche Beschreibung aus der Masterarbeit von Norman Offel übernommen.

Ereignismuster (ECT) Die Ereignisbedingung führt die in der Regel betrachteten und mit Operatoren verknüpften Ereignistypen auf. Aufgrund der Baumstruktur macht es Sinn, die Operatoren als Wurzel und die Operanden als Kinder davon einzusetzen. Mögliche Kinder sind dann jedoch nicht nur Ereignistypen, sondern auch weitere Operatoren. Diese Struktur zieht sich fort, bis in den Blättern nur noch Ereignisse stehen. Mögliche Operatoren zum Verknüpfen der Ereignisse sind für die CEP üblichen, die schon im Abschnitt 2.1 vorgestellt wurden ($\rightarrow, \wedge, \vee, \neg$).

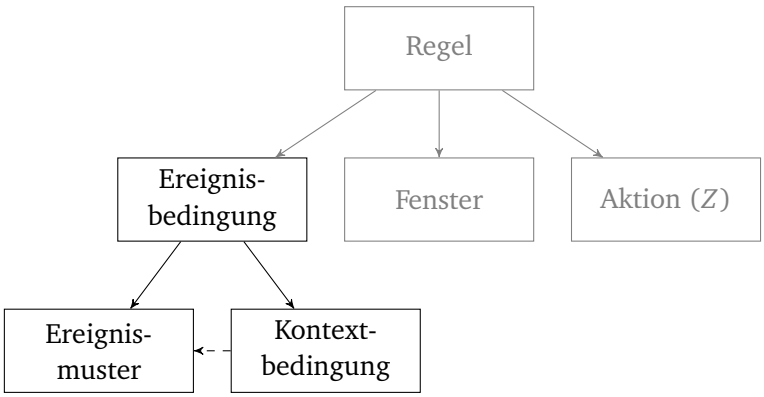


Abbildung 14: Aufteilung der Regelbedingung in Ereignismusterteilbaum (ECT) und Kontextbedingungsteilbaum (ACT).

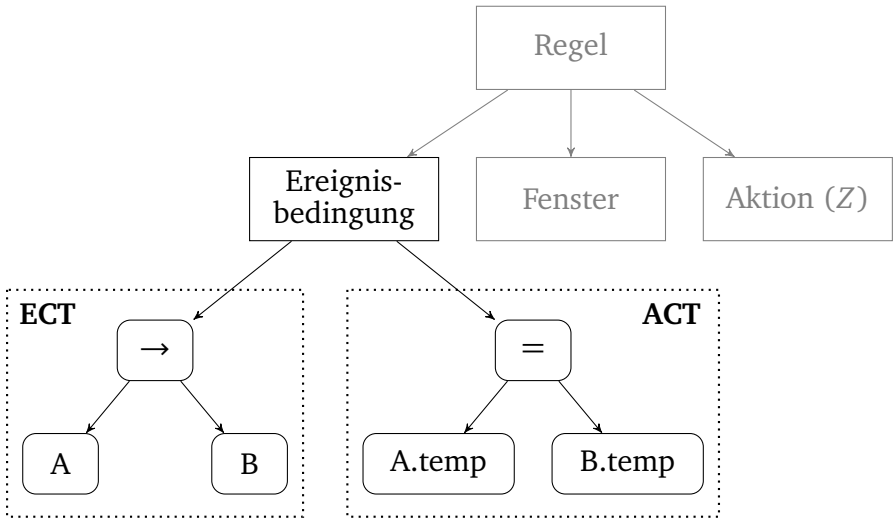


Abbildung 15: Darstellung des ECT und ACT für die beispielhafte Bedingung: $(A \rightarrow B) \wedge (A.temp = B.temp)$.

Kontextbedingung (ACT) Der ACT bildet die in der Regel betrachteten Kontextbedingungen ab und stellt Bezüge zu den entsprechenden Ereignissen im ECT her. Genauer formuliert: im ACT sind genau diejenigen Attribute abgelegt, die den Ereignissen im ECT angehören und in der Regel von Interesse sind. Hinzu kommen noch die üblichen Vergleichsoperatoren aus Abschnitt 2.1 $(+, -, *, / <, >, \leq, \geq, =, \neq)$.

Abbildung 15 illustriert den Aufbau und Zusammenschluss von ACT zu ECT. Beide Ereignisse A und B besitzen das Attribut *temp*. Damit die Bedingung erfüllt ist, muss B auf A folgen und die Werte beider Attribute müssen gleich sein. Für den ACT gilt dieselbe Struktur wie für den ECT: Operatoren sind Wurzeln mit weiteren Operatoren oder Attributen als Kinder. In den Blättern stehen nur Attribute.

5.1.2 Regelfenster

Der zweite Teilbaum der Regelbedingung ist das Zeit- beziehungsweise Längenfenster. Es ergänzt die Ereignisbedingung insofern, als nur diejenigen Ereignis-Instanzen betrachtet werden, die innerhalb des Fensters auftreten. Der Beschreibung in Abschnitt 2.1.2 gemäß, kann eine Regel entweder ein Zeitfenster oder ein Längenfenster haben. In der von BatCEP verwendeten CEP-Engine nehmen beide Fenstertypen denselben, skalaren Eingabewert entgegen und bilden diesen auf ihre Einheit ab (Zeit oder Anzahl von Folgeereignissen). Abbildung 16 zeigt das Fenster im Regelbaum. Ein Zeitfenster bemisst sich in

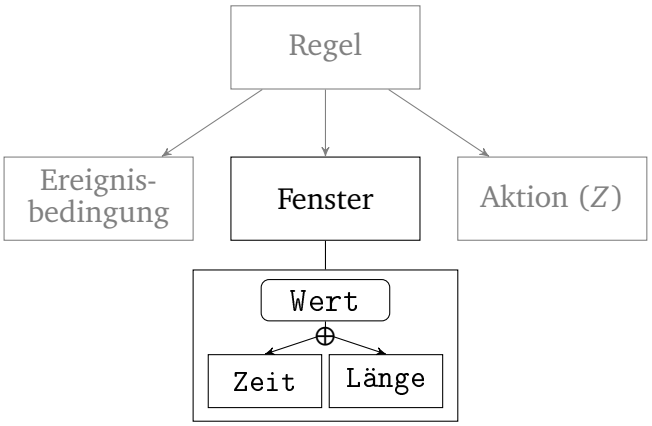


Abbildung 16: Darstellung des Regelfensters im Regelbaum.

Sekunden, dies ist die verwendete Zeiteinheit. Ein Längenfenster wird wie bereits erwähnt als Anzahl von Ereignissen formuliert. *Beispiel:* ist die Fenstergröße (Wert) gleich 3, dann bemisst sich im Falle eines Zeitfensters, dieses auf drei Sekunden. Ein entsprechendes Längenfenster betrachtet stets drei Ereignisse in Folge.

5.1.3 Zugriffe auf Knoten im Regelbaum

Die Darstellung einer CEP-Regel als Baum ermöglicht einen einfachen Zugriff auf jeden ihrer Knoten. Wesentlich ist dabei die Möglichkeit den Baum effizient zu durchlaufen (traversieren) und die für BatCEP verwendete Engine durchläuft den Baum in *pre-order* [9]. Alle in den nachfolgenden Abschnitten beschriebenen Operationen basieren darauf. Das bedeutet sie durchlaufen den Baum in pre-order bis zum Erreichen des operationsabhängigen Zielknotens und manipulieren diesen direkt.

5.2 Konkrete Anpassungsmaßnahmen des Bat-Algorithmus für BatCEP

Dieser Abschnitt beschäftigt sich mit der Kernkomponente von BatCEP: dem vom BA inspirierten Optimierungsalgorithmus mit seinen angepassten Aktualisierungsoperationen.

[9] `https://en.wikipedia.org/wiki/Tree_traversal#Pre-order_2` (zuletzt zugegriffen am 21. August 2017)

Dazu werden Ansätze aus dem vorherigen Kapitel 4 aufgegriffen. Es beginnt zunächst mit der konkreten Abbildungen der Fledermaus-Attribute und dem Initialisierungsprozesses, der den ersten Fledermaus-Schwarm erzeugt.

5.2.1 Attribute

Im Kapitel 4.2 wurden bereits Möglichkeiten aufgezeigt, um die Fledermaus-Attribute so anzupassen, dass sie das Lernen von CEP-Regeln unterstützen. Einige dieser Möglichkeiten werden für BatCEP übernommen und sind nachfolgend detailliert beschrieben.

Position und Positionswechsel Die Position x repräsentiert eine CEP-Regel in der zuvor beschriebenen Baumform. Jeder einzelne ihrer Knoten kann individuell durch gezielte Operationen verändert werden. BatCEP wandelt den Zeitpunkt der Positionswechsel zusammen mit den Positionsevaluierungen im BA geringfügig ab. Anstelle die neue Position x_i^t im Zuge des Zufallsfluges direkt zu Beginn eines Zeitschrittes t aus der Frequenz f_i^t und der Geschwindigkeit v_i^t zu berechnen, verknüpft BatCEP den Positionswechsel für den Zufallsflug und für die lokale Suche direkt mit der Positionsevaluierung. Kurzgefasst: zu Beginn eines jeden Zeitschrittes t wird die Frequenz f_i^t und die Geschwindigkeit v_i^t berechnet. Dann startet die Fledermaus i gemäß des BA in einen Zufallsflug und gegebenenfalls auch in eine lokale Suche und im Zuge dessen wird ihre Position x_i^t auf Grundlage von f_i^t und v_i^t neu berechnet und jeweils auch direkt evaluiert. Dementsprechend sind die Aktualisierungsoperationen, die die Positionswechsel realisieren und auf Abschnitt 4.2.3 basieren, in den nachfolgenden Abschnitten 5.2.2 (Zufallsflug) und 5.2.3 (lokale Suche) definiert.

Frequenz und Geschwindigkeit Frequenz und Geschwindigkeit sind kausal so abgebildet, wie sie bereits in Abschnitt 4.2.4 beschreiben wurden. Demnach wird die Differenz Δ_v^f zwischen der Frequenz f_i^t und Geschwindigkeit v_i^t einer Fledermaus i im Zeitschritt t als Anzahl durchzuführender Aktualisierungsoperationen an der Position x_i^{t-1} interpretiert.

Dazu wird die Frequenz mit der originalen Gleichung 3 aktualisiert. Die Gleichung der Geschwindigkeit wird wie in Abschnitt 4.2.4 beschrieben angepasst und in der nachfolgenden Gleichung 9 noch ein Mal aufgegriffen:

$$v_i^t = v_i^{t-1} + (F_{opt}(x_i^{t-1}) - F_{opt}(x^*))f_i^t. \tag{9}$$

Pulsrate und Lautstärke Pulsrate und Lautstärke behalten ihre Bedeutungen als Indikatoren für die Qualität von Positionen und damit CEP-Regeln bei. Die Pulsrate wird in ihrer Funktion nicht verändert und lenkt nach wie vor das Verhältnis zwischen Erkundung und Ausbeutung durch den Vergleich mit einer Zufallsvariable aus ihrem Wertebereich [0,1]. BatCEP aktualisiert sie mit der originalen Gleichung 5 des BA aus dem Abschnitt 2.3. Die Aktualisierung der Lautstärke geschieht ebenfalls mit der originalen Gleichung 6 und sie ist nach wie vor ein Akzeptanzkriterium für neue Lösungskandidaten. Dazu wird sie weiterhin mit einer Zufallsvariable aus dem Wertebereich $[0,a^0]$ verglichen.

Die Wirkung der Durchschnittslautstärke a^t pro Zeitschritt t wird geringfügig und gemäß der Beschreibung aus Abschnitt 4.2.5 für eine Neuinterpretation der Gleichung 8 angepasst: sie beeinflusst die Aktualisierung einer neuen Position in der lokalen Suche, indem sie die Anzahl der durchzuführenden Aktualisierungsoperationen vorgibt. Ein Beispiel hierzu: ist a^4 gleich 3,5 und ϵ gleich 0,9 so wird die Position x_i^4 von jeder Fledermaus i, die im Zeitschritt 4 eine lokale Suche durchführt $\lfloor 3,5 * 0,9 \rfloor$ also *drei* mal mit den Operationen der lokalen Suche aktualisiert. Diese Anpassung geht ebenfalls darauf zurück, dass x kein skalarer Wert, sondern seine CEP-Regel ist. Die Operationen der lokalen Suche werden später konkret beschrieben.

5.2.2 Der Zufallsflug

BatCEP interpretiert den Zufallsflug des BA neu und unterscheidet ihn in zwei Arten von Flügen: den *rein zufälligen Flug* und den *bedingt zufälligen Flug*, der einer Optimierung entspricht. Welcher der beiden Flüge von einer Fledermaus i durchgeführt wird, hängt maßgeblich von der Qualität ihrer Position x_i^t, also der Fitness der korrespondierenden CEP-Regel $F_{opt}(x_i^t)$ ab. Ist ihre Fitness besser als die durchschnittliche Fitness pro Zeitschritt $F_{opt}(x)^t$ über alle Fledermäuse, dann entscheidet sich die Fledermaus für den bedingt zufälligen Flug, womit sie ihre Position zwar verbessern aber nicht verschlechtern kann. Ist ihre Fitness hingegen schlechter, dann führt sie einen rein zufälligen Flug durch, im Zuge dessen sie ihre Position auch verschlechtern kann. Algorithmus 5 stellt den BatCEP-spezifischen Zufallsflug in Form von Pseudocode dar.

Algorithmus 5 Zufallsflug

```
1: procedure ZUFALLSFLUG
2:     if F_opt(x_i) > F_opt(x)^t then
3:         bedingt zufälliger Flug;
4:     else
5:         rein zufälliger Flug;
6:     end if
7: end procedure
```

Die Entscheidung für einen zweigeteilten Zufallsflug für BatCEP begründet sich mit der Komplexität von CEP-Regeln; mit der Tatsache dass eine einzige CEP-Regel an verschiedenen Punkten aktualisiert werden kann und dem Fakt, dass kurze CEP-Regeln für den Benutzer aussagekräftiger sind als lange. Kurzum: manchmal werden CEP-Regeln auch im Zufallsflug bewusst hinsichtlich ihrer Qualität *verbessert*. Der rein zufällige Flug verknüpft den zufälligen Positionswechsel mit der Evaluierung der neuen Position. Beim bedingt zufälligen Flug geschieht das gleiche, jedoch mit anderen Operationen für den Positionswechsel. Beide Flüge sind nachfolgend beschrieben.

Bedingt zufälliger Flug (Optimierung) Ist die Fitness einer Position x_i^t, besser als die durchschnittliche Fitness über alle Positionen im aktuellen Zeitschritt, dann scheint die

Fledermaus i „auf dem richtigen Weg“ zu sein und optimiert ihren Flug weiter. Dazu führt sie im bedingt zufälligen Flug die nachfolgend beschrieben Aktualisierungsoperationen durch. Dabei gilt immer die Bedingung: Aktualisierungen werden nur dann tatsächlich übernommen, wenn sie zu einer Verbesserung der Position führen.

Explizite Fenster-Aktualisierung mit einer Wahrscheinlichkeit von 0,4 ändert sie Typ und Wert ihres Regelfensters. Der Wert wird dabei zufällig aus der Längen- oder Zeit-Differenz des ersten und des letzten Ereignisses im Arbeitsdatensatz ausgewählt. Ein Beispiel hierzu: per Zufall wurde bestimmt, dass die Regel ein Längenfenster bekommt und der Datensatz beinhaltet 1000 Ereignisse. Die Größe des neuen Fensters wird dann zufällig aus dem Bereich $[0, 1000]$ ausgewählt.

Mit einer Wahrscheinlichkeit von 0,6 führt sie eine *Reduktion ihres aktuellen Fensters* durch. Dabei dient ihr aktueller Fenster-Wert als obere und der geringste Fenster-Wert als untere Grenze für den neuen Wert. Dieser wird zufällig daraus ausgewählt. Der Fenster-Typ ändert sich dabei nicht. Bezogen auf das Beispiel zuvor wäre hierbei die obere Grenze nicht 1000, sondern entspräche der aktuellen Fenstergröße von x_i^t.

Explizite ECT-Aktualisierung ein zufällig ausgewählter Knoten aus dem ECT wird zufällig durch einen neuen Knoten ersetzt. Der neue Knoten ist entweder ein einzelnes neues Ereignis oder ein Ereignis-Operator, mit zufällig gewählten Ereignissen als Operanden. Um dies zu realisieren traversiert BatCEP den ECT von x_i^t und stoppt an einem zufällig ausgewählten Knoten. Dieser wird ersetzt und dann gegebenenfalls auch der ACT entsprechend angepasst. BatCEP kennt alle im Datensatz vorkommenden Ereignistypen aus einem initial erzeugten Ereignismodell.

Explizite Komplexitätsverringerung mit einer Wahrscheinlichkeit von 0,4 wird eine ECT-Aktualisierung wie aus dem vorherigen Punkt durchgeführt und mit einer Wahrscheinlichkeit von 0,6 eine Reduktion der ECT-Komplexität. Realisiert wird dies durch den Tausch eines Operators durch ein einzelnes, zufällig ausgewähltes Ereignis. Einfach formuliert: ein Operator mit zwei Operanden (insgesamt drei Knoten) wird aus dem Regelbaum entfernt und durch ein primitives Ereignis (ein Knoten) ersetzt. Dies verringert die Größe der Regelbedingung und damit einhergehend auch die Regel-Komplexität.

Algorithmus 6 Bedingt zufälliger Flug

```
1: procedure OPTIMIERUNG
2:     temp ← x_i^t;
3:     while v_i^t < f_i^t do
4:         explizite Fenster-Aktualisierung an temp;
5:         if F_opt(temp) > F_opt(x_i^t) then
6:             x_i^t ← temp;
7:         end if
8:         explizite ECT-Aktualisierung an temp;
9:         if F_opt(temp) > F_opt(x_i^t) then
10:            x_i^t ← temp;
11:        end if
12:        exlpizite Komplexitätsaktualisierung an temp;
13:        if F_opt(temp) > F_opt(x_i^t) then
14:            x_i^t ← temp;
15:        end if
16:        inkrementiere v_i^t;
17:    end while
18: end procedure
```

Die erhöhte Tendenz zur Durchführung von Fenster- und Komplexitätsreduktionen basieren auf der Annahme, dass es sich dabei um sehr gezielte Operationen handelt. Diese sollten verhältnismäßig häufiger durchgeführt werden.

Alle drei Operationen werden wie zuvor schon erwähnt, in einer Schleife über die Differenz Δ_v^f durchgeführt. Die Schleife startet mit dem Wert der Geschwindigkeit v_i^t der Fledermaus i. Sie inkrementiert v_i^t in jeder Iteration und stoppt, wenn v_i^t den Wert der Frequenz f_i^t erreicht hat. Algorithmus 6 beschreibt den bedingt zufälligen Flug als Pseudocode.

Rein zufälliger Flug Der rein zufällige Flug stellt keine Bedingung an die Aktualisierungen. Eine strikte Verbesserung ist demnach nicht vorgegeben, sodass die Fledermaus ihre Position auch verschlechtern kann. Dennoch ist der rein zufällige Flug zielführend, weil die Position der Beute (der perfekten CEP-Regel beziehungsweise der tatsächlichen Ursache von Z) im Suchraum nicht bekannt ist. Somit kann der Flug zu einer zunächst schlechteren Position längerfristig betrachtet wieder zu einem guten Ergebnis führen.

Genau wie bei dem bedingt zufälligen Flug werden auch die hier durchgeführten Operationen in der Schleife von v_i^t bis f_i^t durchgeführt. Zudem gibt es noch eine Entwurfsentscheidung: unter Einbeziehung der Lautstärke a_i^t der Fledermaus i im aktuellen Zeitschritt t, wird hier entweder eine Aktualisierung des ECT, des ACT oder des Regelfensters durchgeführt. Insgesamt werden die hier durchführten Operationen wie folgt definiert:

ECT- oder Fenster-Aktualisierung wenn die Lautstärke a_i^t größer ist als eine Zufallsvariable aus dem Wertebereich $[0, a^0]$, dann wird eine einfache ECT- oder Fenster-Aktualisierung durchgeführt. Diese wählt zunächst zufällig einen Knoten aus dem

ECT aus. Handelt es sich dabei um einen inneren oder einen Blattnoten, dann wird dieser genau wie bei der expliziten ECT-Aktualisierung des bedingt zufälligen Fluges durch ein Ereignis oder einen Ereignis-Operator ausgetauscht. Handelt es sich jedoch um die Wurzel des ECT, so wird stattdessen das Regelfenster mit Typ und Wert zufällig ausgetauscht und der ECT bleibt unverändert.

ACT-Aktualisierung wenn a_i^t kleiner ist als die Zufallsvariable, dann bedingt dies eine Aktualisierung des ACT. Dabei werden zufällig ausgewählte Knoten des ACT durch andere ersetzt. So wird beispielsweise `a.temp = b.temp` aktualisiert zu `a.druck > b.temp`. Auch hierbei gilt jedoch, dass nur tatsächlich vorhandene Attribute oder Konstanten eingesetzt werden können.

Algorithmus 7 repräsentiert den rein zufälligen Flug als Pseudocode.

Algorithmus 7 Rein zufälliger Flug

```
1: procedure AKTUALISIERUNG
2:     while v_i^t < f_i^t do
3:         if ρ < a_i^t then                          ▷ ρ ∈ [0, a^0]
4:             ECT-Aktualisierung an x_i^t;
5:         else
6:             ACT-Aktualisierung an x_i^t;
7:         end if
8:         inkrementiere v_i^t;
9:     end while
10: end procedure
```

Dabei ist ρ die Zufallsvariable aus dem Wertebereich $[0, a^0]$.

5.2.3 Die lokale Suche

Die Wahrscheinlichkeit dass eine lokale Suche durchgeführt wird, bestimmt sich für BatCEP genauso wie für den originalen BA, mit der Pulsrate. Dazu wird die Pulsrate mit einer Zufallsvariable aus ihrem Wertebereich $[0, 1]$ verglichen und nur wenn diese größer ist als die Pulsrate, wird die lokale Suche mit ihren nachfolgend beschriebenen Operationen durchgeführt.

Explizite Komplexitätsverringerung genau wie in Abschnitt 5.2.2.

ACT-Aktualisierung genau wie in Abschnitt 5.2.2.

Fenster-Optimierung mit einer Wahrscheinlichkeit von 0,7 wird eine „Radius-Optimierung“ durchgeführt, die speziell für BatCEP entwickelt wurde. Dabei handelt es sich um eine probabilistische und dennoch gezielte Verkleinerung des Regelfensters in Abhängigkeit von der Regel-Fitness. Ihre Funktionsweise ist wie folgt definiert: zuerst wird der Radius-Faktor (RF) bestimmt, der die verhältnismäßige Größe des Radius vorgibt:

$$RF = \frac{Fitness_{max} - Fitness_R}{Fitness_{max} - Fitness_{min}} * 0,35.$$

Wobei $Fitness_{max}$ der maximal erreichbare Wert der verwendeten Fitness-Funktion und $Fitness_R$ die aktuelle Fitness der zu optimierenden CEP-Regel ist. Für $Fitness_{min}$ wird hier 0 angenommen. Der Nenner sorgt dann für die Abbildung des Ergebnisses auf den Wertebereich $[0,1]$. Der Wert $0,35$ ist ein vordefinierter Maximalwert des Radius-Faktors, der nur dann erreicht wird, wenn der Fitness-Wert der CEP-Regel gleich 0 ist. Eine bessere Fitness verkleinert den Wert, denn es gilt: je schlechter die Fitness der CEP-Regel, desto größer soll der Radius bemessen werden, weil „sich die Beute offenbar nicht in der näheren Umgebung befindet“. Je besser die Fitness ist, desto „näher befindet sich die Fledermaus an der Beute“ und umso kleiner kann der Radius bemessen werden.

Im nächsten Schritt wird der Radius mithilfe des Radius-Faktors aus der Differenz Δ_e^l zwischen dem ersten und dem letzten Ereignis des Trainingsdatensatz berechnet. Dabei bezieht sich die Differenz auf den Fenster-Typ der CEP-Regel und bemisst sich entweder in Zeit (Sekunden) oder Länge (Anzahl Ereignisse):

$$Radius = \Delta_e^l * RF$$

Anschließend wird mit dem Radius sowohl die untere Grenze (UG) als auch die obere Grenze (OG) für den Wertebereich des neuen Fensters bestimmt:

$$UG = L - Radius;\ OG = L + Radius$$

Wobei L die aktuelle Größe des Fensters ist. Damit die Grenzen im gültigen Bereich zwischen dem ersten und dem letzten Ereignis bleiben, werden sie darauf beschränkt. So ist der kleinste Wert für UG eines Längenfensters gleich 0 und der größte Wert für OG gleich dem Wert des letzten Ereignisses. Bei einem Zeitfenster wird UG auf den Zeitstempel des ersten Ereignisses im Datensatz gesetzt und OG auf den Zeitstempel des letzten Ereignisses im Datensatz. Zuletzt wird aus [UG, OG] zufällig ein neuer Wert für die Fenstergröße ausgewählt.

Für ein besseres Verständnis sei noch die folgende Ergänzung gegeben: die Radius-Operation legt einen imaginären Radius um die aktuelle Fenstergröße einer CEP-Regel. Dieser Radius ist bei einer sehr schlechten CEP-Regel (Fitness=0) maximal groß ($\Delta_e^l * 0,35$) und bei einer sehr guten Regel (Fitness=1) maximal klein (0). Abbildung 17 illustriert dies an einem Beispiel: gegeben ist eine CEP-Regel mit einer

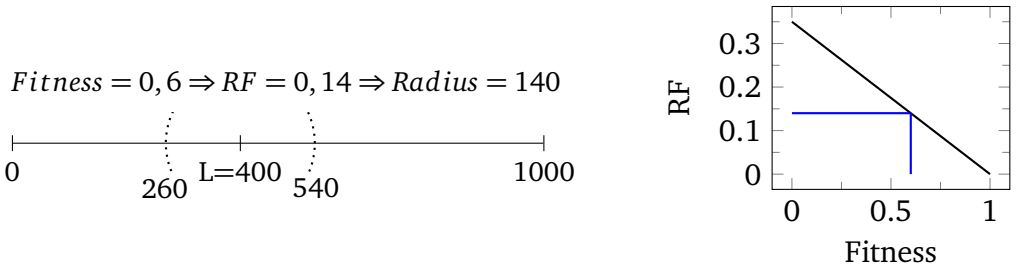


Abbildung 17: Veranschaulichung der Radius-Aktualisierung an einem Längenfenster.

Fitness von 0,6 aus dem Wertebereich [0,1] und einer aktuellen Fenstergröße von $L = 400$. Der Datensatz beinhaltet 1000 Ereignisse. Aus der Fitness ergibt sich ein Radius-Faktor RF von 0,14 und dieser wird mit der Differenz zwischen dem ersten und dem letzten Ereignis im Datensatz $\Delta_e^l = 1000$ multipliziert. Daraus resultiert der Radius von 140, der um die aktuelle Fenstergröße L gelegt wird. Folglich ergibt sich als untere Grenze UG der Wert 260 und als obere Grenze OG 540. Die neue Fenstergröße wird zufällig aus dem Bereich zwischen UG und OG ausgewählt.

Die zweite mögliche Fenster-Operation in der lokalen Suche ist die bereits in Abschnitt 5.2.2 beschriebene *explizite Fenster-Aktualisierung*. Sie wird nur mit einer Wahrscheinlichkeit von 0,3 ausgewählt, weil ihre Ergebnisse weniger präzise sind, als die der Radius-Aktualisierung. Dennoch hat sie ihre Daseinsberechtigung, weil die Radius-Aktualisierung nicht perfekt ist, sodass die korrekte Fenstergröße in seltenen Fällen außerhalb des Radius liegt. Die explizite Fenster-Aktualisierung korrigiert diesen Fehler, indem sie einen vom Radius unabhängigen Zufallswert aus dem Bereich Δ_e^l auswählt.

Im Zuge einiger Tests hat sich ergeben, dass sich der ECT von CEP-Regeln mit einer verhältnismäßig guten Fitness, zumeist aus den korrekten Ereignistypen und -Operatoren zusammensetzt. Wobei sich „korrekt“ auf einen Vergleich mit der perfekten CEP-Regel ($R*$) bezieht, mit der der Trainingsdatensatz erzeugt worden ist. Folglich sind am Ende falsche Attribute oder Operatoren im ACT und falsche (meist zu große) Fenstergrößen das Problem. Aus diesem Grund wurden die hier aufgeführten Operationen für die lokale Suche ausgewählt.

Weiterhin arbeitet die lokale Suche mit einer Schleife, deren Anzahl der Durchläufe in etwa proportional zur durchschnittlichen Lautstärke pro Zeitschritt (a^t) ist. Dieser Ansatz stellt zugleich die Abbildung der originalen Gleichung 8 für die lokale Suche auf BatCEP dar. Algorithmus 8 zeigt die lokale Suche als Pseudocode.

Algorithmus 8 Lokale Suche

```
1: procedure LokaleSuche(x_i^t, x*)
2:     temp ← x*; j ← 0;
3:     while j < ⌊a^t ϵ⌋ do                ▷ Durchschnittslautstärke im Zeitschritt t
4:         switch η do                      ▷ η ∈ {1, 2, 3}
5:             case 1
6:                 explizite Komplexitätsverringerung an temp;
7:             case 2
8:                 ACT-Aktualisierung an temp;
9:             case 3
10:                if σ < 0,7 then          ▷ σ ∈ [0, 1]
11:                    Fenster-Radius-Optimierung an temp;
12:                else
13:                    explizite Fenster-Aktualisierung an temp;
14:                end if
15:        inkrementiere j;
16:    end while
17:    if temp besser als x* then
18:        x_i^t ← temp;
19:    end if
20: end procedure
```

Dabei ist η eine gleich verteilte Zufallsvariable, die eine der Aktualisierungsoperationen auswählt und σ eine Zufallsvariable aus dem Wertebereich $[0, 1]$. Gelingt es damit die gute Position x^* (beziehungsweise eine temporäre Kopie dessen) noch weiter zu verbessern, dann ersetzt sie die schlechte Position der Fledermaus, die die lokale Suche initiiert hat. Ist jedoch keine Verbesserung der guten Position erfolgt, dann behält die Fledermaus ihre Lösung x_i^t unverändert bei.

Abbildung 18 zeigt den von BatCEP modifizierten Durchführungsablauf des Zufallsfluges und der lokalen Suche. Die wiederholte Positionsevaluierung resultiert aus den Anpassungen der Frequenz, der Geschwindigkeit und der Durchschnittslautstärke als Schleifenkriterien.

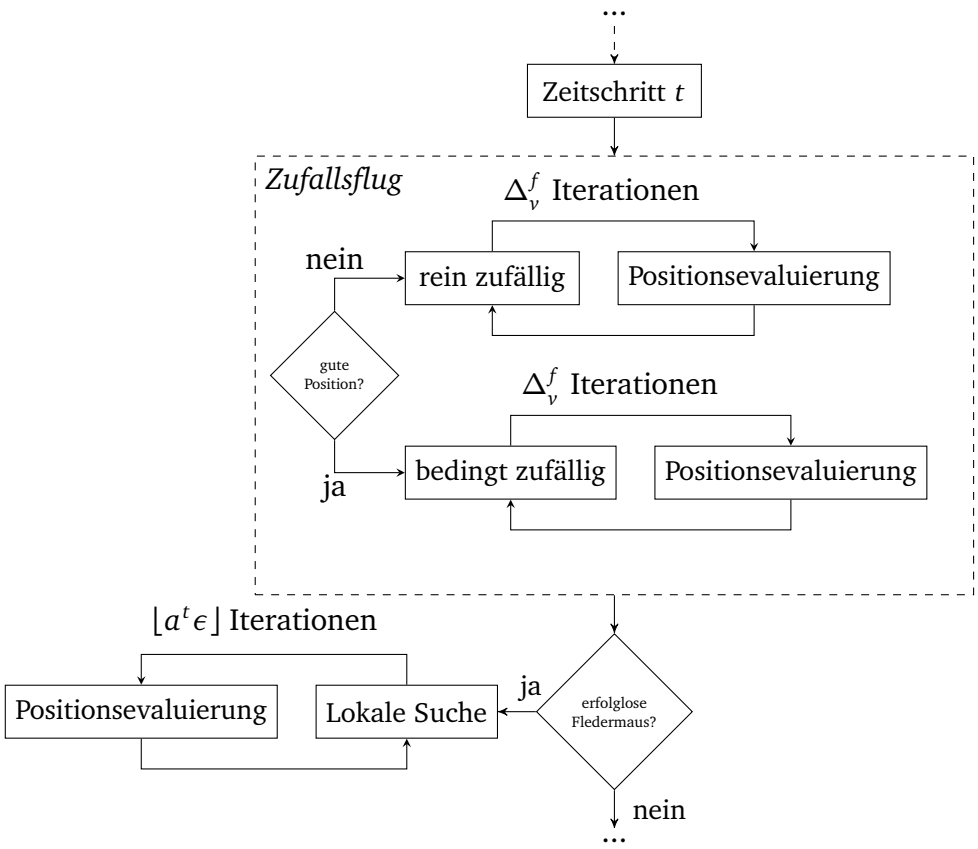


Abbildung 18: BatCEP: modifizierter Durchführungsablauf des Zufallsfluges und der lokalen Suche.

5.2.4 Modifikation von Pulsrate und Lautstärke

Genau wie beim BA finden auch bei BatCEP Aktualisierungen von Pulsrate und Lautstärke statt. Immer dann, wenn eine Fledermaus ihre Lösung verbessern konnte und diese als neues globales Maximum akzeptiert wurde, wird ihre Pulsrate r_i^t erhöht und ihre Lautstärke a_i^t verringert. Dies geschieht mit den originalen Gleichungen 5 und 6, die hier erneut aufgegriffen werden:

$$r_i^{t+1} = r_i^0[1 - exp(-\gamma t)]$$
$$a_i^{t+1} = \alpha a_i^t.$$

Diese Modifikationen ziehen folgende Konsequenzen mit sich:

- Die Wahrscheinlichkeit für die Durchführung einer lokalen Suche verringert sich für die Fledermaus. Das ist zielführend, denn sie hat ja bereits eine gute Position.
- Neue von ihr erzeugte Lösungen werden mit verringerter Wahrscheinlichkeit als globales Maximum akzeptiert. Darin spiegelt sich das für den BA typische Verhältnis zischen Erkundung und Ausbeutung wider, denn zugleich erhöht sich damit die Wahrscheinlichkeit dafür, dass die Lösungen anderer Fledermäuse akzeptiert werden.

- Sie senkt die Durchschnittslautstärke ab, wodurch weniger Aktualisierungsoperationen in der lokalen Suche durchgeführt werden. Zugleich führt sie mit ihrer guten (global besten) Lösung schlechtere Fledermäuse, die ihrerseits eine lokale Suche durchführen, an eine gute Position. Dort sind für weitere Verbesserungen weniger Aktualisierungsoperationen notwendig, als an anderen Positionen.
- Während des Zufallsfluges führt sie als *gute* Fledermaus zunehmend mehr ACT- als ECT-Aktualisierungen durch. Das ist zielführend, denn wie bereits in vorherigen Abschnitten erwähnt, müssen bei bereits guten CEP-Regeln eher die Kontextbedingungen als Ereignismuster verbessert werden.

Die ersten beiden Punkte beschreiben dasselbe Verhalten wie auch beim originalen BA. Die letzten beiden Punkte hingegen sind für BatCEP angepasst und durch die Verarbeitung von CEP-Regeln anstelle skalarer Variablen geprägt. Insgesamt resultiert jedoch dasselbe Verhalten und dasselbe Verhältnis zwischen Erkundung und Ausbeutung wie beim BA.

5.2.5 Definition der Fitness-Funktion

BatCEP misst die Qualität seiner CEP-Regeln mithilfe der Verhältnisse *Recall* und *Precision*. Der Recall setzt den Anteil der durch die CEP-Regel gefundenen Ereignisse mit der Anzahl aller Ereignisse im Trainingsdatensatz in Beziehung und beantwortet damit die Frage: „Wieviele Vorkommen von Z aus allen Vorkommen von Z im Trainingsdatensatz hat die CEP-Regel gefunden?“ Formal ist er definiert als:

$$Recall = \frac{TP}{TP + FN}.$$

Die Precision hingegen beantwortet die Frage: „Wie häufig hat die CEP-Regel genau dann gefeuert, wenn Z das nächste Ereignis ist? Wie häufig hat sie also genau im richtigen Moment gefeuert?“. Die Definition der Precision ist:

$$Precision = \frac{TP}{TP + FP}.$$

Um CEP-Regeln mithilfe von Recall und Precision zu messen, ist es zwingend erforderlich, beide in Kombination miteinander zu betrachten. Weder der Recall noch die Precision kann allein als Qualitätsmaß für CEP-Regeln herangezogen werden, was durch das nachfolgende Beispiel gezeigt werden soll:

CEP-Regel:

```
(A AS a ∨ C AS c) ∧
(a.wert < c.wert)[win:len:4] ⇒ Z
```

Datensatz:

```
2017-07-01 09:03:20 A; wert=1 !
2017-07-01 09:03:23 C; wert=2 !
Z
2017-07-01 09:03:25 A; wert=1 !
2017-07-01 09:03:45 B; wert=1
Z
2017-07-01 09:03:48 B; wert=7
2017-07-01 09:03:53 C; wert=4 !
Z
```

Informationen:

- TP: 2 / TN: 1
- FP: 2 / FN: 1

Alle Stellen an denen die CEP-Regel feuert, sind zur Vereinfachung mit dem Ausrufezeichen (!) markiert. Es ist zu erkennen, dass die Regel an zwei von drei Stellen feuert, an denen Z das nächste Ereignis ist und daraus ergibt sich ein Recall von 67%:

$$Recall = \frac{2}{2+1} = 0,67 = 67\%.$$

Würde der Recall allein die Fitness der CEP-Regel beschreiben, so erscheint dies zunächst sinnvoll, denn die 67% beschreiben genau das Verhältnis zwischen gefundenen und nicht gefundenen Z. Zugleich werden damit jedoch die beiden Fälle ignoriert, in denen die Regel fälschlicherweise gefeuert hat. Dieses Problem lässt sich am besten mit einem Extremfall beschreiben: würde die CEP-Regel auf diesem Datensatz sechs Mal feuern, also bei jedem primitiven Ereignis, dann hat dies einen Recall von 100% zur Folge. Die CEP-Regel scheint demnach besonders gut zu sein, obwohl sie faktisch besonders schlecht ist, denn sie feuert einfach immer – drei mal wenn sie nicht feuern sollte und drei mal wenn sie feuern sollte.

Als nächstes sei die Precision genauer betrachtet. Sie liegt im obigen Beispiel bei 50%:

$$Precision = \frac{2}{2+2} = 0,5 = 50\%.$$

Das bedeutet, dass die CEP-Regel in der Hälfte aller Fälle in denen sie feuert, korrekt feuert und diese Aussage erscheint zunächst sinnvoll. Bei genauerer Betrachtung fällt jedoch auf, dass die Precision den Fall ignoriert, in dem die Regel fälschlicherweise nicht feuert (nach dem ersten B-Ereignis). Wird die Fitness einer CEP-Regel also allein durch die Precision gemessen, so fallen alle Fälle in denen die Regel feuern sollte dies aber nicht tut, aus der Wertung und das ist nicht zielführend. Erst die Kombination aus Recall und Precision ergeben ein repräsentatives Maß für die Fitness.

BatCEP kombiniert Recall und Precision mit der sogenannten F1Score, die als

$$F1Score = 2 * \frac{Recall * Precision}{Recall + Precision}$$

definiert ist. Damit handelt es sich bei der F1Score um das harmonische Mittel zwischen Recall und Precision. Für das Beispiel würde sich demnach eine F1Score von 57% ergeben.

Eine F1Score von 100% kann nur dadurch erreicht werden, dass die CEP-Regel *ausschließlich an den korrekten Stellen* feuert und zugleich *alle Vorkommen von Z* findet – also nur dann wenn beide, der Recall und die Precision zu 100% erfüllt sind. Dann werden die Ursachenquellen für *Z* von der Regel-Bedingung korrekt repräsentiert.

Obwohl es mit Recall und Precision im Grunde zwei Objektive Funktionen gibt, die zunächst gegensätzlich erscheinen, handelt es sich nicht um ein Pareto-Optimum. Zur Erinnerung: das Pareto-Optimum ist ein Zustand in dem eine Eigenschaft nur verbessert werden kann, wenn dafür eine andere Eigenschaft verschlechtert wird. Auch wenn sich durch eine Verbesserung des Recalls die Precision verschlechtert (oder andersherum), ist es dennoch möglich beide zusammen auf 100% zu optimieren – mit der perfekten CEP-Regel, die immer nur im richtigen Moment feuert.

5.2.6 Initialisieren des ersten Fledermaus-Schwarmes

Ausgangspunkt für die Erzeugung des ersten Schwarmes ist das aus dem Trainingsdatensatz erzeugte Ereignismodell. Auf diesem als Grundlage, wird der erste Schwarm in Form von Regelbäumen initialisiert. Die Fitness dieses Schwarmes kann entscheidenden Einfluss auf die Effizienz des gesamten Optimierungsverfahrens haben. Grund hierfür ist, dass ein generell schlecht positionierter Schwarm wesentlich häufiger optimiert werden muss, als einer in dem einige Fledermäuse bereits gut positioniert sind. Darüber hinaus kann es passieren, dass sich das Optimierungsverfahren mit einem schlechten ersten Schwarm mehr und mehr von einer guten Lösung entfernt und letztlich gar keine akzeptable Lösung hervorbringt und demnach nicht konvergiert.

Die Wichtigkeit einer guten Verteilung der Fledermäuse auf den Suchraum ist demnach nicht zu unterschätzen. Für die Initialisierung stehen verschiedene Techniken zur Verfügung, von denen einige in [33] übersichtlich dargelegt sind. Grundsätzlich lassen sie sich in drei nicht klar getrennte Kategorien einteilen: *Zufallsorientiert*, *Kombiniert* und *Generalisierend*.

Die *zufallsorientierte Initialisierung* generiert einen Schwarm mithilfe stochastischer aber auch deterministischer Verfahren. Individuen dieses Schwarmes werden beispielsweise durch Zufallsgeneratoren (PRNG), Generatoren basierend auf chaotischen Systemen (CNG) oder Quasi-Randomisierung erzeugt. Quasi-Randomisierung ist ein deterministisches Verfahren, das folglich nicht als wirklich zufällig bezeichnet werden kann. Wenn jedoch nur wenig Vorwissen über das zu lösende Problem vorhanden ist, kann Quasi-Randomisierung vollkommen ausreichend sein – andersherum: echter Zufall verteilt den Schwarm dann nicht zwingend besser im Suchraum.

In die Kategorie *Kombiniert* fallen genau die Initialisierungstechniken, die sich aus mehreren Techniken zusammensetzen, wobei man hierbei zwei Arbeitsweisen unterscheidet: wenn jeder Teilschritt auch einzeln eine komplette Initialisierung durchführen kann (Beispiele hierfür sind PRNG oder CNG), dann wird die Technik als *hybrid* bezeichnet. Hybride Techniken sind parallelisierbar und vereinen die Vor- und Nachteile ihrer in den Teilschritten angewandten Techniken. Als *mehrstufig* bezeichnet man hingegen Techniken, deren Teilschritte kausal von einander abhängen. Eine häufig verwendete, mehrstufige Initialisie-

rungstechnik ist das sogenannte „Opposite-based learning“ (OBL) – frei übersetzt: „gegensätzliches Lernen“. Dabei wird zuerst ein (auf dem Zufall basierender) Schwarm $S(1, ..., n)$ erzeugt. Im Anschluss daran wird jedes Individuum i mit einer Fitness-Funktion evaluiert. Für alle i deren Fitness-Wert unterhalb einer akzeptierten Grenze liegt, wird „der gegensätzliche Wert“ berechnet [67, 33]. Diesem Vorgehen liegt die Annahme zugrunde, dass sich eine durch den Zufall schlecht erzeugte Lösung, einfach auf der „gegensätzlichen“ Seite des Suchraumes befinden kann. Mehrstufige Techniken erzeugen in der Regel gute Ergebnisse, weil sie eine Fitness-Funktion mit einbeziehen und damit schon während der Erzeugung die Qualität messen.

Generalisierende Initialisierungstechniken teilen sich in zwei Klassen auf: *generisch* und *applikationsspezifisch*. Generische Techniken können für beliebige Optimierungsprobleme eingesetzt werden. Alle in den vorherigen Abschnitten genannten Techniken fallen hierunter. Im Gegensatz dazu sind applikationsspezifische Techniken auf eine bestimmte Domäne ausgelegt und dahingehend optimiert. Sie liefern für ihre bestimmten Zwecke gute Ergebnisse, können aber auf andere Probleme gegebenenfalls nicht korrekt abgebildet werden oder arbeiten dort ineffizient.

BatCEP verwendet die sogenannte „Ramped half-and-half-Initialization“-Methode (RHH), die ihren Ursprung im Bereich der evolutionären Algorithmen, insbesondere der genetischen Programmierung hat [38] und die auch schon für CepGP eingesetzt wurde. RHH ist ein hybrider und generischer Ansatz, bestehend aus zwei Teilschritten: der *Full-Initialization*-Methode und der *Grow-Initialization*-Methode. Beide erzeugen mittels unterschiedlicher Verfahren zufällige CEP-Regelbäume als Positionen und beide werden in den kommenden Abschnitten tiefer gehend erläutert. BatCEP erweitert RHH zudem um die Initialisierung der anderen Fledermaus-Attribute: Frequenz, Geschwindigkeit, Pulsrate und Lautstärke.

***Full-Initialization*-Methode** Alle zu erzeugenden Regelbäume – in diesem Fall der ECT und der ACT – werden bis zu einer vom Benutzer vorgegebenen Tiefe $T - 1$ aus *zufällig* gewählten Operator-Knoten zusammengesetzt. Sobald die Tiefe erreicht wurde, folgen nur noch Ereignisse beziehungsweise Attribute. Abbildung 19 zeigt ein Beispiel für die Full-Initialization-Methode mit einer benutzerdefinierten Tiefe von $T = 2$. Der fertige Baum besteht aus dem Wurzelknoten (Ebene 0) und den beiden Ebenen 1 und 2. Verwendet werden die Operatoren *Sequenz* ($\rightarrow$), das logische *ODER* ($\vee$) und *UND* ($\wedge$), sowie die Ereignistypen A, B, C, D. Die verwendeten Operatoren stammen immer aus der Menge der für CEP möglichen Operatoren, die in Abschnitt 2.1.2 beschreiben sind.

***Grow-Initialization*-Methode** Entgegen der Full-Initialization-Methode ist bei der Grow-Initialization-Methode die Auswahl der Knoten bis zu der benutzerdefinierten Tiefe T nicht nur auf Operatoren beschränkt. Stattdessen stehen auch Ereignisse beziehungsweise Attribute zur Verfügung. Abbildung 20 zeigt ein Beispiel dazu. Zu sehen ist die Grow-Initialization-Methode mit einer benutzerdefinierten Tiefe von $T = 2$. Bereits in der Ebene 1 wird ein Ereignis (A) eingesetzt.

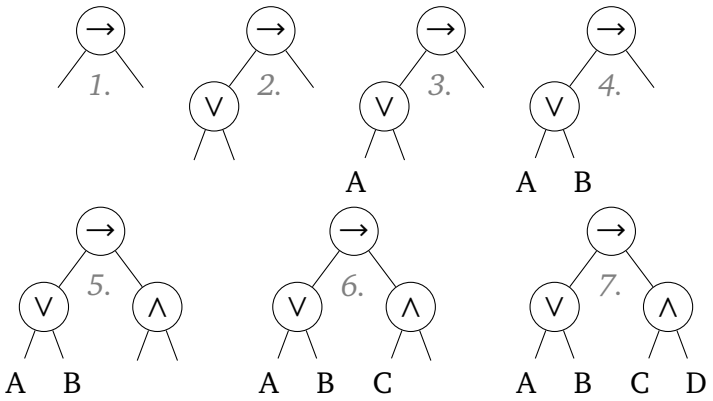


Abbildung 19: Ein Beispiel der Full-Initialization-Methode.

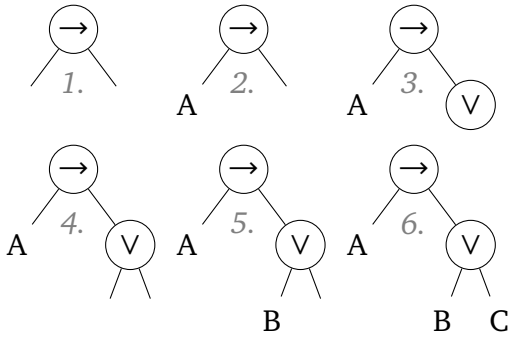


Abbildung 20: Ein analoges Beispiel zur Abbildung 19, hier jedoch unter Verwendung der Grow-Initialization-Methode.

Ramped half-and-half Mit der Kombination beider Methoden zu gleichen Anteilen wird eine Population erzeugt, deren Individuen sich zumeist sehr voneinander unterscheiden und das ist gut so. Deutlich wird dies am Beispiel eines Fledermaus-Schwarmes: unterschiedliche Fledermäuse starten im übertragenen Sinne von unterschiedlichen Positionen im Raum. Daraus folgt, dass sie sich aus verschiedenen, mehr oder weniger gut geeigneten Richtungen der Beute annähern – deren Position sie aber nicht kennen. Einige werden die Beute erreichen und andere nicht. Entscheidend ist, dass der Raum somit gut erkundet wird und letztlich zählt, dass die Beute von mindestens einer Fledermaus erreicht wird.

RHH bezieht keine Fitness-Funktion ein und daher hat der erzeugte Schwarm keine kontrollierbare Qualität. Im Abschnitt 7.3.2 wird eine Idee vorgestellt, mit dem sich die Qualität jedoch auf ein gewisses Minimum regulieren lassen könnte.

5.2.7 Zusammenfassung

Tabelle 1 zieht einen abschließenden Vergleich der Eigenschaften vom BA und von BatCEP und stellt diesen übersichtlich dar.

Tabelle 1: Vergleich der Eigenschaften, Arbeitsweisen und Attribute zwischen dem BA und BatCEP

	BA	**BatCEP**
Lösungsrepräsentation, Position	x ist ein skalarer Funktions-Parameter.	x ist eine CEP-Regel in Baumform.
Positionswechsel	Mittels arithmetischer Operationen.	Mit typischen Baum-Operationen wie Knoten tauschen, entfernen oder hinzufügen.
Zufallsflug	Eine zufallsbedingte Aktualisierung von x_i^t.	Die Differenz Δ_v^f dient als Schleifenkriterium und damit zugleich als Anzahl der Operationsdurchführungen am Regelbaum zur Aktualisierung von x_i^t $(v_i \rightarrow f_i^t)$.
Lokale Suche	x_i^t wird mit arithmetischen Operationen aus x^* neu berechnet.	a^t fungiert als Schleifenkriterium und bestimmt die Anzahl der Operationsdurchführungen am Regelbaum, mit denen x_i^t aus x^* berechnet wird $(0 \rightarrow \lfloor a^t \epsilon \rfloor)$.
Fitness-Funktion	kontinuierliche, problemspezifische Funktion.	F1Score

5.3 Der Einsatz mehrerer Fledermaus-Schwärme

Aufgrund der Tatsache, dass der BA keine äußeren Abhängigkeiten hat, ist er parallelisierbar. BatCEP nutzt diesen Vorteil aus und stellt mehrere Schwärme zur Verfügung, die zeitgleich nach Lösungen suchen. Dabei kommunizieren die Schwärme miteinander, indem sie ihre besten Lösungen austauschen. Heraguemi, Kamel und Drias stellen in [28] drei kooperative Strategien vor, auf denen die Kommunikation basieren kann.

Dieser Ansatz führt zusätzlich zum globalen Extremwert noch einen Schwarm-übergreifenden Extremwert ein. Damit nachfolgend keine Verwirrung um die Begrifflichkeiten der Extremwerte entsteht, werden diese zunächst neu definiert.

Repräsentation der Extremwerte unter Einsatz mehrerer Schwärme Im originalen BA repräsentiert jede Fledermaus i mit ihrer Position x_i eine Lösung: die jeweilige *lokale Lösung*. Die beste Lösung des ganzen Schwarmes stellt zugleich den *globalen Extremwert* x^* dar. Unter Verwendung mehrerer Schwärme gibt es jedoch noch einen dritten Extremwert und zwar die beste Lösung über alle Schwärme. Die lokale Lösung x_i behält ihre Bezeichnung weiterhin bei und die anderen beiden werden für alle nachfolgenden Abschnitte

definiert als:

- Die beste Lösung innerhalb eines Schwarmes ist ab sofort das ***Schwarm-lokale Maximum*** $\mathbf{x}^*$.
- Die beste Lösung über alle Schwärme ist ab sofort das ***Globale Maximum*** $\mathbf{x}^{**}$

Damit sind die Begriffe klar getrennt und es folgt die Beschreibung der drei kooperativen Strategien:

Ring Bei der *Ring-Strategie* werden alle Schwärme in einer Ring-Topologie angeordnet und jeder gibt seine beste Lösung x^* an den rechten oder linken Nachbarn weiter. Die Richtung wird anfangs vorgegeben. Ein solcher Austausch findet jedoch nur dann statt, wenn ein Schwarm für eine gewisse (vom Benutzer vorgegebene) Zeit keine Verbesserung hervorbringt. Das globale Maximum x^{**} ist während des Optimierungsprozesses nicht an einem festen Punkt zugreifbar, stattdessen ist es immerzu die beste der im Austausch befindlichen Lösungen.

Master-Slave Im Zuge der *Master-Slave-Strategie* gibt es einen Master-Schwarm und mehrere Slave-Schwärme, die folgendermaßen miteinander kommunizieren: jeder Slave-Schwarm startet seine Suche nach Lösungen und teilt dem Master-Schwarm seine beste Lösung x^* nach dem ersten Zeitschritt mit. Wenn dies von allen Slave-Schwärmen erledigt wurde, dann startet der Master-Schwarm ebenfalls eine Suche und bringt dabei die besten, ihm mitgeteilten Lösungen ein. Darunter befindet sich logischerweise auch das aktuelle, globale Maximum x^{**}. Seine Suche umfasst dabei auch nur einen Zeitschritt. Wenn er diesen durchlaufen hat, dann teilt er seine beste Lösung x^* und damit zugleich das neue globale Maximum x^{**} den Slave-Schwärmen mit, die damit in den nächsten Zeitschritt starten. Das ganze wiederholt sich für jeden Zeitschritt, bis das Verfahren konvergiert.

Hybrid Die *hybride Strategie* kombiniert Ring und Master-Slave miteinander: die Slave-Schwärme kommunizieren ihre Schwarm-lokalen besten Lösungen mit ihren Nachbarn, wenn dies erforderlich ist. Zudem informieren sie in jedem Zeitschritt den Master-Schwarm ebenfalls mit ihren neuen besten Lösungen und der berechnet das neue globale Maximum x^{**}, gibt dies aber nicht an die Slave-Schwärme weiter. Auch hierbei geschieht die Kommunikation einmal pro Zeitschritt.

Abbildung 21 illustriert alle drei Topologien. Durch Experimente in [28] hat sich herausgestellt, dass alle drei Strategien im Durchschnitt bessere Lösungen hervorbringen als ein nicht-paralleler Ansatz mit nur einem Schwarm. „Bessere Lösungen“ meint Lösungen mit einer höheren Fitness. Die Ring-Strategie hatte zumeist die kürzesten Laufzeiten, was darauf zurückzuführen ist, dass sie weniger Abhängigkeiten zwischen den Schwärmen hat als Master-Slave und Hybrid. Im Vergleich der maximalen Fitness-Werte (also dem Durchschnitt über alle globalen Maxima und Testläufe pro Strategie) auf verschiedenen Datensätzen kamen alle Strategien in etwa auf gleiche Ergebnisse. Selbiges gilt auch für die Anzahl erzeugter Lösungen.

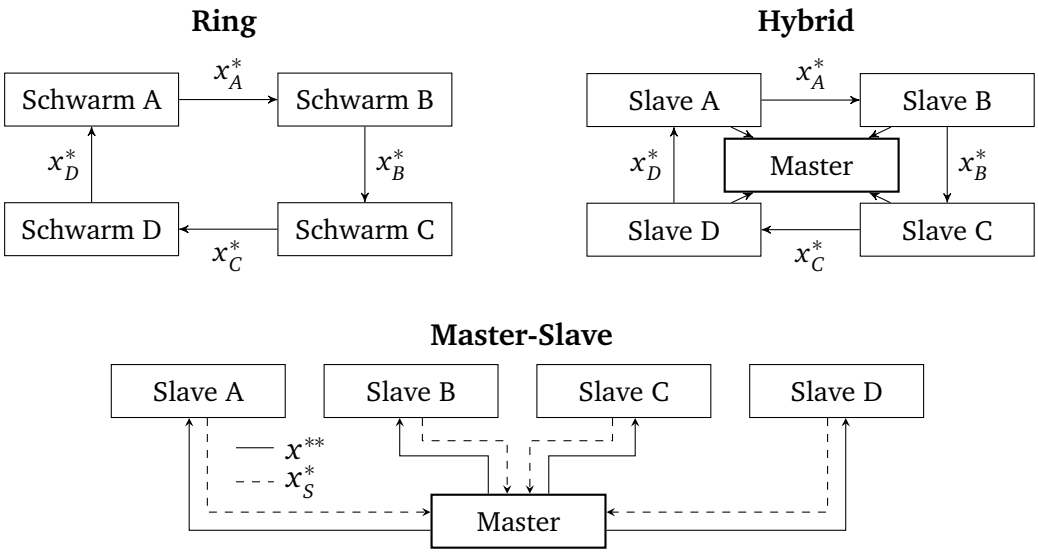


Abbildung 21: Topologien der drei kooperativen Strategien: Ring*,* Master-Slave *und* Hybrid*, mit denen mehrere Schwärme untereinander kommunizieren. Dabei beinhaltet der Informationsaustausch immer die beste Lösung eines Schwarmes.*

Ein weiteres Modell für einen Multi-Schwarm BA ist in [68] vorgestellt. Es soll das Verhältnis zwischen Erkundung und Ausbeutung durch unterschiedliche Attribut-Werte der Schwärme verbessern. Dazu werden die Schwärme derart initialisiert, dass manche von ihnen stark zur Erkundung und andere stark zur Ausbeutung tendieren. Zudem können noch Schwärme mit weniger starken Tendenzen eingebracht werden, sodass sich insgesamt eine Mischung ergibt, die das Spektrum zwischen Erkundung und Ausbeutung gut abdeckt.

Die Intention für dieses Modell basiert auf der der Tatsache, dass der BA in seiner originalen Form bereits ein gutes Verhältnis zwischen Erkundung und Ausbeutung hat. Jedoch nur dann, wenn die Werte der entsprechenden Einflussfaktoren *Pulsrate* und *Lautstärke* „gut gewählt“ sind. Doch wann sind sie das? Die Pulsrate wird im Wertebereich $[0, 1]$ und die Lautstärke in $[0, \infty]$ angegeben. Dabei wird sofort ersichtlich, dass es unendlich viele Kombinationen der beiden Attribute gibt. Die *eine*, richtige Kombination zu finden (sofern es überhaupt nur eine gibt) ist also annähernd unmöglich.

Das Multi-Schwarm Modell aus [68] (Abbildung 22) soll genau diesem Umstand entgegen wirken. Zwei Operatoren sind dabei charakteristisch für das Modell: zum einen der „Migrations-Operator“, mit dem die beste Lösung (x^*) beziehungsweise die beste Fledermaus eines Schwarmes X in einen Schwarm Y migriert wird und zugleich die schlechteste Fledermaus in Y ersetzt. Zum anderen der „Selektions-Operator“, mit dem die beste Fledermaus jedes Schwarmes in jedem Zeitschritt in den Elite-Schwarm übertragen wird. Zugleich lässt sich mit diesem Operator die minimale Fitness für den Elite-Schwarm regulieren, indem die selektierte Fledermaus nur dann in den Elite-Schwarm aufgenommen wird, wenn ihre Fitness über einer vom Benutzer vorgegebenen Grenze liegt.

Aufgrund der Erkenntnisse aus dem Ansatz mit den drei kooperativen Strategien und dem zuletzt erläuterten Modell, lässt sich vereinfacht sagen: geringe Abhängigkeiten zwi-

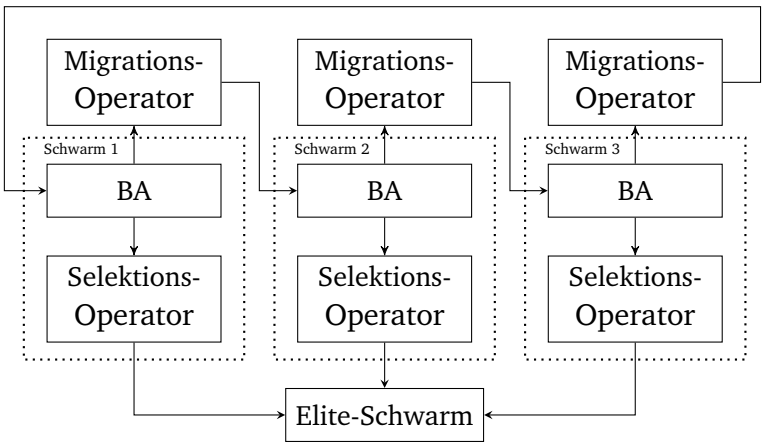


Abbildung 22: Das Multi-Schwarm-Modell aus [68] am Beispiel von drei Schwärmen.

schen den Schwärmen sorgen für schnelle Laufzeiten und gute Ergebnisse. Zudem muss eine gewisse Kommunikation zwischen den Schwärmen stattfinden, damit sie nicht einfach parallel und unabhängig von einander arbeiten, sondern voneinander profitieren.

Daher wurde für BatCEP prototypisch ein Modell entwickelt, in dem die global beste Lösung x^{**} zwischen allen Schwärmen kommuniziert wird. Es ist in Abbildung 23 dargestellt. Die global beste Lösung x^{**} wird unter allen Schwärmen kommuniziert. Jeder

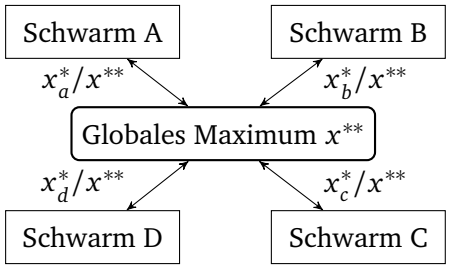


Abbildung 23: Die in BatCEP eingesetzte, kooperative Strategie.

Schwarm bezieht sie in seinen Optimierungsprozess mit ein und aktualisiert sie mit seiner Schwarm-lokal besten Lösung x^*, wenn er sie damit verbessert. Damit arbeiten alle Schwärme immer mit der global besten Lösung. Zugleich bestehen keine Abhängigkeiten die sich negativ auf die Laufzeit auswirken, weil die Schwärme nicht aufeinander warten müssen.

Im Zuge einiger Tests erwies sich diese Strategie als effektiv. Für Messergebnisse sei auf den Abschnitt 7.2.1 verwiesen, in dem BatCEP mit unterschiedlich vielen Schwärmen evaluiert wird.

5.3.1 BatCEP als Pseudocode

Das wesentliche Konzept hinter BatCEP ergibt sich aus den Inhalten von Abschnitt 5 bis 5.2.6. Zudem wird mit dem Einsatz mehrerer Schwärme eine Möglichkeit geschaffen, um BatCEP

zu parallelisieren, womit Abhängigkeiten zwischen den einzelnen Schwärmen einhergehen. Um BatCEP in seiner Gesamtheit übersichtlich darzustellen wird es nachfolgend als Pseudocode beschrieben. Dies beginnt zunächst mit der Initialisierungsphase, in Algorithmus 9.

Algorithmus 9 Initialisierung der Population

```
1:  procedure INITIALISIERUNG
2:      for all Schwärme i do
3:          for all Fledermäuse j do
4:              f^0_{i,j} ← α;                          ▷ α ∈ [0, f_max]
5:              r^0_{i,j} ← init. Pulserate * β;        ▷ β ∈ [0.85, 1]
6:              a^0_{i,j} ← init. Lautstärke * β;
7:              v^0_{i,j} ← f^0_{i,j} * γ;              ▷ γ ∈ [0, 1]
8:              x^0_{i,j} ← RHH;                        ▷ mittels RHH erzeugte CEP-Regel
9:          end for
10:     end for
11: end procedure
```

Dabei stehen die Indizes i, j und 0 für die j-te Fledermaus des Schwarmes i im Zeitschritt 0. α ist ein Zufallswert aus $[0, f_{max}]$ und hinter β steckt ein Zufallswert aus dem Bereich $[0.85, 1]$, mithilfe dessen die Werte für $a^0_{i,j}$ und $r^0_{i,j}$ eingegrenzt werden. Der Wertebereich für β hat sich im Zuge einiger Tests als gut erwiesen und passt zum Initialisierungsprozess des BA, in dessen Definition für alle a^0 und r^0 lediglich initiale Werte vorgeschlagen aber nicht vorgeschrieben werden. Die dritte Variable, γ, ist ein Zufallswert aus dem Bereich $[0, 1]$. Damit hat die Geschwindigkeit $v^0_{i,j}$ einen zufälligen Wert kleiner als $f^0_{i,j}$. Prinzipiell könnte die Geschwindigkeit auch einfach mit 0 initialisiert werden. Der Grund hierfür ist, dass sie bereits direkt nach der Initialisierung einen neuen Wert $v^t_{i,j}$ gemäß der Gleichung 9 erhält, in den zwar ihr Initialwert eingeht aber auch die aktuelle Fitness der zugehörigen Fledermaus und dessen Frequenz $f^t_{i,j}$. Dennoch wurde diese Art der Initialisierung ausgewählt, weil sie der Definition des BA entspricht.

Als Weiterführung listet Algorithmus 10 den Pseudocode von BatCEP im Ganzen auf. Dabei bedeutet „parallele Ausführung“, dass der wesentliche Optimierungsalgorithmus von Zeile 6 bis 24 von jedem Schwarm zeitgleich durchgeführt wird. Die in Zeile 25 auszuliefernde Lösung ist dann die zwischen allen Schwärmen kommunizierte, global beste Lösung x^{**}. θ ist ein Zufallswert aus dem Wertebereich der initialen Pulsrate r, [0,1] und π ein Zufallswert aus dem Wertebereich der initialen Lautstärke a, [0,a]. Sowohl θ als auch π wird für jede Fledermaus und in jedem Zeitschritt neu initialisiert. Das Stopp-Kriterium ist auch bei BatCEP das frühzeitige Erzeugen einer CEP-Regel mit einer Fitness von 100%.

Algorithmus 10 BatCEP

```
1: procedure BatCep
2:     Definition der F_F1Score
3:     Schwarm initialisieren (Algorithmus 9);
4:     Deklaration der global besten Position x**;
5:     for all Schwärme do                                   ▷ parallele Ausführung
6:         Alle Positionen hinsichtlich ihrer F1Score messen;
7:         Die beste Position x* merken;
8:         while t < Zeitschritte & Stopp-Kriterium nicht erfüllt do
9:             for all Fledermäuse i do
10:                Aktualisiere f_i^t und v_i^t;
11:                Führe einen Zufallsflug durch (Algorithmus 5);
12:                if θ > r_i^t then                                ▷ θ ∈ [0,1]
13:                    Führe eine lokale Suche durch (Algorithmus 8);
14:                end if
15:                if π < a_i^t & F_F1Score(x_i^t) ≥ F_F1Score(x*) then   ▷ π ∈ [0,A]
16:                    if F_F1Score(x_i^t) ≥ F_F1Score(x**) then
17:                        Akzeptiere die Lösung x_i^t als globales Maximum x**;
18:                    end if
19:                    Akzeptiere x_i^t als Schwarm-lokales Maximum x*;
20:                    Erhöhe r_i^t und reduziere a_i^t (Gleichungen 5 und 6);
21:                end if
22:            end for
23:        end while
24:    end for
25:    Liefere die global beste Position x** als Lösung aus;
26: end procedure
```

6 Implementierung

BatCEP wurde in Java programmiert und realisiert damit die bisher beschriebenen Konzepte. Dieses Kapitel beschreibt die wesentlichen Entwurfsentscheidungen und Prozesse der Implementierung, angefangen mit der Ein- und Ausgabe von Trainingsdatensatz, Konfiguration und CEP-Regeln. Weiter über den Optimierungsalgorithmus mit seinen spezifischen Eigenschaften wie beispielsweise der Verknüpfung einer Fledermaus mit einer CEP-Regel und der initialen Schwarm-Erzeugung. Bis hin zur Evaluierung mit der F1Score. Die zugrundeliegende CEP-Engine, wurde von CepGP [51] übernommen und wird nicht weiter beschrieben.

6.1 Ein- und Ausgabe

Konfigurationsdatei Um BatCEP zu konfigurieren muss der Benutzer eine Konfigurationsdatei im XML-Format [10] schreiben und diese zum Programmstart als Argument für BatCEP angeben. Sie ist die einzige, konfigurierbare Schnittstelle zum Benutzer und hat folgende Struktur:

```
<?xml version="1.0"?>
<batcep>
  <swarm id="A">
    <swarmsize>20</swarmsize>
    <timesteps>100</timesteps>
    <frequency>1.0</frequency>
    <pulserate>0.6</pulserate>
    <gamma>0.9</gamma>
    <loudness>1.0</loudness>
    <alpha>0.9</alpha>
  </swarm>
  <swarm id="B">
    ...
  </swarm>
  ...
</batcep>
```

Jeder Schwarm ist als Element mit einem Attribut `id` anzugeben. Mit dem Attribut kann der Benutzer den Schwarm benennen, wobei der Name vorerst keine weitere Verwendung hat. Relevant wird er dann, wenn BatCEP in zukünftigen Projekten gemäß einer anderen kooperativen Strategie wie beispielsweise die Ring-Strategie (Abschnitt 5.3) arbeiten soll. Dann können sich die Schwärme anhand ihrer Namen referenzieren. Jedes Schwarm-Element in der XML-Datei enthält weitere Elemente mit Werten für die Attribute der Fledermäuse, somit lassen sich die Schwärme unabhängig voneinander definieren. Nach dem Einlesen der Konfigurationsdatei wird ein korrespondierendes `BatConfig`-Objekt daraus erzeugt. Dieses wird an verschiedene Objekte, die Informationen über die Schwarm- beziehungsweise Fledermaus-Konfiguration benötigen, als Parameter weitergereicht. So beispielsweise an die Initialisierungskomponente oder an den sogenannten `MeasureLogger`,

[10] `https://de.wikipedia.org/wiki/Extensible_Markup_Language` (zuletzt zugegriffen am 21. August 2017)

der relevante Messwerte aus einem Programmablauf bezieht. Genauere Informationen zu BatCEP-spezifischen Messwerten sind im Abschnitt 7.1 zu finden.

Datensatz und das spezielle Ereignis Z Bevor das Lernverfahren als Kernkomponente von BatCEP gestartet werden kann, muss der Trainingsdatensatz in einem Vorverarbeitungsschritt von der CEP-Engine eingelesen worden sein. Dies ist ein notwendiger Schritt, weil alle nachfolgenden Aktionen wie das Instantiieren von Ereignissen und Zusammensetzen von Regel-Bäumen darauf basieren. Dazu muss der Trainingsdatensatz in dem nachfolgend dargestellten Format vorliegen und BatCEP als Argument überreicht werden:

```
...
2017-07-22 05:11:05; C; ID: 3.0; c1: -15.0; c2: -86.0
2017-07-22 05:20:21; E; ID: 4.0; e1: 20.0
2017-07-22 05:34:57; F; ID: 3.0; f1: -28.0; f2: 6.0; f3: 59.0
Z
2017-07-22 05:35:01; A; ID: 2.0; a1: -12.0; a2: 3.0
...
```

Es handelt sich also um eine Liste ohne spezielle Präambel, die zeilenweise zeitlich fortlaufende Ereignisse in einem CSV-Format [11] enthält. Das komplexe Ereignis Z befindet sich ebenfalls im Datensatz. Außerdem muss es vom Benutzer zum Programmstart von BatCEP angegeben werden. Ein beispielhafter Programmaufruf hat folgende Form:

```
java -jar BatCEP.jar datensatz.csv Z konfig.xml
```

Ausgabe als Menschen-lesbare CEP-Regel Schon während des Optimierungsprozesses gibt jeder Schwarm seine Schwarm-lokal beste CEP-Regel x^* zusammen mit den zugehörigen Werten für Recall, Precision und F1Score als Menschen-lesbare Zeichenkette in die Standard-Ausgabe. Nach Beendigung des Algorithmus, also nachdem alle Schwärme ihre Optimierung beendet und CEP-Regeln erzeugt haben, wird die global beste Lösung x^{**} im Programm-Verzeichnis in einer einfachen Datei ebenfalls als eine solche Zeichenkette abgelegt. Das folgende Beispiel zeigt das Format der Zeichenkette:

```
Recall: 0.7083333 Precision: 0.5171103 Fitness: 0.597802
((A as A0 → C as C0)∧(A0.ID > 3.5))[win:time:23Seconds] ⇒ HIT
```

„Fitness“ steht dabei für die F1Score.

6.2 CEP-Regeln, Fledermäuse und Aktualisierungsoperationen

CEP-Regeln als Positionen für Fledermäuse Jede Fledermaus ist eine Instanz der Java-Klasse `Bat`, die die typischen Attribute: f, v, r, a und x vorgibt. Die Position x (im Programmcode als `solution` bezeichnet) repräsentiert eine CEP-Regel des Typs `RuleWithFitness`, der bereits von Norman Offel für CepGP implementiert wurde. Damit ist x eine CEP-Regel in der zuvor beschriebenen Baumform und kann mit Zugriffsmethoden über die Instanz einer Fledermaus gelesen und verändert werden.

[11] `https://de.wikipedia.org/wiki/CSV_(Dateiformat)` (zuletzt zugegriffen am 21. August 2017)

Aktualisierungsoperatoren Die im Abschnitt 5.2 definierten Aktualisierungsoperatoren zum Verändern eines CEP-Regelbaumes sind als Methoden in der Klasse `PointUpdate` implementiert:

1. zufälliger Knoten-Austausch im ECT: `updateECT`
2. zufälliger Knoten-Austausch im ECT oder neue Fenstergröße: `updateECTOrWindow`
3. explizite Reduktion der Regel-Komplexität: `explicitComplexityUpdate`
4. Fenstergröße zufällig neu auswählen oder gezielt reduzieren: `explicitWindowUpdate`
5. Radius-Aktualisierung des Regelfensters: `windowRadiusUpdate`
6. zufälliger Knoten-Austausch im ACT: `updateACT`

Die Methoden 2 und 6 wurden unverändert aus der CepGP-Implementierung übernommen. Alle anderen speziell für BatCEP implementiert und letztlich verwenden alle Methoden Funktionalitäten aus der CEP-Engine, um die CEP-Regelbäume zu traversieren.

Der rein zufällige Flug (Methode: `updateSolution`) verwendet die Methoden aus 2 und 6 und der bedingt zufällige Flug (Methode: `optimizeSolution`) verwendet die Methoden aus 1, 3 und 4. Die lokale Suche (Methode: `generateLocalSolution`) verwendet die Methoden aus 3, 4, 5 und 6.

6.3 Programmeinstieg und Prozess des Optimierungsverfahrens

`BatCEP` ist die Hauptklasse und damit der Einstiegspunkt in das Programm. Sie initiiert alle fundamentalen Prozesse, angefangen mit der Verarbeitung der Konfigurationsdatei und der Schwarm-Initialisierung. Der Hauptalgorithmus, der in der Klasse `BatAlgorithm` implementiert ist, wird für jeden Schwarm einzeln aus der `BatCEP`-Methode `fly` gestartet. Dafür ist die Klasse `BatAlgorithm` parallelisierbar (durch implementieren des Java `Callable`-Interface). `BatAlgorithm` entspricht im Wesentlichen dem in Algorithmus 10 dargestellten Pseudocode ohne die Initialisierung und realisiert zudem den Mechanismus für die Schwarm-Kommunikation aus Abbildung 23 mithilfe der statischen Variable `GLOBAL_BEST_BAT` vom Typ `Bat`.

Initialisierung des ersten Fledermaus-Schwarmes Die Implementierung der Ramped half-and-half Initialisierung (RHH) wurde für BatCEP derart angepasst, dass sie nicht nur CEP-Regelbäume erzeugt, sondern auch die initialen Fledermaus-Schwärme. Dazu wurde RHH in die Klasse `SwarmInitializer` verlagert, die vorrangig die Verarbeitungslogik für *mehrere Fledermaus-Schwärme* beinhaltet. Dementsprechend werden alle Informationen aus der Konfigurationsdatei (Anzahl der Schwärme und Attribut-Werte) als Parameter in Form des `BatConfig`-Objektes an den SwarmInitializer weitergereicht und dieser liefert nach erfolgreicher Initialisierung eine Liste mit Fledermaus-Schwärmen zurück (`ArrayList<Bat[]>`). Jeder einzelne Schwarm ist ein `Bat`-Array, dessen Größe der vom Benutzer vorgegebenen initialen Schwarmgröße (XML-Element: `swarmsize`) entspricht.

Verwaltung und Repräsentation der besten Lösungen Jeder Schwarm verwaltet seine Fledermäuse mit den besten Lösungen in der sogenannten Elite-Liste – es wird also nicht nur die beste Fledermaus gespeichert. Die erste Fledermaus, die dieser Liste hinzugefügt wird, ist die mit der besten Lösung direkt nach dem Initialisierungsprozess. Generell wird eine Fledermaus dieser Liste immer dann hinzugefügt, wenn sie bereits die global beste Lösung x^{**} und noch nicht in er Liste enthalten ist oder wenn sie eine einfache Fledermaus aus dem Schwarm ist, deren Lösung als neue Schwarm-lokal beste Lösung x^{*} akzeptiert wurde. Um doppelte Lösungen zu vermeiden, wird als weitere Bedingung geprüft ob exakt diese Lösung (also exakt diese CEP-Regel) noch nicht in der Liste enthalten ist. In jedem Zeitschritt wird die Elite-Liste nach absteigender Fitness sortiert, sodass die beste Lösung einfach am Anfang der Liste zu finden ist, denn während des Verfahrens ist nur eine, nämlich die beste Lösung x^{*} für Vergleiche interessant.

Des Weiteren referenzieren alle Schwärme *dieselbe*, zuvor bereits genannte Variable `GLOBAL_BEST_BAT`, hinter der sich die Fledermaus mit der global besten Lösung x^{**} verbirgt. Zu Beginn jedes Zeitschrittes prüft jeder Schwarm, ob die globale beste Lösung besser ist als seine Schwarm-lokal beste Lösung. Ist dies der Fall, dann wird die global beste Lösung in seine Elite-Liste aufgenommen und ist zugleich auch die Schwarm-lokal beste Lösung.

Nach Beendigung des Optimierungsverfahrens werden die Elite-Listen aller Schwärme in einer endgültigen Ergebnis-Liste zusammengeführt. Auch hierbei werden doppelte Lösungen durch aussortieren vermieden. Die Ergebnis-Liste steht dann in der Hauptmethode (in der Klasse `BatCEP`) zur Verfügung.

Evaluierung mit der Fitness-Funktion Die F1Score wurde bereits von Norman Offel als Methode implementiert. BatCEP verwendet diese Implementierung und berechnet damit die Fitness seiner Lösungen an verschiedenen Stellen im Programm. Vor allem immer dann, wenn eine Lösung aktualisiert wurde. Beispielsweise also während oder nach einem Zufallsflug und während der lokalen Suche. Dabei hat die Klasse `RuleWithFitness` ein Attribut `conditionFitness`, in dem die bemessene F1Score gespeichert wird. Für BatCEP wurden noch zwei weitere Attribute für Recall und Precision hinzugefügt, die ebenfalls in jedem Evaluierungsschritt aktualisiert werden. Weiterhin besitzt die Klasse `RuleWithFitness` die Attribute `windowFitness` und `complexityFitness`, weil CepGP diese in seine Fitness-Berechnung einfließen lässt. BatCEP verwendet diese Attribute in keiner Weise. Dennoch macht es Sinn sie beizubehalten, weil sie in zukünftigen Projekten gegebenenfalls wieder von Nutzen sein können.

6.4 Architektur-Übersicht und Beschreibung der wesentlichen Module

Um die wichtigsten Module übersichtlich darzulegen, zeigt Abbildung 24 ein abstraktes Klassendiagramm der BatCEP-Implementierung. Die dunkel hinterlegten Klassen und Pakete wurden von CepGP übernommen. Im Wesentlichen handelt es sich dabei um Funktionalitäten der CEP-Engine. Im Paket `bat` sind mit den Klassen `BatAlgorithm` und `Swarm-`

Initializer die Kernkomponenten des Algorithmus hinterlegt. Das Paket update (in der CepGP-Implementierung als mutation benannt), beinhaltet mit PointUpdate eine Klasse die alle zuvor aufgeführten Aktualisierungsoperationen implementiert und das Paket util wurde um drei Klassen MeasureLogger, BatConfig und IOHandler erweitert. Der IOHandler implementiert alle lesenden und schreibenden Festplattenzugriffe. Zudem liest er zu Beginn des Programmstarts die vom Benutzer als XML-Datei formulierte Konfiguration und erstellt daraus ein korrespondierendes BatConfig-Objekt. Der MeasureLogger bietet einige BatCEP-spezifische Methoden zum Beziehen verschiedener Messwerte des Algorithmus an, die für die Evaluierung von BatCEP relevant sind. Manche dieser Methoden entnehmen Daten direkt aus der call()-Methode der Klasse BatAlgorithm und manche werden in der Hauptmethode in der Klasse BatCEP eingesetzt. Letztlich handelt es sich bei den Methoden hauptsächlich um solche, die Aktualisierungsoperationen und Lösungskandidaten analysieren beziehungsweise zählen, aufbereiten und ihre Ergebnisse an den IOHandler weiterreichen, um sie persistent abzulegen.

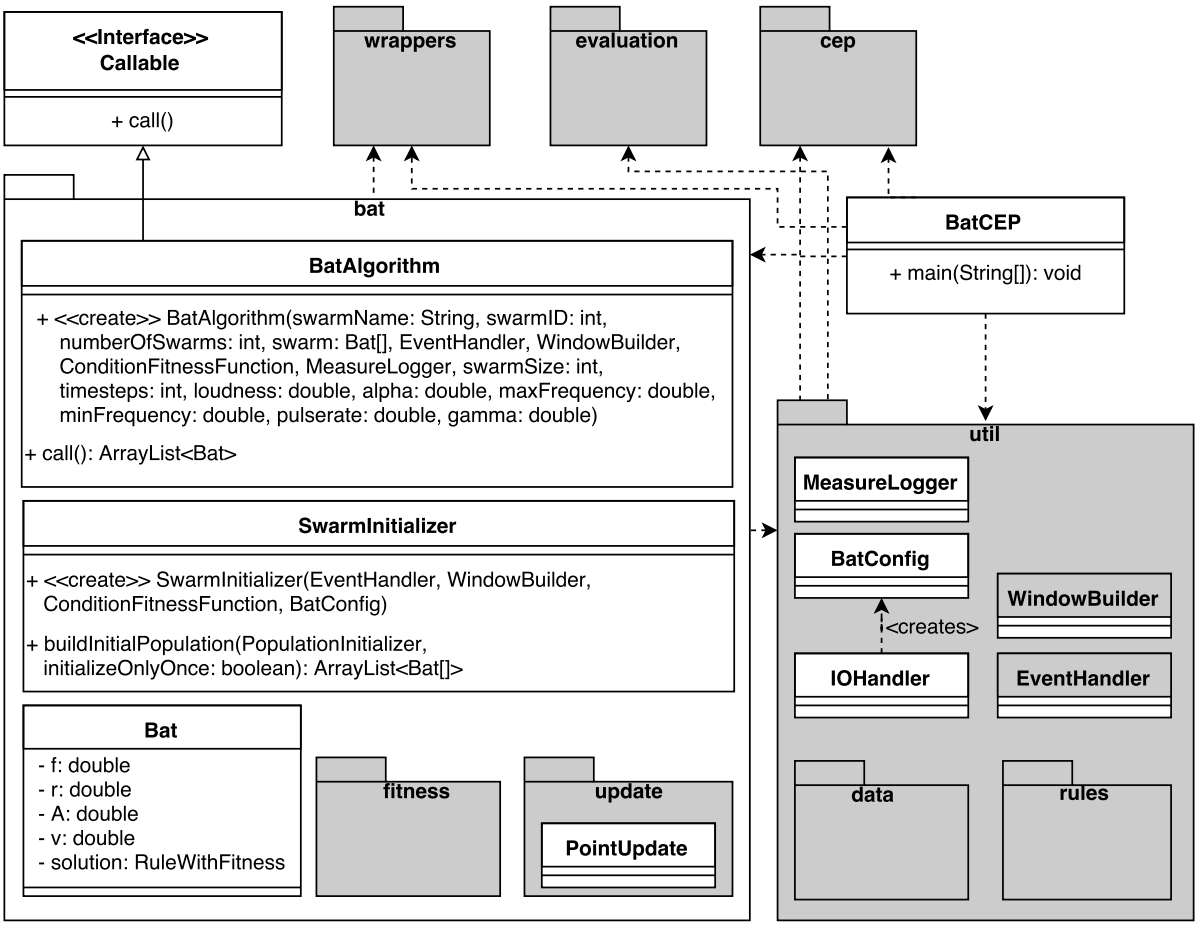


Abbildung 24: BatCEP-Klassendiagramm

7 Evaluierung

Dieses Kapitel widmet sich der Evaluierung von BatCEP. Dabei stehen Vergleiche mit dem artverwandten Verfahren CepGP, sowie Auswirkungen verschiedener Attribut-Werte und Kombinationen derselben im Fokus. Insgesamt soll ein Eindruck über die Leistungsstärke von BatCEP vermittelt werden. Dazu beginnt dieses Kapitel mit notwendigen Berechnungen der „effektiven Schwarmgröße". Weiterhin führt es Details zu den Trainingsdatensätzen auf und erläutert die initialen Attribut-Werte. Zuletzt werden ein paar prototypische Experimente aufgeführt, die einige Entwurfsentscheidungen umkehren oder abwandeln, mit dem Ziel BatCEP (in zukünftigen Projekten) noch weiter zu verbessern.

7.1 Testszenarios

Die BatCEP-Konzepte scheinen in der Theorie zu funktionieren, wodurch BatCEP ein gutes Verfahren zum Lernen von CEP-Regeln zu sein scheint. Doch ist es das wirklich? Um dies zu prüfen liegt es nahe BatCEP mit artverwandten Verfahren zu vergleichen und genau solche Vergleiche bilden den Inhalt dieses Kapitels. Die zugrunde liegenden Trainingsdatensätze werden nachfolgend beschrieben.

Datensätze Die drei eingesetzten Trainingsdatensätze wurden während der Entwicklung von CepGP [51] erzeugt und für die Evaluierung desselben verwendet. Mitunter weil BatCEP die CEP-Engine von CepGP verwendet und beide Verfahren in den kommenden Abschnitten miteinander verglichen werden, bietet es sich an genau diese Datensätze dafür zu verwenden.

Jeder der drei Datensätze wurde folgendermaßen erstellt: seine primitiven Ereignisse werden zunächst zufällig erzeugt und zeitlich nacheinander instantiiert. Im Anschluss daran wird eine spezielle CEP-Regel $R*$ auf den Datensatz angewendet und an allen Stellen an denen $R*$ feuert, wird eine Instanz des Ziel-Ereignistyps Z eingefügt. Ein Verfahren das Regeln erlernt und mit diesen Datensätzen arbeitet, generiert folglich dann das perfekte Ergebnis, wenn dieses genau $R*$ entspricht. Dann hat es eine F1Score von 1,0 sprich 100%.

small beinhaltet 500 Ereignis-Instanzen von insgesamt drei Ereignistypen. Alle Ereignis-Instanzen haben genau ein Attribut. $R*_{small}$ feuert insgesamt 42 Mal und ist definiert als:

```
((A as A0 ∨ (C as C0 ∧ B as B0)) ∧
(B0.ID = C0.ID))[win:time:180Seconds] ⇒ Z
```

medium beinhaltet 1000 Ereignis-Instanzen von insgesamt fünf Ereignistypen. Alle Ereignis-Instanzen haben genau zwei Attribute. $R*_{medium}$ feuert insgesamt 10 Mal und ist definiert als:

```
((A as A0 → E as E0) ∧
(A0.ID = E0.ID))[win:time:600Seconds] ⇒ Z
```

large beinhaltet 2500 Ereignis-Instanzen von insgesamt acht Ereignistypen und die Anzahl der Attribute verteilt sich folgendermaßen auf die Ereignistypen: {A=2, B=2, C=3, D=4, E=2, F=4, G=5, H=5}. $R*_{large}$ feuert insgesamt 158 Mal und ist definiert als:

```
((C as C0 ∧ (B as B0 → (A as A0 ∨ D as D0))) ∧
(C0.ID = B0.ID))[win:length:15] ⇒ Z
```

Anzahl der Lösungen berechnen Aufgrund der Tatsache, dass der BatCEP-Programmablauf stark von seinen initialen Attributen (Abschnitt 5.2.1) sowie vom Zufall beeinflusst wird, ist die konkrete Anzahl zu erzeugender Lösungen beziehungsweise durchzuführender Aktualisierungsoperation nicht vorhersagbar – sie lässt sich jedoch während einer Programmdurchführung mit der Gleichung 13 errechnen. Doch um diese nachvollziehen zu können, müssen noch zwei Einflussfaktoren verstanden sein.

Zum einen die durchschnittliche Anzahl von Aktualisierungsoperationen pro Zeitschritt während der lokalen Suchen:

$$\text{lokalOP}^t.$$

Zur Erinnerung: je nach Höhe der Pulsrate wird mehr oder weniger wahrscheinlich eine lokale Suche durchgeführt. Diese arbeitet mit einer Schleife, deren Anzahl der Iteration von $\lfloor a^t \epsilon \rfloor$ vorgegeben wird. Während jeder Schleifen-Iteration wird eine Aktualisierungsoperation durchgeführt, womit jede lokale Suche folglich $\lfloor a^t \epsilon \rfloor$ Operationen durchführt.

Zum anderen muss die durchschnittliche Anzahl an Aktualisierungsoperationen pro Zeitschritt während der Zufallsflüge bekannt sein:

$$\text{zufallOP}^t.$$

Jeder Zufallsflug ist entweder ein bedingt zufälliger Flug (`optimizeSolution` (drei Operationen)) oder ein rein zufälliger Flug (`updateSolution` (eine Operation)) aber beide Flüge arbeiten intern mit derselben Schleife. Diese beginnt mit dem Geschwindigkeitswert v_i^t der aktuell betrachteten Fledermaus i im Zeitschritt t und läuft bis zur Frequenz f_i^t.

Dabei sei noch einmal auf den Abschnitt 5.2.3 verwiesen, in dem genauer beschrieben ist, wie die Gleichungen für die lokale Suche und den Zufallsflug auf BatCEP abgebildet werden. Während einer Programmdurchführung werden zur Ermittlung der beiden Einflussfaktoren die folgenden Daten gesammelt:

- Anzahl durchgeführter lokaler Suchen pro Zeitschritt $[local^t]$
- Durchschnittslautstärke pro Zeitschritt multipliziert mit einer Zufallsvariable $[a^t \epsilon]$
- Anzahl durchgeführter Zufallsflüge pro Zeitschritt, unterschieden in:
  - `updateSolution` $[update^t]$
  - `optimizeSolution` $[optimize^t]$
- Durchschnittliche Differenz zwischen der Geschwindigkeit v^t und der Frequenz f^t pro Zeitschritt $[(\Delta_v^f)^t]$

$$\text{lokalOP}^t = \lfloor local^t * a^t\epsilon * 10 \rfloor \tag{10}$$

$$\text{zufallOP}^t = \lfloor (update^t + (optimize^t * 3)) * (\Delta_v^f)^t * 10 \rfloor \tag{11}$$

Die Berechnung der beiden Einflussfaktoren ergibt sich somit wie folgt (Gleichungen 10 und 11):
Die Multiplikation mit 10 ist notwendig, um die Werte von $a^t\epsilon$ und $(\Delta_v^f)^t$ auf ganze Zahlen abzubilden. Weiterhin kann mithilfe von lokalOP^t und zufallOP^t auch die durchschnittliche Anzahl von Operationen pro Zeitschritt und Fledermaus errechnet werden (Gleichung 12): Aus dieser Gleichung lässt sich auch entnehmen, dass sich durch die einfache Addition

$$\text{fledermausOP}^t = \frac{\text{lokalOP}^t + \text{zufallOP}^t}{\text{initiale Schwarmgröße}} \tag{12}$$

von lokalOP^t und zufallOP^t oder durch die Multiplikation der initialen Schwarmgröße mit fledermausOP^t die Anzahl durchgeführter Operationen pro Zeitschritt ergibt – die zugleich der Anzahl tatsächlich erzeugter Lösungen pro Zeitschritt entspricht und die ist zumeist größer als die Anzahl der Fledermäuse im Schwarm. Diese Anzahl wird als ***effektive Schwarmgröße*** bezeichnet. Kurzum: jede Fledermaus hat zwar immer eine Position, aber diese wird häufig aktualisiert und ändert sich folglich. Damit produziert jede Fledermaus im Laufe des Algorithmus mehrere Lösungskandidaten und die Menge all dieser Lösungskandidaten wird als effektive Schwarmgröße bezeichnet. Die initiale Schwarmgröße, also die unveränderliche und von RHH initial erzeugte Anzahl der Fledermäuse pro Schwarm ist hingegen vom Benutzer vorgegeben (Element `swarmsize` in der XML-Datei).

Zuletzt werden lokalOP^t und zufallOP^t in die Gleichung 13 eingesetzt und dessen Berechnung ergibt die Anzahl erzeugter Lösungen für eine komplette Programmdurchführung:

$$\text{OPs} = \text{Schwärme} * \text{Zeitschritte} * (\text{lokalOP}^t + \text{zufallOP}^t) \tag{13}$$

Schwärme bezeichnet dabei die Anzahl parallel eingesetzter Schwärme und *Zeitschritte* sind entsprechend die Zeitschritte pro Schwarm, die jeder Schwarm durchläuft.

Alle hier erläuterten Gleichungen gelten wohlgemerkt nur für BatCEP. Die Anzahl der Lösungen der anderen Verfahren mit denen BatCEP im Laufe der nachfolgenden Abschnitte verglichen wird, berechnet sich anders. Wie genau die Berechnungen dabei durchgeführt werden, wird an entsprechenden Stellen angegeben.

Initiale Attribut-Werte BatCEP wurde im Zuge seiner Entwicklung einige Male getestet, um Entwurfsentscheidungen zu validieren. Eine konkrete Evaluation mit Hinsicht auf Vergleichswerte und optimale Attribut-Werte wurde jedoch bisher nicht durchgeführt und folgt daher im Zuge dieses Abschnitts.

Dazu werden zunächst intuitiv und in Anlehnung an die Beschreibungen des originalen BA [69] initiale Attribut-Werte ausgewählt. Tabelle 2 listet diese auf. Sie gelten für alle nachfolgenden Testszenarien, sofern nicht explizit etwas anderes angegeben ist.

Tabelle 2: Grundlegende Konfiguration für alle nachfolgenden Testszenarien

Frequenz (min./max.)	0.0 / 1.3
Lautstärke / α	1.0 / 0.9
Pulsrate / γ	0.7 / 0.9

7.1.1 BatCEP im Vergleich mit RHH

Wie bereits in Abschnitt 5.2.6 erwähnt, verwendet BatCEP die Ramped half-and-half Initialisierung (RHH), um die erste Population zu erzeugen. Ein Vergleich zwischen den Ergebnissen von BatCEP und denen der isoliert betrachteten RHH, bietet sich als erstes Testszenario an. Aufgrund der Tatsache, dass die Arbeitsweise von RHH grundlegend auf dem Zufall basiert und auch BatCEP viel Zufälligkeit beinhaltet, kann man zurecht die Frage stellen: „Generiert RHH allein schon ähnlich gute oder sogar bessere Lösungen als BatCEP?“ Dieses Testszenario, dessen abstrakter Ablauf in Abbildung 25 beschrieben ist, soll die Frage beantworten. Im Zuge dessen wird zunächst sowohl BatCEP als auch RHH

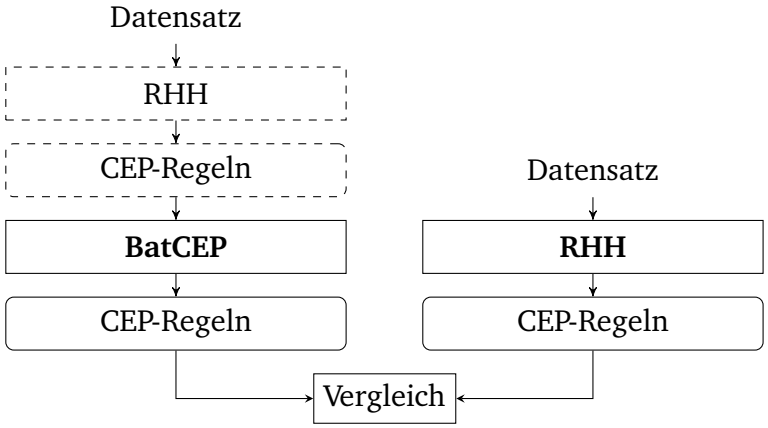


Abbildung 25: Ablauf des Testszenarios 7.1.1, in dem BatCEP mit RHH hinsichtlich der Güte (gemessen in Recall, Precision und F1Score) der jeweils erzeugten CEP-Regeln verglichen wird. ***Links*** *der abstrakte Testverlauf von BatCEP und* ***rechts*** *der von RHH.*

„unabhängig“ voneinander auf den Datensatz **small** angewendet. Tabelle 3 führt die spezifischen Kennwerte des ersten Vergleiches auf Basis des Datensatzes small auf. Es sei noch ein Mal darauf hingewiesen, dass BatCEP in seinem Initialisierungs-Schritt einmal RHH anwendet und mit den daraus erzeugten Schwärmen weiterarbeitet. Daher ist logischerweise eine gewisse Abhängigkeit zwischen BatCEP und RHH vorhanden. Abbildung 26 zeigt die

Messergebnisse auf. Dargestellt ist jeweils für BatCEP und für RHH der arithmetische Mittelwert der *besten* Fledermaus aus jedem Zeitschritt. Zur Erinnerung: die Gesamtanzahl erzeugter Fledermäuse in einem BatCEP-Testlauf, berechnet sich nach der Gleichung 13. Für RHH ist die Berechnung einfacher: Schwärme * Schwarmgröße * Zeitschritte. Einfacher deswegen, weil RHH pro Zeitschritt eine komplette Population neu erzeugt – ohne Abgleich mit einer Fitness-Funktion.

*Tabelle 3: Parameter und Kennwerte für den Vergleich von BatCEP mit RHH auf dem Datensatz **small***

	BatCEP	**RHH**
Schwärme	1	
Initiale Schwarmgröße	20	153
∅ Operationen pro Fledermaus in jedem Zeitschritt	7.65	1
Effektive Schwarmgröße	153 (7.65 * 20)	153
Zeitschritte	250	
Testläufe	10	
Erzeugte Lösungen pro Testlauf	38250	
Erzeugte Lösungen insgesamt	382500	
∅ Laufzeit pro Testlauf	64s	118s
Laufzeit über alle Testläufe	640s	1188s

small Der qualitative Unterschied gemessen in *Precision*, *Recall* und *F1Score* ist anfänglich groß, fällt aber im Bereich der späteren Zeitschritte verhältnismäßig gering aus. Während die Werte der von BatCEP erzeugten Lösungen mit zunehmenden Zeitschritten ansteigen, ist dies bei RHH nicht der Fall. Der wesentliche Grund hierfür ist, dass BatCEP die initiale Population inkrementell aktualisiert, mit einer Fitness-Funktion misst und darauf hin erneut aktualisiert und verbessert. Somit werden anfänglich eher schlechte Lösungen Schritt für Schritt optimiert. Bei RHH ist dies nicht der Fall, hier entstehen rein zufällige Lösungen in jedem Zeitschritt. Insgesamt ist die Precision bei RHH stets höher als der Recall und das lässt sich auf die Komplexität der erzeugten Regeln zurückführen. Diese sind häufig so spezifisch, dass sie an vielen korrekten Stellen feuern und das ist gut so. Zugleich ist diese Präzision auch hinderlich, denn sie verhindert ein Erreichen aller korrekter Stellen. Wie könnte diesem Umstand entgegnet werden? Grundlegend durch verallgemeinern der Bedingung, also durch mindern der Komplexität – genau das macht BatCEP.

Um noch weitere Vergleichswerte zu haben, werden BatCEP und RHH auch auf die anderen beiden Datensätze medium und large angewendet. Im nachfolgenden Abschnitt sind die Ergebnisse zunächst für den Datensatz **medium** dargestellt.

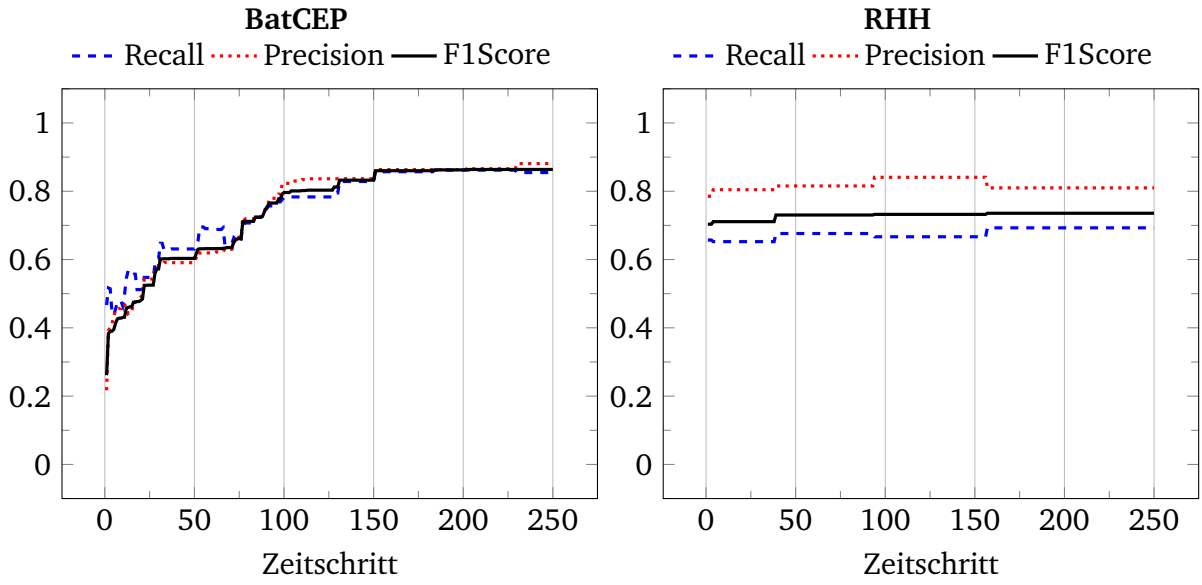


*Abbildung 26: Messergebnisse des Vergleichs von BatCEP mit RHH mit dem Datensatz **small**.*

medium Tabelle 4 listet die neuen Kennwerte auf, die sich durch die zufallsbedingte Veränderung der *effektiven Schwarmgröße* ergeben und in Abbildung 27 sind die Ergebniskurven dargestellt.

*Tabelle 4: Parameter und Kennwerte für den Vergleich von BatCEP mit RHH auf dem Datensatz **medium***

	BatCEP	**RHH**
Schwärme	1	
Initiale Schwarmgröße	20	143
∅ Operationen pro Fledermaus in jedem Zeitschritt	7.15	1
Effektive Schwarmgröße	143 (7.15 * 20)	143
Zeitschritte	250	
Testläufe	10	
Erzeugte Lösungen pro Testlauf	35750	
Erzeugte Lösungen insgesamt	357500	
∅ Laufzeit pro Testlauf	144s	387s
Laufzeit über alle Testläufe	1447s	3877s

Die Anstiege beider Kurven sind denen aus dem ersten Vergleich sehr ähnlich. Auffällig ist zum einen dass beide Verfahren schlechtere Ergebnisse hervorbringen. Zum anderen fällt für RHH die verhältnismäßig hohe *Precision* im Bereich um 0,55 bei einem gleichzeitigen Recall im Bereich 0,28 auf. Diese Differenz ist noch etwas höher als im vorhe-

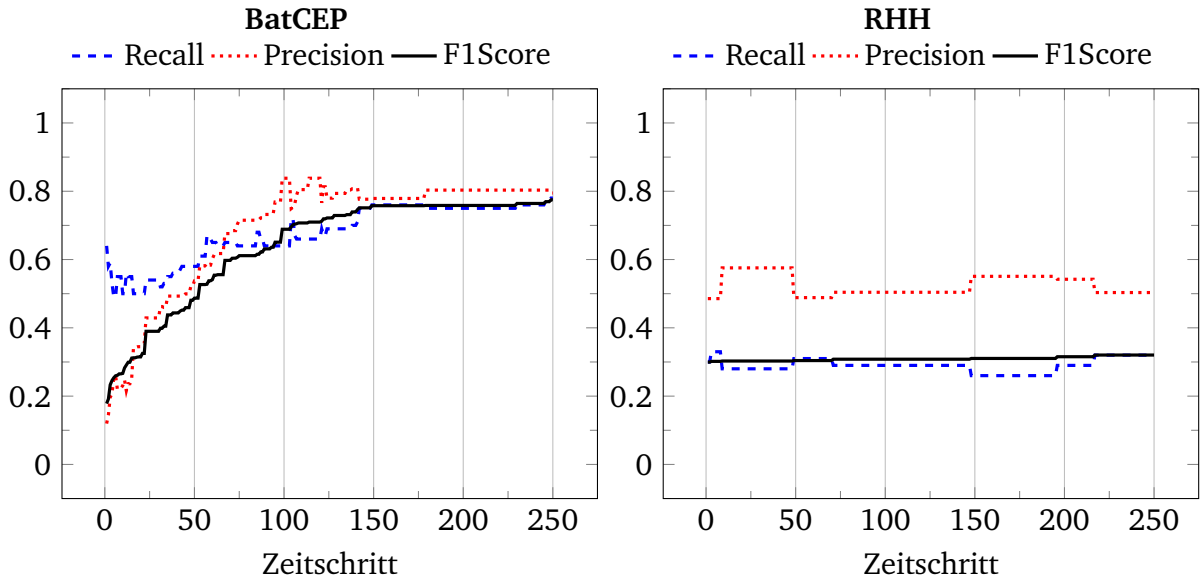


Abbildung 27: Messergebnisse des Vergleichs von BatCEP mit RHH mit dem Datensatz ***medium****.*

rigen Vergleich. Wie kann man sich das erklären? Sehr wahrscheinlich liegt das an der Kombination aus dem kurzen Bedingungsteil von $R*_{medium}$ und an ihrem vergleichsweise großen Zeitfenster von 600 Sekunden. Einige der von RHH erzeugten Regeln feuern im Bereich dieses Fensters, was zurecht als korrekt bewertet wird und folglich eine erhöhte Precision bedingt. Insgesamt feuern die erzeugten Regeln jedoch zu selten, was zu dem geringen Recall führt. Würde der Bedingungsteil mehr Ereignistypen einbeziehen, so würden mehr Regeln feuern und der Recall ansteigen – sehr wahrscheinlich jedoch zum Preis einer verringerten Precision. Zur Erinnerung: der Recall beantwortet die Frage: „Wieviele der relevanten Stellen wurden von der Regel gefunden?“, während die Precision die Frage beantwortet: „Wieviele der gefundenen Stellen sind wirklich relevant?“. Daher gibt letztlich erst die Kombination beider Messungen Aufschluss über die Güte der Regel und wie an der F1Score-Kurve von RHH in Abbildung 27 zu erkennen ist, fällt die Fitness im Schnitt mit etwa 0.3 recht gering aus.

Ein weiterer Grund für die schlechteren Ergebnisse ist die vergrößerte Differenz Δ_e^l zwischen der ersten und der letzten Ereignis-Instanz im Datensatz medium verglichen mit small. Dieser Umstand ist besonders für RHH problematisch, weil RHH die Zeitfenster für seine Regeln rein zufällig aus dem Bereich Δ_e^l auswählt.

Aus der Ergebniskurve von BatCEP lassen sich ebenfalls Schwankungen erkennen. Hier ist es zu Anfang der Recall, der verhältnismäßig hoch ausfällt. Im Zeitschritt 60 etwa scheinen sich Recall und Precision in etwa anzugleichen und dann steigt die Precision wieder an, während der Recall absinkt. Ab Zeitschritt 150 gleichen sich beide Messungen wieder an. Dieser Verlauf lässt sich mit der Arbeitsweise von BatCEP erklären: zu Beginn startet das Verfahren mit der Population von RHH, also mit einer erhöhten Precision und einem geringen Recall. Dann unternehmen alle Fledermäuse Zufallsflüge, im Zuge derer einige rein zufällig fliegen (so wie bei RHH) und andere bedingt zufällig, um ihre Position zu verbessern und diese Verbesserung beinhaltet (mit Verweis auf Abschnitt 5.2.2) eine Re-

duktion zum einen des Regel-Fensters und zum anderen der Regel-Komplexität. Weil die Fenster also verkleinert werden und auch weniger Ereignisse in den Regelbedingungen vorkommen, feuern die Regeln zunächst häufiger: der Recall steigt an und die Precision sinkt ab.

Dann wird gelegentlich eine lokale Suche durchgeführt, in der die Regel-Komplexität ebenfalls verringert wird. Zudem wird das Fenster in einem relevanten Radius optimiert. Das führt zu einer Verbesserung der Regel, die sich in einem Angleichen von Precision und Recall widerspiegelt. Unter Betrachtung dieser Ergebnisse scheinen die entwickelten Aktualisierungsoperationen für BatCEP zielführend zu sein.

large Zuletzt werden beide Verfahren auf den Datensatz large angewendet. Tabelle 5 führt die erneut aktualisierten Kennwerte für diesen Vergleich auf und die entsprechenden Ergebniskurven sind in Abbildung 28 einzusehen.

*Tabelle 5: Parameter und Kennwerte für den Vergleich von BatCEP mit RHH auf dem Datensatz **large***

	BatCEP	**RHH**
Schwärme	1	
Initiale Schwarmgröße	20	110
∅ Operationen pro Fledermaus in jedem Zeitschritt	5.5	1
Effektive Schwarmgröße	110 (5.5 * 20)	110
Zeitschritte	250	
Testläufe	10	
Erzeugte Lösungen pro Testlauf	27500	
Erzeugte Lösungen insgesamt	275000	
∅ Laufzeit pro Testlauf	857s	9111s
Laufzeit über alle Testläufe	8579s	91110s

Erneut ist eine Verschlechterung der Ergebnisse zu erkennen, bei der BatCEP eine maximale F1Score von etwa 0,6 erreicht. Der Recall bewegt sich die meiste Zeit im Bereich zwischen 0,6 und 0,8 und liegt damit im Mittel, während die Precision mit Werten unter 0,2 startet und sich fortlaufend verbessert. Insgesamt bringt BatCEP auch nicht mehr als 27.500 Lösungen hervor. Das sind etwa 10.000 weniger als bei den vorherigen Testläufen. Womit lässt sich das begründen? Zunächst ist der Datensatz verhältnismäßig groß, was das Finden guter Lösungen erschwert. Allein die maximale Differenz Δ_e^l ist mehr als doppelt so groß wie die von medium und somit lassen sich geeignete Fenstergrößen wesentlich schlechter erraten. Hinzu kommt, dass large noch mehr Ereignistypen verwendet, was für den Recall förderlich ist aber die Precision absenkt. Förderlich ist dies für den Recall weil mehr mögliche Ereignistypen dafür sorgen, dass die erzeugten Regeln häufiger feuern –

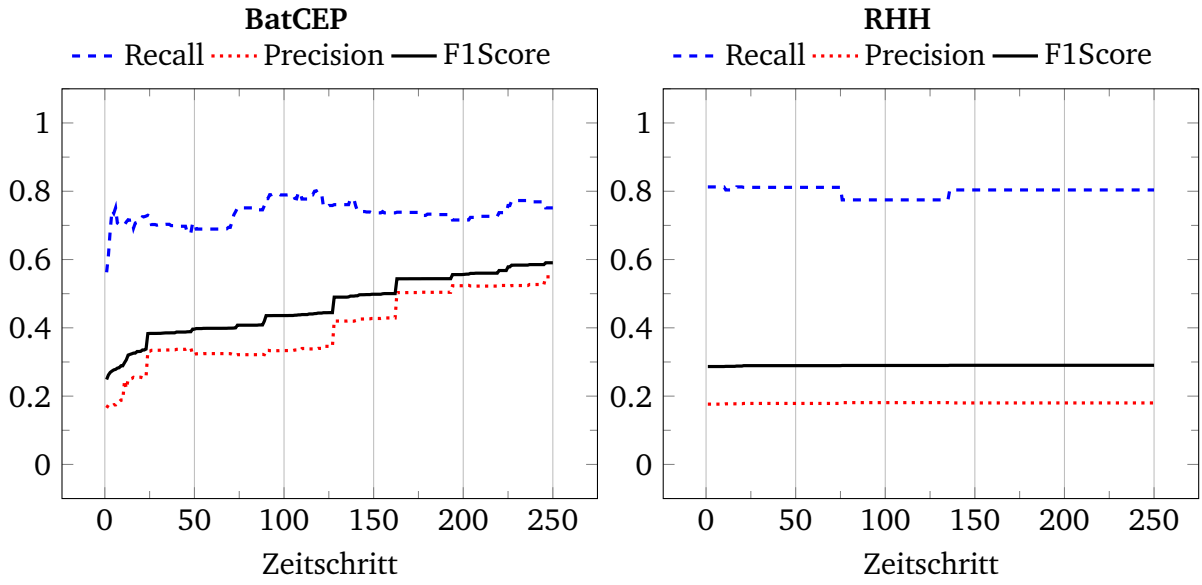


*Abbildung 28: Messergebnisse des Vergleichs von BatCEP mit RHH mit dem Datensatz **large**.*

jedoch zu häufig und damit auch an vielen falschen Stellen, was wiederum die Precision absenkt.

Bleibt noch zu klären, wieso hier insgesamt weniger Lösungen hervorgehen als aus den vorherigen Tests: dies geht auf den Ablauf von BatCEP zurück. Immer dann wenn die Fitness der zu aktualisierenden Lösung besser ist als die durchschnittliche Fitness pro Zeitschritt, wird sie im Zuge des Zufallsfluges *optimiert*. Liegt ihre Fitness hingegen unterhalb der durchschnittlichen Fitness, so wird sie *rein zufällig* aktualisiert und diese beiden Verfahren unterscheiden sich im Aufwand (siehe Abschnitt 5.2.2 und Pseudocode 5). Eine Optimierung bedingt demnach drei Mal so viele Aktualisierungsoperationen wie die rein zufällige Aktualisierung und weil in diesem Test der Großteil aller Lösungen mit derselben (schlechten) Fitness startet, werden wesentlich mehr zufällige Aktualisierungen als Optimierungen durchgeführt.

Insgesamt ist festzustellen, dass BatCEP seine Lösungen auf allen Datensätzen kontinuierlich verbessert. RHH hingegen tut dies nicht und das erwartet man intuitiv, denn RHH referenziert keine Fitness-Funktion. In der Tabelle 6 sind die Maximalwerte dieses Testszenarios für einen abschließenden Überblick dargestellt.

Tabelle 6: Überblick über die Maximalwerte von BatCEP und RHH im Testszenario 7.1.1

	BatCEP			**RHH**		
	small	medium	large	small	medium	large
Recall	0,85	0,78	0,75	0,69	0,31	0,80
Precision	0,88	0,79	0,54	0,80	0,50	0,18
F1Score	0,86	0,77	0,59	0,73	0,32	0,29

7.1.2 BatCEP im Vergleich mit CepGP

Wie bereits mehrfach erwähnt implementiert BatCEP die CEP-Engine von CepGP. BatCEP und CepGP unterscheiden sich demnach hauptsächlich in der Art des jeweiligen Optimierungsalgorithmus und diese einheitliche Grundlage ist ein guter Ausgangspunkt für einen direkten Vergleich. Der abstrakte Testverlauf ist in Abbildung 29 dargestellt. BatCEP wird

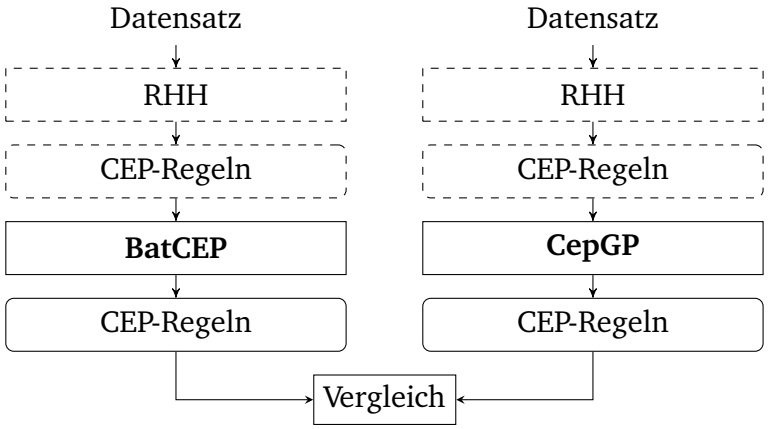


Abbildung 29: Ablauf des Testszenarios 7.1.1, in dem BatCEP mit CepGP hinsichtlich der Güte (gemessen in Recall, Precision und F1Score) der jeweils erzeugten CEP-Regeln verglichen wird. ***Links*** *der abstrakte Testverlauf von BatCEP und* ***rechts*** *der von CepGP.*

zuerst ausgeführt, weil die Anzahl der Lösungen nicht exakt vorhergesagt werden kann. Durch eine bewusste Wahl der Start-Parameter ist diese jedoch abschätzbar. Dabei werden hier erstmals mehrere Schwärme eingesetzt, um mehrere Prozesse (und damit gegebenenfalls auch mehr CPU-Kerne) zu nutzen, was in erster Linie die Gesamtlaufzeit in einem akzeptablen Rahmen halten soll, denn in diesem Testszenario steigt die Anzahl der Lösungen mit etwa 150.000 pro Testdurchlauf auf annähernd das Dreifache der vorherigen Testszenarien an. Im Zuge aller Testdurchläufe werden somit knapp 3 Millionen Lösungen erzeugt. CepGP wird mit angepassten Startwerten aufgerufen, sodass es genauso viele Lösungen erzeugt wie BatCEP. Dabei sind die als *optimal* ausgezeichneten Startwerte für CepGP aus [51] Seite 111 entnommen, lediglich eine geringfügige Anpassung der Population muss vorgenommen werden, um auf die Anzahl der benötigten Lösungen zu kommen. Letztlich ist es genau diese Konfiguration von CepGP, aus der die 150.000 Lösungen resultieren.

small In diesem Testszenario sind zwei Konfigurationstabellen angegeben. Tabelle 7 für BatCEP und Tabelle 8 für CepGP. Grund hierfür ist, dass BatCEP und CepGP sehr unterschiedliche Attribute haben, was wiederum den Unterschieden zwischen Schwarmintelligenz und Genetischer Programmierung geschuldet ist. Abbildung 30 zeigt die Ergebniskurven.
Es ist deutlich zu erkennen dann beide Verfahren gute Ergebnisse hervorbringen. CepGP steigt kontinuierlich bis auf eine maximale F1Score von 0,86 an und BatCEP erreicht sogar 0,93. Während CepGP einen kontinuierlichen Kurvenanstieg über die gesamte Laufzeit

Tabelle 7: Parameter und Kennwerte für BatCEP im Vergleich von BatCEP mit CepGP. Datensatz: ***small***

	BatCEP
Schwärme	4
Initiale Schwarmgröße	120
∅ Operationen pro Fledermaus in jedem Zeitschritt	10.65
Effektive Schwarmgröße	1278 (10.65 * 120)
Zeitschritte	30
Testläufe	10
Erzeugte Lösungen pro Testlauf	153360
Erzeugte Lösungen insgesamt	1533600

Tabelle 8: Parameter und Kennwerte für CepGP im Vergleich von BatCEP mit CepGP. Datensatz: ***small***

	CepGP
Populationsgröße	5112
Generationen	30
Crossover-Rate	0.8
Mutationsrate	0.05
Elitism-Rate	0.1
Testläufe	10
Erzeugte Lösungen pro Testlauf	153360
Erzeugte Lösungen insgesamt	1533600

aufweist, sticht der Kurvenverlauf von BatCEP im besonderen Maße ins Auge, denn dieser steigt bis zum Zeitschritt 12 stark an und erreicht dort sein Maximum. Im Umkehrschluss bedeutet dies, dass BatCEP mit der hier gewählten Konfiguration weit weniger Lösungen erzeugen muss, um im Durchschnitt dieselbe Fitness zu erreichen – und zwar reichen etwa: $4 * 120 * 10,65 * 12 = 61344$ erzeugte Lösungen aus. Was ist der Grund hierfür? Die Anzahl erzeugter Lösungen bis zum Maximum bei Zeitschritt 12 übersteigt mit 61344 weit die Anzahl der erzeugten Lösungen im ersten Testszenario 7.1.1. Dort hat BatCEP auf demselben Datensatz wie hier eine maximale F1Score von 0.86 erreicht. Das bessere Ergebnis hier kann demnach einfach aus der größeren Menge erzeugter Lösungen resultieren. Doch dieser Vergleich hinkt, denn im ersten Testszenario wurde nur ein Schwarm eingesetzt und auch die initiale Schwarmgröße und die Anzahl der Zeitschritte unterscheiden sich. Genau diese Unterschiede in den Attribut-Werten sind von großem Interesse und damit sei auf den Abschnitt 7.2 verwiesen.

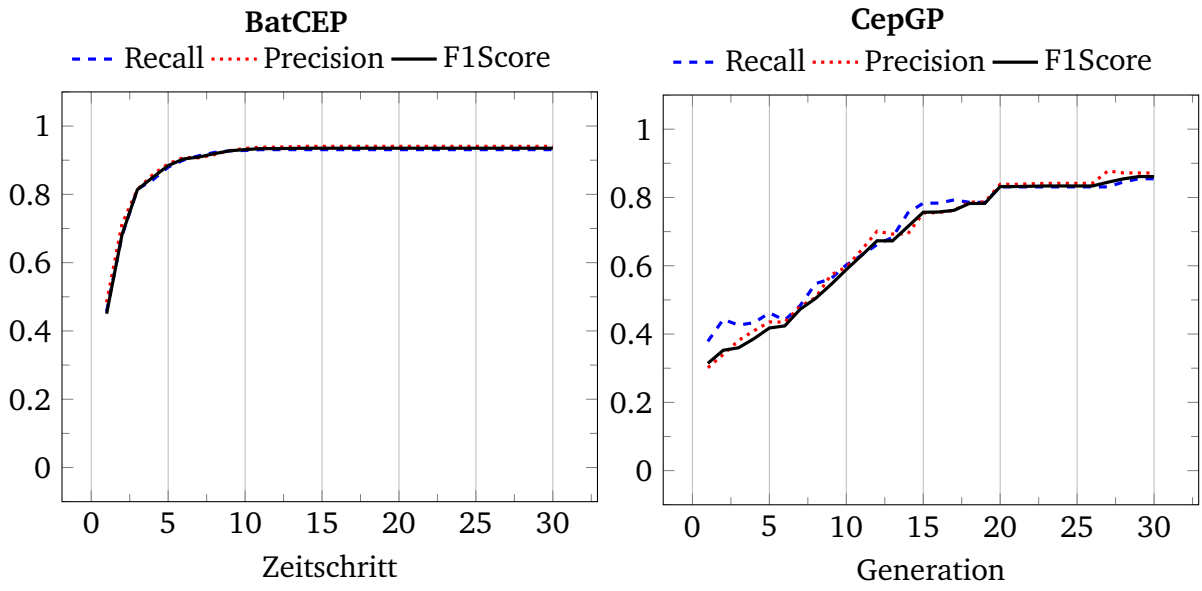


Abbildung 30: Messergebnisse des Vergleichs von BatCEP mit CepGP. Datensatz: ***small****.*

medium Nachfolgend sind die Konfigurationen und Ergebniskurven für den Test auf dem nächstgrößeren Datensatz medium aufgeführt (Tabellen 9 und 10 sowie Abbildung 31).

Tabelle 9: Parameter und Kennwerte für BatCEP im Vergleich von BatCEP mit CepGP. Datensatz: ***medium***

	BatCEP
Schwärme	4
Initiale Schwarmgröße	120
∅ Operationen pro Fledermaus in jedem Zeitschritt	10.3
Effektive Schwarmgröße	1236 (10.3 * 120)
Zeitschritte	30
Testläufe	10
Erzeugte Lösungen pro Testlauf	148320
Erzeugte Lösungen insgesamt	1483200

Hier ist ebenfalls eine verhältnismäßig frühe Konvergenz von BatCEP zu beobachten, während CepGP sich stetig über alle Generationen verbessert. Weiterhin ist zu erkennen, dass die Kurven von CepGP offenbar mit noch mehr Generationen noch weiter ansteigen würden – das Maximum scheint noch nicht erreicht zu sein.

Es bleibt festzuhalten, dass BatCEP mit der hier gewählten Konfiguration schnell sein Maximum erreicht (bereits nach 15 Zeitschritten und somit etwa nach 74160 erzeugten Lösungen). Zugleich ist das Maximum mit einer F1Score von 0,87 noch verbesserungswür-

Tabelle 10: Parameter und Kennwerte für CepGP im Vergleich von BatCEP mit CepGP. Datensatz: ***medium***

	CepGP
Populationsgröße	4944
Generationen	30
Crossover-Rate	0.8
Mutationsrate	0.05
Elitism-Rate	0.1
Testläufe	10
Erzeugte Lösungen pro Testlauf	148320
Erzeugte Lösungen insgesamt	1483200

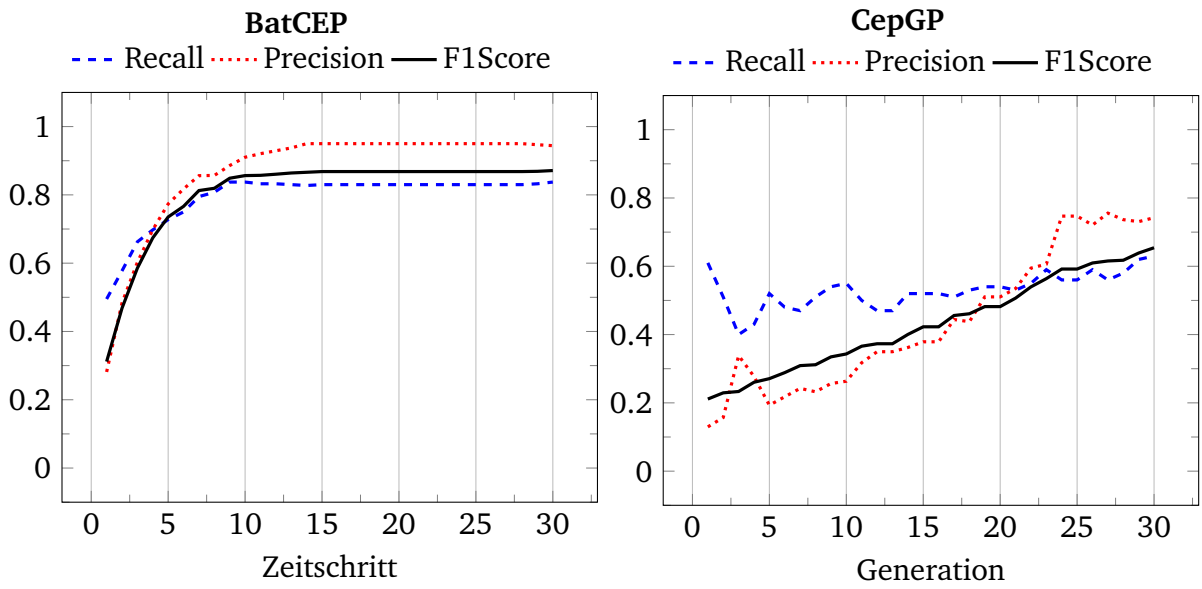


Abbildung 31: Messergebnisse des Vergleichs von BatCEP mit CepGP. Datensatz: ***medium****.*

dig. CepGP scheint hingegen noch mehr als die vollen 148320 Lösungen zu benötigen um sein Maximum zu erreichen, somit lässt dieser Test keine Aussage darüber zu, ob BatCEP oder CepGP am Ende eine bessere Fitness erreicht.

large Zuletzt steht der Vergleich beider Verfahren auf Grundlage des Datensatz *large* an. Aufgrund seiner Größe ist zu erwarten, dass sowohl BatCEP als CepGP noch schlechtere Ergebnisse hervorbringen werden, als beim vorherigen Test mit medium. Grund hierfür ist zum einen die erschwerte Eingrenzung geeigneter Regel-Fenster und zum anderen die wesentlich größere Anzahl möglicher Kombinationen, besonders weil *large* mehr Ereignistypen mit teilweise auch noch mehr Attributen enthält, als die beiden anderen Datensätze.

Unter Betrachtung der Ergebnisse lässt sich feststellen, dass beide Verfahren tatsächlich

Tabelle 11: Parameter und Kennwerte für BatCEP im Vergleich von BatCEP mit CepGP. Datensatz: ***large***

	BatCEP
Schwärme	4
Initiale Schwarmgröße	120
∅ Operationen pro Fledermaus in jedem Zeitschritt	6.46
Effektive Schwarmgröße	776 (6.46 * 120)
Zeitschritte	30
Testläufe	10
Erzeugte Lösungen pro Testlauf	93120
Erzeugte Lösungen insgesamt	931200

Tabelle 12: Parameter und Kennwerte für CepGP im Vergleich von BatCEP mit CepGP. Datensatz: ***large***

	CepGP
Populationsgröße	3104
Generationen	30
Crossover-Rate	0.8
Mutationsrate	0.05
Elitism-Rate	0.1
Testläufe	10
Erzeugte Lösungen pro Testlauf	93120
Erzeugte Lösungen insgesamt	931200

noch schlechter abschneiden. BatCEP erreicht eine F1Score von 46% und CepGP kommt auf 36%. Erstaunlich ist jedoch, dass die Fenster in beiden besten Lösungen aus allen zehn Testläufen, wider Erwarten genau dem Fenster von $R*_{large}$ entsprechen. Demnach steckt die Schwierigkeit darin geeignete Kombinationen im ECT und ACT zu finden. Zugleich hat die beste Lösung von BatCEP eine F1Score von 99% und die beste Lösung von CepGP eine F1Score von 39% erreicht.

7.2 Einfluss der BatCEP-Parameter

Während der letzten Abschnitte wurde der Wunsch nach einem besseren Verständnis für die Konfiguration der BatCEP-Attribute laut. BatCEP hat mit unterschiedlichen Konfigurationen unterschiedliche Ergebnisse hervorgebracht und daher stellt sich die Frage nach der optimalen Konfiguration, also nach den optimalen Start-Parametern aus denen sich alle At-

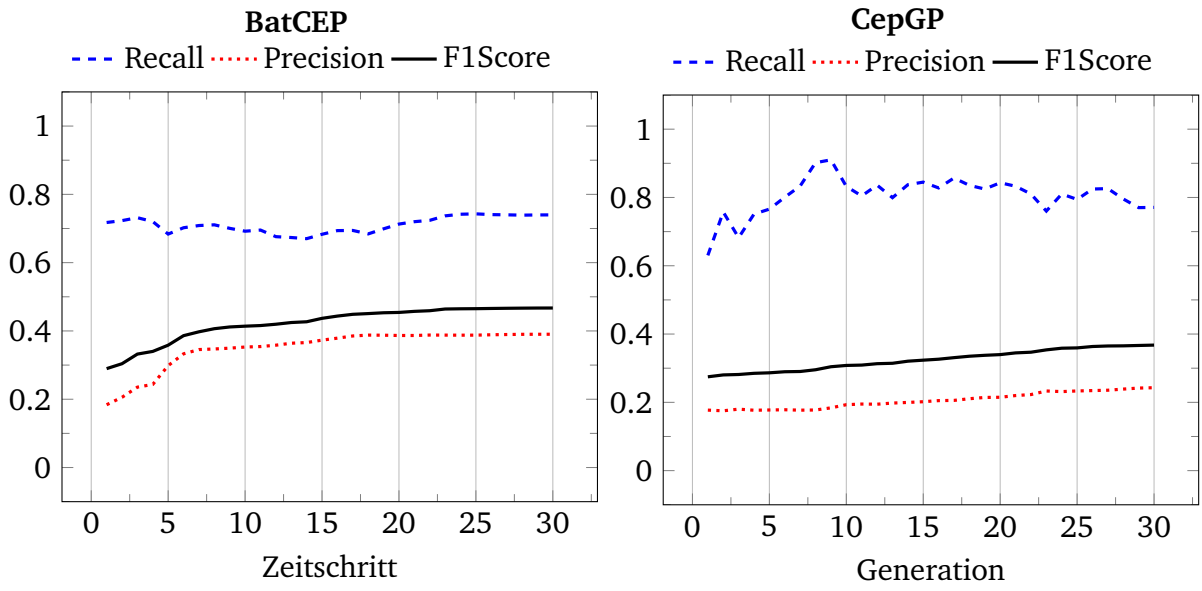


Abbildung 32: Messergebnisse des Vergleichs von BatCEP mit CepGP. Datensatz: ***large****.*

tributwerte jeder Fledermaus ableiten. Die nachfolgenden Abschnitte beschäftigen sich mit genau dieser Fragestellung und helfen dabei, mittels weiterer Testszenarien gute und weniger gute Konfigurationen herauszustellen. Um dabei einen einheitlichen Ausgangspunkt zu haben, verwenden alle Testszenarien als Grundlage die Konfiguration aus Tabelle 13 und verändern immer nur ein Attribut. Als Datensatz wird stets ***small*** eingesetzt. Um dabei aussagekräftige Ergebnisse zu erzeugen, wird für jedem Test der arithmetische Mittelwert über *zehn* Testdurchläufe gebildet – wie auch in allen Testszenarien zuvor.

Tabelle 13: Grundlegende Konfiguration für alle nachfolgenden Testszenarien

Schwärme	1
Initiale Schwarmgröße	20
Zeitschritte	250
Frequenz (min./max.)	0.0 / 1.3
Pulsrate / γ	0.7 / 0.9
Lautstärke / α	1.0 / 0.9

7.2.1 Anzahl der Schwärme

Die Anzahl der Schwärme ist aus zweierlei Hinsicht interessant: zum einen kommunizieren die Schwärme miteinander indem sie ihre besten Lösungen miteinander teilen und das kann sich positiv auf das Gesamtergebnis auswirken. Zum anderen können mehrere Schwärme die Laufzeit verbessern, denn diese sind parallel implementiert und arbeiten unabhängig von einander – mit Ausnahme der kommunizierten besten Lösungen. Um einen

Eindruck zu vermitteln, wie BatCEP von der Anzahl der Schwärme beeinflusst wird, sind Testergebnisse mit unterschiedlichen Schwarmgrößen in Tabelle 14 und Abbildung 33 dargelegt. Eine weitere Information: der Test-Computer verfügt über 4 CPU-Kerne.

Tabelle 14: BatCEP: Kennwerte für den Einsatz mehrerer Schwärme

Anzahl der Schwärme	**1**	**2**	**3**	**4**
Effektive Schwarmgröße	139	156	180	175
Lösungen pro Testlauf	34750	78000	135000	175000
Lösungen insgesamt	347500	780000	1350000	1750000
Erreichte Fitness	0,73	0,86	0,93	**0,99**
∅ Laufzeit pro Testlauf	60s	86s	73s	93s
Laufzeit über alle Testläufe	600s	868s	736s	932s

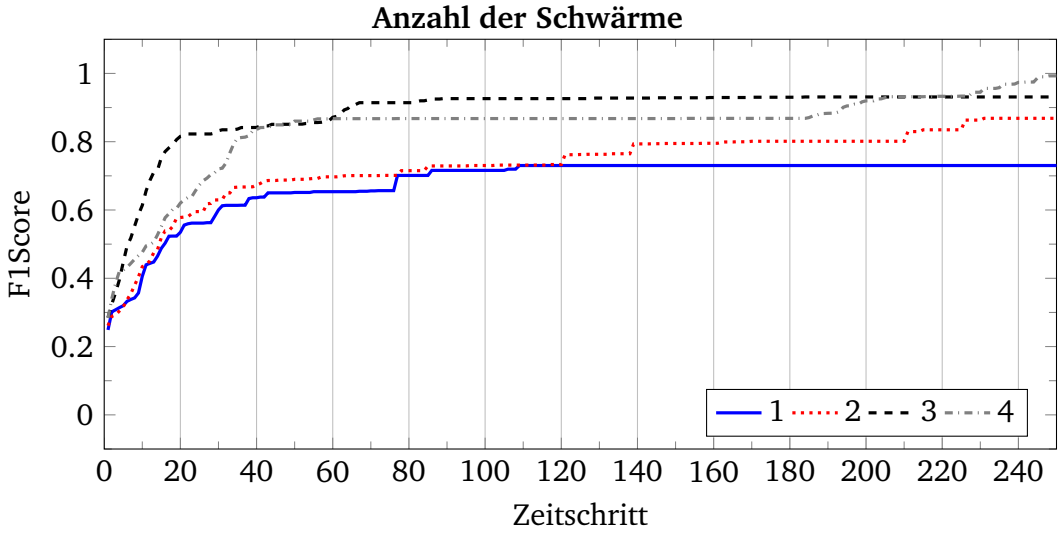


Abbildung 33: BatCEP: Kurvenverläufe für den Einsatz mehrerer Schwärme

Betrachtet man die Ergebniskurven so fällt auf, dass alle Tests in etwa den gleichen Anstieg aufweisen. Auffällig ist, dass sowohl mit einem als auch mit drei Schwärmen etwa ab Zeitschritt 100 das Maximum erreicht ist. Mit zwei und vier Schwärmen ist dies nicht der Fall, dort erfolgt ein weiterer Anstieg etwa ab Zeitschritt 200, wobei die Gründe hierfür nicht direkt ersichtlich sind.

Weiterhin sind hier interessante Laufzeiten zu beobachten: zwei Schwärme laufen länger als drei und erreichen zugleich eine schlechtere maximale Fitness. Am besten schneidet der Test mit vier Schwärmen ab. Dabei wurden zum einen die meisten Lösungen erzeugt und zum anderen eine hervorragende Fitness von annähernd 1,0 (also 100%) erreicht. Insgesamt bleibt festzuhalten: je mehr Schwärme eingesetzt werden, desto besser wird das Ergebnis.

In zukünftigen Projekten könnten diesbezüglich weitere Evaluierungen durchgeführt werden. Dabei wäre es interessant zu wissen wie BatCEP mit noch mehr CPU-Kernen und Schwärmen konvergiert aber auch wie das Verhalten mit mehreren Threads auf vier Kernen ist. Zudem wurden hier alle Schwärme mit denselben Attribut-Werten initialisiert. Es wäre jedoch interessant zu wissen, wie sich unterschiedliche Attribut-Werte für verschiedene Schwärme auf das Optimierungsverfahren auswirken. So könnte besonders die Kombination von Schwärmen für eine lokale Optimierung und Schwärmen für eine globale Optimierung zu guten Ergebnissen führen.

7.2.2 Schwarmgröße

Die Schwarmgröße hat einen nicht unerheblichen Einfluss auf die Güte der Lösungen, denn intuitiv würde man erwarten: je größer der initiale Schwarm ist, desto besser ist der Suchraum abgedeckt und umso einfacher lassen sich daraus gute Lösungen berechnen. Die nachfolgenden Ergebnisse sollen Aufschluss darüber bringen, ob sich dies für BatCEP bestätigt (Tabelle 15 und Abbildung 34).

Tabelle 15: BatCEP: Kennwerte für unterschiedliche Schwarmgrößen

Initiale Schwarmgröße	**10**	**40**	**70**	**100**
Effektive Schwarmgröße	67	353	686	896
Lösungen pro Testlauf	16750	88250	171500	224000
Lösungen insgesamt	167500	882500	1715000	2240000
Erreichte Fitness	0,77	0,86	**0,93**	0,86
∅ Laufzeit pro Testlauf	28s	128s	212s	326s
Laufzeit über alle Testläufe	280s	1280s	2120s	3265s

Die intuitive Annahme bestätigt sich hier nicht, denn das beste Ergebnis wird bereits mit einer Schwarmgröße von 70 Fledermäusen erreicht. Weiterhin fällt auf, dass alle Tests ihre maximale Fitness bereits weit vor dem letzten Zeitschritt erreichen. Im Durchschnitt zwischen Zeitschritt 50 und 150. Das ist eine nicht unwesentliche Feststellung für das Finden der optimalen Konfiguration, denn die Anzahl der Zeitschritte hat logischerweise einen erheblichen Einfluss auf die Laufzeit.

7.2.3 Zeitschritte

Ähnlich der Schwarmgröße sind auch die Zeitschritte maßgeblich an der Abdeckung des Suchraums beteiligt. Stellt man sich einen kleinen, initialen Schwarm vor der jedoch überdurchschnittlich häufig aktualisiert (also neu positioniert) wird und dabei im Idealfall nicht in einem falschen globalen Maximum mündet, dann deckt dieser kleine Schwarm den Suchraum ebenfalls gut ab. Zu erwarten ist theoretisch, dass sich das Ergebnis mit zunehmender Anzahl an Zeitschritten verbessert. Jedoch wurde im vorherigen Abschnitt

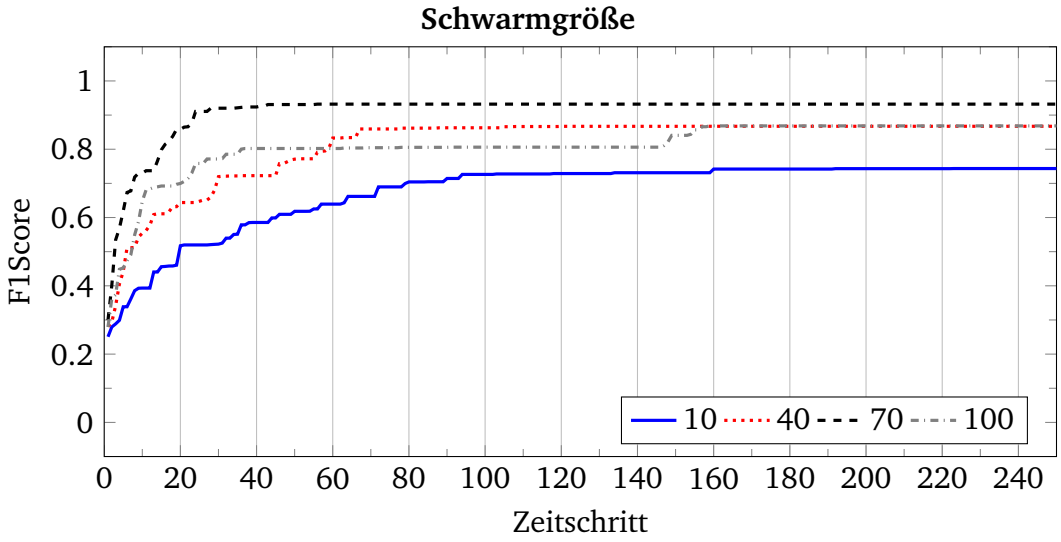


Abbildung 34: BatCEP: Kurvenverläufe für unterschiedliche Schwarmgrößen

festgestellt, dass sich die Ergebnisse ab einem gewissen Zeitschritt nicht mehr signifikant verbessern. Nachfolgend sind die entsprechenden Testergebnisse aufgeführt (Tabelle 16 und Abbildung 35).

Tabelle 16: BatCEP: Kennwerte für unterschiedlich viele Zeitschritte

Zeitschritte	**50**	**150**	**300**	**450**
Effektive Schwarmgröße	129	136	141	151
Lösungen pro Testlauf	6450	20400	42300	67950
Lösungen insgesamt	64500	204000	423000	679500
Erreichte Fitness	0,64	0,72	0,85	**0,86**
∅ Laufzeit pro Testlauf	9s	32s	69s	99s
Laufzeit über alle Testläufe	96s	322s	695s	994s

Die Erwartungen bestätigen sich. Es ist beobachten, dass mit zunehmenden Zeitschritten auch die Güte der Lösungen ansteigt. Insbesondere ab Zeitschritt 430 steigt der Fitness-Wert noch ein mal geringfügig an. Gleichzeitig erhöht sich die Laufzeit, was auch zu erwarten ist. Kurzum: je mehr Zeitschritte man vorgibt, desto öfter wird aktualisiert und umso besser ist das Ergebnis am Ende. Dabei muss man jedoch beachten, dass nicht die Anzahl der Zeitschritte allein maßgebend ist, denn für diesen Test hätte prinzipiell auch ein einziger Durchlauf ausgereicht, dennoch wurde BatCEP der Vollständigkeit halber vier mal gestartet. Die Schwankungen zwischen den Kurven zeigen dabei wie sehr BatCEP (beziehungsweise der BA) vom Zufall an vielen Stellen beeinflusst wird.

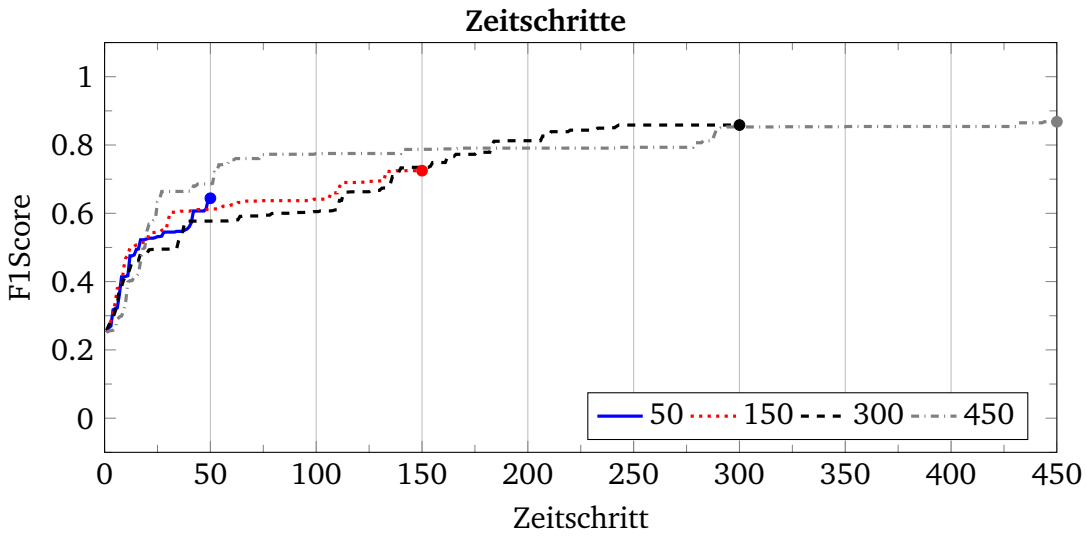


Abbildung 35: BatCEP: Kurvenverläufe für unterschiedlich viele Zeitschritte

7.2.4 Frequenz

Frequenz und Geschwindigkeit bestimmen die Anzahl der Iterationen während des Zufallsfluges (sowohl in `updateSolution` als auch in `optimizeSolution`). Je höher die Frequenz ist, desto mehr Operationen werden dort durchgeführt und das sollte sich zum einen in der Güte der Endergebnisse und zum anderen an der Laufzeit bemerkbar machen (Tabelle 17 und Abbildung 36).

Tabelle 17: BatCEP: Kennwerte für unterschiedliche Frequenzen

Frequenz	**0.1**	**0.6**	**1.5**	**2.5**
Effektive Schwarmgröße	29	64	149	249
Lösungen pro Testlauf	7250	16000	37250	62250
Lösungen insgesamt	72500	160000	372500	622500
Erreichte Fitness	0,48	0,66	0,65	**0,79**
∅ Laufzeit pro Testlauf	37s	39s	69s	86s
Laufzeit über alle Testläufe	378s	394s	699s	864s

Die Annahme bestätigt sich insofern, als eine höhere Frequenz zu mehr Iterationen und damit auch zu mehr Lösungen führt. Zugleich erhöht sich die Laufzeit. Es bleibt also festzuhalten, dass die Frequenz zusammen mit der Geschwindigkeit praktisch als Schleifen-Kriterium fungiert und ansonsten keine weiteren Einflüsse ausübt.

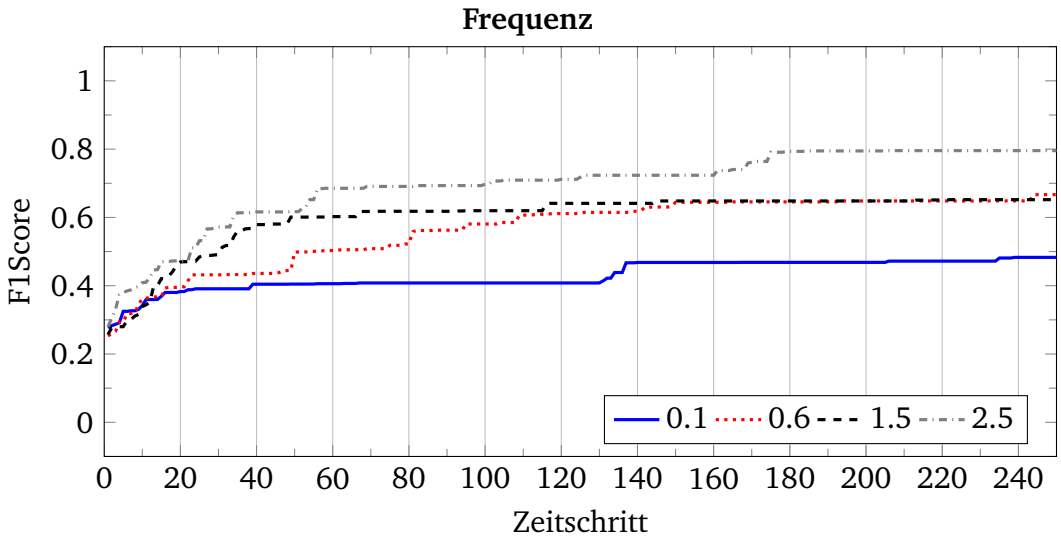


Abbildung 36: BatCEP: Kurvenverläufe für unterschiedliche Frequenzen

7.2.5 Pulsrate und γ

Die Pulsrate ist schon während des Programmablaufs ein Hinweis auf die aktuelle Fitness einer Lösung. Dabei ist ein niedriger Wert (<5) ein Indiz für eine schlechte Lösung und in diesem Fall wird mit erhöhter Wahrscheinlichkeit eine lokale Suche durchgeführt. Zur Erinnerung: im ersten Schritt der lokalen Suche kopiert sich die betrachtete Fledermaus die Lösung einer besseren Fledermaus (aus der Elite-Liste). Im zweiten Schritt aktualisiert sie ihre Kopie. Daraus lässt sich schlussfolgern, dass die lokale Suche eine eher schlechte Fledermaus zum globalen Maximum führt und das ist ein zielführender Schritt. Zugleich verringert das die Abdeckung des Suchraums, weil die beiden Fledermäuse zuerst unterschiedlich positioniert waren und sich jetzt in etwa an der gleichen Stelle befinden.

Von einer niedrigen, initialen Pulsrate erwartet man folglich viele lokale Suchen schon zu Anfang aber auch während des ganzen Programmablaufs – also eine starke Tendenz zu einer verfrühten Übernahme des globalen Maximums und das muss nicht zwingend gut sein, weil es den Suchraum schlecht abdeckt. Eine hohe Pulsrate sollte das Gegenteil bewirken: viele Zufallsflüge und wenige lokale Suchen. Das kann sich als sinnvoll erweisen und gute Ergebnisse hervorbringen, weil Zufallsflüge ebenfalls gute Lösungen produzieren können. Aber es kann sich auch als schlecht erweisen und zwar genau dann, wenn sich viele (oder sogar alle) Fledermäuse durch die Zufallsflüge vom Ziel entfernen (analog zur Schwarmexplosion bei der Partikelschwarmoptimierung). Die Messergebnisse mit unterschiedlichen Pulsraten sind aus Tabelle 18 und Abbildung 37 zu entnehmen.
Am besten schneidet BatCEP mit einer niedrigen Pulsrate von 0,1 ab. Daraus folgt, dass die Durchführung vieler lokaler Suchen als positiv anzusehen ist und das bestärkt die Annahme, dass die in der lokalen Suche durchgeführten Operationen wie beispielsweise die Radius-Aktualisierung zielführend sind.

Basierend auf der *besten Pulsrate (0,1)*, wird die Veränderung von γ untersucht. γ ist ein wesentlicher Einflussfaktor, denn er erhöht die Pulsrate immer dann, wenn die Fleder-

Tabelle 18: BatCEP: Kennwerte mit unterschiedlichen Pulsraten

Pulsrate	**0.1**	**0.3**	**0.6**	**0.9**
Effektive Schwarmgröße	254	206	163	91
Lösungen pro Testlauf	63500	51500	40750	22750
Lösungen insgesamt	635000	515000	407500	227500
Erreichte Fitness	**0,863**	0,860	0,73	0,76
∅ Laufzeit pro Testlauf	81s	107s	66s	45s
Laufzeit über alle Testläufe	811s	1076s	661s	452s

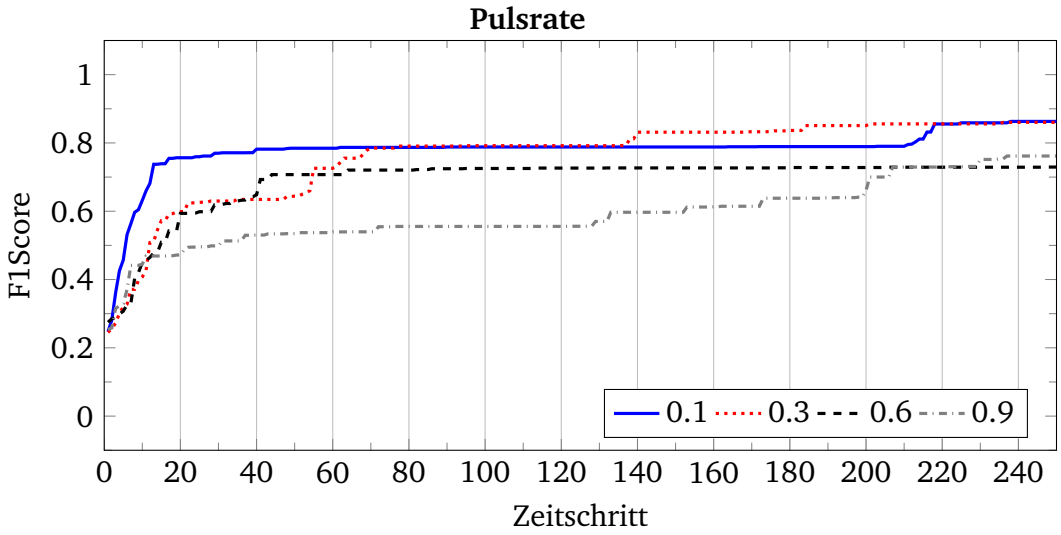


Abbildung 37: BatCEP: Kurvenverläufe mit unterschiedlichen Pulsraten

maus die nächste global beste Lösung hervorgebracht hat. Ein großer Betrag von γ (~1) verändert die Pulsrate sehr schnell. Folglich stagniert der Programmablauf hinsichtlich der Pulsrate nach wenigen Zeitschritten. Ein niedriger Betrag (~0, aber nicht genau 0!) hingegen, gleicht die Pulsrate nur langsam ihrem Endwert r_i^0 an, was die Pulsrate über viele Zeitschritte hinweg stetig und geringfügig verändert. Die Messergebnisse sind in Tabelle 19 und in Abbildung 38 einzusehen.
Das ideale γ scheint in dieser Konfiguration einen Betrag von 1 zu haben. Generell sind die Unterschiede in den Ergebnissen aber nicht allzu groß und ein lineares Verhalten beim Anstieg von γ liegt auch nicht vor. Wie also lassen sich die Ergebnisse interpretieren? Bei einem γ von 1 nähert sich die Pulsrate r_i^t sehr schnell der initialen Pulsrate r_i^0 an. Sehr schnell bedeutet, dass die Differenz zwischen r_i^t und r_i^0 etwa nach dem zehnten Zeitschritt unwesentlich klein ist. Ein γ mit einem Betrag knapp über 0 hingegen benötigt wesentlich mehr Zeitschritte. Beispiel: $\gamma = 0,01$ erfordert 1000 Zeitschritte *mit* Annäherung an die Beute für eine ähnlich geringe Enddifferenz.

Tabelle 19: BatCEP: Kennwerte für unterschiedliche γ

Pulsrate	0.1			
γ	**0.1**	**0.3**	**0.6**	**1.0**
Effektive Schwarmgröße	243	225	245	236
Lösungen pro Testlauf	60750	56250	61250	59000
Lösungen insgesamt	607500	562500	612500	590000
Erreichte Fitness	0,8	0,75	0,8	**0,86**
∅ Laufzeit pro Testlauf	106s	94s	99s	106s
Laufzeit über alle Testläufe	1069s	946s	990s	1061s

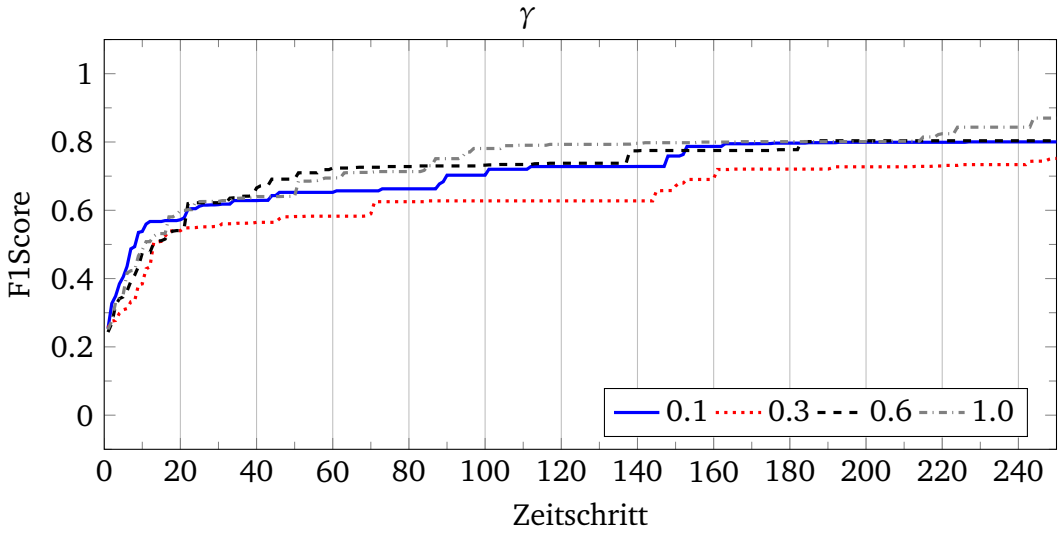


Abbildung 38: BatCEP: Kurvenverläufe für unterschiedliche γ

7.2.6 Lautstärke und α

Die Lautstärke wirkt an drei Stellen im Algorithmus: zunächst hat sie Einfluss auf die Aktualisierungsoperation im Zufallsflug, indem mit ihr zwischen Aktualisierung des ECT oder des ACT entschieden wird. Realisiert ist dies durch einen Vergleich mit einer Zufallsvariable β. Ist die Lautstärke dabei größer als β, dann wird der ECT aktualisiert und der ACT sonst.

Die zweite Stelle an der sie mitwirkt ist die Entscheidung, ob eine neue Lösung akzeptiert wird oder nicht (siehe Pseudocode 10). Auch hier wird sie mit β verglichen und nur wenn die Lautstärke größer ist, hat die neue Lösung eine Chance übernommen zu werden.

An dritter Stelle dient der Betrag der Durchschnittslautstärke, die sich logischerweise aus der Lautstärke aller Fledermäuse ableitet, als Schleifen-Bedingung. Einfach formuliert: eine große Lautstärke bedeutet eine große Durchschnittslautstärke und diese bewirkt eine große Anzahl an Schleifendurchläufen in der lokalen Suche. Bei einer großen Lautstärke

sind also mehr Aktualisierungsoperationen zu erwarten als bei einer niedrigen. Intuitiv erwartet man folglich bessere Ergebnisse bei einer höheren Lautstärke. Die Testergebnisse sind in Tabelle 20 und Abbildung 39 einzusehen.

Tabelle 20: BatCEP: Kennwerte für unterschiedliche Lautstärken

Lautstärke	**0.1**	**0.6**	**1.3**	**2.0**
Effektive Schwarmgröße	42	86	166	201
Lösungen pro Testlauf	10500	21500	41500	50250
Lösungen insgesamt	105000	215000	415000	502500
Erreichte Fitness	0,6	0,55	0,79	**0,91**
∅ Laufzeit pro Testlauf	37s	40s	66s	82s
Laufzeit über alle Testläufe	374s	409s	662s	828s

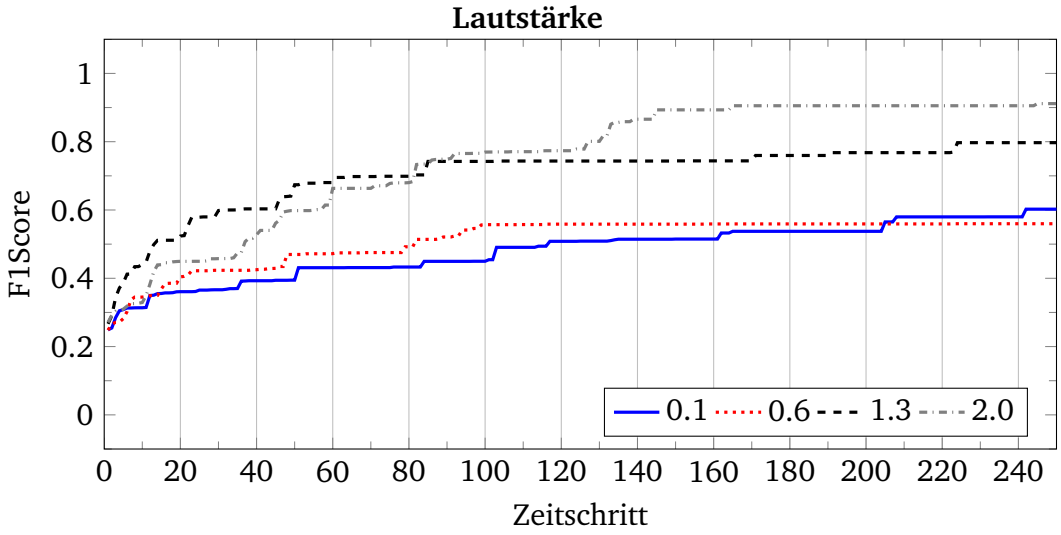


Abbildung 39: BatCEP: Kurvenverläufe für unterschiedliche Lautstärken

Ein wesentlicher Einflussfaktor an dieser Stelle ist α, denn damit wird die Lautstärke reduziert. Das geschieht immer dann, wenn die zugehörige Lösung als neue global beste Lösung übernommen wird und damit hat α entscheidenden Einfluss auf den Verlauf des Verfahrens. Somit ist bei einem klein gewählten α ein schnelles Absenken und bei einem groß gewählten ein langsames Absenken der Lautstärke zu erwarten und das wirkt sich folgendermaßen auf den Programmablauf aus (Tabelle 21):
Die Testergebnisse zeigen, dass die effektive Schwarmgröße zusammen mit α ansteigt. Mit dem schnellen Absenken der Lautstärke bei einem kleinen α geht ein schnelles Absenken der Durchschnittslautstärke einher und somit werden weniger Operationen pro Zeitschritt (während der lokalen Suche) durchgeführt. Bei einem großen α tritt der gegensätzliche Fall ein.

Tabelle 21: Programmverhalten bei unterschiedlichen Beträgen von α

klein: $\alpha \sim 0$	**groß: $\alpha \sim 1$**
Der rein zufällige Flug wird bereits nach kurzer Zeit mehr ACT-Aktualisierungen, als ECT-Aktualisierungen durchführen.	Der rein zufällige Flug führt häufiger ECT-Aktualisierungen durch.
Neue Lösungen von bereits erfolgreichen Fledermäusen werden schon nach wenigen Zeitschritten nicht mehr akzeptiert.	Neue Lösungen von bereits erfolgreichen Fledermäusen werden während des Programmablauf zumeist akzeptiert (vorausgesetzt sie sind besser als das globale Maximum).
Die Anzahl der Aktualisierungsoperationen sinkt mit zunehmenden Zeitschritten stark ab, weil die Durchschnittslautstärke rasant absinkt.	Die Anzahl der Aktualisierungsoperationen bleibt von Programmstart bis -ende annähernd gleich.

Tabelle 22: BatCEP: Kennwerte für unterschiedliche α

Lautstärke	2.0			
α	**0.1**	**0.3**	**0.6**	**1.0**
Effektive Schwarmgröße	76	94	174	259
Lösungen pro Testlauf	19000	23500	43500	64750
Lösungen insgesamt	190000	235000	435000	647500
Erreichte Fitness	0,5	0,57	0,83	**1,0**
∅ Laufzeit pro Testlauf	44s	43s	69s	77s
Laufzeit über alle Testläufe	441s	439s	696s	775s

Gleichzeitig erreicht das beste Testergebnis mit einem α von 1 hier erstmals eine F1Score von 100% über 10 Testdurchläufe und das lässt sich logischerweise auf die Effektivität der lokalen Suche zurückführen, denn diese führt bei jedem Aufruf mit dieser Konfiguration *zwanzig* Aktualisierungsoperationen durch.

7.2.7 Zusammenfassung – Die „ideale“ Konfiguration

Tabelle 23 repräsentiert die „theoretisch“ ideale Konfiguration. Ob sie tatsächlich ideal ist, muss im Zuge zukünftiger Projekte überprüft werden. Grund hierfür ist die gegenseitige Beeinflussung aller Attribute, für die es unzählige Kombinationen gibt. Beispielsweise ist es häufig so, dass gute Endergebnisse aus einer großen Anzahl insgesamt erzeugter Lösun-

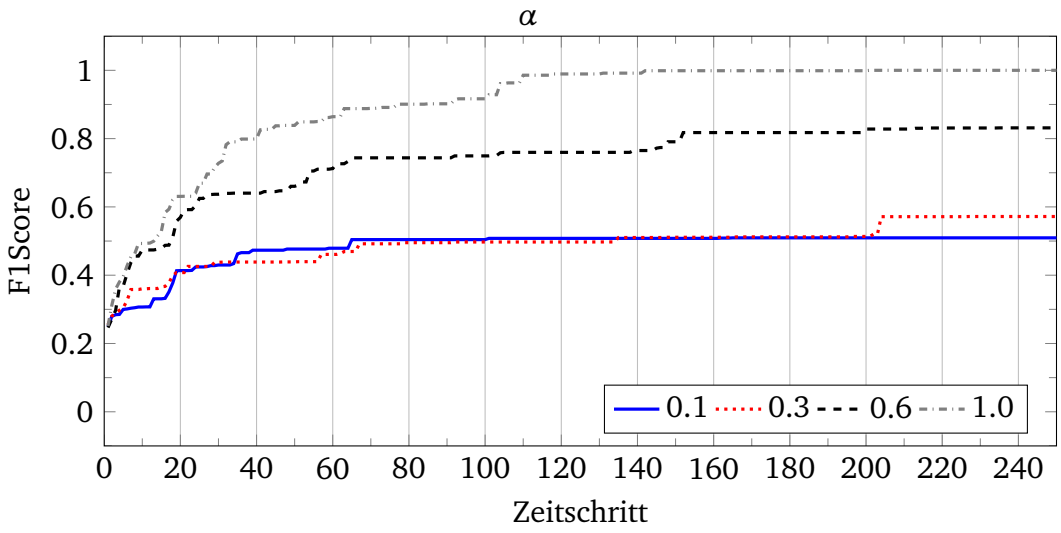


Abbildung 40: BatCEP: Kurvenverläufe für unterschiedliche α

gen resultieren und der Einsatz von n Schwärmen erzeugt die n-fache Anzahl an Lösungen pro Zeitschritt. Daraus folgt, dass womöglich ein Viertel der in der idealen Konfiguration angegebenen Zeitschritte ausreichend wären. Eine repräsentative Anzahl aus allen Kombinationen kann im Rahmen dieser Masterarbeit nicht evaluiert werden, weil dies den Rahmen sprengen würde. Zu der Tabelle 23 seien noch die folgenden, zusammengefass-

Tabelle 23: Die ideale BatCEP-Konfiguration gemäß der vorangegangenen Testergebnisse

Anzahl der Schwärme	4
Initiale Schwarmgröße	50 - 70
Zeitschritte	100 - 450
Frequenz	≥2,5
Pulsrate	0,1 - 0,3
γ	1,0
Lautstärke	≥2,0
α	1,0

ten Informationen gegeben:

- Die Anzahl der Schwärme ist mit *vier* nicht ausreichend intensiv getestet worden. Es kann durchaus sein, dass noch mehr Schwärme (gegebenenfalls sogar mehr Schwärme als vorhandene CPU-Kerne) zu noch besseren Ergebnissen führen.
- Beobachtungen zufolge kann man die Anzahl der Zeitschritte verringern, wenn man zugleich mehr Schwärme einsetzt. Dies ist nachvollziehbar, denn eine Halbierung der Zeitschritte wird durch eine Verdopplung der Schwärme hinsichtlich der erzeugten Lösungen in etwa kompensiert.

- Frequenz und Lautstärke können mit noch größeren Beträgen sehr wahrscheinlich noch bessere Ergebnisse hervorbringen, denn letztlich führen größere Beträge zu mehr Aktualisierungsoperationen.
- Geringe Pulsraten führen zu vielen lokalen Suchen und damit – wie zu beobachten war – zu guten Ergebnissen. Doch darf nicht außer Acht gelassen werden, dass dies einer starken Tendenz zur *Ausbeutung* entspricht, wodurch ein Ungleichgewicht zwischen Erkundung und Ausbeutung geschaffen wird.

Insgesamt erreicht BatCEP zumeist gute Ergebnisse und muss dabei oftmals weniger Lösungen erzeugen als vergleichbare Verfahren. Zugleich scheint die lokale Suche seine besondere Stärke zu sein, denn gute Ergebnisse entstehen häufig durch eine Vielzahl lokaler Suchen.

7.3 Ansätze zur Verbesserung von BatCEP

Im Zuge der Entwicklung von BatCEP entstanden zu einigen Verfahren Verbesserungsideen, die in diesem Abschnitt vorgestellt und experimentell getestet werden. Zunächst die sehr interessante Konfiguration von Wang, Chang und Zhang [68], mit verschieden initialisierten Schwärmen und der Frage, ob diese auch für BatCEP vorteilhaft ist. Dann ein Ansatz zur Verbesserung der Ramped half-and-half-Initialisierung, gefolgt von einer angepassten lokalen Suche, die nicht nur die beste Fledermaus mit einbezieht. Abschließend wird auf den Zufallsflug eingegangen, um zu testen ob der einfache, unbedingte Zufallsflug ausreichend wäre, um genauso gute Ergebnisse zu erzeugen wie mit dem zweigeteilten Zufallsflug.

7.3.1 Unterschiedlich initialisierte Schwärme

In diesem Experiment soll herausgefunden werden, ob BatCEP mit unterschiedlich initialisierten Schwärmen bessere Ergebnisse erzeugt als mit gleich initialisierten. Zur Erinnerung: in [68] wurde ein Multi-Schwarm-Modell für den BA entwickelt, das auf unterschiedlich initialisierte Schwärme setzt (Abschnitt 5.3). Dabei tendiert beispielsweise ein Schwarm zu lokalen Lösungen, ein anderer zu globalen Lösungen und ein dritter hat ein ausgewogenes Verhältnis zwischen lokalen und globalen Lösungen. BatCEP arbeitet wohlgemerkt nicht exakt nach diesem Modell aber eben auch mit mehreren Schwärmen. Somit ist es auch hierbei interessant zu wissen, ob sich unterschiedlich initialisierte Schwärme auf die Qualität der Lösungen auswirken.

In der Tabelle 24 sind die initialen Attribut-Werte der vier Programmdurchläufe in diesem Experiment aufgeführt. Dabei bezeichnet A einen Programmablauf, in dem vier unterschiedlich initialisierte Schwärme $\{S_1, S_2, S_3, S_4\}$ eingesetzt werden. S_1 tendiert stark zu lokalen Lösungen, S_2 und S_3 haben leichte Tendenzen zu lokalen beziehungsweise globalen Lösungen und S_4 tendiert stark zu globalen Lösungen. Bei B, C und D hingegen wurden alle vier Schwärme gleich initialisiert aber dennoch unterscheiden sie sich: bei B tendieren alle Schwärme zu lokalen Lösungen und damit sind es „erkundende“ Schwärme. Bei C haben alle Schwärme ein ausgewogenes Verhältnis zwischen lokalen und globalen Lösungen

und bei D wiederum tendieren alle Schwärme zu globalen Lösungen, damit sind es „ausbeutende“ Schwärme. Alle Schwärme haben dieselbe Größe und Anzahl der Zeitschritte und auch γ und α sind immer gleich.

Tabelle 24: Attribut-Werte aller Schwärme im Experiment 7.3.1, in dem BatCEP mit unterschiedlichen Schwärmen initialisiert wird.

	A				B	C	D
	S_1	S_2	S_3	S_4	$S_{1..4}$	$S_{1..4}$	$S_{1..4}$
Initiale Schwarmgröße	50						
Zeitschritte	100						
Frequenz	1.3	1.0	0.8	0.5	1.3	1.0	0.5
Pulsrate	1.0	0.8	0.4	0.1	1.0	0.6	0.1
γ	0.9						
Lautstärke	0.1	0.3	1.0	2.5	0.1	1.0	2.5
α	0.9						

Tabelle 25: Kennwerte im Experiment 7.3.1 mit unterschiedlich initialisierten Schwärmen.

	A	B	C	D
Anzahl der Schwärme	4	4	4	4
Effektive Schwarmgröße	460	125	458	1032
Lösungen pro Testlauf	184000	50000	183200	409200
Lösungen insgesamt	1840000	500000	1832000	4092000
Erreichte Fitness	0,93	0,81	**1,0**	0.87
∅ Laufzeit pro Testlauf	101s	69s	109s	148s
Laufzeit über alle Testläufe	1018s	692s	1093s	1480s

Unter Betrachtung der Ergebnisse lässt sich feststellen, dass C und damit der Programmdurchlauf mit dem ausgewogenen Verhältnis zwischen lokalen und globalen Lösungen das beste Ergebnis hervorbringt. C erzeugt „die perfekte Lösung“ mit einer F1Score von 100%. Der gemischte Programmablauf A erzeugt das zweit-beste Ergebnis mit einer Fitness von 93%. B und D hingegen überschreiten die 90%-Grenze nicht.

Dieses Resultat entspricht im Grunde genau dem, was man vom originalen BA erwarten würde. Denn dieser zeichnet sich besonders durch ein gutes Verhältnis zwischen lokalen und globalen Lösungen aus und erzeugt damit gute Ergebnisse. Obgleich im letzten Abschnitt die lokale Suche als „besondere Stärke“ bezeichnet wurde. Somit hätte man annehmen können, dass im Zuge dieses Experiments der Programmablauf D mit der starken

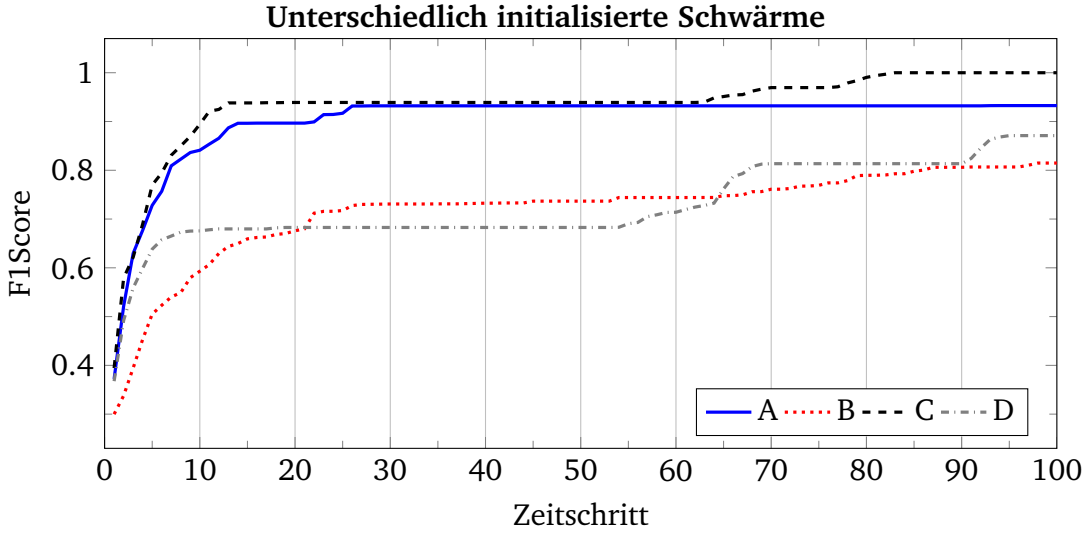


Abbildung 41: Kurvenverläufe des Experiments 7.3.1

Tendenz zur globalen Lösung das beste Ergebnis erzeugt. Doch offenbar ist der Suchraum dabei im Vorfeld nicht genügend abgedeckt und so mündet das Verfahren zu früh in einem globalen aber falschen Maximum, aus dem es auch mithilfe der optimierenden Operationen wie beispielsweise der Radius-Aktualisierung nicht mehr entfliehen kann.

Überdies kann das Experiment unter Verwendung eines anderen Modells, wie beispielsweise dem aus [68] zu ganz anderen Ergebnissen führen. Grund hierfür ist, dass sich die Strategie der dortigen Schwarm-Kommunikation von der in BatCEP verwendeten Strategie unterscheidet (Abschnitt 5.3).

7.3.2 Verbesserte RHH-Initialisierung „Fit-Schwarm“

Im Zuge des ersten Testszenarios aus Abschnitt 7.1.1 entstand der Gedanke einer verbesserten RHH-Initialisierung – obwohl die Erzeugnisse von RHH gemäß der Tabelle 6 bereits gut sind.

Die Grundidee ist durch Vorgabe einer Mindest-Fitness die schlechtesten Fledermäuse im Vorfeld auszusortieren und durch bessere zu ersetzen. Um dies zu realisieren erzeugt man in einer Schleife zunächst einen normalen Schwarm mit RHH und ordnet die Fledermäuse nach absteigender Fitness. Anschließend werden alle Fledermäuse verworfen, deren Fitness kleiner ist als die Grenze: $\tau * F_{best}$. Wobei τ ein vom Benutzer bestimmter Faktor aus dem Wertebereich [0,1] ist und F_{best} bezeichnet den Fitness-Wert der besten Fledermaus. In der nächsten Iteration werden wieder nur diejenigen Fledermäuse übernommen, deren Fitness über der Grenze liegt und die Schleife wird solange wiederholt, bis der Schwarm die geforderte Größe hat. Dazu folgendes Beispiel: τ wird auf 0,8 festgelegt und ein Schwarm aus drei Fledermäusen S={1, 2, 3} initialisiert. Die Fledermaus 1 erreicht eine F1Score von 0,41; 2 erreicht eine F1Score von 0,24 und 3 eine F1Score von 0,37. Damit hat 1 die beste Fitness $F_{best} = 0,41$ und daraus resultiert, dass alle anderen Fledermäuse die Fitness-Grenze von $0,8 * 0,41 = 0,328$ überschreiten müssen oder

ansonsten aussortiert werden. Das bedeutet 2 wird entfernt und 3 bleibt erhalten. Da der Schwarm jetzt nur noch aus zwei Fledermäusen besteht, werden neue Fledermäuse erzeugt und die erste die die Fitness-Grenze von 0,328 überschreitet wird dem Schwarm hinzugefügt.

Leider hat sich dieser Ansatz als nur mäßig erfolgreich gezeigt, denn die Rechenzeit steigt in Abhängigkeit von τ an, während die von BatCEP erzeugten Lösungen, basierend auf dieser Initialisierung, sogar schlechter sind als vorher. Eine mögliche Erklärung hierfür ist, dass sich die so erzeugten Lösungen weniger voneinander unterscheiden und damit schlecht auf den Suchraum verteilt sind. Dennoch erscheint die Idee einer verbesserten RHH-Initialisierung nicht abwegig und könnte in zukünftigen Projekten verbessert und evaluiert werden, um gegebenenfalls doch noch einen Nutzen daraus zu ziehen. Vielleicht kann eine Verbesserung durch messen der „Unterschiedlichkeit" der Regeln (im Englischen: *Diversity*) erzielt werden. Gemeint ist ein Maß mit dem die Verteilung der Fledermäuse im Suchraum gemessen werden kann. Entsprechende Gleichungen sowie Definitionen der *Diversity* in Bezug auf Assoziationsregeln sind unter anderem in [22, 31] zu finden.

7.3.3 Angepasste lokale Suche

Während der Entwicklung der lokalen Suche kam die folgende Frage auf: „Könnte die lokale Suche bessere Ergebnisse hervorbringen, wenn anstelle der global besten Lösung *eine der global besten Lösungen* zufällig ausgewählt wird und als Referenz dient?"

Für einige Datensätze ist es möglich unterschiedliche, CEP-Regeln mit einer Fitness von 100% zu erzeugen. Gemeint sind unterschiedliche Lösungen, von denen jede einzelne die Ursachen für Z korrekt repräsentiert. Kann es dann nicht auch zielführend sein, neben der besten Lösung eines Schwarmes, beispielsweise die besten drei Lösungen in die lokale Suche einzubringen? Dies entspräche einem abgeschwächten Verhältnis zwischen lokalem und globalem Maximum.

Diese Theorie ist durchaus berechtigt, denn sie sorgt gegebenenfalls für eine bessere Verteilung der Fledermäuse im Suchraum und verringert die Wahrscheinlichkeit dafür in einem falschen globalen Maximum zu konvergieren. Um die Theorie zu validieren wurde ein experimenteller Test durchgeführt, dessen Ergebnisse in Tabelle 26 und Abbildung 42 einzusehen sind. Die Attribute wurden der besten Konfiguration aus der Tabelle 23 gemäß initialisiert.

Die mit *A* ausgezeichnete Kurve zeigt die Fitness unter Verwendung der „normalen" lokalen Suche, bei der immer nur *die global beste* Fledermaus referenziert wird. *B* zeigt die Fitness unter Verwendung einer modifizierten lokalen Suche, die *eine der besten* Fledermäuse referenziert.

Unter Anbetracht der Ergebnisse lässt sich feststellen, dass beide Programmausführungen nach weit weniger als 300 Zeitschritten eine Fitness von 100% erreichen – also „eine perfekte Lösung hervorbringen". Dieser Umstand spricht eindeutig dafür, dass die ideale Konfiguration noch verbesserungswürdig ist und zwar insofern, als die gegenseitige Beeinflussung der Attribute dort mit einfließen muss.

Tabelle 26: Kennwerte des Experiments mit einer angepassten lokalen Suche

	A	B
Effektive Schwarmgröße	2143	2188
Lösungen pro Testlauf	2571600	2625600
Lösungen insgesamt	25716000	26256000
Erreichte Fitness	1.0	**1.0**
∅ Laufzeit pro Testlauf	524s	616s
Laufzeit über alle Testläufe	5240s	6160s

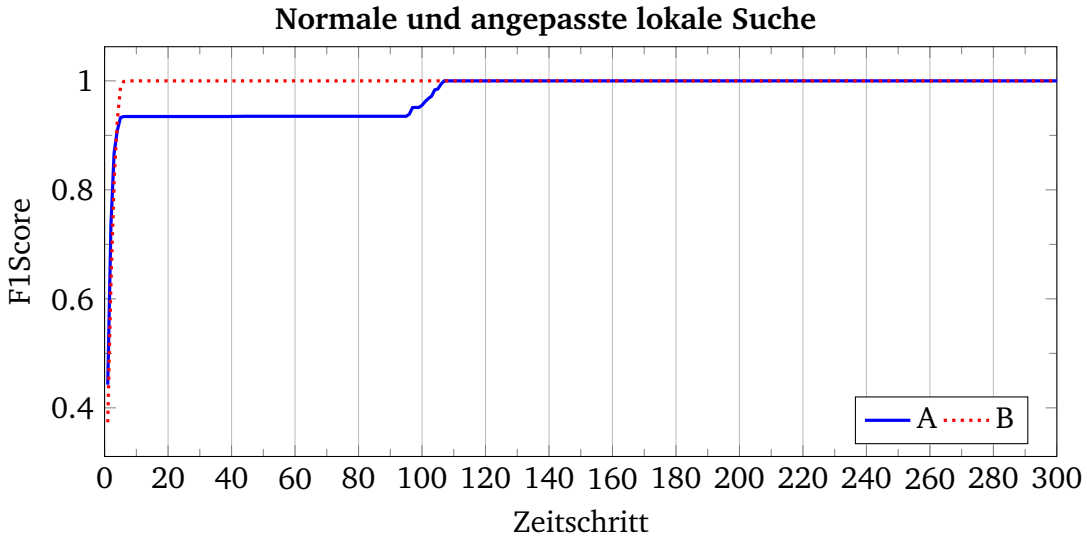


Abbildung 42: Kurvenverläufe des Experiments mit einer angepassten lokalen Suche.

Weiterhin zeigen die Ergebnisse, dass die angepasste lokale Suche (B) mit gerade einmal 7 Zeitschritten wesentlich schneller mit einem guten Ergebnis konvergiert, als die „normale“ lokale Suche (A) mit 107 Zeitschritten. Die entsprechenden Laufzeiten mit derart verringerten Zeitschritten wären dann etwa 14 Sekunden für B und 3 Minuten für A. Dabei handelt es sich wohlgemerkt um Durchschnittswerte über zehn Testdurchläufe. Insgesamt erzeugen beide Varianten der lokalen Suche gute Lösungen. In zukünftigen Projekten könnte geprüft werden, ob es tatsächlich Unterschiede hinsichtlich der Abdeckung des Suchraums gibt.

7.3.4 Nur der „einfache“ Zufallsflug

Der originale BA verwendet stets den einfachen Zufallsflug, bei dem sich die Fledermaus rein zufällig im Suchraum bewegt. Auf BatCEP übertragen hieße das, dass nur der in Abschnitt 5.2.2 beschriebene, *rein zufällige Zufallsflug* eingesetzt wird. Auf den optimierenden, bedingt zufälligen Flug würde verzichtet. In diesem Experiment wird getestet, wie

sich dies auf BatCEP auswirken würde. Die Ergebnisse sind in Tabelle 27 und Abbildung 43 dargestellt und die Attribute wurden der besten Konfiguration aus Tabelle 23 gemäß initialisiert.

Tabelle 27: Kennwerte des Experiments mit dem „einfachen“ Zufallsflug

	A	B
Effektive Schwarmgröße	2192	1980
Lösungen pro Testlauf	2630400	2376000
Lösungen insgesamt	26304000	23760000
Erreichte Fitness	**1.0**	0.87
∅ Laufzeit pro Testlauf	567s	511s
Laufzeit über alle Testläufe	5670s	5110s

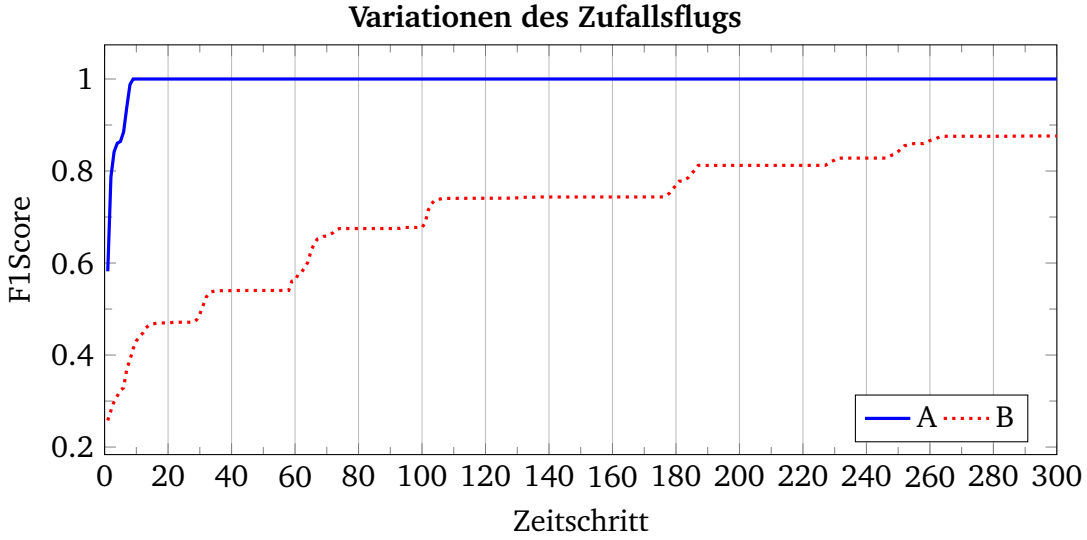


*Abbildung 43: Kurvenverläufe des Experiments mit dem „einfachen“ Zufallsflug. Die mit **A** ausgezeichnete Kurve zeigt die Fitness unter Verwendung des in Abschnitt 5.2.2 beschriebenen Zufallsfluges, der sowohl den rein zufälligen als auch den bedingt zufälligen Flug umschließt. **B** zeigt die Fitness unter Verwendung des rein zufälligen (einfachen) Zufallsfluges.*

Zunächst ist erkennbar, dass BatCEP nur mit dem rein zufälligen Flug ebenfalls gut funktioniert und akzeptable Ergebnisse hervorbringt. So wird über zehn Testdurchläufe mit je 300 Zeitschritten eine Fitness von 87% erreicht. Zugleich weisen die Ergebniskurven deutlich auf die *Nützlichkeit des optimierenden, bedingt zufälligen Flugs* hin – beziehungsweise auf die Kombination beider Flüge. So erreicht BatCEP mit der Optimierung bereits nach dem neunten Zeitschritt eine Fitness von 100%.

7.4 Erproben der Berechnungsgrenzen von BatCEP

Die in diesem Abschnitt gezeigten Experimente sollen die Berechnungsgrenzen von BatCEP aufzeigen. Ziel ist es einen Überblick darüber zu verschaffen, bei welchen Arten von Daten gute und bei welchen schlechte Ergebnisse erzeugt werden. Eigens dafür wurden 18 Datensätze basierend auf vier CEP-Regeln $R*_{1..4}$ erzeugt, die zunehmend komplexer werden. Alle Trainingsdatensätze haben dieselbe Grundkonfiguration:

Ereignistypen: 3

Ereignisinstanzen: 1500

Anzahl der Sensoren (ID): 1

Zeitdifferenz zwischen den Ereignisinstanzen: 1 Sekunde

Anzahl der Attribute pro Ereignisinstanz: 1 (nur ID)

Davon weicht jeder Datensatz wie nachfolgend beschrieben ab, um zusätzlich zu den verschieden komplexen CEP-Regeln nochmals zunehmend mehr Komplexität zu erzeugen:

Datensätze 1.1 bis 1.6 basieren alle auf der CEP-Regel $R*_1$

$(A \rightarrow C)[\text{win:time:20sec}] \Rightarrow Z$

1.1 steht für die Grundkonfiguration; 1.2 hat 3 ID-Werte; 1.3 hat 6 ID-Werte; 1.4 hat 3 Ereignistypen; 1.5 hat 9 Ereignistypen und 1.6 hat 12 Ereignistypen.

Datensätze 2.1 bis 2.6 basieren alle auf der CEP-Regel $R*_2$

$(A \wedge B)[\text{win:time:20sec}] \Rightarrow Z$

2.1 steht für die Grundkonfiguration; 2.2 hat 3 ID-Werte; 2.3 hat 6 ID-Werte; 2.4 hat 3 Ereignistypen; 2.5 hat 9 Ereignistypen und 2.6 hat 12 Ereignistypen.

Datensätze 3.1 bis 3.3 basieren alle auf der CEP-Regel $R*_3$

$(A \wedge B) \wedge (A.ID = B.ID)[\text{win:time:20sec}] \Rightarrow Z$

3.1 steht für die Grundkonfiguration; 3.2 hat 3 ID-Werte und 3.3 hat 6 ID-Werte.

Datensätze 4.1 bis 4.3 basieren alle auf der CEP-Regel $R*_4$

$((A \wedge B) \rightarrow C) \wedge (A.ID = B.ID \wedge B.ID = C.ID)[\text{win:time:20sec}] \Rightarrow Z$

4.1 steht für die Grundkonfiguration; 4.2 hat 3 IDs und 4.3 hat 6 IDs.

Für jeden Testlauf wurde BatCEP mit der Konfiguration aus der Tabelle 28 initialisiert. Diese entspricht nicht ganz der idealen Konfiguration, was mit der Größe der Trainingsdatensätze (1500 Ereignisse pro Datensatz) zu begründen ist. Mit einer Konfiguration die der idealen Konfiguration entspricht, würde die Laufzeit für alle Testläufe bei dieser Menge solch großer Datensätze nicht vertretbar sein. Grund hierfür ist die Tatsache, dass jede einzelne Positionsevaluation einen Suchlauf über den ganzen Datensatz erfordert. Wie schon in jedem Szenario zuvor, werden auch hier jeweils zehn Testdurchläufe pro Datensatz durchgeführt. Die Ergebniskurven zeigen die darüber gemittelten Werte.

Tabelle 28: Konfiguration zum Erproben der Berechnungsgrenzen von BatCEP.

Anzahl der Schwärme	4
Initiale Schwarmgröße	50
Zeitschritte	100
Frequenz	1,0
Pulsrate	0,6
γ	0,9
Lautstärke	1,0
α	0,9

Test mit den Datensätzen 1.1 bis 1.6

Abbildung 44 zeigt die Ergebniskurven für den Test der Datensätze 1.1 bis 1.6. Zunächst ist darauf erkennbar, dass die Mehrheit der Testläufe mit einem sehr guten Ergebnis beendet. Schwierigkeiten hat BatCEP mit dem Datensatz 1.2, der drei verschiedene Werte für das ID-Attribut vorhält.

Tabelle 29: Kennwerte für die Evaluation von BatCEP auf den Datensätzen 1.1 bis 1.6

Datensatz	1.1	1.2	1.3	1.4	1.5	1.6
Effektive Schwarmgröße	176	249	217	190	244	219
Lösungen pro Testlauf	70400	99600	86800	76000	97600	87600
Lösungen insgesamt	704000	996000	868000	760000	976000	876000
Erreichte Fitness	1.0	0.9	1.0	1.0	1.0	0.98
∅ Laufzeit pro Testlauf	355s	579s	197s	314s	159s	219s
Laufzeit über alle Testläufe	3550s	5790ss	1970s	3140s	1590s	2190s

Tabelle 29 führt die Kennwerte für den Test der Datensätze 1.1 bis 1.6 auf. Auch hier ist anhand der stark erhöhten Laufzeit zu erkennen, das BatCEP viele Aktualisierungsschritte durchgeführt hat – letztlich waren scheinbar viele davon nur wenig erfolgreich. Da stellt sich die Frage nach dem Grund für die Schwierigkeiten bei einem Datensatz, der genau drei unterschiedliche Werte für das ID-Attribut vorhält. Zumal die CEP-Regel $R*_1$ nicht einmal Kontextbedingungen stellt. Sehr wahrscheinlich spielen dabei folgende Gegebenheiten zusammen: zunächst einmal sind die Datensätze recht groß, was das Erraten korrekter Fenstergrößen erschwert. Dann kommt dazu, dass die Vergleichsoperatoren „>“ und „<“ in der Kontextbedingung zu wenige Ereignisse im Fenster ausschließen. Mit mehr möglichen Werten für das ID-Attribut würden damit gegebenenfalls mehr Ereignisse aus einem Fenster ausgeschlossen werden, was dazu führt, dass die Regel weniger häufig feuert.

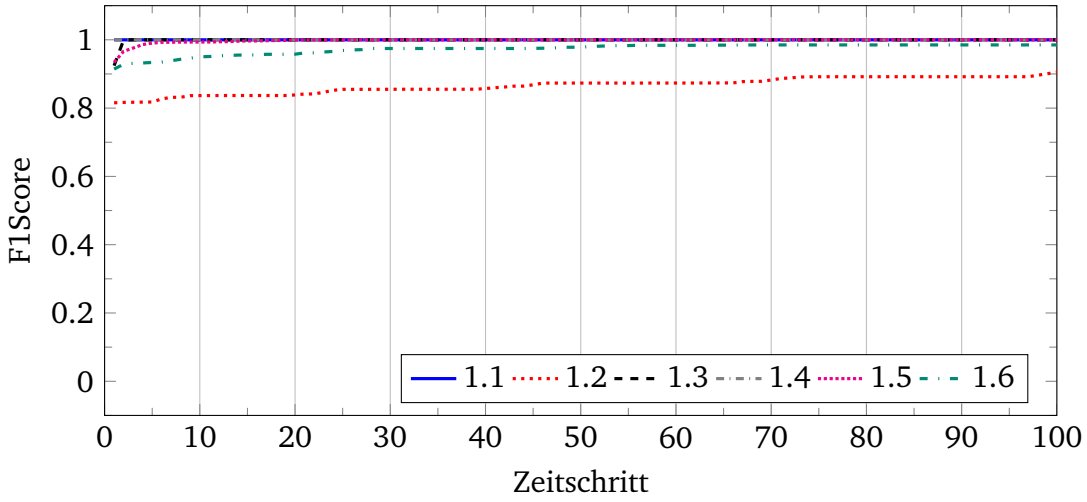


Abbildung 44: BatCEP Kurvenverläufe der Evaluation von BatCEP auf den Datensätzen 1.1 bis 1.6

Test mit den Datensätzen 2.1 bis 2.6

Abbildung 45 zeigt die Ergebniskurven über die Datensätze 2.1 bis 2.6. Hier ist das gleiche Muster zu erkennen. BatCEP hat auch hierbei Probleme mit dem Datensatz, der genau drei unterschiedliche Werte für die ID beinhaltet.

Tabelle 30: Kennwerte für die Evaluation von BatCEP auf den Datensätzen 2.1 bis 2.6

Datensatz	2.1	2.2	2.3	2.4	2.5	2.6
Effektive Schwarmgröße	171	306	191	231	250	262
Lösungen pro Testlauf	68400	122400	76400	92400	100000	104800
Lösungen insgesamt	684000	1224000	764000	924000	1000000	1048000
Erreichte Fitness	1.0	0.93	1.0	1.0	1.0	1.0
∅ Laufzeit pro Testlauf	316s	230s	311s	152s	135s	126s
Laufzeit über alle Testläufe	3160s	2300s	3110s	1520s	1350s	1260s

Die Kennwerte in Tabelle 30 spiegeln dies ebenfalls wider. Erfreulicherweise erreicht BatCEP bei allen anderen Datensätzen eine Fitness von 100% – und das schon nach bedeutend weniger Zeitschritten.

Test mit den Datensätzen 3.1 bis 3.3

Abbildung 46 zeigt die Ergebniskurven der Datensätze 3.1 bis 3.3. Die zur Erzeugung dieser Datensätze eingesetzte Regel $R*_3$ ist insofern komplexer, als sie erstmals eine Kontextbedingung stellt. Auf dem Datensatz 3.1 erreicht BatCEP eine Fitness von 100%, was darauf zurück zu führen ist, dass nur ein Wert für das ID-Attribut existiert, womit die Kontextbedingung stets erfüllt ist.

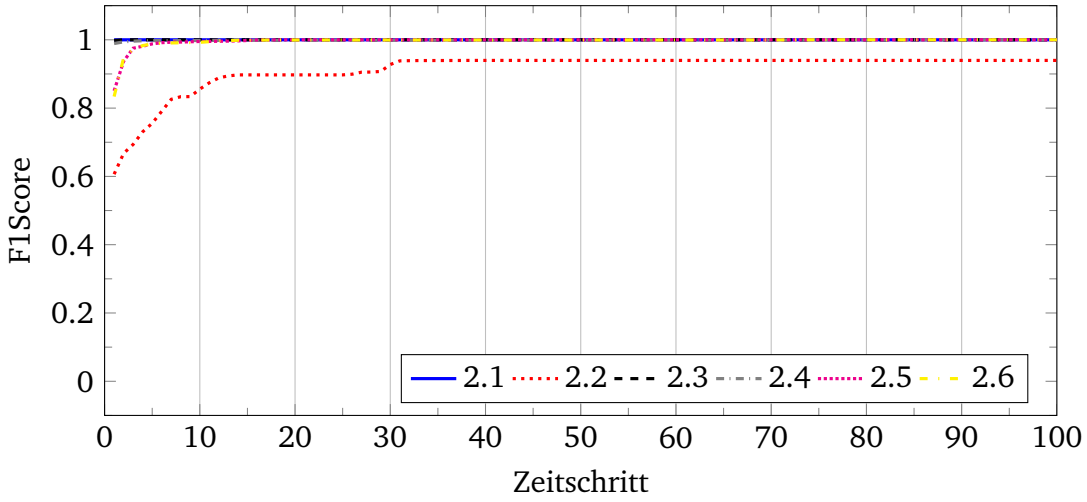


Abbildung 45: BatCEP Kurvenverläufe der Evaluation von BatCEP auf den Datensätzen 2.1 bis 2.6

Tabelle 31: Kennwerte für die Evaluation von BatCEP auf den Datensätzen 3.1 bis 3.3

Datensatz	3.1	3.2	3.3
Effektive Schwarmgröße	165	237	245
Lösungen pro Testlauf	66000	94800	98000
Lösungen insgesamt	660000	948000	980000
Erreichte Fitness	1.0	0.66	0.59
∅ Laufzeit pro Testlauf	338s	200s	188s
Laufzeit über alle Testläufe	3380s	2000s	1880s

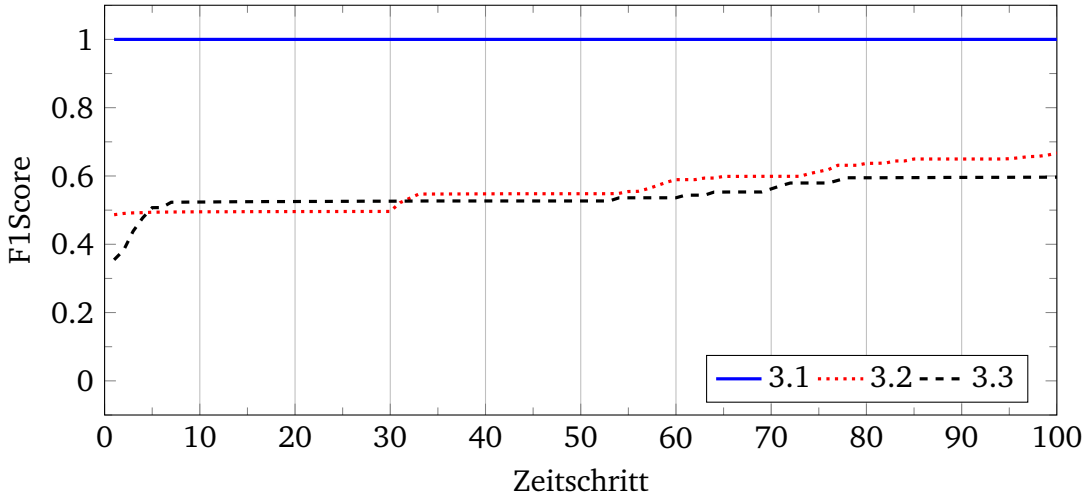


Abbildung 46: BatCEP Kurvenverläufe der Evaluation von BatCEP auf den Datensätzen 3.1 bis 3.3

Dieses Verhalten ändert sich für die Datensätze 3.2 und 3.3, weil beide die Anzahl unterschiedlicher Werte für das ID-Attribut erhöhen. Damit vergrößert sich auch der Suchraum, sodass BatCEP im Rahmen der 100 Zeitschritte keine geeignete Lösung darin finden kann. Tabelle 31 zeigt die Kennwerte.

Test mit den Datensätzen 4.1 bis 4.3

Die Komplexität der Regel $R*_4$, mit der die Datensätze 4.1 bis 4.3 erzeugt wurden, ist erneut angestiegen. Zum einen wurde das Ereignismuster größer und auch die Kontextbedingung wurde erweitert. Dies schlägt sich auch in den Ergebniskurven in Abbildung 47 nieder. Im Grunde ist dasselbe Muster wie zuvor erkennbar: der Datensatz 1.1 mit nur einem Wert für das ID-Attribut kann recht einfach analysiert werden, sodass BatCEP dort eine Fitness von 100% erreicht – offenbar schon direkt nach der Initialisierung oder im ersten Zeitschritt.

Tabelle 32: Kennwerte für die Evaluation von BatCEP auf den Datensätzen 4.1 bis 4.3

Datensatz	4.1	4.2	4.3
Effektive Schwarmgröße	172	250	236
Lösungen pro Testlauf	68800	100000	94400
Lösungen insgesamt	688000	1000000	944000
Erreichte Fitness	1.0	0.43	0.3
∅ Laufzeit pro Testlauf	305s	249s	169s
Laufzeit über alle Testläufe	3050s	2490s	1690s

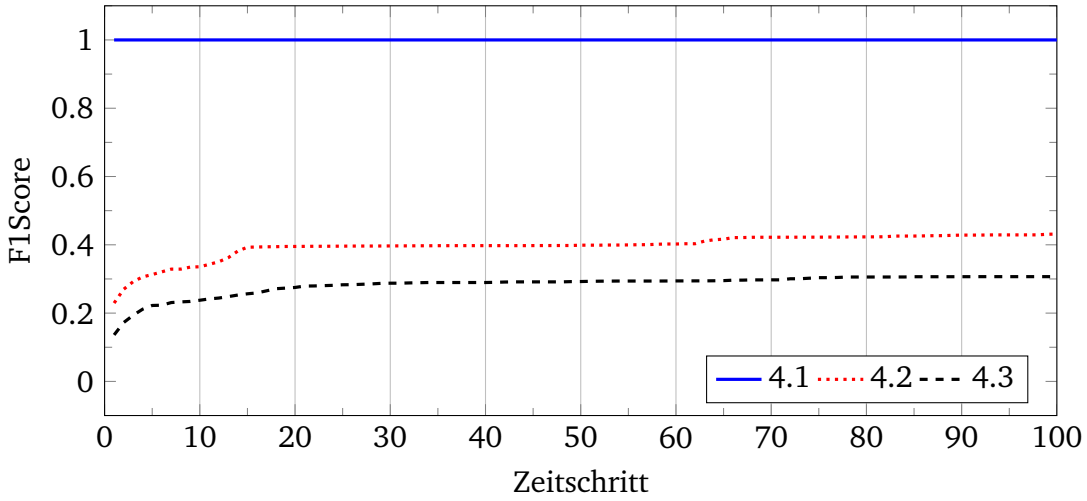


Abbildung 47: BatCEP Kurvenverläufe der Evaluation von BatCEP auf den Datensätzen 4.1 bis 4.3

Dies ändert sich auf dieselbe Weise wie schon im Test zuvor, wenn die Kontextbedingung komplexer wird. Mit Fitnesswerten von 43% und 30% fällt das Ergebnis sogar noch schlechter aus, als bei den Datensätzen 3.2 und 3.3. Dies geht auf die erhöhte Komplexität im Ereignismuster zurück. Weiterhin ist zu beobachten, dass die Kurven nach dem Zeitschritt 100 noch weiter ansteigen würden. Mit mehr Zeitschritten (oder gegebenenfalls mehr Schwärmen) und damit mehr erzeugten Lösungen würde BatCEP vielleicht noch eine gute Lösung finden – jedoch mit einem nicht wirklich vertretbaren Zeitaufwand. Die Kennwerte sind in Tabelle 32 zu sehen.

8 Zusammenfassung und Ausblick

In dieser Masterarbeit wurde mit „BatCEP“ ein einfach zu konfigurierendes Verfahren entwickelt, das Ursachen für komplexe Ereignisse in Datenströmen erkennt und diese in Form von CEP-Regeln zum Ausdruck bringt. Um die Ursachen für ein komplexes Ereignis zu finden benötigt BatCEP lediglich einen Trainingsdatensatz als Mitschnitt realer Ereignisdaten, in dem primitive Ereignisse zusammen mit dem komplexen Ereignis enthalten sind. Dann bringt es primitive Ereignisse in spezielle Konstellationen und klassifiziert diese hinsichtlich des komplexen Ereignisses. Damit handelt es sich bei BatCEP um ein assoziatives Lernverfahren, das speziell auf Complex Event Processing zugeschnitten ist. Dabei geht der Begriff „lernen“ Programm-intern auf die Tatsache zurück, dass BatCEP jede seiner erzeugten CEP-Regeln mit einer Fitness-Funktion bewertet und vergleicht. Somit lernt es Schritt für Schritt welche Eigenschaften gut und welche schlecht sind und auf Basis dieses Wissens werden zunehmend bessere CEP-Regeln erzeugt. Gesamtheitlich betrachtet bedeutet „lernen“, dass BatCEP relevante Ereignismuster aus einem Trainingsdatensatz erlernt und das Gelernte in Form von CEP-Regeln an den Benutzer weiterreicht. Der Benutzer (zumeist wohl ein Fachexperte) bringt die CEP-Regeln in den realen Betrieb mit ein, indem er sie fortan auf Echtzeit-Datenströme anwendet. Überdies vereint BatCEP das Complex Event Processing mit der Schwarmintelligenz, einer von der Natur inspirierten Form der künstlichen Intelligenz. So basiert die Kernkomponente von BatCEP auf dem sogenannten Bat-Algorithmus, einem effektiven Optimierungsalgorithmus für kontinuierliche Funktionen, der die Echo-Ortung kleiner Fledermaus-Arten als Vorbild hat. Seine charakteristischen Eigenschaften wurden im Zuge dieser Masterarbeit auf das kombinatorische Problem des Lernens von CEP-Regeln übertragen.

Insgesamt wurden schon zahlreiche assoziative Lernverfahren entwickelt, viele davon für die sogenannte „Warenkorb-Analyse“. Solche Verfahren klassifizieren Artikel von Kundentransaktionen in Supermärkten und können anschließend Vorhersagen treffen wie: „Wer Brot und Kaffee kauft, der kauft sehr wahrscheinlich auch Milch.“ Die Übertragung solcher Assoziationen auf das Complex Event Processing ist jedoch schwieriger, denn dabei spielen nicht nur die Artikel selbst eine Rolle, sondern auch die exakte Reihenfolge in der sie gekauft werden, in welchen Zeiträumen sie gekauft werden und welche Produkte explizit nicht gekauft werden. Speziell für CEP-Regeln wurden bisher nur wenige Lernverfahren entwickelt. Zwei populäre sind *iCEP* und *autoCEP* und hinzu kommt noch *CepGP*. Letzteres lernt CEP-Regeln mithilfe der genetischen Programmierung, womit es sich genau wie bei BatCEP um ein biologisch inspiriertes Lernverfahren handelt und zudem um das erste dieser Art. BatCEP wurde in dieser Masterarbeit hinreichend evaluiert und unter anderem auch mit CepGP hinsichtlich der Qualität der erzeugten CEP-Regeln und der Laufzeit verglichen. Dabei hat BatCEP zumeist besser abgeschnitten, was wohl auf seine effektiven Aktualisierungsoperationen und seine probabilistische Funktionsweise zurück geht.

Ausblick

BatCEP funktioniert bereits sehr gut als Lernverfahren für CEP-Regeln und hat das Potential zukünftig für den realen Einsatz auch auf populäre CEP-Produkte wie *Esper* übertragen zu werden. In diesem Abschnitt werden noch ein paar Verbesserungs- und Erweiterungsvorschläge angesprochen, die Substanz für zukünftige Projekte bieten.

Eine andere CEP-Engine BatCEP verwendet die CEP-Engine von CepGP und diese erwies sich als gute Grundlage. Sie ist elegant implementiert, so können beispielsweise neue Fitness-Funktionen recht einfach hinzugefügt werden. Mit der schlanken Regel-Sprache lassen sich Ereignismuster einfach formulieren und CEP-Regeln sind bereits in effizient zu bearbeitenden Baum-Strukturen abgebildet. Dennoch könnte BatCEP zukünftig auf ein populäres CEP-Produkt abgebildet werden und dessen Regel-Sprache annehmen. So könnte beispielsweise *Esper* mit der Esper-spezifischen Regel-Sprache Anwendung finden.

Eine andere Initialisierungstechnik BatCEP könnte auch eine andere Initialisierungstechnik erhalten. RHH erwies sich als nützlich und dennoch gibt es noch weitere, vielversprechende Techniken wie beispielsweise das sogenannte „Opposite-based learning“, das schlecht im Suchraum platzierte Lösungskandidaten „auf der anderen Seite des Suchraumes“ neu platziert. Dafür muss es dann auch eine Abbildungsmethode geben, mit der sich der Suchraum ausmessen lässt. Dabei stellt sich die generelle Frage, ob dies für CEP-Regeln überhaupt möglich ist. Zudem kann die Idee der verbesserten RHH-Initialisierung mit dem „Fit-Schwarm“ weiter verfolgt werden.

Andere Abbildungen der Attribute Frequenz, Geschwindigkeit und Lautstärke Für BatCEP wird die Differenz zwischen der Geschwindigkeit und der Frequenz als Anzahl durchzuführender Aktualisierungsoperationen verwendet. Dabei wurden die entsprechenden Aktualisierungsgleichungen möglichst getreu übernommen. Ähnlich ist dies bei der Durchschnittslautstärke pro Zeitschritt. Im Zuge zukünftiger Projekte könnten hierfür andere Abbildungen entwickelt werden. Hierzu könnten die einzelnen Elemente einer CEP-Regel beispielsweise mit einem speziellen Verfahren als skalare Werte abgebildet werden, die dann mit arithmetischen Operationen aktualisierbar sind. Dann könnten Frequenz, Geschwindigkeit und Lautstärke mit ihren Werten direkten Einfluss auf die Aktualisierung einer neuen Position nehmen. Im selben Zuge könnte durch Einbringen neuer Aktualisierungsoperationen das leichte Ungleichgewicht zwischen Erkundung und Ausbeutung ausgeglichen werden. Denn wie die Evaluation gezeigt hat, liefert BatCEP dann die besten Ereignisse, wenn überdurchschnittlich häufig lokale Suchen durchführt, was einer Tendenz zur Ausbeutung entspricht.

Eine andere Kommunikations-Strategie Während der Beschreibung des konzeptionellen Ansatzes wurden verschiedene Kommunikations-Strategien vorgeschlagen, nach denen die parallel eingesetzten Schwärme kommunizieren. BatCEP setzt hierbei auf das allseits

geteilte globale Maximum. Zukünftig könnte geprüft werden, ob sich andere Strategien als besser erweisen. Die Implementierung ist bereits ansatzweise darauf ausgelegt.

Eine bessere Kodierung für effiziente Fitness-Berechnungen Jede Fitness-Messung für jede CEP-Regel erfordert einen kompletten Suchlauf über den Trainingsdatensatz, wodurch die Laufzeit bei großen Datensätzen in einen nicht vertretbaren zeitlichen Bereich ansteigt. Zugleich wurden in einigen Berichten über andere Lernverfahren wie beispielsweise in [46] spezielle Kodierungen vorgeschlagen, mit denen nur ein Suchlauf über den Datensatz erforderlich ist. So müssen anstelle mehrerer Suchläufe beispielsweise nur logische Verknüpfungen direkt an einer CEP-Regel durchgeführt werden, um ihren Fitness-Wert zu berechnen. In einem zukünftigen Projekt könnte für BatCEP ebenfalls eine solche Kodierung entwickelt werden; gegebenenfalls kann diese sogar zusätzlich zur Baumrepräsentation implementiert werden.

Methodenaufrufe in Kontextbedingungen zulassen Prinzipiell sollte es auch möglich sein domänenspezifische Methodenaufrufe in die CEP-Regeln einzubauen, um Attribut-Werte damit zu vergleichen. Als Beispiel sei die folgende CEP-Regel gegeben:

`(A AS a1` $\rightarrow$ `A AS a2)`$\wedge$`(a1.temp > DB.getMaxTemp(a1))[win:time:2min]` $\Rightarrow$ Z

Damit BatCEP eine solche Methode tatsächlich aufrufen kann, muss eine Verbindung zum realen Anwendungssystem hergestellt werden.

Literatur

[1] A. Agarwal und N. Nanavati. „Association rule mining using hybrid GA-PSO for multi-objective optimisation“. In: *2016 IEEE International Conference on Computational Intelligence and Computing Research (ICCIC)*. Dez. 2016, S. 1–7. DOI: 10.1109/ICCIC.2016.7919571.

[2] M. Agrawal, M. Mishra und S. P. S. Kushwah. „Association rules optimization using improved PSO algorithm“. In: *International Conference on Communication Networks (ICCN)*. Nov. 2015, S. 395–398. DOI: 10.1109/ICCN.2015.76.

[3] R. Agrawal und R. Srikant. „Fast Algorithms for Mining Association Rules in Large Databases“. In: *Proceedings of the 20th International Conference on Very Large Data Bases*. VLDB '94. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1994, S. 487–499. URL: http://dl.acm.org/citation.cfm?id=645920.672836.

[4] S. Akhtar, A. R. Ahmad und E. M. Abdel-Rahman. „A Metaheuristic Bat-Inspired Algorithm for Full Body Human Pose Estimation“. In: *Ninth Conference on Computer and Robot Vision*. Mai 2012, S. 369–375. DOI: 10.1109/CRV.2012.55.

[5] A. de Almeida, P. G. Toracio und A. T. R. Pozo. „Multiple objective particle swarm for classification-rule discovery“. In: *IEEE Congress on Evolutionary Computation*. Sep. 2007, S. 684–691. DOI: 10.1109/CEC.2007.4424537.

[6] J. D. Altringham. *Bats*. 2. Aufl. New York: Oxford University Press, 2001, S. 352.

[7] S. Binitha und S. Siva. „A Survey of Bio inspired Optimization Algorithms“. In: *International Journal of Soft Computing and Engeneering (IJSCE)* 2.2 (2012). ISSN: 2231-2307.

[8] E. Bonabeau, M. Dorigo und G. Theraulaz. *Swarm Intelligence: From Natural to Artificial Systems*. New York, NY, USA: Oxford University Press, Inc., 1999.

[9] T. C. Bora, L. d. S. Coelho und L. Lebensztajn. „Bat-Inspired Optimization Approach for the Brushless DC Wheel Motor Problem“. In: *IEEE Transactions on Magnetics* 48.2 (Feb. 2012), S. 947–950. ISSN: 0018-9464. DOI: 10.1109/TMAG.2011.2176108.

[10] R. Bruns und J. Dunkel. *Complex Event Processing*. Springer Vieweg, 2015. DOI: 10.1007/978-3-658-09899-5.

[11] M. Chen und S. A. Ludwig. „Discrete Particle Swarm Optimization with local search strategy for Rule Classification“. In: *2012 Fourth World Congress on Nature and Biologically Inspired Computing (NaBIC)*. Nov. 2012, S. 162–167. DOI: 10.1109/NaBIC.2012.6402256.

[12] M. Clerc und J. Kennedy. „The particle swarm - explosion, stability, and convergence in a multidimensional complex space“. In: *IEEE Transactions on Evolutionary Computation* 6.1 (Feb. 2002), S. 58–73. ISSN: 1089-778X. DOI: 10.1109/4235.985692.

[13] R. Damodaram und M. Valarmathi. „Phishing website detection and optimization using Modified bat algorithm“. In: *International Journal of Engineering Research and Applications* 2 (1 Jan. 2012), S. 870–876. ISSN: 2248-9622.

[14] Y. Djenouri, H. Drias, Z. Habbas und H. Mosteghanemi. „Bees Swarm Optimization for Web Association Rule Mining“. In: *Proceedings of the The IEEE/WIC/ACM International Joint Conferences on Web Intelligence and Intelligent Agent Technology - Volume 03*. WI-IAT '12. Washington, DC, USA: IEEE Computer Society, 2012, S. 142–146. URL: http://dx.doi.org/10.1109/WI-IAT.2012.148.

[15] P. Domingos. „A Few Useful Things to Know About Machine Learning“. In: *Commun. ACM* 55.10 (Okt. 2012), S. 78–87. ISSN: 0001-0782. DOI: 10.1145/2347736.2347755. URL: http://doi.acm.org/10.1145/2347736.2347755.

[16] M. Dorigo und G. D. Caro. „Ant colony optimization: a new meta-heuristic“. In: *Proceedings of the Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406)*. Bd. 2. 1999, 1477 Vol. 2. DOI: 10.1109/CEC.1999.782657.

[17] Eberhart und Y. Shi. „Particle swarm optimization: developments, applications and resources“. In: *Proceedings of the Congress on Evolutionary Computation (IEEE Cat. No.01TH8546)*. Bd. 1. 2001, 81–86 vol. 1. DOI: 10.1109/CEC.2001.934374.

[18] E. Farhana und S. Heber. „Biogeography-based Rule Mining for Classification“. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. GECCO '17. Berlin, Germany: ACM, 2017, S. 417–424. ISBN: 978-1-4503-4920-8. DOI: 10.1145/3071178.3071221. URL: http://doi.acm.org/10.1145/3071178.3071221.

[19] A. Faritha Banu und C. Chandrasekar. „An optimized appraoch of modified bat algorithm to record deduplication“. In: *International Journal of Computer Applications* 62.1 (2012), S. 10–15.

[20] Y. Gao, J. Q. Hu und X. L. Tang. „The Application of Hybrid Ant Colony Algorithm in Association Rule Mining“. In: *9th International Symposium on Computational Intelligence and Design (ISCID)*. Bd. 2. Dez. 2016, S. 329–333. DOI: 10.1109/ISCID.2016.2085.

[21] A. Garg, P. K. Mahapatra und A. Kumar. „Error optimization using Bat and PSO algorithms for machine vision system based tool movement“. In: *International Conference on Advances in Computing, Communications and Informatics (ICACCI)*. Sep. 2014, S. 2689–2693. DOI: 10.1109/ICACCI.2014.6968298.

[22] L. Geng und H. J. Hamilton. „Interestingness Measures for Data Mining: A Survey“. In: *ACM Comput. Surv.* 38.3 (Sep. 2006). ISSN: 0360-0300. DOI: 10.1145/1132960.1132963. URL: http://doi.acm.org/10.1145/1132960.1132963.

[23] Y. Gigras, K. Gupta, Vandana und K. Choudhary. „A Comparison between Bat Algorithm and Cuckoo Search for Path Planning“. In: *International Journal of Innovative Research in Computer and Communication Engineering* 3 (2015). ISSN: 2320-9801.

[24] M. Gupta und S. Ram. „Application of Weighted Particle Swarm Optimization in Association Rule Mining“. In: Bd. 1. 2012.

[25] J. Han, J. Pei und Y. Yin. „Mining Frequent Patterns Without Candidate Generation“. In: *SIGMOD Rec.* 29.2 (Mai 2000), S. 1–12. ISSN: 0163-5808. DOI: 10.1145/335191.335372. URL: http://doi.acm.org/10.1145/335191.335372.

[26] K. Hassani und W. S. Lee. „An Incremental Parallel Particle Swarm Approach for Classification Rule Discovery from Dynamic Data“. In: *12th International Conference on Machine Learning and Applications*. Bd. 1. Dez. 2013, S. 430–435. DOI: 10.1109/ICMLA.2013.87.

[27] K. E. Heraguemi, N. Kamel und H. Drias. „Association Rule Mining Based on Bat Algorithm“. In: *Bio-Inspired Computing - Theories and Applications: 9th International Conference, BIC-TA 2014, Wuhan, China, October 16-19, 2014. Proceedings*. Hrsg. von L. Pan. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, S. 182–186. DOI: 10.1007/978-3-662-45049-9_29. URL: http://dx.doi.org/10.1007/978-3-662-45049-9_29.

[28] K. E. Heraguemi, N. Kamel und H. Drias. „Multi-swarm bat algorithm for association rule mining using multiple cooperative strategies“. In: *Applied Intelligence* 45.4 (2016), S. 1021–1033. ISSN: 1573-7497. DOI: 10.1007/s10489-016-0806-y. URL: http://dx.doi.org/10.1007/s10489-016-0806-y.

[29] N. Holden und A. A. Freitas. „A hybrid particle swarm/ant colony algorithm for the classification of hierarchical biological data“. In: *Proceedings IEEE Swarm Intelligence Symposium, 2005. SIS 2005.* Juni 2005, S. 100–107. DOI: 10.1109/SIS.2005.1501608.

[30] N. P. Holden und A. A. Freitas. „A Hybrid PSO/ACO Algorithm for Classification“. In: *Proceedings of the 9th Annual Conference Companion on Genetic and Evolutionary Computation*. GECCO '07. London, United Kingdom: ACM, 2007, S. 2745–2750. DOI: 10.1145/1274000.1274080. URL: http://doi.acm.org/10.1145/1274000.1274080.

[31] R. A. Huebner. „Diversity-based interestingness measures for association rule mining“. In: *Proceedings of ASBBS* 16.1 (Feb. 2009). URL: https://pdfs.semanticscholar.org/a1f2/08b5ea6d09d6a34c38791982ec21ea5964c1.pdf.

[32] J. Ji, N. Zhang, C. Liu und N. Zhong. „An Ant Colony Optimization Algorithm for Learning Classification Rules“. In: *IEEE/WIC/ACM International Conference on Web Intelligence (WI 2006 Main Conference Proceedings)(WI'06)*. Dez. 2006, S. 1034–1037. DOI: 10.1109/WI.2006.35.

[33] B. Kazimipour, X. Li und A. K. Qin. „A review of population initialization techniques for evolutionary algorithms“. In: *IEEE Congress on Evolutionary Computation (CEC)*. Juli 2014, S. 2585–2592. DOI: 10.1109/CEC.2014.6900618.

[34] J. Kennedy und R. Mendes. „Population structure and particle swarm performance“. In: *Evolutionary Computation. Proceedings of the Congress on CEC*. Bd. 2. 2002, S. 1671–1676. DOI: 10.1109/CEC.2002.1004493.

[35] J. Kennedy und R. Eberhart. *Particle Swarm optimization*. 1995. URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=488968.

[36] J. Kennedy und R. C. Eberhart. *Swarm Intelligence*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001.

[37] K. Khan und A. Sahai. „A fuzzy c-means bi-sonar-based Metaheuristic Optimization Algorithm“. In: *International Journal of Artificial Intelligence and Interactive Multimedia* 1.7 (2012), S. 26–32. DOI: 10.9781/ijimai.2012.173.

[38] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA, USA: MIT Press, 1992.

[39] L. Li und Y. Zhou. „A Novel Complex-valued Bat Algorithm“. In: *Neural Computer Applications* 25.6 (Nov. 2014), S. 1369–1381. ISSN: 0941-0643. DOI: 10.1007/s00521-014-1624-y. URL: http://dx.doi.org/10.1007/s00521-014-1624-y.

[40] J.-H. Lin, C.-W. Chou, C.-H. Yang und H.-L. Tsai. „A Chaotic Levy Flight Bat Algorithm for Parameter Estimation in Nonlinear Dynamic Biological Systems“. In: *Journal of Computer and Information Technology* 2.2 (Feb. 2015), S. 56–63.

[41] B. Liu, H. A. Abbas und B. McKay. „Classification rule discovery with ant colony optimization“. In: *IEEE/WIC International Conference on Intelligent Agent Technology, 2003. IAT 2003.* Okt. 2003, S. 83–88. DOI: 10.1109/IAT.2003.1241052.

[42] B. Liu, H. A. Abbass und B. McKay. „Density-Based Heuristic for Rule Discovery with Ant-Miner“. In: (2007).

[43] Y. Liu, Z. Qin, Z. Shi und J. Chen. „Rule Discovery with Particle Swarm Optimization“. In: *Content Computing: Advanced Workshop on Content Computing, AWCC 2004, ZhenJiang, JiangSu, China, November 15-17, 2004. Proceedings*. Hrsg. von C.-H. Chi und K.-Y. Lam. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, S. 291–296. DOI: 10.1007/978-3-540-30483-8_35. URL: http://dx.doi.org/10.1007/978-3-540-30483-8_35.

[44] D. Luckham und W. R. Schulte. *Event Processing Glossary - Version 2.0*. 2011. URL: http://www.complexevents.com/wp-content/uploads/2011/08/EPTS_Event_Processing_Glossary_v2.pdf (besucht am 20. 07. 2017).

[45] V. Mangat. „Swarm Intelligence Based Technique for Rule Mining in the Medical Domain“. In: *International Journal of Computer Applications* 4.1 (Juli 2010). Published By Foundation of Computer Science, S. 19–24.

[46] Manju und C. Kant. „Mining association rules directly using ACO without generating frequent itemsets“. In: *2015 International Conference on Energy Systems and Applications*. Okt. 2015, S. 390–395. DOI: 10.1109/ICESA.2015.7503377.

[47] A. Margara, G. Cugola und G. Tamburrelli. „Learning from the Past: Automated Rule Generation for Complex Event Processing“. In: *Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems*. DEBS '14. Mumbai, India: ACM, 2014, S. 47–58. DOI: 10.1145/2611286.2611289. URL: http://doi.acm.org/10.1145/2611286.2611289.

[48] M. K. Marichelvam und T. Prabaharam. „A bat algorithm for realistic hybrid flow-shop schedulihng problems to minimize makespan and mean flow time“. In: *ICTACT Journal on Soft Computing* 3.1 (2012), S. 428–433.

[49] R. Mousheimish, Y. Taher und K. Zeitouni. „Automatic Learning of Predictive Rules for Complex Event Processing: Doctoral Symposium“. In: *Proceedings of the 10th ACM International Conference on Distributed and Event-based Systems*. DEBS '16. Irvine, California: ACM, 2016, S. 414–417. DOI: 10.1145/2933267.2933430. URL: http://doi.acm.org/10.1145/2933267.2933430.

[50] P. Musikapun und P. Pongcharoen. „Solving Multi-Stage Multi-Machine Multi-Product Scheduling Problem Using Bat Algorithm“. In: *IACSIT Press* (2012).

[51] N. Offel. *Master Thesis: A Genetic Programming Algorithm To Derive Complex Event Processing Rules For A Given Event Type*. Aug. 2016. URL: http://sw-architecture.inform.hs-hannover.de/joomla/images/thesis/ma_offel_endversion_20160829.pdf.

[52] E. Osaba, X.-S. Yang, F. Diaz, P. Lopez-Garcia und R. Carballedo. „An Improved Discrete Bat Algorithm for Symmetric and Asymmetric Traveling Salesman Problems“. In: *Eng. Appl. Artif. Intell.* 48.C (Feb. 2016), S. 59–71. ISSN: 0952-1976. DOI: 10.1016/j.engappai.2015.10.006. URL: http://dx.doi.org/10.1016/j.engappai.2015.10.006.

[53] R. S. Parpinelli, H. S. Lopes und A. A. Freitas. „Data mining with an ant colony optimization algorithm“. In: *IEEE Transactions on Evolutionary Computation* 6.4 (Aug. 2002), S. 321–332. ISSN: 1089-778X. DOI: 10.1109/TEVC.2002.802452.

[54] D. M. W. Powers. „Evaluation: From precision, recall and f-measure to roc., informedness, markedness & correlation“. In: *Journal of Machine Learning Technologies* 2.1 (2011), S. 37–63.

[55] J. R. Quinlan. *C4.5: Programs for Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993.

[56] B. Ramesh, V. C. J. Mohan und V. V. Reddy. „Application of Bat Algorithm for Economic Load Dispatch Problem with Valve-Point Effect“. In: *The IUP Journal of Electrical & Electronics Engineering* 4.2 (Mai 2013), S. 38–46.

[57] C. J. V. Rijsbergen. *Information Retrieval*. 2nd. Newton, MA, USA: Butterworth-Heinemann, 1979. ISBN: 0408709294.

[58] A. L. Samuel. „Computing Bit by Bit or Digital Computers Made Easy“. In: *Proceedings of the Institute of Radio Engineers* 41.10 (Okt. 1953), S. 1223–1230. ISSN: 0096-8390. DOI: 10.1109/JRPROC.1953.274271.

[59] A. L. Samuel. „Some Studies in Machine Learning Using the Game of Checkers“. In: *IBM J. Res. Dev.* 3.3 (Juli 1959), S. 210–229. ISSN: 0018-8646. DOI: 10.1147/rd.33.0210. URL: http://dx.doi.org/10.1147/rd.33.0210.

[60] A. Secker, M. N. Davies, A. A. Freitas, E. B. Clark, J. Timmis und D. R. Flower. „Hierarchical Classification of G-Protein-Coupled Receptors with Data-Driven Selection of Attributes and Classifiers“. In: *Int. J. Data Min. Bioinformatics* 4.2 (März 2010), S. 191–210. ISSN: 1748-5673. DOI: 10.1504/IJDMB.2010.032150. URL: http://dx.doi.org/10.1504/IJDMB.2010.032150.

[61] P. Sehrawat, Manju und H. Rohil. „Association Rule Mining Using Firefly Algorithm“. In: *International Journal of Latest Trends in Engineering and Technology (IJLTET)* 3 (2 Nov. 2013).

[62] P. Sharma, S. Tiwari und M. Gupta. „Association Rules Optimization using Artificial Bee Colony Algorithm with Mutation“. In: *International Journal of Computer Applications* 116.13 (Apr. 2015).

[63] M. A. M. Shukran, Y. Y. Chung, W.-C. Yeh, N. Wahid und A. M. A. Zaidi. „Artificial Bee Colony based Data Mining Algorithms for Classification Tasks“. In: *Modern Applied Science* 5.4 (Aug. 2011). DOI: 10.5539/mas.v5n4p217.

[64] J. Smaldon und A. A. Freitas. „A New Version of the Ant-Miner Algorithm Discovering Unordered Rule Sets“. In: *2006 Genetic and Evolutionary Computation Conference*. Hrsg. von M. Keijzer. Bd. 1. New York, New York (USA): ACM Press, Juli 2006, S. 43–50. URL: http://kar.kent.ac.uk/14462/.

[65] O. S. Soliman und E. A. Elhamd. „A Chaotic Levy Flights Bat Algorithm for Diagnosing Diabetes Mellitus“. In: *International Journal of Computer Applications* 111.1 (Feb. 2015), S. 36–42.

[66] A. Song, X. Ding, J. Chen, M. Li, W. Cao und K. Pu. „Multi-objective association rule mining with binary bat algorithm.“ In: *Intelligent Data Analysis* 20.1 (2016), S. 105–128. URL: http://dblp.uni-trier.de/db/journals/ida/ida20.html#SongDCLCP16.

[67] H. R. Tizhoosh. „Opposition-Based Learning: A New Scheme for Machine Intelligence“. In: *International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce (CIMCA-IAWTIC'06)*. Bd. 1. Nov. 2005, S. 695–701. DOI: 10.1109/CIMCA.2005.1631345.

[68] G. G. Wang, B. Chang und Z. Zhang. „A multi-swarm bat algorithm for global optimization“. In: *2015 IEEE Congress on Evolutionary Computation (CEC)*. Mai 2015, S. 480–485. DOI: 10.1109/CEC.2015.7256928.

[69] X.-S. Yang. „A New Metaheuristic Bat-Inspired Algorithm“. In: *Nature Inspired Cooperative Strategies for Optimization (NICSO 2010)*. Hrsg. von J. R. González, D. A. Pelta, C. Cruz, G. Terrazas und N. Krasnogor. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, S. 65–74. DOI: 10.1007/978-3-642-12538-6_6. URL: http://dx.doi.org/10.1007/978-3-642-12538-6_6.

[70] X.-S. Yang und X. He. „Bat Algorithm: Literature Review and Applications“. In: *Int. J. Bio-Inspired Comput.* 5.3 (Juli 2013), S. 141–149. ISSN: 1758-0366. DOI: 10.1504/IJBIC.2013.055093. URL: http://dx.doi.org/10.1504/IJBIC.2013.055093.

[71] L. Ye und E. Keogh. „Time Series Shapelets: A New Primitive for Data Mining“. In: *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '09. Paris, France: ACM, 2009, S. 947–956. ISBN: 978-1-60558-495-9. DOI: 10.1145/1557019.1557122. URL: http://doi.acm.org/10.1145/1557019.1557122.

[72] S. Yilmaz, E. Ugur Kucuksille und Y. Cengiz. „Modified Bat Algorithm“. In: *Electronics & Electrical Engineering* 20 (Feb. 2014), S. 71.

[73] J. W. Zhang und G. G. Wang. „Image Matching Using a Bat Algorithm with Mutation“. In: *Review of Modern Engineering Solutions for the Industry*. Bd. 203. Applied Mechanics and Materials. Trans Tech Publications, Nov. 2012, S. 88–93. DOI: 10.4028/www.scientific.net/AMM.203.88.

[74] Z. Zhang. „An improved rule mining technology based on swarm intelligence computation“. In: *School of Computer Science and Technology in Heilongjiang University* (Sep. 2014).