Fachhochschul-Masterstudiengang
SOFTWARE ENGINEERING
4232 Hagenberg, Austria

Integration selbstadaptiver Selektionsdrucksteuerung in Evolutionsstrategien

Masterarbeit

zur Erlangung des akademischen Grades
Master of Science in Engineering

Eingereicht von

Günther Teufl, BSc

Begutachter: Prof. (FH) Priv.-Doz. DI Dr. Michael Affenzeller

Dezember 2014

Erklärung

Ich erkläre eidesstattlich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen nicht benutzt und die den benutzten Quellen entnommenen Stellen als solche gekennzeichnet habe. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt.

Hagenberg, am 16. Dezember 2014

Günther Teufl

Inhaltsverzeichnis

Erklärung i

Kurzfassung iv

Abstract v

1 Einleitung 1
- 1.1 Hintergrund und Motivation . . . 1
- 1.2 Problemstellung und Zielsetzung . . . 2

2 Evolutionäre Algorithmen 3
- 2.1 Iteratives Problemlösen durch Evolution . . . 3
  - 2.1.1 Von der Biologie zur Optimierung . . . 4
  - 2.1.2 Rekombination . . . 8
  - 2.1.3 Mutation . . . 12
  - 2.1.4 Selektion . . . 14
- 2.2 Evolutionsstrategien und Genetische Algorithmen . . . 15
  - 2.2.1 Die (1+1) - Evolutionsstrategie . . . 16
  - 2.2.2 (μ+λ) und (μ,λ)-Notation . . . 17
  - 2.2.3 MSC-ES und CMA-ES . . . 20
  - 2.2.4 Rekombination in Evolutionsstrategien . . . 28
  - 2.2.5 Genetische Algorithmen . . . 30
- 2.3 Selektionsdruck-Steuerung . . . 34
  - 2.3.1 Selektion in Genetischen Algorithmen . . . 34
  - 2.3.2 Selektion in Evolutionsstrategien . . . 35
  - 2.3.3 Offspring Selection . . . 35

3 OS-ES: Offspring Selection Evolution Strategy 39
- 3.1 Erweiterung der Evolutionsstrategie . . . 39
  - 3.1.1 Offspring Selection in der (μ,λ)-ES . . . 39
  - 3.1.2 Offspring Selection in der CMA-ES . . . 44
- 3.2 Umsetzung der Algorithmen . . . 46
  - 3.2.1 HeuristicLab als Entwicklungsumgebung . . . 47
  - 3.2.2 Implementierung der OS-ES . . . 52

3.2.3 Implementierung der OS-CMA-ES 57

4 Empirische Untersuchung 62
4.1 Aufbau der Experimente 62
4.1.1 Testen mit HeuristicLab 62
4.1.2 Ausgewählte Probleme 63
4.2 Untersuchung der OS-ES 65
4.2.1 Ergebnisse ohne Rekombination 65
4.2.2 Ergebnisse mit Rekombination 70
4.2.3 Rekombination mit multiplen Operatoren 75
4.2.4 Weitere Experimente 80
4.2.5 Evaluation der Ergebnisse 83
4.3 Untersuchung der OS-CMA-ES 84
4.3.1 Aufbau der Tests 84
4.3.2 Ergebnisse 85
4.3.3 Evaluation der OS-CMA-ES 89

5 Zusammenfassung 90

Quellenverzeichnis 91
Literatur 91
Online-Quellen 95

Kurzfassung

Bei der Lösung von Optimierungsproblemen mit *Evolutionären Algorithmen* zählen unter anderem die *Evolutionsstrategien* (*ES*) zu den wichtigsten Verfahren. Sie ermöglichen eine kontinuierliche, zielgerichtete Verbesserung von potentiellen Lösungen in einer Weise, die es erlaubt beliebige Problemstellungen zu bewältigen. Die Wahl der passenden Algorithmus-Parameter und der dadurch entstehende *Selektionsdruck* sind dabei ausschlaggebend für die erreichte Qualität der ermittelten Lösung. In den aktuell etablierten Arten der Evolutionsstrategie muss der Selektionsdruck in Abhängigkeit des zu lösenden Problems und dessen Komplexität richtig gewählt werden. Der Selektionsdruck bleibt dann im Laufe der Optimierung zu jedem Zeitpunkt konstant. Es kann jedoch auch sinnvoll sein, den Druck während der Ausführung des Verfahrens zu erhöhen oder wieder zu senken, um einen besseren Fortschritt der Optimierung zu ermöglichen. Durch eine *selbstadaptive Selektionsdruck-Steuerung* ist der Algorithmus in der Lage, den ausgeübten Selektionsdruck selbstständig anzupassen. Ein Konzept zur Realisierung einer solchen selbstadaptiven Steuerung ist die *Offspring Selection*. Durch die Integration dieses Konzeptes in die $(\mu/\rho^{+},\lambda)$-ES und die CMA-ES evaluiert diese Arbeit die Auswirkung einer selbstadaptiven Steuerung im Bezug auf die erreichte Lösungsqualität. Für die CMA-ES konnten durch den Einsatz von *Offspring Selection* keine besseren Resultate erzielt werden. Die $(\mu/\rho^{+},\lambda)$-ES erreichte jedoch für viele Probleme durch die selbstadaptive Steuerung des Selektionsdruckes eine bessere Lösungsqualität.

Abstract

In the field of solving optimization problems by applying *Evolutionary Algorithms*, the so called *Evolution Strategies* (*ES*) belong to the most important methods. They implement a continuous and goal-oriented improvement of possible solutions in a way, that can be applied to many different optimization problems. The use of appropriate parameters for the algorithm and the resulting selection pressure are then critical for reaching a good solution quality. For the currently well-known types of evolution strategies, an appropriate selection pressure has to be chosen depending on the underlying problem and its complexity. The selection pressure then remains constant during the process of optimization, although it may be useful to increase or reduce the selection pressure over time. By implementing a *self-adaptive selection pressure steering*, the algorithm is able to modify the applied selection pressure automatically. A concept for realizing this self-adaption is given by the *Offspring Selection*. This thesis evaluates the effect of a self-adaptive steering in terms of the achieved solution quality by applying this concept to the $(\mu/\rho^{+},\lambda)$-ES and to the CMA-ES. Although it was not possible to improve the results of the CMA-ES, a better solution quality could be reached for many problems by applying a self-adaptive selection pressure steering to the $(\mu/\rho^{+},\lambda)$-ES.

Kapitel 1

Einleitung

1.1 Hintergrund und Motivation

Das Gebiet der Optimierung beschäftigt sich mit Verfahren, die es ermöglichen die optimalen Parameter eines meist komplexen Systems herauszufinden. Die Anwendungsgebiete sind dabei vielseitig. In der Finanzwelt können durch die Optimierung beispielsweise Aktien-Portfolios entwickelt werden, die den Gewinn maximieren oder ein minimales Risiko aufweisen. Betriebswirtschaftliche Anwendungsgebiete umfassen die Planung von optimalen Losgrößen und Auftragsreihenfolgen in der Produktion oder eine möglichst sinnvolle Personaleinsatzplanung. Auch im technisch-naturwissenschaftlichen Bereich finden sich zahlreiche Anwendungen, z. B. für den optimalen Entwurf von Mikroprozessoren um deren Geschwindigkeit zu verbessern [Nis97].

Die *Evolutionsstrategien* (*ES*) zählen neben einigen anderen Algorithmen zu einer bestimmten Gattung an Optimierungsverfahren, die speziell in den letzten Jahren immer mehr an Bedeutung gewannen und unter dem Begriff der *Evolutionären Algorithmen* zusammengefasst sind. Diese Verfahren werden oft auch als *naturanalog* bezeichnet, da sie aus Vorgängen in der Natur inspiriert sind. Sie ermöglichen durch eine abstrakte Definition der Abläufe die Bearbeitung verschiedenster Problemstellungen. Strukturelles Wissen über die Probleme lässt sich ohne hohen Modellierungs- oder Formalisierungsaufwand in diese Methoden integrieren [GKK13]. Je nach Komplexität der Problemstellung kann die Optimierung jedoch sehr rechen- oder zeitintensiv sein, weshalb sich mittlerweile zahlreiche Forschungsgruppen intensiv mit der Weiterentwicklung solcher Verfahren auseinandersetzen. Regelmäßig finden auch große Konferenzen in Europa oder den U.S.A zu diesen Themen statt.

1.2 Problemstellung und Zielsetzung

Komplexe Probleme weisen häufig viele für den Algorithmus scheinbar optimale Belegungen der Lösungs-Variablen auf, die es schwierig machen die tatsächlich beste Lösung zu finden. Für die *Evolutionsstrategien* zeigt sich dieses Problem durch ein Stagnieren des Algorithmus an einem solch scheinbar optimalen Punkt. Es ist in diesem Fall nicht mehr möglich, durch eine weitere Ausführung des Verfahrens noch bessere Lösungen zu finden. Wie schnell eine Evolutionsstrategie zu einem solchen Punkt konvergiert, und somit auch die erreichte Qualität der Lösungen, lässt sich durch die Wahl einiger Konfigurations-Parameter beeinflussen. Durch diese Parameter entsteht ein bestimmter, fix definierter Druck, der für den Fortschritt der Optimierung verantwortlich ist. Aufgrund der Analogie zu den Vorgängen in der Natur spricht man auch von einem bestimmten *Selektionsdruck*. Dieser sollte je nach Problemstellung passend gewählt sein. Bei den aktuell etablierten Evolutionsstrategien bleibt der Selektionsdruck anschließend für den gesamten Optimierungsprozess konstant, obwohl es sinnvoll sein könnte je nach Fortschritt der Optimierung nach einiger Zeit wieder etwas mehr oder weniger Druck auszuüben.

Diese Arbeit widmet sich der Umsetzung einer selbstadaptiven Steuerung des Selektionsdruckes für die Evolutionsstrategie, die den Selektionsdruck anschließend zu jedem Zeitpunkt der Ausführung selbstständig anpasst. Diese selbstadaptive Steuerung ist anhand des Konzeptes der *Offspring Selection*, das bereits erfolgreich in anderen Optimierungsverfahren eingesetzt wurde, in die $(\mu/\rho^{+},\lambda)$-ES und die CMA-ES zu integrieren. Die Funktionsweise dieser neuen Evolutionsstrategien soll anschließend anhand einiger bekannter Benchmark-Probleme getestet und evaluiert werden.

Kapitel 2

Evolutionäre Algorithmen

2.1 Iteratives Problemlösen durch Evolution

Die ersten Ansätze und Ideen in Richtung *Evolutionärer Algorithmen* gab es bereits in den 50er und 60er Jahren [Wei02]. Dabei wurden verschiedenste Betrachtungsweisen und Zielsetzungen verfolgt, unter anderem auch die Lösung von Optimierungsproblemen. Einige Zeit später, in den 70er und 80er Jahren, entstanden unter dem Fokus der Optimierung einige noch heute relevante Arten an Algorithmen, die sich unter dem Begriff der *Evolutionären Algorithmen* (kurz *EA*) sammelten und alle eine Gemeinsamkeit teilen: Sie sind inspiriert durch den in der Natur vorkommenden, biologischen Evolutionsprozess. Ein bekannter Vorreiter in der Entwicklung von Evolutionären Algorithmen, John H. Holland, formulierte dazu folgende Zeilen [Hol92a]:

> *Lebewesen sind vollendete Problemlöser. In der Vielzahl der Aufgaben, die sie bewältigen, übertreffen sie die besten Computerprogramme bei weitem - zur Frustration der Programmierer, die Monate oder gar Jahre harter geistiger Arbeit für einen Algorithmus aufwenden, während Organismen ihre Fähigkeit durch den scheinbar ziellosen Mechanismus der Evolution erwerben.*

Dieses Zitat deutet bereits darauf hin, dass der Mechanismus der Evolution in Wirklichkeit sehr zielgerichtet arbeitet. Durch den Einfluss der natürlichen Evolution sind die unterschiedlichen Organismen der Erde perfekt an die jeweilige Umgebung und die damit verbundenen Probleme und Herausforderungen angepasst. Die Evolutionären Algorithmen nutzen das Prinzip der Evolution zur Lösung von Optimierungsproblemen und basieren auf dem Modell der natürlichen, biologischen Evolution, das Charles Darwin bereits im Jahre 1859 erstmals definierte [Dar06]. Die Vielfalt und Komplexität der Lebensformen, die im Laufe der Zeit durch die Evolution auf unserem Planeten entstanden sind, können laut Darwin auf einige wenige Mechanismen zurückgeführt werden. Das Zusammenspiel dieser Mechanis-

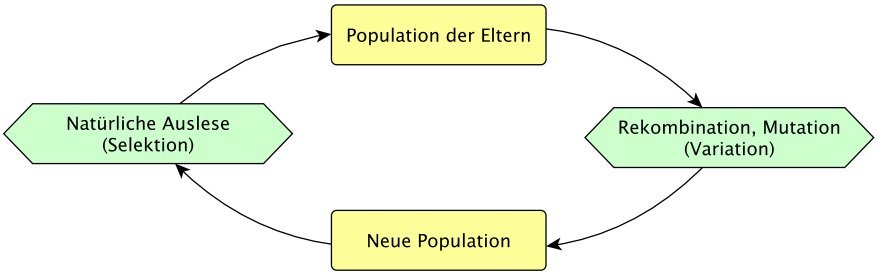

Abbildung 2.1: Der grundlegende Zyklus der Evolution.

men führt letztendlich zur laufenden Anpassung an den vorherrschenden Lebensraum und die damit verbundenen Lebensbedingungen. Ein solcher Mechanismus ist unter anderem die Fortpflanzung von Individuen und die damit verbundene Weitergabe von Erbinformationen von einer Generation zur nächsten. Die Individuen einer neuen Generation unterscheiden sich jedoch von ihren Eltern, da bei der Fortpflanzung Faktoren wie *Mutation* oder *Rekombination* die Erbinformationen verändern oder vermischen. Die so erzeugten Nachkommen weisen unterschiedliche Merkmale auf und sind in Anbetracht ihres Lebensraumes mehr oder weniger konkurrenzfähig. Die *natürliche Auslese* sorgt dann dafür, dass sich die besser angepassten Individuen durchsetzen und wiederum ihre Erbinformationen weitergeben können. Es entsteht ein fortwährender Zyklus der Evolution, der zur kontinuierlichen Weiterentwicklung und Anpassung einer Spezies führt.

2.1.1 Von der Biologie zur Optimierung

Dieser grundlegende Evolutionszyklus ist in Abb. 2.1 grafisch dargestellt. Durch das Wechselspiel aus *Variation* (Mutation und Rekombination) und *Selektion* (natürliche Auslese) lässt sich die schrittweise Entstehung der unterschiedlichen Lebensformen und ihre Anpassung an die verschiedenen Lebensräume erklären. Die *Reproduktion*, also das Vermischen von Erbmaterial, sowie spontane Veränderungen des Erbmaterials durch *Mutation* sind verantwortlich für die Erzeugung neuer, mehr oder weniger konkurrenzfähiger Individuen. An dieser unterschiedlichen Konkurrenzfähigkeit kann die natürliche Auslese ansetzen. Die Weiterentwicklung einer Spezies anhand dieses Prinzips wird oft auch mit dem Begriff *„survival of the fittest“* verbunden, der vom britischen Sozialphilosophen H. Spencer stammt und später von Darwin übernommen wurde. Im Kontext von nur einer Spezies und deren Anpassung an die Umwelt besteht der Evolutionsprozess aus folgenden Punkten:

- Aus einer vorhandenen *Population* von Individuen einer Spezies

- entstehen neue Individuen durch *Rekombination* und *Mutation*,
- welche abhängig ihrer *Fitness* in der aktuellen Umwelt entweder
- in die nächste *Population* einfließen oder
- durch natürliche *Selektion* ausscheiden.

Durch Umlegen dieses Prinzips auf das Problem der Optimierung entsteht folgender Ablauf:

- Aus einer *Population* an potentiellen Lösungen eines Problems
- entstehen neue potentielle Lösungskandidaten durch *Rekombination* und *Mutation*,
- welche abhängig ihrer *Fitness* zur Lösung des Problems
- in die nächste *Population* einfließen oder
- durch *Selektion* ausscheiden.

Konkret kann dies beispielsweise zur Optimierung des *Problems eines Handlungsreisenden* (*Traveling Salesman Problem*, kurz *TSP*) eingesetzt werden. Das *Traveling Salesman Problem* ist eines der am intensivsten untersuchten Probleme im Bereich *Computational Mathematics*[1] [ABC07]. Ausgehend von einer Menge an Städten und deren Distanzen zueinander soll eine Rundreise (nachfolgend auch Tour genannt) erstellt werden, die den kürzest möglichen Weg darstellt, alle Städte zu bereisen und wieder zur Ausgangsstadt zurückzukehren. Es gilt also eine bestimmte Kombination an Städten zu finden, die den minimalsten Weg einer Rundreise beschreibt. Die Schwierigkeit dieses kombinatorischen Optimierungsproblems lässt sich anhand der Berechnung der *Anzahl an möglichen Touren*[2] für eine gewisse Anzahl an Städten verdeutlichen. Diese errechnet sich wie folgt, wobei n die Anzahl an Städten und N die Anzahl an möglichen Touren beschreibt:

$$N = (n-1)! \tag{2.1}$$

Bereits bei nur 14 Städten gibt es mehr als sechs Milliarden mögliche Rundreisen. Bei 22 Städten sind es genau 51.090.942.171.709.440.000 Touren:

> *This computation would require even the fastest supercomputers to roll up its sleeves and prepare for a hard day's work, but with patience it may be possible to carry out the search. If, however, we had one hundred cities or so, then checking all routes to select the shortest is out of the question, even devoting the entire planet's computing resources to the task.*

So beschreibt William J. Cook, bekannt durch seine Arbeiten zum TSP, die Komplexität des Problems [Coo11]. Alle möglichen Lösungen zu berechnen, um danach die beste auszuwählen, ist bei sehr vielen Städten nahezu

[1] Computerunterstütztes Lösen mathematischer Probleme, mehr dazu in [Yan08]
[2] berechnet durch die Anzahl an Permutationen, mehr über Kombinatorik in [JJ03]

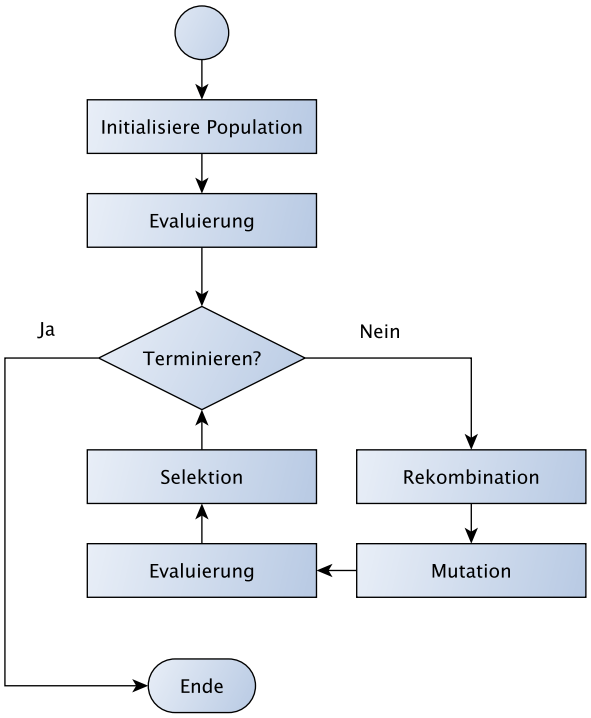

Abbildung 2.2: Der grundlegende Ablauf eines Evolutionären Algorithmus.

unmöglich. Durch die Anwendung eines Evolutionären Algorithmus kann jedoch eine sehr gute, eventuell sogar die optimale Lösung ermittelt werden. Es ergibt sich folgender Ablauf:

- Aus einer vorhandenen *Menge* an möglichen Rundreisen (*Population*)
- entstehen neue mögliche Rundreisen durch den Einsatz von *Rekombination* und *Mutation*,
- welche abhängig der *gesamten Weglänge* der Rundreise (*Fitness*)
- in die nächste *Population* einfließen oder
- ausscheiden (*Selektion*).

Anstelle von Organismen in der Biologie werden hier also TSP-Rundreisen von einer Generation zur nächsten evolviert. So ist es möglich in absehbarer Zeit eine akzeptable Lösung zu finden, ohne jede einzelne der möglichen Touren berechnen zu müssen.

Abb. 2.2 zeigt die wichtigsten Schritte eines EA als Ablaufdiagramm [Wei02]. Bevor der Evolutionszyklus startet wird zunächst die Basispopulation initialisiert und anschließend die *Fitness* der Individuen berechnet. In obigem Beispiel wäre dies das Erstellen einer Menge möglicher TSP-Rundreisen und das Ermitteln der gesamten Weglänge pro Rundreise. Nach

der Fitness-Evaluierung betritt der Algorithmus seine Hauptschleife und erzeugt neue Lösungskandidaten durch den Einsatz von Rekombinations- und Mutations-Methoden. Abhängig vom behandelten Problem sind verschiedene Arten an Operatoren für Rekombination und Mutation möglich. Im Falle des TSP kann z. B. die Rekombination auf Basis zweier Rundreisen eine gänzlich neue generieren. Die Mutation könnte wiederum ganz zufällig die Reihenfolge von zwei Städten einer Rundreise vertauschen. Es ist auch möglich, dass ein Evolutionärer Algorithmus gänzlich auf Rekombination verzichtet und nur Mutation einsetzt. Dies wäre vergleichbar mit Organismen, die keinen Paarungs-Partner benötigen, sondern sich selbst reproduzieren können. Zuletzt wird die Fitness der neuen Lösungen evaluiert und aus den Individuen der neuen und alten Generation die nächste Population selektiert. Wieder am Beispiel des TSP würde dies bedeuten, dass der Prozess der Selektion eine neue Menge an Rundreisen wählt, welche die anfangs erstellte Erstpopulation ersetzt und als Basis für die nächste Generation dient. Man spricht in diesem Zusammenhang auch von einer *Selektion der Überlebenden*. Am Ende jeder Schleifen-Iteration muss entschieden werden, ob der Algorithmus weiterlaufen soll. Gründe für die Terminierung des Algorithmus sind beispielsweise:

- Das Erreichen einer bestimmten maximalen Anzahl an Generationen
- Das Überschreiten einer Grenze der Anzahl an evaluierten Lösungen
- Erreichen eines Zeitlimits
- Erkennen einer Stagnation des Algorithmus (der Algorithmus schafft es nicht mehr bessere Lösungen zu generieren).

Je nach Typ des Algorithmus oder Problemstellung können verschiedene Abbruchbedingungen zum Einsatz kommen.

Algorithmus 2.1: Struktur eines Evolutionären Algorithmus

```
BASIC EVOLUTIONARY ALGORITHM
  t ← 0
  initialize : P(0) ← {s_1, ..., s_n}
  evaluate P(0) : {Φ(s_1), ..., Φ(s_n)}
  while terminate(...) ≠ true do
    recombine : P' ← rec(P(t))
    mutate : P' ← mut(P')
    evaluate P' : {Φ(s'_1), ..., Φ(s'_m)}
    select : P(t + 1) ← sel(...)
    t ← t + 1
  end while
end
```

Der oben dargestellte Algorithmus 2.1 gießt den in Abb. 2.2 präsentierten Ablauf in Pseudocode und umreißt grob die grundlegende Struktur eines EA-

Programms [Nis97][Bäc96]. t entspricht der Anzahl der bisher berechneten Generationen. Die Menge P stellt eine Population dar und enthält die n Lösungen s_i einer bestimmten Generation. In der Zwischenpopulation P' sind die m erzeugten Nachkommen jeder Generation temporär verwaltet. Damit ein Programm die Lösungen verarbeiten kann, müssen diese eventuell in geeigneter Form kodiert werden, z. B. im Binärformat, als Baum oder als Vektor. Die *Fitnesswerte* dieser Lösungen werden durch Anwenden der *Fitnessfunktion* Φ ermittelt. Sie übernimmt üblicherweise zwei Schritte:

- Das Anwenden der *Dekodierungsfunktion* Γ
- und die anschließende Berechnung der Fitness über die *Zielfunktion* F

$$\Phi(s_i) = F(\Gamma(s_i)) \tag{2.2}$$

In vielen Anwendungen ist der Schritt des Dekodierens nicht notwendig, da die Zielfunktion direkt die Fitness berechnen kann. Aus diesem Grund werden die Begriffe Zielfunktion und Fitnessfunktion oft synonym verwendet. Beim TSP ist die Zielfunktion dafür verantwortlich die gesamte Tourlänge einer Rundreise zu berechnen. Der EA soll danach die Tour mit der minimalsten Länge finden. Es handelt sich somit um ein Minimierungsproblem. Ebenso kann der EA eingesetzt werden, um ein Maximum der Zielfunktion zu suchen. Bei der Lösung von Maximierungs- und Minimierungsproblemen besteht grundsätzlich kein Unterschied. Jedes Minimierungsproblem mit der Zielfunktion f ist in ein Maximierungsproblem mit der Zielfunktion $\hat{f} \leftarrow -f$ transformierbar [GKK13]. Die *terminate*-Methode ermittelt für jede Generation ob der Algorithmus beendet werden soll. Das Ergebnis der Methode ist oft durch einige zuvor definierte Zusatzparameter bestimmt, welche die Abbruchbedingungen regeln. Dazu zählt z. B. ein Generationslimit (s. oben). Details zur Funktionsweise der Rekombinations-, Mutations- und Selektions-Operatoren sind in den nachfolgenden Abschnitten 2.1.2, 2.1.3 sowie 2.1.4 beschrieben. Die Evolutionären Algorithmen unterscheiden sich durch den der Biologie entsprungenen Ablauf stark von anderen Optimierungs- und Suchverfahren. Die biologische Terminologie ist deshalb sehr wesentlich für den Umgang mit den EA [Nis97]. Tabelle 2.1 gibt Aufschluss über die wichtigsten Begriffe und deren praktische Bedeutung.

2.1.2 Rekombination

Die Rekombination mischt normalerweise zwei Individuen der Elternpopulation durch den Einsatz eines *Crossover*-Operators und erzeugt Nachkommen, die Eigenschaften beider Elternteile enthalten. Je nach Problemstellung und der dafür gewählten Lösungsdarstellung können sehr unterschiedliche *Crossover*-Operatoren zum Einsatz kommen [Bäc96]. Eine weit verbreitete Darstellung ist die binäre Kodierung, die Lösungen als Bit-Strings verwaltet. Für diese Darstellung existieren viele etablierte Crossover-Operatoren,

Tabelle 2.1: EA-Terminologie im Überblick.

Begriff	*Praktische Bedeutung*
Individuum	Lösung (in geeigneter Weise repräsentiert, z. B. Vektor, Bitstring, Liste, Baum, usw.)
Population (von Individuen)	Menge von Lösungen
Eltern	zur Reproduktion selektierte Lösungen
Kinder, Nachkommen	aus den Eltern erzeugte Lösungen
Mutation	Operator, der jeweils ein Individuum modifiziert
Rekombination, Crossover	Operator, der Merkmale zweier Individuen vermischt
Fitness	Qualität der Lösung bezogen auf die Ziele
Generation	Verfahrensiteration

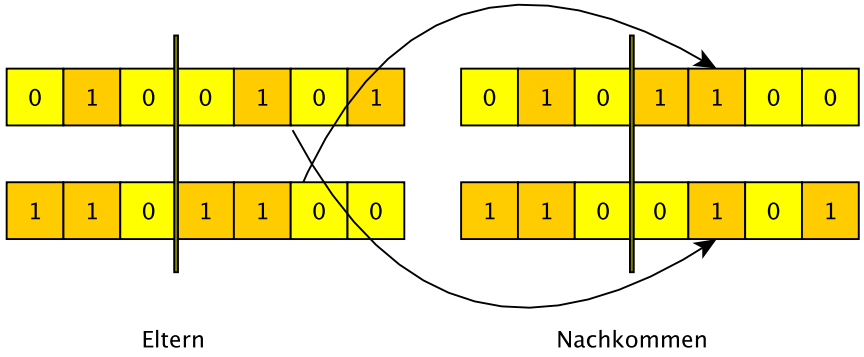

Abbildung 2.3: Rekombination mittels Single Point Crossover (SPX).

die aus zwei Bit-Strings einen neuen kombinieren. Abb. 2.3 zeigt ein Beispiel des *Single Point Crossover* (*SPX*). Der Operator wählt eine zufällige Schnittposition und erzeugt zwei Nachkommen durch einfaches Vertauschen der Bits nach dem Schnittpunkt. Weitere Operatoren zur Rekombination in binärer Darstellung werden in dieser Arbeit nicht angeführt, sind aber in anderen Werken ausführlich behandelt [Gol89][Hol92b].

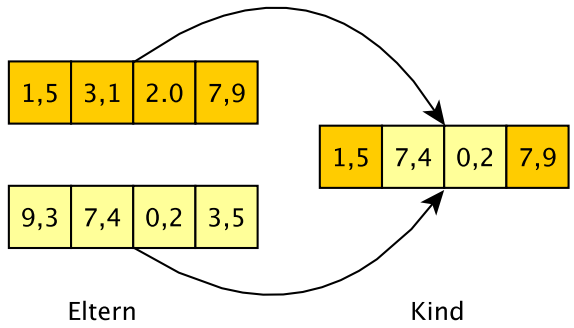

Abbildung 2.4: Funktionsweise der Diskreten Rekombination.

Kodierungsformen von Lösungen

Soll durch die Lösungen ein Graph repräsentiert werden, eignet sich anstelle der binären Darstellung der Einsatz eines Integer-Vektors. Ein Beispiel dafür findet sich wiederum bei der Optimierung des Traveling Salesman Problems [KKK12][RK14][Ahm07]. *Crossover* oder *Mutation* können bei diesem Problem aber ungültige Kind-Lösungen erzeugen, die keine korrekte Rundreise im Sinne des TSP darstellen. Solche ungültigen Lösungen müssen entweder korrigiert, oder in der Fitnessfunktion stark bestraft werden, damit sie durch die Selektion ausscheiden. Eine weitere Möglichkeit wäre das Einführen eines speziellen, problemspezifischen Operators, der von Beginn an nur korrekte Lösungen erzeugt [Üço02]. Ein solcher Operator ist jedoch meist rechenintensiver und langsamer als einfache Operatoren. Dieses Problem der illegalen Routen kommt ebenso in binärer Darstellung vor, wäre dort aber weitaus schwieriger zu handhaben. Solch neue Repräsentationen für spezifische Probleme und ihre zugehörigen Operatoren weisen häufig nur wenig Gemeinsamkeiten mit der klassischen binären Kodierung auf. Jedoch zeigte sich durch zahlreiche Untersuchungen, dass speziell entwickelte Darstellungen und Operatoren im praktischen Einsatz zu sehr effizienten Lösungen führen können [Mic98].

Sehr häufig sind Lösungen auch als reellwertige Vektoren dargestellt. Dies ist die Norm für Zielfunktionen, die n-reellwertige Eingangsparameter (Entscheidungsvariablen) erwarten und daraus den Fitnesswert ermitteln, also Funktionen der Form $F : \mathbb{R}^n \to \mathbb{R}$. Für eine konkrete Zielfunktion f kann der Fitnesswert für $\vec{x} \in \mathbb{R}^n$ dann durch $f(\vec{x})$ ermittelt werden [Nis97]. Ein Beispiel zu einem Crossover in dieser Darstellung ist die in Abb. 2.4 gezeigte *Diskrete Rekombination*. Dieser Operator wählt für jede Entscheidungsvariable des erzeugten Individuums mit gleicher Wahrscheinlichkeit entweder den Wert des einen oder des anderen Elters. Aus Gründen der

Vollständigkeit ist als letzte wichtige Darstellungsform an dieser Stelle noch die Baumstruktur zu nennen, die hauptsächlich im Bereich der *Genetischen Programmierung*[3] ihren Einsatz findet.

Eltern-Selektion

Rekombination bedeutet das Erzeugen von neuen Individuen durch Vermischen von Individuen der Eltern-Population. Die *Eltern-Selektion* trägt dabei die Verantwortung jene Individuen auszuwählen, die als Eltern für die Kreuzung dienen [Wei09]. Realisiert wird dies durch einen Pool an Individuen, der auf Basis eines bestimmten Selektions-Mechanismus aus der aktuellen Population befüllt wird. Der gewählte *Crossover*-Operator erzeugt nun aus den Individuen dieses *Paarungs-Pools* (engl. *Mating Pool*) die Nachkommen. Die *Eltern-Selektion* kann unterschiedliche Strategien verfolgen. Durch ein Befüllen des Paarung-Pools ausschließlich mit besonders guten Individuen sind schlechte Lösungen von der Paarung ausgeschlossen und erzeugen keine Nachkommen. Weiters ist es möglich ein bestimmtes Individuum mehrfach in den Paarungs-Pool zu wählen. Ein solches Individuum dient dann in der selben Generation bei unterschiedlichen Paarungen als Elter und erzeugt mehrfach Nachkommen. Die *Fitnessproportionale Selektion* selektiert z. B. Individuen auf Basis ihrer Fitness im Verhältnis zur durchschnittlichen Fitness der Population. Je besser das Individuum im Gegensatz zum Durchschnitt abschneidet, desto öfter kommt es im Paarungs-Pool vor. Der Paarungs-Pool könnte aber auch durch einen reinen Zufallsmechanismus befüllt werden, und somit keine Individuen bei der Paarung bevorzugen.

Da die Eltern-Selektion Individuen von der Paarung ausschließen und andere mehrfach wählen kann, ist sie vergleichbar mit der am Ende jeder Generation stattfindenden *Selektion der Überlebenden*, und kann diese gegebenenfalls sogar ersetzen.

- Die Eltern-Selektion kann Individuen von der Paarung ausschließen. Diese erzeugen somit keine Nachkommen
- Die Überlebenden-Selektion kann Individuen von der nächsten Population ausschließen. Diese erzeugen ebenso keine Nachkommen mehr.

Der Evolutionäre Fortschritt entsteht durch das Wechselspiel von *Variation* und *Selektion*. Die Wahl einer passenden Lösungsdarstellung, der richtigen Operatoren, sowie eines geeigneten Selektions-Mechanismus ist von größter Wichtigkeit für den Erfolg eines Evolutionären Algorithmus. Mehr dazu in Abschnitt 2.1.4.

[3] Ein Spezialgebiet der Evolutionären Algorithmen, das darauf abzielt Programme zu generieren [Koz93].

2.1.3 Mutation

Wie bereits bei den *Crossover*-Operatoren existieren abhängig der Lösungskodierung verschiedenste *Mutations*-Operatoren zur Veränderung eines Individuums. Holland beschrieb die Mutation als „Hintergrund-Operation“, die ab und zu bei einigen Nachkommen der binär kodierten Lösungen ein zufälliges Bit invertiert [Hol75]. Sie kann aber ebenso als Haupt-Operator fungieren und auf jedes Individuum einer Generation einwirken. Unabhängig von der Häufigkeit der Anwendung hat der Operator immer die Aufgabe zufällige Veränderungen an den Nachkommen vorzunehmen. Ein simpler Operator bei reellwertiger Kodierung modifiziert z. B. jede Komponente des n-dimensionalen Lösungsvektors $\vec{x}$ auf Basis einer bestimmten Standardabweichung σ [BS02]:

$$\vec{x'} \leftarrow \vec{x} + \sigma(N_1(0,1), \ldots, N_n(0,1)) \tag{2.3}$$

Wobei $N_i(0,1)$ jeweils einen unabhängigen normalverteilten Zufallswert zwischen Null und Eins darstellt. Der Strategieparameter σ kann bei diesem Operator nun dazu eingesetzt werden, die *Mutationsweite*, also das Ausmaß der Zufallsänderungen, zu steuern. Die Mutationsweite ist maßgeblich für die Qualität der erzielten Lösungen eines Algorithmus, der diesen Operator einsetzt. Die korrekte Wahl der Mutationsweite ist jedoch sehr schwierig und abhängig von der Beschaffenheit der so genannten *Fitnesslandschaft*, die durch die Zielfunktion über den *Suchraum* gespannt wird.

Fitnesslandschaft und Suchraum

Als *Suchraum* bezeichnet man die Menge aller möglichen Lösungen eines Optimierungsproblems. Im Beispiel des TSP umfasst der Suchraum also alle möglichen Touren. Bei reellwertigen Optimierungsproblemen ist der Suchraum bei einem Lösungsvektor mit n-Dimensionen, sofern keine Beschränkungen existieren, der gesamte Raum $\mathbb{R}^n$. Ordnet man nun jedem Punkt dieses Suchraums, also jeder Lösung, den durch die Zielfunktion ermittelten *Fitnesswert* zu, ergibt sich die *Fitnesslandschaft* des Optimierungsproblems. Man kann sich diese Fitnesslandschaften als einen Blick auf eine ländliche Gegend aus der Vogelperspektive vorstellen, wobei die Höhe jedes Punktes analog zum Fitnesswert der entsprechenden Lösung ist [LP02]. Abb. 2.5 zeigt eine mögliche Fitnesslandschaft eines zwei-dimensionalen Suchraumes. Da hier nur zwei Parameter zu optimieren sind, lässt sich die Zielfunktion als eine drei-dimensionale Gebirgslandschaft darstellen. In höheren Dimensionen mit n-Entscheidungsvariablen muss die Zielfunktion als abstrakte $(n+1)$-dimensionale Gebirgslandschaft gesehen werden, die sich jedoch nicht mehr darstellen lässt [GKK13]. Durch multi-dimensionale Skalierung kann versucht werden einen höher-dimensionalen Datensatz in eine niedrigere Dimension zu skalieren, wobei die Struktur möglichst erhalten bleiben sollte.

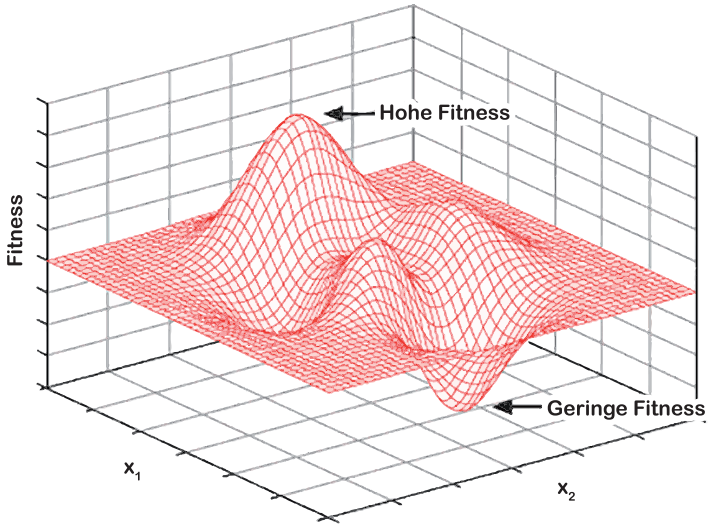

Abbildung 2.5: Eine beispielhafte Fitnesslandschaft eines 2-dimensionalen Suchraums.

Denn die Fitnesslandschaft ist nur dann aussagekräftig, wenn ähnliche Lösungen auch als benachbarte Punkte in der Landschaft abgebildet sind. Ein simpler Optimierer kann dann als kurzsichtiger Wanderer gesehen werden, der versucht das tiefste Tal oder den höchsten Berg der Landschaft zu finden. Ausgehend von einem zufälligen Punkt auf der Karte versucht er dieses Ziel mit einer möglichst kleinen Anzahl an Schritten zu erreichen. Die *Mutationsweite* regelt dabei die maximale Schrittweite, also Distanz, die eine Lösung auf der Landschaft durch den Mutations-Operator zurücklegen kann. Bei einer sehr kleinen Schrittweite könnte der Algorithmus bei einem niedrigeren Berg „hängen bleiben“. Man bezeichnet einen solchen Berg auch als *lokales Optimum*. Das *globale Optimum*, also der höchste Berg, wird dann nicht gefunden. Bei einer zu großen Schrittweite endet das Ergebnis der Mutation eher in einem zufälligen Sprung an einen anderen Ort, als in einer kontrollierten, schrittweisen Suche. Die Mutation sollte also immer ein auf das Problem passendes Ausmaß an Veränderung einbringen. Bei manchen Lösungsdarstellungen und anderen Mutations-Operatoren, ist es grundsätzlich gar nicht möglich das Ausmaß der Veränderung einer Mutation vorauszusagen. Das Vertauschen von zwei zufälligen Städten einer TSP-Rundreise könnte beispielsweise die Rundreise nur minimal, aber auch extrem stark verändern.

Zusammenspiel von Rekombination und Mutation

Rekombination und Mutation sind verantwortlich für das Finden neuer Lösungen im Suchraum. Sie werden daher auch als *Suchoperatoren* bezeichnet.

Dabei spielt die Balance zwischen *Breitensuche* (*Exploration*) und *Tiefensuche* (*Exploitation*) eine wichtige Rolle. Am Beispiel einer Fitnesslandschaft bedeutet die Tiefensuche das schrittweise Finden eines Berges oder Tals in der Nachbarschaft einer Lösung. Die Breitensuche wäre hingegen in der Lage, bisher unbekannte, weiter entfernte Berge oder Täler zu finden. Die Suchoperatoren können je nach Problemstellung und Parametrierung eher explorativ oder exploitativ arbeiten. Eine grundsätzlich sehr starke Mutation wäre beispielsweise eher explorativ. Der Einsatz passender Suchoperatoren ist demnach ausschlaggebend für die Qualität der erzielten Lösungen. Es ist auch möglich Rekombination als eher exploitativen und Mutation als mehr explorativen Operator zu implementieren, um eine ausgewogene Mischung der Suchvarianten zu erreichen.

2.1.4 Selektion

Während Rekombination und Mutation verantwortlich für die Suche nach neuen Lösungen sind, hat die *Selektion* die Aufgabe den Optimierungsprozess in bestimmte, besonders vielversprechende Bereiche des Suchraumes zu treiben. Selektions-Operatoren wählen die Menge jener Lösungskandidaten, die in der nächsten Iteration untersucht werden [Wei09]. Sie können entweder eine kleine Gruppe der besten Lösungen oder eine große Breite an Kandidaten selektieren. Lässt die Selektion eine große Breite verschiedener Individuen zu, arbeitet der Algorithmus eher *explorativ*. Das Einschränken auf einige wenige, besonders fitte Individuen jeder Generation ist stark *exploitativ*. Man spricht in diesem Zusammenhang auch von einem niedrigen oder hohen *Selektionsdruck*.

Eine mögliche Selektionsstrategie bei einer Population von n ist das Auswählen der n besten Individuen aus einer Menge von m erzeugten Nachkommen, wobei $m > n$ gilt. So könnten z. B. in jeder Iteration die besten 20 Individuen aus 50 erzeugten Nachkommen als nächste Population gewählt werden. Das Ergebnis dieser *Überlebenden-Selektion* dient später auch als Basis für eine mögliche Paarung durch den Einsatz von *Rekombination*. Die dabei notwendige *Eltern-Selektion* kann den vorhandenen Selektionsdruck, der bereits durch die *Überlebenden-Selektion* besteht, noch zusätzlich erhöhen. Typischerweise fokussieren konkrete Algorithmen meist nur eine der beiden Selektionsformen. In obigem Beispiel wäre demnach im Anschluss an die *Überlebenden-Selektion* eine rein zufällige Selektion der Eltern die typische Wahl. Weitere Selektionsstrategien erlauben neben den Nachkommen auch das Einbinden der Eltern in den Selektionsprozess. Dies würde Ermöglichen, dass besonders gute Lösungen über viele Generationen erhalten bleiben.

Zusammenspiel von Selektions- und Suchoperatoren

Um die optimale Lösung eines Problems zu finden, muss der Algorithmus ein gutes Verhältnis zwischen Exploitation und Exploration umsetzen. Dabei spielt die richtige Kombination von Selektions- und Suchoperatoren eine zentrale Rolle. Der Einsatz von sehr explorativen Suchoperatoren, kann beispielsweise durch einen höheren Selektionsdruck ausgeglichen werden. Evolutionäre Algorithmen, die Exploitation über Exploration stellen, konvergieren schneller, jedoch riskieren sie dabei, die optimale Lösung nicht zu finden. Auf der anderen Seite können Algorithmen, die zu stark explorativ arbeiten, ihre Lösungen eventuell nie gut genug verbessern um das globale Optimum zu finden - oder es wird nur per Zufall nach sehr langer Zeit entdeckt. Zahlreiche Werke widmen sich bereits diesem Problem der Ausgewogenheit von Breiten- und Tiefensuche, das bei jedem Optimierungsalgorithmus eine Rolle spielt [ECS89][Hol92b][ES98].

2.2 Evolutionsstrategien und Genetische Algorithmen

Seit den ersten Versuchen, Optimierungsprobleme auf Basis der Evolution zu lösen, wurde viel auf diesem Gebiet geforscht. Speziell auch die rasante Entwicklung der Computertechnik und die damit verbundenen Rechenkapazitäten sorgten stark für immer mehr Interesse und neue Anwendungsgebiete. Heute unterscheidet man vier Hauptströmungen der Evolutionären Algorithmen, die in Abb. 2.6 dargestellt sind [Nis97]:

- Der Bereich *Genetische Algorithmen* (Genetic Algorithms, GA),
- die *Evolutionäre Programmierung* (Evolutionary Programming, EP),
- die *Evolutionsstrategien* (Evolution Strategies, ES)
- und die *Genetische Programmierung* (Genetic Programming, GP).

Bereits 1965 arbeiteten Ingo Rechenberg und Hans-Paul Schwefel in Berlin an der Optimierung des Windwiderstandes eines stromlinienförmigen Körpers. Nachdem sie das Problem mathematisch nicht lösen konnten entwickelten sie die Idee, den Körper auf Basis einfacher evolutionärer Prinzipien zu optimieren. So entstand die *Evolutionsstrategie* [Rec73][Sch74]. Ungefähr zeitgleich begann Lawrence J. Fogel in San Diego mit dem Entwurf künstlicher, intelligenter Automaten und entwickelte dabei die *Evolutionäre Programmierung* [Fog66]. Die *Genetischen Algorithmen* entstanden Mitte der 70ger durch die Forschungen von John H. Holland und sind heute die wahrscheinlich meist verbreitetste Form der Evolutionären Algorithmen [Hol75]. Neben den vielen Varianten, die zu Genetischen Algorithmen bisher entwickelt wurden, konnte speziell die *Genetische Programmierung*, stark beeinflusst durch John R. Koza, besondere Bedeutung erlangen [Koz93]. Algorithmen dieses Bereichs versuchen ganze Programme zu generieren, indem

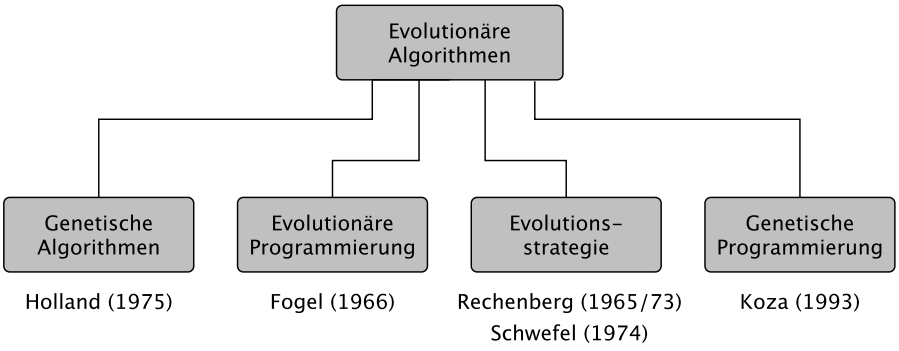

Abbildung 2.6: Taxonomie der Evolutionären Algorithmen.

Lösungen in Form von Computerprogrammen dargestellt und mit Mechanismen der Evolution verändert werden. Diese Arbeit widmet sich der möglichen Verbesserung der *Evolutionsstrategie* durch Implementieren eines neuen Konzepts, das zuvor erfolgreich in *Genetischen Algorithmen* umgesetzt wurde. Im folgenden werden daher nur diese beiden Gebiete, mit Fokus auf die ES, näher beschrieben.

ES setzen als Suchoperator hauptsächlich die Mutation ein und erzielen evolutionären Fortschritt durch eine Selektion der Überlebenden am Ende jeder Generation. In ihrer einfachsten Form wird sogar ganz auf Rekombination verzichtet. Ein weiteres Merkmal der ES ist die Tatsache, dass sie hauptsächlich zur Optimierung kontinuierlicher Entscheidungsvariablen, also reellwertiger Vektor-Lösungen, eingesetzt wird. Die erste Variante der Evolutionsstrategie war noch nicht populations-basiert, sondern verbesserte pro Generation immer nur ein Individuum. Diese spezielle Form der Evolutionsstrategie ist auch als (1+1)-Evolutionsstrategie bekannt.

2.2.1 Die (1+1) - Evolutionsstrategie

Die (1+1)-Evolutionsstrategie erstellt eine zufällige Lösung, erzeugt durch Mutation genau einen Nachkommen und ersetzt die Eltern-Lösung eventuell, falls die neue Lösung besser ist. Es existiert also immer nur eine aktuelle Lösung in der Population, die so lange mutiert wird, bis eine bessere gefunden wird. Die Mutation wird dabei über zufällige, normalverteilte Änderungen jeder Komponente des Lösungsvektors durchgeführt:

$$\vec{x'} \leftarrow \vec{x} + \sigma(N_1(0,1), \ldots, N_n(0,1)) \tag{2.4}$$

Wobei jede Komponente x_i der Lösung über die Schrittweite σ modifiziert wird. Untersuchungen zeigten, dass die Konvergenzgeschwindigkeit verbessert werden kann, indem die Mutationsweite laufend angepasst wird [Rec73].

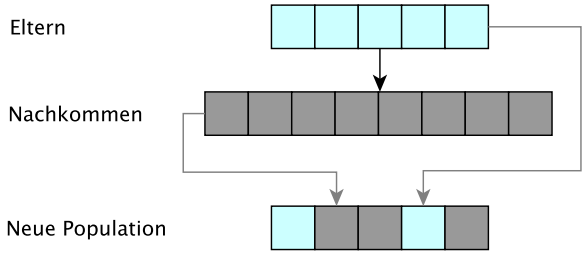

Abbildung 2.7: Selektion einer neuen Population bei der $(\mu + \lambda)$-ES.

Dabei kristallisierte sich heraus, dass durchschnittlich 1/5 der Mutationen zu einem besseren Ergebnis führen sollten, der Rest sollte schlechtere Lösungen produzieren. Wird dieser Grenzwert unter- oder überschritten muss die Mutationsweite σ erhöht oder gesenkt werden. Dieses Konzept ist heute auch als *1/5 Erfolgsregel* bekannt.

Später wurden auch populations-basierte Evolutionsstrategien entwickelt, wobei sich auch dort eine selbstadaptive Anpassung der Mutationsweite als wichtig herausstellte [HOG95][MB07]. Zur Spezifikation der Populationsgröße und der Anzahl an daraus erzeugten Nachkommen existiert bei den ES eine etablierte Darstellungsform im Stil $(\mu + \lambda)$. Übersetzen würde man dies in eine ES mit Populationsgröße μ und λ erzeugten Kindern pro Generation.

2.2.2 (μ+λ) und (μ,λ)-Notation

Die erste populations-basierte Evolutionsstrategie, die $(\mu + 1)$-ES, erzeugte wie auch die $(1 + 1)$-ES genau einen Nachkommen in jeder Generation. Dieser Nachkomme wurde jedoch durch Rekombination zweier Eltern einer Population von μ Individuen erzeugt, und zusätzlich noch mutiert. Studien zeigten, dass dieser Algorithmus dazu tendiert die Mutationsweite laufend zu verringern, egal ob dies vorteilhaft ist oder nicht, weshalb Schwefel daraufhin zwei weitere Varianten entwickelte [BS02]:

- Die $(\mu + \lambda)$-ES, die eine Menge $\lambda \geq 1$ an Nachkommen ohne Rekombination erzeugt und anschließend mutiert. Abb. 2.7 stellt den Selektionsprozess grafisch dar. Bei dieser Form der ES wird die neue Population aus den besten Individuen der Eltern und der Nachkommen zusammengestellt.
- Die (μ, λ)-ES, die eine Menge $\lambda > \mu$ an Nachkommen erzeugt, jedoch die Eltern in jedem Fall von der nächsten Population ausschließt und die neue Population nur aus den Nachkommen λ wählt (Abb. 2.8).

Das Selektionsverfahren ist in beiden Fällen, abgesehen von einem möglichen Einbinden der Eltern, immer gleich. Durch einfache *Deterministische Selektion* werden die jeweils μ besten Individuen zur neuen Population ge-

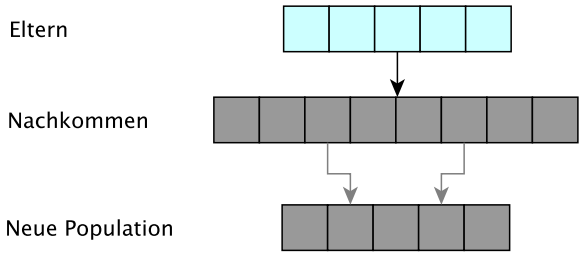

Abbildung 2.8: Selektion einer neuen Population bei der (μ, λ)-ES.

wählt. Bei der (μ, λ)-ES umfasst die Lebensdauer eines Individuums maximal eine Generation, da es danach prinzipiell von der Selektion ausgeschlossen ist. Es kann daher vorkommen, dass gute Zwischen-Lösungen wieder verloren gehen, sofern durch die Mutation ausschließlich schlechtere Lösungen entstehen. Die beste bisherige Lösung muss in dieser Form daher separat gespeichert werden. Weiters ist es bei der $(\mu + \lambda)$-ES einfacher Konvergenz zu gewährleisten, da im schlimmsten Fall eine verfrühte Stagnation auftritt, z. B. wenn die Mutationsweite zu klein wird, bevor das Optimum erreicht ist. Im Gegensatz dazu kann die Komma-Variante sogar divergieren, vor allem wenn die Mutationsweite zu groß ist. Die Idee zu den populations-basierten Varianten war hauptsächlich aus zwei Gründen motiviert:

- Auf parallelen Rechner mit λ Kernen, könnte die Menge der Nachkommen gleichzeitig berechnet werden. Die Geschwindigkeit der Evolution einer Generation wäre somit stark erhöht.
- Selbstadaption zu einer optimalen Mutationsweite erfordert einen gewissen Nachkommens-Überschuss, da ansonsten möglicherweise kleinere Mutationsweiten bevorzugt werden. Weiters kann die Komma-ES eine Selbstadaption unterstützen, indem sogar bei einem negativen Fortschritt durch zu große Mutationsweiten die Nachkommen mit dem geringsten Rückschritt bevorzugt werden.

Anstelle der Selbstadaption einer einheitlichen Schrittweite σ wurde mit den neuen populations-basierten ES auch die Möglichkeit eingeführt, jede Komponente der Lösungsvektoren durch eine eigene Schrittweite zu optimieren. Denn für unterschiedliche Dimensionen des Lösungsraumes sind wahrscheinlich auch unterschiedliche Mutationsweiten vorteilhaft:

$$\vec{x'} \leftarrow \vec{x} + (N_1(0, \sigma_1), \ldots, N_n(0, \sigma_n)) \tag{2.5}$$

Wobei jede Lösungskomponente x_i über die zugehörige Schrittweite σ_i angepasst wird. Die Schrittweite ist dabei anfangs fix gewählt und wird danach gemeinsam mit den Lösungskomponenten als Bestandteil der Lösung

mitentwickelt[4]. Somit werden die Schrittweiten im Laufe des Algorithmus mitoptimiert. Der Lösungsvektor umfasst dann die Menge der Zielvariablen, sowie ebenso die Menge der Standardabweichungen. Die Selbstadaption der Schrittweiten übernimmt danach die Evolutionsstrategie, da bei der Selektion schlechte Lösungen ausscheiden, die durch falsche Schrittweiten entstehen. In jeder Iteration muss dafür aber noch zusätzlich vor der Mutation der Eingangsvariablen x_i jede Schrittweite σ_i zufällig mutiert werden. Dies geschieht durch Multiplikation mit einer logarithmisch normalverteilten Zufallsgröße [Nis97]:

$$\sigma_i' \leftarrow \sigma_i \; exp(\tau_1 \; N(0,1) + \tau_2 \; N_i(0,1)) \tag{2.6}$$

Die Strategieparameter τ_1 und τ_2 regeln dabei das Ausmaß der Veränderung, also die Mutationsweite, für σ. Der Teil $\tau_1 \; N(0,1)$ gilt dabei global für alle σ und wird einmal pro Generation ermittelt. $\tau_2 \; N_i(0,1)$ hingegen ist unabhängig für jedes σ_i zu berechnen. Traditionell galt für die Wahl der Parameter folgende Empfehlung, bei einer Dimensionsgröße von n :

$$\tau_1 = (\sqrt{2n})^{-1}, \tag{2.7}$$

$$\tau_2 = (\sqrt{2\sqrt{n}})^{-1} \tag{2.8}$$

Spätere Forschungen zeigten jedoch, dass diese Werte nicht immer optimal sind, weshalb günstige Werte für beide Parameter aktuell eher bei $0,1$ bis $0,2$ liegen.

Algorithmus 2.2: Aufbau der (μ^+,λ)-Evolutionsstrategie

```
BASIC EVOLUTION STRATEGY
  choose parameters: μ, λ, σ_i, τ_1, τ_2, ...
  t ← 0
  initialize : P(0) ← {s_1, ..., s_μ}        ▷ s_i ← (x_1, ..., x_n, σ_1, ..., σ_n)
  evaluate P(0) : {Φ(s_1), ..., Φ(s_μ)}
  while terminate(...) ≠ true do
    mutate : P' ← mut(P(t), τ_1, τ_2)          ▷ mutate all σ_i and x_i
    evaluate P' : {Φ(s'_1), ..., Φ(s'_λ)}
    select : P(t+1) ← sel(P'|P' ∪ P(t))        ▷ different for , and +-ES
    t ← t + 1
  end while
end
```

Alg. 2.2 veranschaulicht den Ablauf einer (μ^+,λ)-ES. Als erster Schritt ist die Wahl der Strategieparameter vorzunehmen. Bei einer Verwendung von

[4] Bäck empfiehlt für sämtliche Standardabweichungen einen Startwert $\sigma_i = 3,0$ [Bäc96].

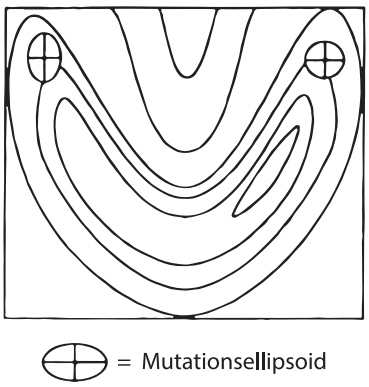

Abbildung 2.9: Grafische Darstellung einer unkorrelierten Mutation.

Standardwerten der Parameter für die Selbstadaption der Schrittweiten ist hauptsächlich die Größe der Population μ und die Anzahl der Nachkommen λ zu wählen. Hierbei ist anzumerken, dass bei großen Populationen mit sehr unterschiedlichen Werten der Zielvariablen und Schrittweiten in der Population die Unterschiede zwischen der Komma- und der Plus-Variante der ES immer mehr verblassen.

2.2.3 MSC-ES und CMA-ES

Ein Problem der Evolutionsstrategie in dieser Grundform ist, dass die Mutationen in den einzelnen Problemdimensionen, also den Entscheidungsvariablen, unabhängig voneinander stattfinden. Man bezeichnet dies auch als *Unkorrelierte Mutation*. Dadurch ist es nicht möglich, der lokal besten Suchrichtung unmittelbar zu folgen. Veranschaulichen lässt sich dies anhand eines zwei-dimensionalen Suchraumes durch das Einzeichnen der möglichen Lösungen, die durch Mutation einer Lösung erreicht werden können (Abb. 2.9). Jede Lösungskomponente wird anhand einer bestimmten Mutationsweite angepasst, wodurch bei zwei Dimensionen ein *Mutationsellipsoid* aufgespannt wird, das die durch die Mutation erreichbaren Lösungen beschreibt. Dieses Ellipsoid liegt bei einer unkorrelierten Mutation immer parallel zu den Achsen des Suchraumes. Versteht man die Linien im Bild als Höhenlinien[5] einer bestimmten Fitnesslandschaft wird klar, dass der Algorithmus schneller ans Ziel kommen würde, wenn die Ellipsoiden in Richtung Gipfel des Berges gedreht wären. Abb. 2.10 zeigt die selbe Landschaft mit entsprechenden Ellipsoiden bei *Korrelierter Mutation*. Die Ausrichtung der Ellipsoiden ändert sich in diesem Fall, denn sie können sich dem Verlauf der Fitnesslandschaft anpassen und ermöglichen der Evolutionsstrategie somit schneller bei der Optimierung voranzuschreiten.

[5] Alle Punkte einer Höhenlinie weisen denselben Fitnesswert auf.

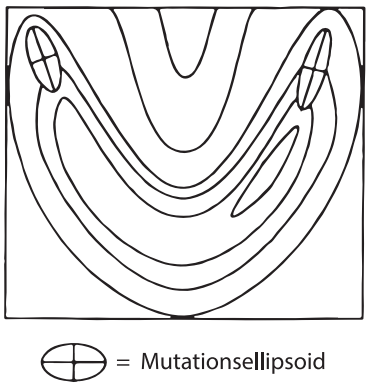

Abbildung 2.10: Grafische Darstellung der korrelierten Mutation.

Eine solche Darstellung eines *Mutationsellipsoids* kann dabei auch als eine in der Statistik übliche Visualisierung der Dichte einer zwei-dimensionalen Normalverteilung betrachtet werden[6] [HE06]. Sämtliche Punkte, die über die Mutation erzeugt werden, fallen normalverteilt in diesen Raum. Die Abbildung der Varianz von mehrdimensionalen Normalverteilungen ist in der Statistik üblicherweise durch eine *Kovarianzmatrix* Σ definiert. Diese Kovarianzmatrix generalisiert die Notation der Varianz in mehreren Dimensionen, denn eine einzige Zahl reicht nicht mehr aus um die Streuung von zufälligen Punkten eines beispielsweise zwei-dimensionalen Raumes anzugeben. In diesem Fall wäre dann bereits eine (2 x 2)-Kovarianzmatrix zur Beschreibung notwendig. Pro Dimension würde eine weitere Zeile und Spalte hinzukommen, also ergibt sich bei n-Dimensionen eine (n x n)-Kovarianzmatrix. Es existieren drei spezielle Formen der Kovarianzmatrix für bestimmte Normalverteilungen [Han98]:

1. Die Kovarianzmatrix Σ entspricht einem Vielfachen der Einheitsmatrix I. Das bedeutet, dass nur eine Varianz σ^2 für sämtliche Dimensionen der Matrix definiert ist: $\Sigma = \sigma^2\ I$. Abb. 2.11a zeigt die Dichteverteilung dieser Form bei zwei Dimensionen. Da nur ein Parameter σ frei wählbar ist ergibt sich hier ein Kreis, bei drei Dimensionen eine Kugel, usw.
2. Die Kovarianzmatrix hat die Struktur einer Diagonalmatrix. Jedes Element der Diagonale ist frei wählbar, wodurch bei n Dimensionen n Variablen möglich sind. In Abb. 2.11b ist diese Form für zwei Dimensionen grafisch dargestellt. Es ergeben sich achsparallele Ellipsen, bzw. bei höheren Dimensionen Hyperellipsen. Jedes Element der Diagonale einer solchen Matrix kann als individuelle Standardabweichung σ_i einer der Achsen gesehen werden. Diese Form entspricht dem zuvor

[6] Punkte im Raum streuen abhängig der Varianz normalverteilt um den Mittelpunkt.

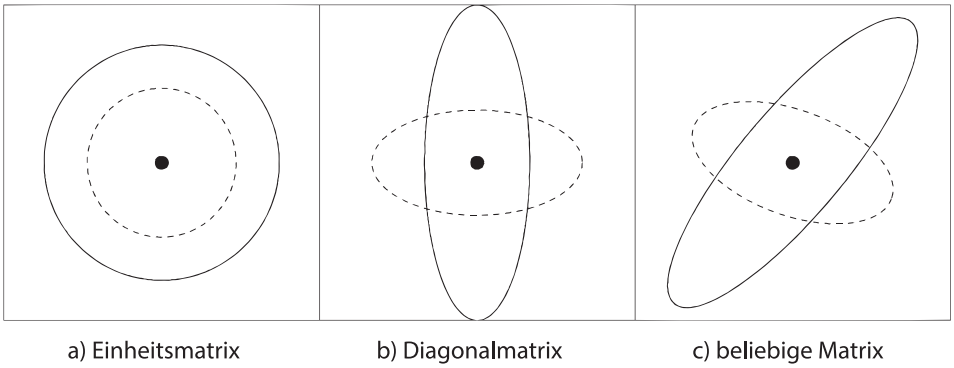

Abbildung 2.11: Spezielle Formen der Kovarianzmatrix und die Visualisierung ihrer Dichte bei einer zwei-dimensionalen Normalverteilung.

dargestellten Ergebnis der bisher *Unkorrelierten Mutation*.

3. Die Kovarianzmatrix ist eine beliebige positiv definite, symmetrische Matrix[7]. Das Ergebnis einer solchen Matrix ist in Abb. 2.11c dargestellt und entspricht einer beliebig orientierten (Hyper-)Ellipse. In diesem Fall weist die Matrix $(n^2+n)/2$ freie Parameter auf. Um *Korrelierte Mutationen* zu erzeugen, muss also diese Form eingesetzt werden. Häufig betrachtet man zunächst eine achsparallele Ellipse (Diagonalmatrix), die anschließend mit Hilfe einer orthogonalen linearen Transformation gedreht wird.

MSC-ES

Die (μ, λ)-MSC-Evolutionsstrategie[8] war die erste ES, die fähig war korrelierte Mutationen anhand einer Kovarianzmatrix durchzuführen [BFK13]. Der Algorithmus wurde von Schwefel entwickelt und ist auch bekannt unter dem Namen CORR-ES. In dieser Strategie wird die Kovarianzmatrix durch $(n^2 - n)/2$ Rotations-Matrizen ermittelt, die auf die bereits existente Diagonalmatrix der bisherigen unkorrelierten Mutation multiplikativ angewendet werden. Die dazu notwendigen Rotationswinkel sind als zusätzliche Strategieparameter realisiert. Neben den n Standardabweichungen erhält jedes Individuum nun auch n_α Rotationswinkel $\alpha_m \in [-\pi, \pi]$ $(m = 1, 2, \ldots, n_\alpha)$ als interne Parameter, sodass eine Lösung aus drei Teilen besteht: $\vec{s}_i \leftarrow (\vec{x}_i, \vec{\sigma}_i, \vec{\alpha}_i)$. Die genaue Anzahl der Rotationswinkel[9] beträgt in diesem Fall $n_\alpha = (n^2-n)/2$ [SR95]. Das ergibt eine Summe von insgesamt

[7] Kovarianzmatrizen müssen immer symmetrisch und positiv definit sein.

[8] MSC ist eine Abkürzung für mutative self-adaption of covariances.

[9] Bei einer unterschiedlichen Anzahl an Schrittweiten n_σ und Dimensionen n wird die Anzahl der Winkel über $n_\alpha = (n - n_\sigma/2)\ (n_\sigma - 1)$ ermittelt.

$(n^2 + n)/2$ Strategieparametern. Da die Qualität der ermittelten Lösungen stark von den Strategieparameterwerten abhängt, ist es wieder vorteilhaft diese selbstadaptiv zu wählen. Die Selbstadaption einer Vielzahl von Parametern kann dabei aber durchaus zeitaufwändig sein, vor allem bei großen Populationen.

Der Ablauf der Mutation erfolgt wie bisher, außer dass zusätzlich noch jeder Rotationswinkel α_j zu mutieren ist, bevor die Entscheidungsvariablen mutiert werden:

$$\alpha'_j \leftarrow \alpha_j + \beta\ N_j(0,1) \tag{2.9}$$

Es wird also wieder ein zufälliger, normalverteilter Wert zu jedem Winkel addiert. Die Mutationsweite β empfiehlt Schwefel auf den Wert $\beta \approx 0,0873$ ($\approx 5°$) zu setzen. Sollte der zulässige Bereich $[-\pi, \pi]$ über- oder unterschritten werden, wird eine zirkuläre Abbildung auf diesen Bereich angewandt, wodurch ungültige Werte vermieden werden. Ist der neue Winkel z. B. um c_α größer als π wird der Wert $-\pi + c_\alpha$ verwendet.

Die Formel zur Mutation der Entscheidungsvariablen ist in der Literatur nun wie folgt definiert [Nis97]:

$$\vec{x}' \leftarrow \vec{x} + \vec{N}(\vec{0}, \vec{\sigma}', \vec{\alpha}') \tag{2.10}$$

$\vec{N}(\vec{0}, \vec{\sigma}', \vec{\alpha}')$ bezeichnet dabei einen normalverteilten Zufallsvektor, generiert mit dem Erwartungswertvektor $\vec{0}$ und der durch die mutierten Standardabweichungen und Winkel beschriebenen Kovarianzmatrix. Zur Berechnung eines solchen Vektors wird mit Hilfe der Standardabweichungen wie bisher ein Zufallsvektor $\vec{\Delta} = \vec{N}(\vec{0}, \vec{\sigma})$ erzeugt. Dieser enthält normalverteilte, aber unkorrelierte Komponenten. Anschließend folgt eine ebenenweise Rotation anhand n_α Multiplikationen mit den Rotationsmatrizen $R_{p,q}(\alpha_{pq})$. Der dafür zu wählende Winkel α_{pq} ermittelt sich durch die Indextransformation $pq = (1/2)\ (2n - p)\ (p + 1) - 2n + q$. Jede Multiplikation entspricht einer Koordinatentransformation hinsichtlich der Achsen p und q anhand des zugehörigen Rotationswinkels α_{pq}. Die n-dimensionalen Rotationsmatrizen $R_{p,q}$ sind definiert über die Einheitsmatrix und folgenden Ausnahmen:

- An den Stellen (p, p) und (q, q) gilt $\cos \alpha_{pq}$.
- Für (p, q) gilt $-\sin \alpha_{pq}$.
- Der Wert für (q, p) ist definiert über $-(p, q) = \sin \alpha_{pq}$

Auf diese Weise werden die normalverteilten aber unkorrelierten Mutationen anhand der Rotationswinkel transformiert. Bei nur zwei Dimensionen existiert genau ein Winkel, bzw. eine Rotationsmatrix. Diese wäre entsprechend obiger Formel wie folgt aufgebaut:

$$R_{1,2}(\alpha) = \begin{pmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{pmatrix} \tag{2.11}$$

Das Erzeugen eines mutierten Lösungsvektors entspricht danach:

$$\vec{x}' \leftarrow \vec{x} + \vec{N}(\vec{0}, \vec{\sigma}') \cdot R_{1,2} = \vec{x} + \begin{pmatrix} \Delta_1^{unkor} \\ \Delta_2^{unkor} \end{pmatrix} \cdot \begin{pmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{pmatrix} \tag{2.12}$$

In Summe erfordert diese Variante keine großen Erweiterungen an der Implementierung der ES-Grundform, jedoch kann der verursachte Rechenaufwand beträchtlich sein, vor allem bei größeren Populationen. Hinzu kommt weiters, dass die hier eingesetzte Selbstadaption von individuellen Schrittweiten durch Mutation laut Ostermeier im Falle kleiner Populationen nur schwer möglich ist [OGH94], wobei zwei Hauptgründe genannt werden: Erstens ist eine gute Mutation der Entscheidungsvariablen nicht unbedingt das Ergebnis von guten Strategieparametern, da auch einfach eine vorteilhafte Instanz des normalverteilten Zufallsvektors auftreten kann. Weiters steht das Ziel, eine große Varianz an Schrittweiten in der Population zu behalten, im Konflikt mit dem Ziel, große Sprünge der Schrittweiten in aufeinanderfolgenden Generationen zu vermeiden. Die Erwartung beliebige Topologien selbstadaptiv durch korrelierte Mutationen bearbeiten zu können erfüllt sich letztendlich nicht, denn die Konvergenzraten liegen nur unwesentlich höher, jedoch bei weit größerem Rechenaufwand. Aus diesem Grund wurde an verschiedenen *Derandomisierten ES* geforscht, die von einer Selbstadaption der Strategieparameter durch Mutation absehen, und versuchen auf andere Weise eine flexible Anpassung an die Fitnesslandschaft umzusetzen. Der Aufbau dieser Varianten unterscheidet sich teilweise stark von der zuvor beschriebenen Standard-ES. Nach einiger Zeit mündeteten die Forschungen und Zwischenvarianten in der heute etablierten CMA-Evolutionsstrategie.

CMA-ES

Die Kovarianzmatrix-Adaption (covariance matrix adaption, CMA) wählt eine der lokalen Topologie angepasste Mutationsverteilung durch ein Auswerten der über die Selektion realisierten Mutationsschritte. Eine passende Kovarianzmatrix wird demnach anhand der überlebenden Individuen ermittelt [Han98]. Die Grundidee basiert auf dem Prinzip, dass die Struktur einer Normalverteilung und damit auch die Kovarianzmatrix, die diese Verteilung beschreibt, auf Basis einer Menge von Punkten gewonnen werden kann. Abb. 2.12 zeigt sechs Vektoren in einem zwei-dimensionalen Raum. Diese Vektoren beschreiben eine durch die Ellipse dargestellte Normalverteilung. Die Kovarianzmatrix C einer solchen Verteilung lässt sich durch das Anwenden folgender Formel für n Vektoren v schätzen:

$$C_{emp} = \frac{1}{n-1} \sum_{i=1}^{n} (v_i - \frac{1}{n} \sum_{j=1}^{n} v_j) \, (v_i - \frac{1}{n} \sum_{j=1}^{n} v_j)^T \tag{2.13}$$

Wurden diese Punkte zuvor anhand einer Kovarianzmatrix C zufällig erzeugt, entspricht die empirische Kovarianzmatrix C_{emp} einer Schätzung der

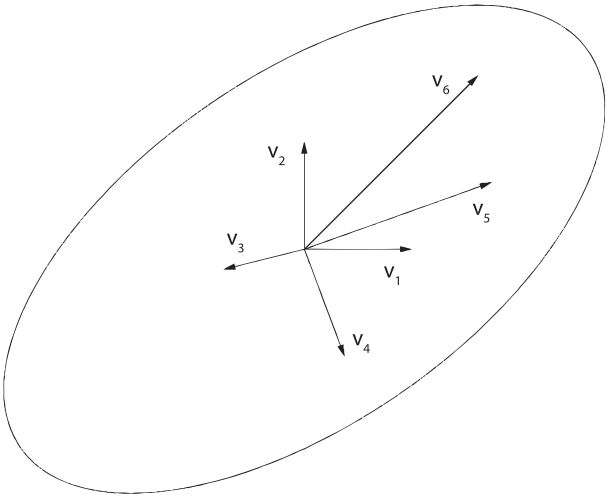

Abbildung 2.12: Visualisierung einer zwei-dimensionalen Normalverteilung anhand der Vektoren $v_1, \ldots, v_6$

tatsächlichen Kovarianzmatrix C. Dieses Prinzip bildet die Grundlage der CMA-Evolutionsstrategie.

Bevor jedoch die genaue Funktionsweise der Adaption der Kovarianzmatrix beschrieben werden kann, ist ein grundsätzlicher Unterschied der CMA-Evolutionsstrategie zur Standard-ES zu beachten: Die Mutations- und Rekombinations-Methoden operieren nicht wie bisher auf jedem Individuum der Population einzeln, sondern betreffen die gesamte Population. Der folgende Abschnitt widmet sich der Umsetzung dieser speziellen Mutation und Rekombination des Algorithmus. Die Mutation der CMA-ES basiert in jeder Generation auf einer Kovarianzmatrix C, die im Laufe des Algorithmus über die selektierten Lösungen an die Topologie der Fitnesslandschaft angepasst wird. Die Nachkommen jeder Generation werden anhand dieser Matrix und genau einer Standardabweichung σ erzeugt:

$$x_{i:\lambda} \leftarrow m + \sigma\, N(0, C) \qquad i = 1, \ldots, \lambda \tag{2.14}$$

Sämtliche Nachkommen $x_{i:\lambda}$ sind gemäß der Kovarianzmatrix C um den Mittelpunkt m zufällig normalverteilt. Es wird also im Gegensatz zur Standard-ES nicht jede Lösung der Population individuell mutiert. Sämtliche Nachkommen sind durch eine Normalverteilung um den Mittelwert der Population zu erzeugen. Der Evolutionäre Fortschritt entsteht dann für eine (μ, λ)-CMA-ES durch folgende Operationen:

1. Die Mutation erzeugt normalverteilt um den Mittelwert von μ Eltern eine Menge an λ Nachkommen.
2. Die Selektion wählt wie bisher die besten μ Nachkommen.

3. Die Rekombination errechnet einen neuen Mittelwert m aus den selektierten Nachkommen.
4. Die Kovarianzmatrix wird anhand der selektierten Lösungen adaptiert.
5. Anhand des neuen Mittelwertes und der angepassten Kovarianzmatrix kann durch Mutation die nächste Generation erzeugt werden.

Die Rekombination ist im Falle der CMA-ES demnach dafür verantwortlich, die selektierten Individuen zu einem neuen Mittelwert m zu kombinieren. Hier sind wiederum verschiedene Rekombinations-Verfahren möglich, eine etablierte Variante ist die Berechnung des neuen Mittelwertes m' als *Gewichteter Durchschnitt* der selektierten Lösungsvektoren [Han07]:

$$m' = \sum_{i=1}^{\mu} w_i \, x_{i:\lambda} \tag{2.15}$$

μ entspricht der Anzahl der Eltern und demnach auch der Anzahl an selektierten Individuen der Nachkommen. Diese Individuen $x_{i:\lambda}$ werden unter Einbezug der Gewichte $w_i \in \mathbb{R}_+$ aufsummiert. Ein Wert von $w_i = 1/\mu$ für alle Gewichte entspricht dabei einem ungewichteten Mittelwert der Punkte. Höhere Gewichte für bessere Lösungen würden den Mittelwert stärker in Richtung dieser Lösungen rücken.

Die Adaption der Kovarianzmatrix baut auf die Berechnung einer empirischen Kovarianzmatrix gemäß Formel 2.13 auf. Es kann durch ein Umlegen dieser Formel auf die Population der Nachkommen die Verteilung der erzeugten Punkte $x_{i:\lambda}$ beschrieben werden [Han07]:

$$C_{emp} = \frac{1}{\lambda - 1} \sum_{i=1}^{\lambda} (x_{i:\lambda} - \frac{1}{\lambda} \sum_{j=1}^{\lambda} x_{j:\lambda}) \, (x_{i:\lambda} - \frac{1}{\lambda} \sum_{j=1}^{\lambda} x_{j\lambda})^T \tag{2.16}$$

C_{emp} entspricht in diesem Fall einer Schätzung der Kovarianzmatrix C, welche die Varianz der generierten Lösungen beschreibt. Ein etwas anderer Ansatz zur Ermittlung der Matrix ist in folgender Formel gegeben:

$$C_\lambda = \frac{1}{\lambda} \sum_{i=1}^{\lambda} (x_{i:\lambda} - m) \, (x_{i:\lambda} - m)^T \tag{2.17}$$

Auch hier entspricht C_λ einer Schätzung der Kovarianzmatrix C, jedoch liegt der Unterschied zwischen den Formeln 2.16 und 2.17 im Referenz-Mittelwert. Bei C_{emp} ist dies der Mittelwert der erzeugten Punkte. Bei C_λ kommt der echte Mittelwert m, der als Basis für die Mutation diente, zu tragen. Während C_{emp} also die Varianz der Lösungen rund um deren Mittelwert beschreibt, schätzt C_λ die Varianz der Lösungen zum originalen Mittelwert, also eher die echte Kovarianzmatrix. Das Ziel der CMA-ES ist es nun, eine bessere Kovarianzmatrix für die nächste Generation zu finden. Um dies zu

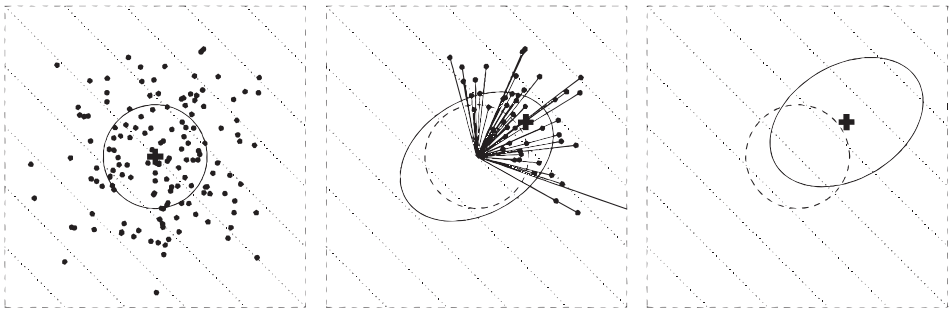

Abbildung 2.13: Ein beispielhafter Ablauf der Kovarianzmatrix-Adaption.

Erreichen kommt die bereits bei der Rekombination eingesetzte *Gewichtete Selektion* zum Einsatz:

$$C' = \sum_{i=1}^{\mu} w_i (x_{i:\lambda} - m)\ (x_{i:\lambda} - m)^T \tag{2.18}$$

Die so ermittelte Matrix C' ist, ebenso wie auch C_λ zuvor, eine Schätzung der Varianz bezogen auf den originalen Mittelpunkt, jedoch jetzt nach dem Anwenden der Selektion. C' beschreibt nun jene Varianz, die auf Basis des gleichen Mittelwerts m nur die besten μ Nachkommen der Generation umfasst. Die Kovarianzmatrix ändert sich also in Richtung der besseren Lösungen. Sie ersetzt anschließend die aktuelle Kovarianzmatrix C und dient als Basis für die Mutation der nächsten Generation.

Abb. 2.13 zeigt ein Beispiel einer Kovarianzmatrix-Adaption nach diesem Prinzip in einem zwei-dimensionalem Raum:

- Ausgehend von einer Einheitsmatrix als initiale Kovarianzmatrix C (kreisförmige Verteilung) werden zufällige Nachkommen erzeugt. Diese sind als Punkte im Raum dargestellt (Abb. 2.13a).
- Die Selektion wählt in Abb. 2.13b die besten μ Punkte. In diesem Beispiel liegt das Optimum oben rechts im Bild, weshalb Punkte in dessen Nähe selektiert werden. Die Rekombination ermittelt den neuen Mittelwert m', und auf Basis der selben Punkte wird die neue Kovarianzmatrix C' geschätzt. Es entsteht eine Kovarianzmatrix, welche rund um den ursprünglichen Mittelpunkt die Form einer Ellipse in Richtung des Optimums beschreibt.
- Abb. 2.13c veranschaulicht den nächsten Mutationsschritt, der den zuvor erzeugten Mittelwert und die neue Varianz kombiniert um neue Lösungen zu erzeugen, die mehr in Richtung des Optimums streuen.

Alg. 2.3 beschreibt die grundlegende Struktur einer CMA-ES Implementierung und setzt die zuvor beschriebenen Methoden und Formeln für Muta-

Algorithmus 2.3: Struktur der CMA-Evolutionsstrategie

```
Basic CMA Evolution Strategy
  choose parameters: μ, λ, σ, w_i, ...
  t ← 0
  C ← I
  repeat
    t ← t + 1
    for i = 1, ..., λ do                                  ▷ create λ children
      mutate : x_{i:λ} ← m + σ N(0, C)
      evaluate : Φ(x_{i:λ})
    end for
    select : sel(x_{1:λ}, ..., x_{λ:λ})                  ▷ select μ best offspring
    recombine : m = Σ_{i=1}^{μ} w_i x_{i:λ}               ▷ move mean value
    C = Σ_{i=1}^{μ} w_i (x_{i:λ} − m) (x_{i:λ} − m)^T     ▷ adapt covariance matrix
    σ ← σ exp(...)                                        ▷ update step width σ
  until terminate(...) ≠ true
end
```

tion, Selektion, Rekombination und Selbstadaption ein. Die Selbstadaption der Kovarianzmatrix durch einfaches Schätzen anhand der selektierten Nachkommen bringt jedoch einige Probleme mit sich. Die so gewonnene adaptierte Kovarianzmatrix ist nur dann zuverlässig, wenn die Anzahl an selektierten Individuen groß genug ist, um die Verteilung gut zu beschreiben. Für eine stark zielgerichtete Suche (im Gegensatz zu einer eher explorativen Suche) ist es jedoch nötig, dass die Anzahl der Nachkommen λ eher gering gehalten ist. Aus diesem Grund wird in moderneren CMA-Varianten die Adaption der Matrix durch komplexere Methoden ersetzt. Diese Methoden unterstützen auch kleinere Populationsgrößen besser, indem auch Informationen vergangener Generationen in die Adaption der Matrix einfließen. Weiters ist auch eine geeignete Form zur Adaption der Schrittweite σ im beispielhaften Algorithmus nicht näher angeführt. Üblicherweise setzt die CMA-ES dafür das Prinzip der *Kumulativen Schrittweitenadaption* (cumulative stepsize control, CSA) ein. Im Rahmen dieser Arbeit ist es ausreichend ein grobes Verständnis für die Funktionsweise der CMA-ES zu schaffen. Für genauere Beschreibungen und neuere Varianten des Algorithmus wird an dieser Stelle an die Werke [Han98], [Han07] und [BFK13] verwiesen.

2.2.4 Rekombination in Evolutionsstrategien

Der Einsatz von Rekombination erweitert die (μ^+,λ)-Notation der Evolutionsstrategien um die Anzahl ρ der pro Rekombination verwendeten Eltern. Es ergibt sich dadurch die erweiterte Schreibweise $(\mu/\rho^+,\lambda)$ [HAA15]:

- Die Elternpopulation besteht aus μ Individuen.

- Für die Rekombination werden jeweils ρ aus μ Individuen verwendet. Demnach gilt $\rho \leq \mu$.
- λ beschreibt die Nachkommen pro Generation.
- $^+_,$ gibt Aufschluss über die Art der Selektion.

Bei der CMA-ES ist die Rekombination fixer Bestandteil des Algorithmus, da sie das Verschieben des Mittelwertes der Population beschreibt. Auch für die CMA-ES ist die Angabe der Anzahl der Eltern ρ üblich. Die in Abschnitt 2.2.3 beschriebene CMA-ES bildet den Mittelwert aus allen μ selektierten Eltern, es gilt in diesem Fall $\rho = \mu$. Häufig wird bei der CMA-ES zusätzlich noch die Art der eingesetzten Rekombination angedeutet. Für eine *Gewichtete Rekombination* anhand der Gewichte w ergibt sich dann $\rho = \mu_w$. Die korrekte Beschreibung einer solchen CMA-ES lautet dann $(\mu/\mu_w, \lambda)$. In der Literatur ist diese Definition auch oft in der Form (μ_w, λ) abgekürzt dargestellt.

Im Gegensatz dazu ist die Rekombination in der Standard-ES optional. Eine $(15, 100)$-ES erzeugt 100 Nachkommen über Mutation aus einer Population von 15 Individuen. Erweitert man diese ES zu $(15/2, 100)$ werden alle 100 Nachkommen durch eine Rekombination von jeweils zwei Individuen erzeugt und anschließend mutiert. Ein Wert $\rho = 2$ ist sehr üblich, jedoch sind auch höhere Werte möglich. Die Rekombination kombiniert dann ein neues Individuum bei z. B. $\rho = 3$ aus den Bestandteilen von drei Eltern. Rekombination ist für die ES keinesfalls unwichtig, wie teilweise behauptet wird. Sie begünstigt die Selbstadaption der Strategieparameter und kann somit die Optimierung beschleunigen. Untersuchungen zeigten, dass es von Vorteil ist unterschiedliche Operatoren für die Rekombination der Entscheidungsvariablen und der Strategieparameter anzuwenden. In Abschnitt 2.1.2 wurde bereits die *Diskrete Rekombination* diskutiert, die häufig als Operator bei den Entscheidungsvariablen eingesetzt wird. Die Rekombination der Strategieparameter ist hingegen meist durch eine Mittelwertbildung der Werte aller Eltern definiert, man bezeichnet dies auch als *Intermediäre Rekombination*. Formal kann man diese Rekombinationen für $\rho = 2$ wie folgt ausdrücken:

$$x_{comb}(j) = x_{E1}(j) \text{ oder } x_{E2}(j) \qquad (j = 1, 2, \ldots, n), \tag{2.19}$$

$$\sigma_{comb}(k) = 0,5 \cdot (\sigma_{E1}(k) + \sigma_{E2}(k)) \qquad (k = 1, 2, \ldots, \sigma_n) \tag{2.20}$$

Jede Komponente j des erzeugten Nachkommens x_{comb} erhält entweder den Wert des Elters E_1 oder E_2, wobei $E_1, E_2 \in (1, 2, \ldots, \mu)$. Die rekombinierten Standardabweichungen σ_{comb} errechnen sich über den Mittelwert der einzelnen Standardabweichungen k. Im Falle einer MSC-ES, die korrelierte Mutationen über mehrere Rotationswinkel α_m beschreibt, müssen diese als zusätzliche Strategieparameter ebenso rekombiniert werden. Üblicherweise kommt dabei der selbe Operator wie auch bei den Standardabweichungen

σ, in unserem Fall eine Intermediäre Rekombination, zum Einsatz [Nis97]. Alg. 2.4 erweitert abschließend den Ablauf einer $(\mu\overset{+}{,}\lambda)$-ES um Rekombination.

Algorithmus 2.4: Aufbau einer $(\mu/\rho\overset{+}{,}\lambda)$-Evolutionsstrategie

```
BASIC RECOMBINATION EVOLUTION STRATEGY
  choose parameters: μ, λ, ρ, σ_i, τ_1, τ_2, ...
  t ← 0
  initialize : P(0) ← {s_1, ..., s_μ}          ▷ s_i ← (x_1, ..., x_n, σ_1, ..., σ_n)
  evaluate P(0) : {Φ(s_1), ..., Φ(s_μ)}
  while terminate(...) ≠ true do
    recombine : P' ← rec(P(t), ρ)                ▷ generate λ children
    mutate : P' ← mut(P', τ_1, τ_2)              ▷ mutate all children
    evaluate P' : {Φ(s'_1), ..., Φ(s'_λ)}
    select : P(t+1) ← sel(P'|P' ∪ P(t))          ▷ different for , and +-ES
    t ← t + 1
  end while
end
```

2.2.5 Genetische Algorithmen

Neben den Evolutionsstrategien liegen auch die *Genetischen Algorithmen* (Genetic Algorithms, GA) im Fokus dieser Arbeit. Die grundlegende Idee hinter den beiden Arten von Evolutionären Algorithmen ist jedoch ganz verschieden. Während die Evolutionsstrategie hauptsächlich die Mutation als Suchoperator einsetzt, und somit eher einer populations-basierten Nachbarschaftssuche entspricht, arbeiten Genetische Algorithmen großteils mit Rekombination. Die Rekombination vereint die Eigenschaften von verschiedenen Lösungen im Suchraum, die eventuell weit voneinander entfernt sind, zu einem neuen Individuum, das nicht unbedingt in der direkten Nachbarschaft der Eltern liegen muss. Die so erzeugten Nachkommen können somit weniger als benachbarte, sondern eher als „verwandte“, also den Eltern ähnliche, Lösungen klassifiziert werden. Man geht davon aus, dass durch eine Weitergabe von den jeweils guten Komponenten der Eltern ein besseres Kind erzeugt werden kann. Besonders gut funktioniert dieser Ansatz verständlicherweise, wenn auch das Optimierungsproblem und die Suchoperatoren diese Metapher möglichst gut unterstützen. Genetische Algorithmen wurden bisher erfolgreich auf verschiedensten Gebieten, vor allem auch zur Lösung kombinatorischer Optimierungsprobleme, eingesetzt. Am Beispiel des in Abschnitt 2.1.1 vorgestellten Traveling Salesman Problems wird dieser Unterschied deutlich:

- Die Evolutionsstrategie sucht durch Mutation einer Menge an Rundreisen nach besseren Lösungen in deren Umfeld. Je größer die Mu-

Tabelle 2.2: GA Terminologie im Überblick.

Begriff	*Praktische Bedeutung*
Chromosom (besteht aus Genen)	üblicherweise identisch mit Individuum[10]
Gen	Freiheitsgrad, Variable eines Chromosoms
Allel	mögliche Ausprägungen eines Gens
Genotyp (bestimmte Sammlung von Genen aus versch. Chromosomen)	kodierte Lösung
Phänotyp (von außen sichtbare Auswirkung eines Genotyps)	dekodierte Lösung

tation, desto weniger zielgerichtet ist die Suche. Einzelne Bestandteile (Teilstrecken) der Rundreise können durch die Mutationen zufällig entstehen oder wieder verloren gehen.

- Genetische Algorithmen suchen durch Kombinieren von guten Rundreisen nach neuen Lösungen, die aus Komponenten der Rundreisen ihrer Eltern bestehen. Treffen dabei jeweils gute Teilstrecken aufeinander, entsteht eine bessere Rundreise. Die oben beschriebene Metapher passt in diesem Fall sehr gut. Es macht Sinn, zu versuchen eine Lösung, also die Rundreise, nicht in der Gesamtheit, sondern auf Basis ihrer Komponenten, zu betrachten und zu optimieren. Gute Teilstrecken bleiben dabei in der Population erhalten und gehen nicht verloren.

Auch in der Biologie konnten anstelle der eher abstrakten, darwinistischen Evolutionstheorie, speziell durch neue Kenntnisse in den Bereichen Populationsgenetik und Molekularbiologie, genauere Modelle zur Beschreibung der evolutionären Weitergabe von Erbmaterial (Genen) gewonnen werden. Genetische Algorithmen nutzen daher oft die dort vorherrschende Terminologie zur Beschreibung ihrer Konzepte. Tabelle 2.2 übersetzt die wichtigsten neuen Begriffe. Ein praktisches Beispiel gibt Abb. 2.14 anhand einer als Integer-Vektor kodierten TSP-Lösung mit fünf Städten. Jedes Gen beschreibt die Stadt für eine gewisse Position in der Rundreise. Der gesamte Genotyp spiegelt die Reihenfolge der Städte einer Rundreise wider.

[10] Es wäre auch möglich, Individuen mit mehreren Chromosomen zu entwickeln. Dies wird jedoch selten praktiziert.

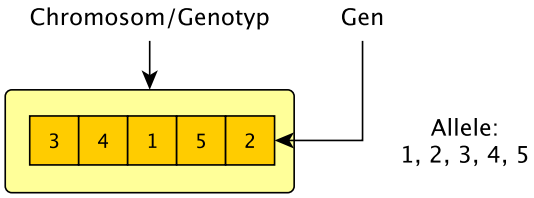

Abbildung 2.14: Die Darstellung einer TSP-Lösung mit fünf Städten als Integer-Vektor.

Eine gute Einführung in die genetischen Algorithmen auf Basis der von John H. Holland entwickelten Konzepte [Hol75] bietet David Goldberg in seinem Werk *„Genetic Algorithms in Search, Optimization and Machine Learning“* [Gol89]. In folgendem Abschnitt wird auf den Ablauf eines einfachen GA in abstrakter Form eingegangen.

Funktionsweise eines GA

Abb. 2.15 beschreibt die wichtigsten Operationen eines Genetischen Algorithmus als Ablaufdiagramm [Scr13]:

- Zunächst muss die Ausgangspopulation mit zufälligen Chromosomen (Lösungen) befüllt und anhand der Fitnessfunktion evaluiert werden. Wie auch schon bei den Evolutionsstrategien ist die Wahl einer passenden Populationsgröße sehr wichtig. Damit die Rekombination gute Lösungen generieren kann, müssen die Komponenten des optimalen Chromosoms in der Population vorhanden sein. Man spricht in diesem Zusammenhang auch von einer ausreichenden *Genetischen Diversität* in der Population. Bestimmte Merkmale (Allele) der optimalen Lösung, die in der Population fehlen, können vom GA nur schwer zurückgewonnen werden.
- Die *Eltern-Selektion* sorgt bei den Genetischen Algorithmen für den nötigen Selektionsdruck. Sie ersetzt die bei den ES übliche Selektion der Überlebenden und bestimmt welche Individuen mit welcher Häufigkeit im Paarungs-Pool vorkommen.
- Jeweils zwei Individuen des Paarungs-Pools erzeugen danach durch das Anwenden des *Rekombinations*-Operators die Nachkommen der Generation. Diese enthalten Allele (Genausprägungen) beider Elternteile. Die Anzahl der erzeugten Nachkommen entspricht der zuvor gewählten Populationsgröße.
- Anhand einer geringen *Mutations*-Wahrscheinlichkeit werden manche Nachkommen zufällig verändert. Allele, die ansonsten in der Population fehlen würden, können so eventuell wieder zurückgewonnen werden.

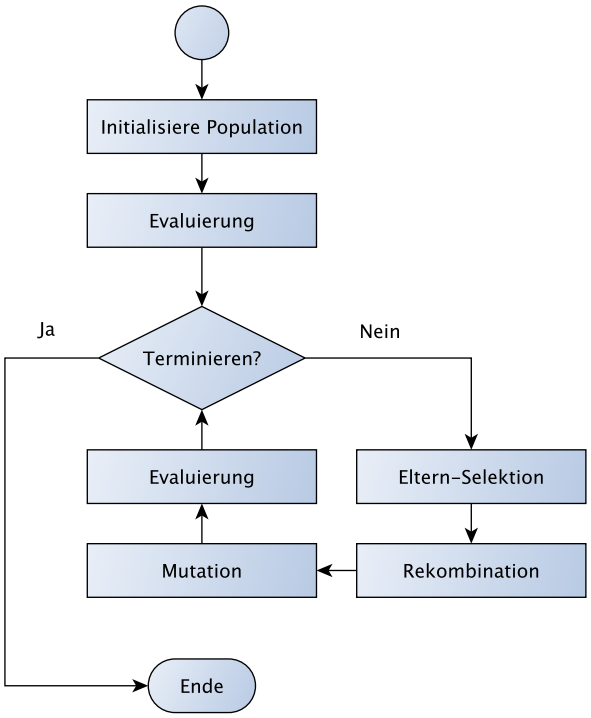

Abbildung 2.15: Der Ablauf eines einfachen Genetischen Algorithmus.

Die Mutation bei GA ist in Bezug auf die Allele also ein explorativer Faktor.

- Die Fitnessfunktion bestimmt anschließend die Qualität der erzeugten Lösungen. Die alte Population wird nun durch die Nachkommen ersetzt. In der Literatur ist dieses Ersetzen als *Generational Replacement*[11] bezeichnet.

Verfrühte Konvergenz

Das größte Problem der Evolutionsstrategien ist eine Stagnation des Algorithmus in einem lokalen Optimum. Die Ursache dieses Problems besteht durch zu kleine Mutationsweiten, also einem zu stark exploitativen Verhalten des Algorithmus. Auf der anderen Seite sind große Mutationsweiten problematisch, da sie sich negativ auf das Konvergenzverhalten einer ES auswirken. Bei Genetischen Algorithmen existiert ein ähnliches Problem, das man als *Verfrühte Konvergenz* bezeichnet, aber durch eine andere Ursache entsteht. Im Gegensatz zur ES stagniert der Algorithmus hier aufgrund von fehlenden

[11] Anstatt eines gänzlichen Ersetzens der Population ist es auch möglich einige besonders gute Eltern zu behalten. Diese Strategie nennt man dann *Elitismus*.

Allelen in der Population, also einer zu geringen *Genetischen Diversität*. Da sich gute Genausprägungen im Laufe der Generationen immer mehr durchsetzen, werden die Lösungen einer Population zunehmend ähnlicher. Die *Mutation* ist dann die einzige Kraft, die zufällig die fehlende Information einbringen kann, welche sich danach in folgenden Generationen eventuell wieder ausbreiten kann. Das zufällige Finden solch fehlender Allele bei der geringen Mutations-Wahrscheinlichkeit eines GA ist sehr schwierig. Eine höhere Mutationsrate würde jedoch dazu führen, dass sich gute Lösungskomponenten durch Rekombination schwerer durchsetzen, da eine Mutation diese guten Lösungen zerstören könnte. Weiters kann ebenso ein zu hoher Selektionsdruck (nur die besten Individuen werden als Eltern selektiert) dazu führen, dass später benötigte Lösungsbestandteile durch die Selektion ausscheiden und aus der Population verschwinden. Ohne genügend Selektionsdruck kann jedoch kein Fortschritt erzielt werden. Aufgrund dieser Problematik entstand die Idee zu einer *selbstadaptiven Selektionsdruck-Steuerung* bei Genetischen Algorithmen, die im nächsten Abschnitt detailliert behandelt wird.

2.3 Selektionsdruck-Steuerung

Bevor detailliert auf die Umsetzung einer selbstadaptiven Selektionsdruck-Steuerung eingegangen werden kann, sollte Klarheit über die Art der Selektion in Genetischen Algorithmen und Evolutionsstrategien bestehen. Die folgenden Abschnitte fassen die wichtigsten Fakten zusammen und erklären die Umsetzung einer selbstadaptiven Selektionsdruck-Steuerung bei Genetischen Algorithmen anhand des Konzeptes der *Offspring Selection*.

2.3.1 Selektion in Genetischen Algorithmen

Die Einstellung des Selektionsdruckes im Kontext der Genetischen Algorithmen basiert hauptsächlich auf dem Selektions-Operator für die Eltern-Selektion und dem vorherrschenden Ersetzungs-Mechanismus [Mic98]. Je stärker die Selektion bei der Auswahl der Eltern ist, desto höher ist der Selektionsdruck. Schreibt der Ersetzungs-Mechanismus anschließend vor, dass einige schlechte Individuen der Nachkommen mit besonders guten Individuen der Eltern ersetzt werden, steigt der Selektionsdruck ebenso. Besonders gute Individuen können dann über viele Generationen in der Population erhalten bleiben und in jeder Generation Nachkommen erzeugen. Diese Form der Ersetzung nennt sich *Elitismus* und kommt üblicherweise schneller zu guten Lösungen. Der Nachteil ist jedoch, dass sich die Population leichter aus sehr ähnlichen Individuen zusammensetzt und somit eine höhere Gefahr der Verfrühten Konvergenz besteht. Hinzu kommt weiters, dass in der grundlegenden Theorie der Genetischen Algorithmen kein geeignetes Modell besteht, das eine kontrollierte Selektionsdruck-Steuerung erlaubt. Aus diesem Grund ist eine Erweiterung des GA-Modells notwendig, die eine sol-

che Steuerung möglich macht. Diese ist inspiriert durch die Steuerung des Selektionsdruckes in Evolutionsstrategien.

2.3.2 Selektion in Evolutionsstrategien

Im Gegensatz zu den Genetischen Algorithmen setzen die Evolutionsstrategien auf eine Selektion der Überlebenden. Der Selektionsdruck ist dadurch bei der $(\mu \overset{+}{,} \lambda)$-ES über eine passende Wahl der Parameter μ und λ konfigurierbar. Betrachtet man den Selektionsdruck als „*Chance eines Individuums, zu überleben*“, wäre eine Formalisierung wie folgt möglich [SHF94]:

$$s = \frac{\mu}{\lambda} \tag{2.21}$$

Bei einer $(10, 100)$-ES errechnet sich dadurch ein Wert von $s = \frac{1}{10}$ oder $0,1$. Nur eines pro zehn erzeugten Individuen schafft es in die nächste Generation. Für eine $(\mu + \lambda)$-ES ist eine solche Berechnung ebenso möglich:

$$s = \frac{\mu}{\mu + \lambda} \tag{2.22}$$

Je kleiner der Wert für s ist, desto geringer ist die Überlebenschance eines Individuums. Niedrige Werte bedeuten demnach einen hohen Selektionsdruck. Liegt der Fokus auf dem tatsächlichen Selektionsdruck beschreibt s diesen nur indirekt proportional. Ein direkter Wert für den Selektionsdruck kann über den Kehrwert von s gewonnen werden:

$$SelPressure = \frac{1}{s} = \frac{\lambda}{\mu} \text{ oder } \frac{\mu + \lambda}{\mu} \tag{2.23}$$

Diese Form der Überlebenden-Selektion und die Möglichkeit der Berechnung des Selektionsdruckes erlauben im Gegensatz zu Genetischen Algorithmen eine kontrollierte Steuerung des Selektionsdruckes. Die *Offspring Selection* bei Genetischen Algorithmen nutzt dieses Prinzip um eine direkte und in weiterer Folge auch selbstadaptive Steuerung des Selektionsdruckes umzusetzen.

2.3.3 Offspring Selection

Um eine Selektion im Stile der Evolutionsstrategien bei Genetischen Algorithmen zu implementieren, ist es notwendig das Modell des typischen Genetischen Algorithmus um eine virtuelle Zwischenpopulation zu erweitern [Aff01]. Diese Zwischenpopulation entspricht der Nachkommenspopulation λ von Evolutionsstrategien und dient als Basis für eine zusätzliche Selektion der Überlebenden.

Abb. 2.16 veranschaulicht die Funktionsweise eines GA mit virtueller Zwischenpopulation. Die über Rekombination und Mutation erzeugten Nachkommen füllen zunächst nur die virtuelle Population auf. Wie auch bei den

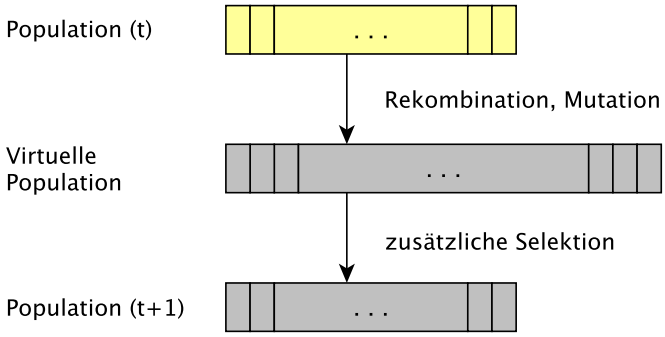

Abbildung 2.16: Integration einer virtuellen Population in den GA.

Evolutionsstrategien erlaubt der Algorithmus danach nur den besten Nachkommen in die nächste Population einzufließen. Der Selektionsdruck dieses zusätzlichen Selektionsschrittes kann nun für eine Population $|POP|$ durch folgende Formel beschrieben werden:

$$s = \frac{|POP|}{|POP_{virtuell}|} \tag{2.24}$$

Wobei $|POP| \geq |POP_{virtuell}|$ gilt. Die Größe der virtuellen Population lässt sich als Vielfaches der echten Populationsgröße darstellen, wodurch sich angelehnt an die Notation eines *Simulated Annealing*-Algorithmus[12] $|POP_{virtuell}| = |POP| \cdot T$ ergibt:

$$s = \frac{|POP|}{|POP| \cdot T} = \frac{1}{T} \qquad (T > 1) \tag{2.25}$$

Ausgehend von diesem erweiterten GA-Modell ist es möglich den Selektionsdruck zielgerichtet zu steuern. Durch eine passende Wahl von T kann die Konvergenz des Algorithmus kontrolliert werden. Biologisch gesehen entspricht diese Selektion einem Sterben von Kindern, bevor diese Nachkommen zeugen können. Die Anzahl der Kinder, die es nicht in die nächste Generation schaffen, entspricht:

$$|POP| \cdot T - |POP| = |POP| \cdot (T - 1) \tag{2.26}$$

Ein Verringern der Kindersterblichkeitsrate, also ein stetiges Senken des Selektionsdruckes während der Optimierung über den Parameter T, ist auch in biologischer Hinsicht sinnvoll, da in besser entwickelten Populationen eine geringere Sterblichkeitsrate der Kinder vorherrscht. Diese Erweiterung des Genetischen Algorithmus arbeitet zwar noch nicht selbstadaptiv, bildet

[12] Durch kontinuierliches Verringern eines Strategieparameters T sucht der Algorithmus zunehmend weniger explorativ und mehr in die Tiefe [KGV83][Čer85].

aber die Grundlage zur Realisierung einer selbstadaptiven Steuerung des Selektionsdruckes.

Selbstadaptive Selektionsdruck-Steuerung

Die zentrale Frage zur Umsetzung einer Selbstadaption ist, wie groß die erzeugte Menge an Nachkommen in einer bestimmten Generation sein soll und welche dieser Individuen in die nächste Population einfließen. Die Größe der virtuellen Population muss bei einer selbstadaptiven Variante variabel sein und je nach Anforderungen an den Algorithmus automatisch angepasst werden. Um eine generische Implementierung zu ermöglichen darf außerdem keine problemspezifische Information für das selbstadaptive Modell herangezogen werden. Es liegt daher nahe, die Strategie-Entscheidungen auf Basis der Fitness der erzeugten Individuen zu treffen. Eine Idee dafür liefert die in Abschnitt 2.2.1 vorgestellte $(1+1)$-Evolutionsstrategie mit ihrer selbstadaptiven Anpassung der Standardabweichung anhand der *1/5 Erfolgsregel*. Die Selbstadaption erfolgt dort über einen Vergleich der erzeugten Lösung mit dem Elter. Die Standardabweichung wird so angepasst, dass in Summe ungefähr 1/5 der erzeugten Lösungen erfolgreich sind, also bessere Ergebnisse liefern als die Eltern-Lösung. Dieses Modell muss zwar für den Einsatz in Genetischen Algorithmen noch adaptiert werden, die Idee der Festlegung eines bestimmten Anteils an erfolgreichen Nachkommen erscheint jedoch durchaus sinnvoll.

Es ergibt sich durch das Anwenden dieser Idee folgender GA-Ablauf [Aff05]: Der erste Selektionsschritt wählt die Eltern für die Rekombination. Entweder zufällig oder in einer anderen Weise, die bei Genetischen Algorithmen üblich ist, z. B. über eine Fitnessproportionale Selektion. Nach der Durchführung von Rekombination und Mutation folgt ein weiterer Selektionsschritt anhand des Erfolges der Reproduktion. Durch einen Vergleich mit den Eltern kann gewährleistet werden, dass die Suche zu jeder Generation mit einer ausreichenden Anzahl an Nachkommen arbeitet, welche die Fitness ihrer Eltern übertreffen. Zu diesem Zweck wird ein neuer Parameter $SuccessRatio \in [0,1]$ definiert, der für jede nächste Generation den Mindestanteil an erfolgreichen Nachkommen in Relation zur Populationsgröße beschreibt. Für $SucessRatio = 0,5$ und $|POP| = 100$ gilt demnach, dass jede Generation mindestens 50 Nachkommen beinhalten muss, die ihre Eltern übertreffen. Die restlichen $|POP| \cdot (1 - SuccessRatio)$ Individuen der Population können sowohl aus erfolgreichen oder nicht erfolgreichen Individuen bestehen. Abb.2.17 präsentiert den Genetischen Algorithmus mit Offspring Selection als Ablaufdiagramm.

Der tatsächliche Selektionsdruck am Ende jeder Generation ist definiert über die Populationsgröße $|POP|$ und der variablen Anzahl an Individuen $|POOL|$, die generiert werden mussten um das Kriterium der $SuccessRatio$ zu erfüllen, aber nicht Teil der neuen Population wurden:

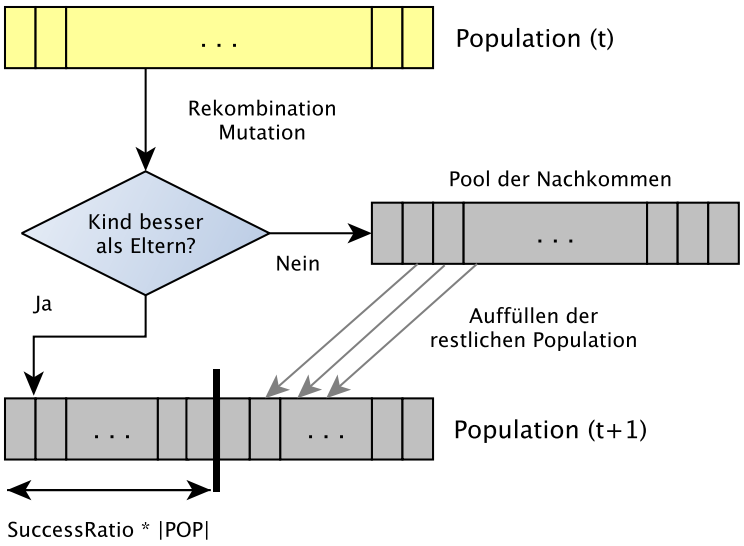

Abbildung 2.17: Ablauf der selbstadaptiven Selektionsdruck-Steuerung über Offspring Selection anhand des Parameters *SuccessRatio*.

$$ActSelPressure = \frac{|POP| + |POOL|}{|POP|} \tag{2.27}$$

Durch den Einsatz der selbstadaptiven Selektionsdruck-Steuerung über den Parameter $SuccessRatio$ und der Berechnung von $ActSelPressure$ in jeder Generation ergibt sich auch die Möglichkeit der Einführung eines maximalen Selektionsdruckes $MaxSelPressure$. Dieser Parameter beschreibt die maximale Anzahl an erzeugten Nachkommen pro Generation und dient als Indikator für das Auftreten einer Verfrühten Konvergenz. Wenn die Nachkommensgrenze ($MaxSelPressure \cdot |POP|$) erreicht ist und noch nicht genügend Reproduktionen ($SuccessRatio \cdot |POP|$) erfolgreich waren, kann der Algorithmus beendet werden. Im Sinne der Genetischen Algorithmen stellt die Offspring Selection nicht unbedingt eine höhere Gefahr auf eine Verfrühte Konvergenz dar. Die Ursache der Verfrühten Konvergenz liegt in fehlender Genetischer Diversität, also sehr ähnlichen Individuen. Die Offspring Selection erlaubt jedoch keine Nachkommen, die ident zu den Eltern sind, da diese keine bessere Fitness aufweisen. Über die Offspring Selection und weiteren darauf aufbauenden GA-Varianten konnten bisher sehr gute Ergebnisse erzielt werden. Vor allem auch das Erkennen von Verfrühter Konvergenz hat sich bei speziellen Algorithmen, die diese Information nutzen und darauf reagieren, als sehr vorteilhaft herausgestellt.

Kapitel 3

OS-ES: Offspring Selection Evolution Strategy

Das Ziel dieser Arbeit ist die Integration einer selbstadaptiven Steuerung des Selektionsdruckes im Stile der Offspring Selection in die Evolutionsstrategien. Dieses Kapitel widmet sich den dazu nötigen Anpassungen am Beispiel der (μ^{+},λ)-ES sowie der CMA-ES.

3.1 Erweiterung der Evolutionsstrategie

3.1.1 Offspring Selection in der (μ,λ)-ES

Im Gegensatz zu den Genetischen Algorithmen arbeiten die Evolutionsstrategien ausschließlich mit einer deterministischen Selektion der Überlebenden am Ende jeder Generation. Dieser Selektionsschritt arbeitet bereits auf einer Menge λ an Nachkommen, die auf Basis ihrer Fitness in die nächste Population kommen oder ausscheiden. Der Selektionsdruck ist dabei über die Populationsgröße μ und die Anzahl der Kinder λ steuerbar.

$$s = \frac{\mu}{\lambda} \tag{3.1}$$

Die selbstadaptive Variante soll nun äquivalent zur Offspring Selection bei Genetischen Algorithmen einen Vergleich mit dem Elter umsetzen. Jene Kinder, die nach dem Anwenden der Mutation eine schlechtere Fitness als das Elter aufweisen, werden in einem Pool von Nachkommen archiviert. Die Größe dieses Pools wächst so lange an, bis das neue Kriterium der *SuccessRatio* erfüllt ist, also bis ein gewisser Anteil der nächsten Population aus erfolgreichen Mutationen besteht. Der aktuelle Selektionsdruck am Ende jeder Generation berechnet sich dann ident zu den Genetischen Algorithmen:

$$ActSelPressure = \frac{|POP| + |POOL|}{|POP|} \tag{3.2}$$

Ein Unterschied zu den Genetischen Algorithmen besteht in der Tatsache, dass diese neue Selektion das alte Selektionsverfahren gänzlich ersetzt und nicht zusätzlich zur Eltern-Selektion wirkt. Zu Anfang wird der Algorithmus relativ einfach erfolgreiche Mutationen durchführen können, im besten Fall ist sogar jedes Kind erfolgreicher als die Eltern, was einem Wert von $ActSelPressure = 1$ entspricht. Die normale (μ, λ)-ES mit deterministischer Selektion würde durch den fixen Überschuss an Nachkommen von Beginn an einen höheren Selektionsdruck aufweisen. In späteren Generationen kommt jedoch die Selbstadaption der Offspring Selection zu tragen und eine variable Anzahl an Nachkommen wird generiert. Nun ist die restliche Population mit Individuen aus diesen Nachkommen zu befüllen. Folgende Punkte sind dabei zu beachten:

- Bei der Offspring Selection dient die $SuccessRatio$ als Kriterium für die Selektion. Dabei ist es egal welche Individuen der Population dieses Kriterium erfüllen, solange die Mutation ein Kind mit besserer Fitness erzeugt.
- Erfolgreiche Kinder, die aus weniger guten Lösungen entstehen, können eine schlechtere Fitness aufweisen, als nicht erfolgreiche Kinder, die aus bereits sehr guten Lösungen erzeugt wurden.
- Auf die gesamte Population der Nachkommen betrachtet ist ein erfolgreiches Kind demnach nicht zwingend auch eine überdurchschnittlich gute Lösung.

Im Falle von niedrigeren Werten für $SuccessRatio$ und einem höheren Überschuss an Nachkommen kann es daher Sinn machen, den Rest der Population nicht zufällig aus dem Nachkommens-Pool zu befüllen, sondern an dieser Stelle eine bei den ES übliche deterministische Selektion durchzuführen. Bei einem Wert von $SuccessRatio = 0.5$, $\mu = 40$ und $|POOL| = 100$ in einer bestimmten Generation würde eine deterministische Selektion dafür sorgen, dass die restlichen 20 Individuen der nächsten Generation nicht zufällig, sondern aus den besten Individuen im Nachkommens-Pool selektiert werden. Dieses Verfahren entspricht eher dem Verhalten einer normalen Evolutionsstrategie, jedoch wird durch die Offspring Selection zusätzlich noch das Kriterium des Mindestanteils an erfolgreichen Kindern in der Population erfüllt. Abb 3.1 beschreibt diese Variante der Offspring Selection.

Wie auch bei den Genetischen Algorithmen kann der berechnete Selektionsdruck $ActSelPressure$ dazu eingesetzt werden eine Stagnation des Algorithmus zu erkennen und über die Definition eines Limits $MaxSelPressure$ als Abbruchkriterium genutzt werden. In den weiteren Ausführungen dieser Arbeit ist diese neue Variante der Evolutionsstrategie als OS-ES (Offspring Selection Evolution Strategy) bezeichnet. Die obigen Ausführungen beziehen sich auf die Adaption der (μ, λ)-ES, bei der eine neue Generation nur aus den erzeugten Nachkommen besteht. In gleicher Weise kann jedoch ebenso die $(\mu + \lambda)$-ES angepasst werden.

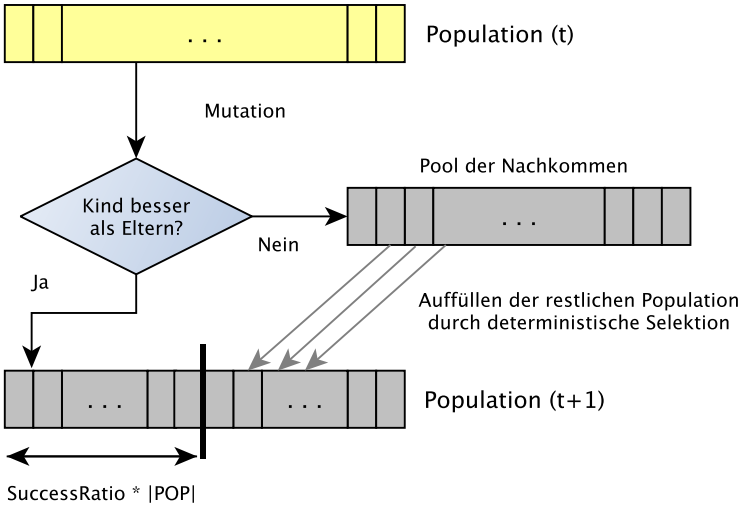

Abbildung 3.1: Integration der Offspring Selection in die ES.

Erweiterung der (μ + λ)-ES

Im Gegensatz zur Komma-Variante der Evolutionsstrategien bindet die $(\mu+\lambda)$-ES auch die gesamte Population der Eltern in die Selektion ein. Ersetzt man diese Selektion durch eine Offspring Selection bieten sich zwei Möglichkeiten zur Einbindung der Eltern:

1. Die deterministische Selektion in der Offspring Selection befüllt die nächste Population mit den besten $\mu \cdot (1 - SuccessRatio)$ Individuen der nicht erfolgreichen Nachkommen. Hier könnte zusätzlich die Eltern-Population miteinbezogen werden. In dieser Variante wären $\mu \cdot SuccessRatio$ Individuen der nächsten Generation immer durch erfolgreiche Kinder bestimmt. Nur die restlichen Plätze können eventuell von guten Eltern eingenommen werden. In Abb. 3.2 ist diese Variante grafisch dargestellt.
2. Die Offspring Selection erzeugt eine (Zwischen-)Population bestehend aus μ Nachkommen. Anschließend kann die neue Population durch eine zusätzliche deterministische Selektion aus diesen Nachkommen und den Eltern bestimmt werden. In dieser Variante ist es auch möglich, dass in der neuen Population die erfolgreichen Nachkommen durch Individuen der Eltern ersetzt werden. Dies wäre der Fall, wenn alle erfolgreichen Kinder von besonders schlechten Eltern abstammen, und dadurch eine geringere Fitness als sämtliche anderen Eltern aufweisen. Im Gegensatz zur Einbindung der Eltern bereits innerhalb der Offspring Selection können die Eltern hier auch bei hohen Wer-

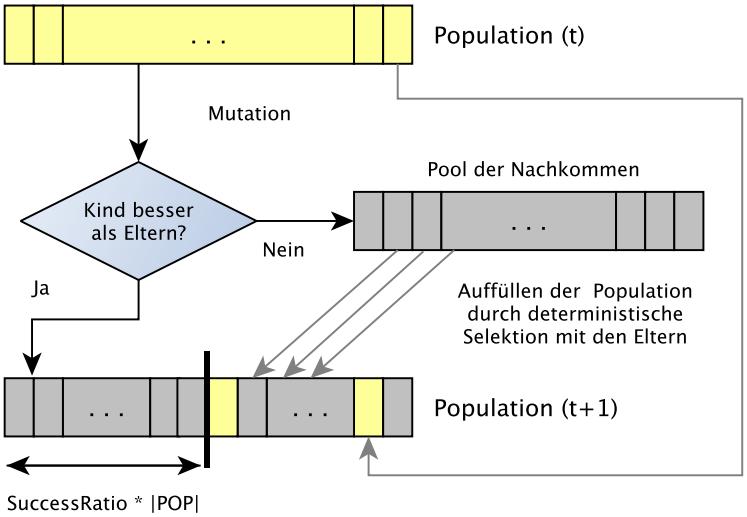

Abbildung 3.2: Ablauf der ES mit Einbezug der Eltern innerhalb der OS.

ten von *SuccessRatio* besser miteinbezogen werden. Bei beispielsweise $SuccessRatio = 0.85$ und $\mu = 40$ wären in der ersten Variante nur noch zehn Plätze der gesamten Population für die Eltern frei.

Typischerweise ist es bei einer $(\mu + \lambda)$-ES möglich, dass die gesamte neue Population nur aus den Eltern besteht. Eine zusätzliche Selektion im Anschluss an die Offspring Selection entspricht eher dieser Metapher, weshalb die hier vorgestellte OS-ES diese Variante umsetzt. Im praktischen Einsatz werden erfolgreiche Kinder der zunehmend besseren Lösungen einer Population in der Regel auch gesamt betrachtet eine gute Fitness aufweisen, und sich somit auch gegen die Eltern durchsetzen. In Abb. 3.3 ist der Ablauf einer solchen Evolutionsstrategie grafisch dargestellt.

OS-ES und Rekombination

Eine selbstadaptive Steuerung des Selektionsdruckes über Offspring Selection ist ebenso in ES-Varianten möglich, die zusätzlich zur Mutation auch Rekombination nützen. Der einzige Unterschied zu den zuvor ausgeführten Abläufen liegt darin, dass sämtliche Nachkommen durch Rekombination und einer anschließenden Mutation erzeugt werden. Abb. 3.4 zeigt diesen leicht geänderten Ablauf. Auf diese Weise kann die Offspring Selection in sämtliche Varianten einer $(\mu/\rho\overset{+}{,}\lambda)$-ES integriert werden. Da bei der OS-Evolutionsstrategie anstelle der Anzahl der Nachkommen λ lediglich die *SuccessRatio* definiert werden kann, ist die obige standardisierte Notation nicht mehr passend. Damit auch für die OS-ES eine solche kurze Schreibwei-

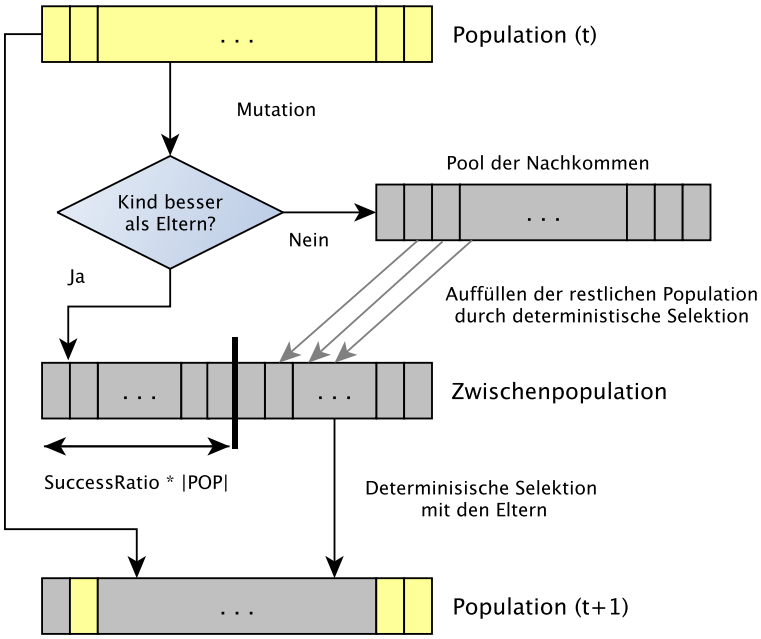

Abbildung 3.3: Ablauf der ES mit Einbezug der Eltern nach der OS.

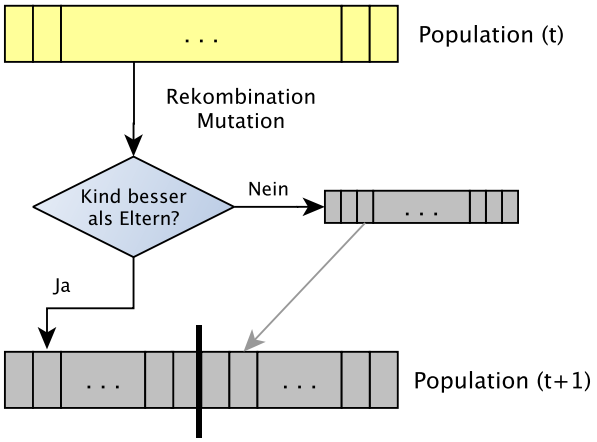

Abbildung 3.4: Ablauf der ES bei Offspring Selection mit Rekombination.

se der Parameter möglich ist, wird in dieser Arbeit eine Notation der Form $(\mu/\rho^{+}, SuccessRatio)$ verwendet. Eine $(10/2, 0.7)$-OS-ES beschreibt nachfolgend z. B. eine Offspring Selection Evolution Strategy ohne Einbezug der Eltern, die Rekombination einsetzt und mit den Parametern $\mu = 10$, $\rho = 2$

und $SuccessRatio = 0.7$ arbeitet. Auf Basis dieser neuen Modelle kann nun der in Abschnitt 2.2.4 vorgestellte ES-Algorithmus erweitert werden. Alg. 3.1 beschreibt diesen neuen Ablauf in Pseudocode.

Algorithmus 3.1: Ablauf der $(\mu/\rho^{+}, SuccessRatio)$-OS-ES

```
Offspring Selection Evolution Strategy
  choose parameters: μ, ρ, σ_i, τ_1, τ_2, ...
  choose os parameters: SuccessRatio, MaxSelPressure
  t ← 0
  initialize : P(0) ← {s_1, ..., s_μ}          ▷ s_i ← (x_1, ..., x_n, σ_1, ..., σ_n)
  evaluate P(0) : {Φ(s_1), ..., Φ(s_μ)}
  while terminate(...) ≠ true do
    CurrSuccRatio ← 0
    POOL ← { }
    P' ← { }
    while CurrSuccRatio < SuccessRatio          ▷ success ratio reached?
        or (|P'| + |POOL|) < |P| do             ▷ enough children generated?
      recombine : x' ← rec(P(t), ρ)             ▷ generate one child
      mutate : x' ← mut(x', τ_1, τ_2)           ▷ mutate child
      evaluate : Φ(x')                          ▷ evaluate child
      if childIsBetter(...) then
        P' ← P' ∪ {x'}                          ▷ add child to next population
        CurrSuccRatio = |P'| / |P|              ▷ update success ratio
      else
        POOL ← POOL ∪ {x'}                      ▷ add child to pool
      end if
    end while
    fill population : fill(P', POOL)
    select : P(t+1) ← sel(P', P(t))             ▷ plus selection?
    t ← t + 1
  end while
end
```

3.1.2 Offspring Selection in der CMA-ES

Neben der $(\mu/\rho^{+}, \lambda)$-Evolutionsstrategie soll diese Arbeit auch für die CMA-Evolutionsstrategie eine mögliche selbstadaptive Selektionsdruck-Steuerung untersuchen. Der größte Unterschied bei der Integration von Offspring Selection für die CMA-Variante liegt in der Tatsache, dass bei der CMA-ES sämtliche Kinder anhand des Mittelwertes der letzten Generation und der adaptierten Kovarianzmatrix erzeugt werden. Um zu entscheiden, ob die Kinder erfolgreich sind, müssen diese demnach alle mit dem gleichen Elter, also dem Mittelwert der vorhergehenden Generation, verglichen werden.

Anhand des *SuccessRatio*-Parameters wird wiederum entschieden, welcher Anteil der so erzeugten Population aus erfolgreichen Kindern bestehen soll. Es ergibt sich folgender Ablauf:

1. Die *Mutation* erzeugt normalverteilt um den Mittelwert von μ Eltern eine variable Menge an Nachkommen, bis dass eine neue Population der Größe μ generiert wurde, die das Kriterium der *SuccessRatio* erfüllt. Die Qualität der erzeugten Kinder ist dabei mit der Qualität des Mittelwertes m der Population zu vergleichen. Jene Nachkommen, die eine bessere Qualität als der aktuelle Mittelwert aufweisen, gelten als erfolgreich und werden ein Teil der neuen Population. Nicht erfolgreiche Nachkommen sind im Nachkommens-Pool aufgehoben.
2. Durch die *Offspring Selection* ist nun die gesamte neue Population definiert. Dabei wird die Population in jeder Generation mit Individuen aus dem Pool der nicht erfolgreichen Nachkommen befüllt, nachdem die erforderliche Menge an erfolgreichen Nachkommen erzeugt wurde. Ein zusätzlicher Selektionsdruck kann durch die angewandte Strategie zur Wahl dieser $\mu \cdot (1 - SuccessRatio)$ nicht erfolgreichen Nachkommen ausgeübt werden. Wie auch im Konzept der OS-ES kommt in der hier vorgestellten Erweiterung der CMA-ES die bei den Evolutionsstrategien übliche deterministische Selektion zum Einsatz.
3. Die Rekombination errechnet den neuen Mittelwert m auf Basis der Population, die durch Offspring Selection erzeugt wurde. Es kann wiederum ein beliebiger Rekombinations-Operator der CMA-ES, z. B. eine gewichtete Rekombination, eingesetzt werden.
4. Die Kovarianzmatrix wird anschließend anhand der neuen Population adaptiert. Die Implementierung der Adaption der Kovarianzmatrix erfolgt wie bisher und erfordert durch den Einsatz von Offspring Selection keine zusätzlichen Anpassungen.
5. Anhand der veränderten Kovarianzmatrix und des neuen Mittelwertes erzeugt der Algorithmus die nächste Generation über Mutation.

Abb. 3.5 zeigt das Zusammenspiel von Rekombination, Mutation und Offspring Selection in der CMA-ES. Die notwendige Adaption der Kovarianzmatrix sowie die Anpassung der Strategieparameter sind in diesem Ablauf nicht enthalten, da die Integration von Offspring Selection keine Auswirkung auf diese Methoden hat.

Diese neue Variante der CMA-ES mit selbstadaptiver Selektionsdruck-Steuerung durch den Einsatz von Offspring Selection ist im Folgenden auch als OS-CMA-ES bezeichnet. Die Definition der Parameter dieses Algorithmus erfolgt ähnlich zur CMA-ES in der Form $(\mu/\rho, SuccessRatio)$. Da die Rekombination typischerweise die gesamte Population μ umfasst gilt $\mu = \rho$, wodurch sich die Notation auf $(\mu, SuccessRatio)$ beschränkt. Im Nachfolgenden bezeichnet z. B. eine $(20, 0.2)$-OS-CMA-ES eine CMA-ES mit Offspring Selection und den Parametern $\mu = 20$ sowie $SuccessRatio = 0.2$.

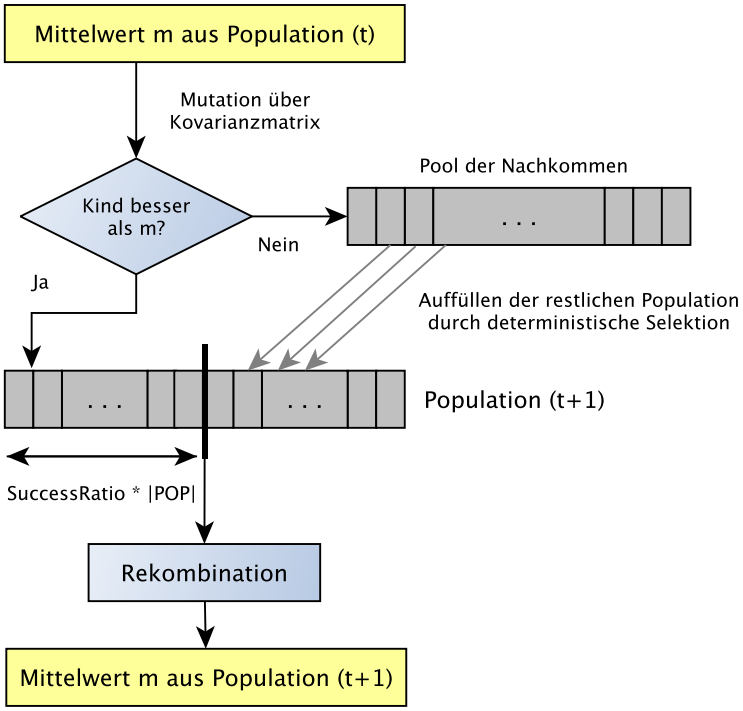

Abbildung 3.5: Ablauf der CMA-ES mit Offspring Selection.

Auch in dieser Variante ist es möglich über den aktuellen Selektionsdruck in einer bestimmten Generation eine Stagnation des Algorithmus zu erkennen. Durch die Definition eines zusätzlichen Parameters $MaxSelPressure$ kann diese Information dazu genutzt werden, ein Limit des Selektionsdruckes als zusätzliches Abbruchkriterium des Algorithmus einzusetzen. Die Erweiterung einer Implementierung der CMA-ES um das Konzept der Offspring Selection ist in Alg. 3.2 dargestellt.

3.2 Umsetzung der Algorithmen

In den folgenden Abschnitten ist die Implementierung der beiden neuen Varianten an Evolutionsstrategien näher ausgeführt. Diese dient anschließend als Basis für die durchgeführten Tests und die Analyse der Ergebnisse. Sowohl die Umsetzung der Algorithmen als auch die Tests und deren Auswertung wurden unter Einsatz des Frameworks *HeuristicLab* durchgeführt.

Algorithmus 3.2: Struktur der OS-CMA-Evolutionsstrategie

OFFSPRING SELECTION CMA EVOLUTION STRATEGY
choose parameters: $\mu, \sigma, w_i, \ldots$
choose os parameters: $SuccessRatio, MaxSelPressure$
$t \leftarrow 0$
$C \leftarrow I$
$m \leftarrow randomPoint(\ldots)$ ▷ initialize mean value
repeat
$t \leftarrow t + 1$
$CurrSuccessRatio \leftarrow 0$
$POOL \leftarrow \{\}$
$P' \leftarrow \{\}$
$evaluate\ mean\ value : \Phi(m)$ ▷ for child comparison
while $CurrSuccRatio < SuccessRatio$ ▷ success ratio reached?
or $(|P'| + |POOL|) < \mu$ **do** ▷ enough children generated?
$mutate : x' \leftarrow m + \sigma\ N(0, C)$ ▷ generate one child
$evaluate : \Phi(x')$ ▷ evaluate child
if $childIsBetter(\ldots)$ **then**
$P' \leftarrow P' \cup \{x'\}$ ▷ add child to population
$CurrSuccRatio = \frac{|P'|}{\mu}$ ▷ update success ratio
else
$POOL \leftarrow POOL \cup \{x'\}$ ▷ add child to pool
end if
end while
$fill\ population : fill(P', POOL)$
$recombine : m = \sum_{i=1}^{\mu} w_i\ x_{i:\lambda}$ ▷ move mean value
$C = \sum_{i=1}^{\mu} w_i (x_i - m)\ (x_i - m)^T$ ▷ adapt covariance matrix
$\sigma \leftarrow \sigma\ exp(\ldots)$ ▷ update step width σ
until $terminate(\ldots) \neq true$
end

3.2.1 HeuristicLab als Entwicklungsumgebung

HeuristicLab ist ein Framework für heuristische und evolutionäre Algorithmen, das 2002 von Mitgliedern des *Heuristic and Evolutionary Algorithms Laboratory* (*HEAL*) entwickelt wurde und laufend weiterentwickelt wird. Die Entstehung von HeuristicLab ist motiviert durch spezielle Anforderungen, die von der großen Anzahl an existierenden Implementierungen und Frameworks für Optimierungs-Algorithmen nicht hinreichend erfüllt werden konnten. Diese Entwicklungen lassen sich grob in zwei Arten gliedern [Aff05]:

- Es existieren einerseits viele Implementierungen von Forschern oder Studenten, die nur die Umsetzung eines bestimmten Algorithmus zur Lösung einer bestimmten Problemstellung behandeln.

- Andererseits finden sich einige Software-Pakete, meistens Bibliotheken, die von Forschungsgruppen entwickelt werden und den Programmierer bei der Implementierung einer bestimmten Algorithmus-Variante für die Lösung beliebiger Probleme unterstützen.

Die erste Form ist im Bezug auf neue Problemstellungen sehr unflexibel, da der Algorithmus nur für ein bestimmtes Problem implementiert wurde. Doch selbst bei den fortschrittlicheren Bibliotheken, die bei der Lösung verschiedener Probleme unterstützen, existiert eine Einschränkung anhand einer bestimmten Art von Algorithmen. HeuristicLab bietet hingegen ein Framework, das die Implementierung und Anwendung unterschiedlichster Algorithmen und Problemstellungen erlaubt. Es ist möglich neue Algorithmen zu entwickeln und diese auf sämtlichen bisher integrierten Problemstellungen zu testen. Genauso kann ein neues Problem mit allen implementierten Algorithmen bearbeitet werden, um danach die Performanz der Algorithmen zu vergleichen. Einen weiteren Grund für die Entwicklung von HeuristicLab als neues Framework lieferten die Programmiersprachen, die andere Bibliotheken einsetzen. Diese sind entweder für den Programmierer zu wenig komfortabel (z. B. C, C++) oder eignen sich aufgrund der Ausführungsgeschwindigkeit nicht gut zur Lösung von Optimierungsproblemen (z. B. Java). Für die Entwicklung von HeuristicLab fiel die Entscheidung auf *C#* als Programmiersprache, da diese als moderne Hochsprache einen sehr guten Programmierkomfort bietet und durch die ausgeklügelte .NET-Runtime[1] auch eine gute Ausführungs-Geschwindigkeit aufweist. Das HeuristicLab Framework ist *Open-Source* und steht kostenlos im Internet zur Verfügung[2]. Es umfasst mittlerweile eine große Menge an Features, unter anderem:

- Ein grafisches User-Interface
- Implementierungen zahlreicher Evolutionärer Algorithmen
- Viele bekannte Benchmark-Probleme
- Tools zur Daten-Analyse und Visualisierung
- Simulations-basierte Optimierung
- u. v. m.

Das Ziel einer losen Kopplung der Algorithmen und Problemstellungen ist in HeuristicLab durch eine stark Plugin-basierte Architektur der Software realisiert. Sämtliche Algorithmen und Problemstellungen sind als Plugins umgesetzt und werden bei jedem Programmstart vom Framework geladen. Um jede der vorhandenen Problemstellungen bearbeiten zu können, müssen die Algorithmen in einer Weise implementiert werden, die nur den Ablauf des

[1] Es kommen spezielle Methoden zum Einsatz, welche die Geschwindigkeit im Vergleich zu einfachen Interpretern enorm verbessern, z. B. JIT-Kompilierung, Pre-jitting, usw.. Mehr über C# und die .NET-Plattform findet sich in [Tro12].

[2] http://dev.heuristiclab.com/

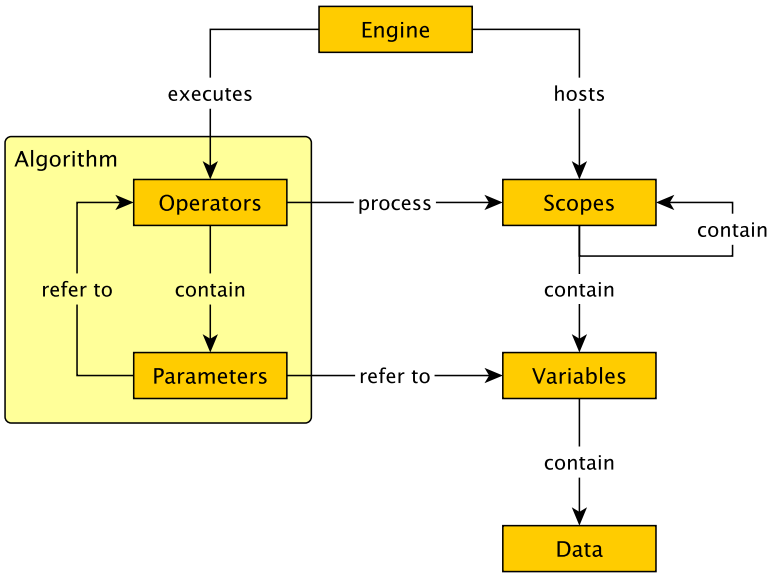

Abbildung 3.6: Klassen-Diagramm zu Algorithmen in HeuristicLab.

Algorithmus festlegt. Sämtliche Operationen, die Abhängigkeiten zu einem bestimmten Problem haben, dürfen nur in abstrakter Form ein Bestandteil dieses Ablaufs sein. Durch die Auswahl eines Problems kommen dann an dieser Stelle die konkreten Operationen zum Einsatz.

Algorithmen in HeuristicLab

Die für die Implementierung von Algorithmen notwendigen Basis-Klassen und deren Zusammenspiel sind in Abb. 3.6 als Klassendiagramm dargestellt. Die HeuristicLab-Engine verwaltet während der Ausführung eines Algorithmus die wichtigsten Variablen in speziellen *Scopes*. Die Scopes können auch weitere Sub-Scopes aufnehmen, welche wiederum eine Menge an Variablen umfassen. Beispielsweise ließe sich auf diese Weise eine Struktur definieren, die auf oberster Ebene (im Haupt-Scope, Main-Scope) globale Variablen wie die Anzahl der Generationen, die bisher beste gefundene Lösung, sowie die Werte für Strategieparameter enthält. Eine Population von Lösungen kann dann in Form von Sub-Scopes für jedes Individuum hinzugefügt werden. Jeder dieser Sub-Scopes ist dann ident aufgebaut und verwaltet z. B. eine konkrete Lösung und deren Qualität. Es ergibt sich durch die Verschachtelung von Scopes auf diese Weise eine Baumstruktur. Ein solcher *Scope-Tree* ist in Abb. 3.7 dargestellt.

Die Algorithmen selbst besitzen Parameter und bestehen aus einer Kette von Operatoren. Die Parameterwerte müssen vor dem Ausführen des Algo-

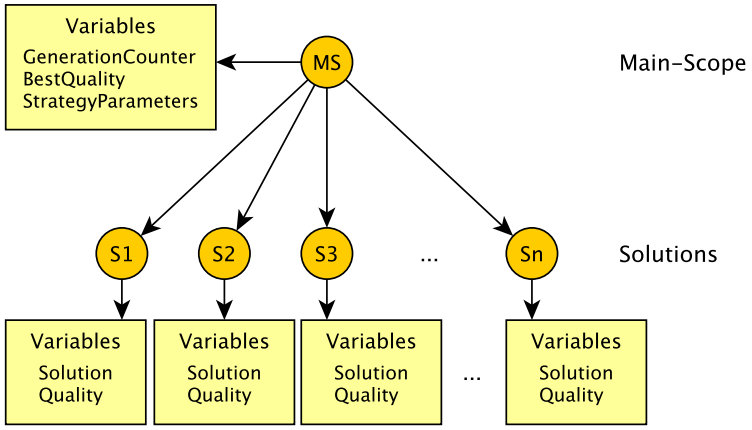

Abbildung 3.7: Beispielhafter Scope-Tree zur Verwaltung der Variablen.

rithmus definiert werden, es ist aber auch möglich Standardwerte festzulegen. Die Parameter verweisen dabei nur auf Variablen im Scope-Tree, die den tatsächlichen Wert beinhalten. Der Ablauf des Algorithmus ist dann definiert durch eine festgelegte Abfolge der Operatoren, die sequentiell von der Engine abgearbeitet werden. Für einen Evolutionären Algorithmus umfasst diese Kette dann z. B. Operatoren für Selektion, Mutation, Rekombination, Fitness-Evaluierung, u. v. m. Ebenso sind Aufgaben wie das Erhöhen des Generationszählers, das Prüfen des Abbruchkriteriums oder das Sammeln von Daten zur späteren Analyse in Form von Operatoren realisiert und in den Algorithmus integriert. Sämtliche Operatoren verarbeiten bestimmte Scopes und können Variablen in diesem Scope oder den Sub-Scopes anlegen oder verändern, neue Sub-Scopes definieren oder auch ganze Sub-Scopes löschen. Der Evaluierungs-Operator startet beispielsweise die problemspezifische Fitness-Evaluierung und fügt die so ermittelte Qualität als neue Variable zum Scope der Lösung hinzu. Ein Selektions-Operator könnte die aktuelle Population in zwei Sub-Scopes aufteilen, welche einerseits die selektierten und andererseits die nicht gewählten Individuen beschreiben. Ein Beispiel eines solchen Selektions-Operators ist in Abb. 3.8 dargestellt.

Die Operatoren sind dabei in einer Weise implementiert, die nur die notwendigsten Informationen über den Aufbau der Scopes voraussetzt. Die Selektion nach obigem Beispiel erwartet etwa, dass unterhalb des aktuellen Scopes die Lösungen als Sub-Scopes vorhanden sind, die jeweils eine Variable enthalten, welche deren Qualität beschreibt. Durch diese Implementierung ist das aktuell gewählte Optimierungs-Problem unwichtig für die Ausführung des Operators. Für Problem-abhängige Operatoren, wie beispielsweise die Lösungs-Evaluierung, enthält der Ablauf nur Platzhalter, die nach der Auswahl eines Problems durch einen bestimmten Operator ersetzt werden.

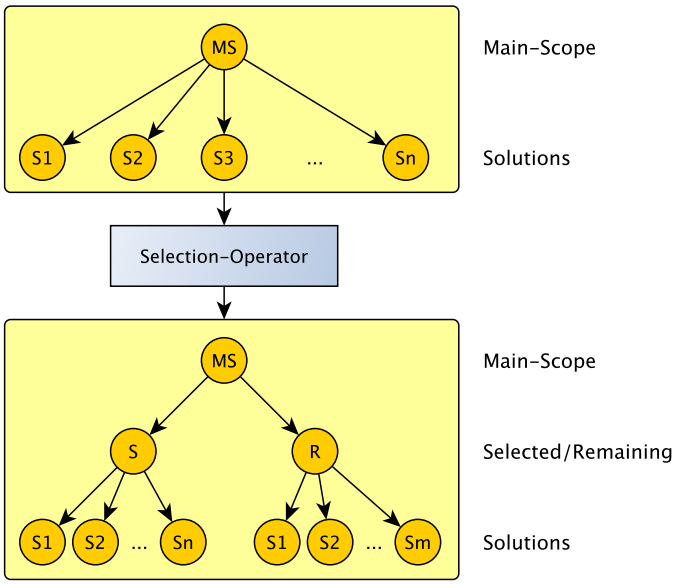

Abbildung 3.8: Beispiel der Veränderung des Scope-Trees durch einen Selektions-Operator, der die Individuen in zwei neue Sub-Scopes aufteilt.

Durch diesen Aufbau ist die Trennung von Problem und Algorithmus klar definiert und eine beliebige Kombination von Problem und Algorithmus ist möglich. Die Definition der Algorithmen in Form von Operator-Ketten hat jedoch noch weitere Vorteile:

- Es existieren verschiedene Implementierungen der Engine zur Ausführung der Operatoren. Neben der *Sequential Engine* ist es auch möglich die *Parallel Engine* zu wählen, welche die Abarbeitung einzelner Sub-Scopes (beispielsweise bei der Evaluierung der Lösungen) automatisch parallel ausführt, wodurch die Geschwindigkeit auf Rechnern mit mehreren Kernen verbessert werden kann. Der Einsatz der *Debug Engine* erlaubt hingegen zu jedem Zeitpunkt der Ausführung die Betrachtung des aktuellen Scope-Trees und die Werte aller Variablen.
- Weiters kann die Ausführung des Algorithmus jederzeit pausiert werden. Die Engine hält in diesem Fall vor dem nächsten Operator an. Dadurch ist es möglich den aktuellen Fortschritt zu persistieren und zu einem späteren Zeitpunkt wieder mit der Ausführung fortzufahren.

Die Software bietet somit ein sehr umfangreiches und flexibles Framework zur Lösung von Optimierungsproblemen. Eine detailliertere Betrachtung der Architektur der Software führt an dieser Stelle jedoch zu weit. Die nächsten Abschnitte widmen sich der Umsetzung der neuen Evolutionsstrategien mit Offspring Selection als HeuristicLab-Plugins.

3.2.2 Implementierung der OS-ES

In der aktuellen Version 3.3.9 von HeuristicLab[3] ist bereits jeweils eine Implementierung der $(\mu/\rho^{+},\lambda)$-ES und der CMA-ES in Form von HeuristicLab-Plugins vorhanden. Diese bereits lauffähigen Plugins bilden die Basis für eine Erweiterung der Algorithmen um Offspring Selection:

- Das Projekt *HeuristicLab.Algorithms.EvolutionStrategy-3.3* der .NET Solution von HeuristicLab beinhaltet das Plugin der $(\mu/\rho^{+},\lambda)$-ES. Dieses Plugin liefert die Grundlage für das Projekt des neuen Algorithmus.
- *HeuristicLab.Algorithms.OffspringSelectionEvolutionStrategy-3.3* stellt eine exakte Kopie des bestehenden ES-Projektes dar. Die Namen für die Algorithmen-Klassen, die Namespaces und das Plugin sind jedoch entsprechend angepasst. Die Implementierung des Algorithmus definiert bis auf einige Änderungen den selben Ablauf und besteht aus den folgenden zwei Klassen:
- Die Klasse *OffspringSelectionEvolutionStrategy* definiert den Algorithmus in grober Form. Hier sind die Parameter des Algorithmus definiert und die wichtigsten Operatoren für die Ausführung durch die Engine angelegt. Zusätzlich kümmert sich diese Klasse auch um das Anlegen verschiedener Analyzer, die Daten für eine spätere Auswertung und Evaluierung der Ergebnisse sammeln. Weiters kann hier auf Events wie z. B. das Laden einer Problemstellung reagiert werden, um Platzhalter-Operatoren automatisch durch konkrete Implementierungen zu ersetzen.
- Die Klasse *OffspringSelectionEvolutionStrategyMainLoop* ist ein spezieller Operator, der in den Ablauf des Algorithmus integriert ist und die eigentlich Logik der Evolutionsstrategie umsetzt. Die Hauptschleife mit all ihren Operationen ist hier in Form einer Verkettung von Operatoren definiert. Dazu zählen Mutation, Rekombination, Selektion, das Prüfen der Abbruchbedingungen und auch die Fitness-Evaluierung. Zusätzlich sind verschiedene Ausführungspfade für Varianten mit Rekombination oder Einbindung der Eltern möglich.

Im Folgenden sind die notwendigen Erweiterungen an diesen Klassen für die Einführung der Offspring Selection beschrieben.

Erweiterung der Algorithmus-Klasse

Der Algorithmus benötigt für die Umsetzung der Offspring Selection einige zusätzliche Parameter:

- Die *SuccessRatio* ersetzt den Parameter *Children* (λ, Anzahl der Nachkommen) und wird zur Konfiguration der selbstadaptiven Variante eingesetzt.

[3] Stable Release vom 11. Oktober 2013

- Der Parameter *MaximumSelectionPressure* legt eine Grenze für den Selektionsdruck fest und dient als zusätzlich Abbruchkriterium.
- Der Parameter *SelectedParents* beschreibt die Menge der Nachkommen, die pro Iteration der Offspring Selection erzeugt werden. Dieser Parameter wird benötigt, da die Implementierung der Offspring Selection in HeuristicLab nicht jeden Nachkommen einzeln erzeugt, um diesen danach zur neuen Population oder zum Pool der Nachkommen hinzuzufügen. Stattdessen wird jeweils eine Menge an Kindern generiert, mit den Eltern verglichen und durch Offspring Selection zur Population hinzugefügt oder im Pool archiviert. Sollten danach noch zusätzliche Kinder benötigt werden, generiert der Algorithmus wiederum diese fixe Menge an Nachkommen, solange bis das Ziel der *SuccessRatio* erreicht ist.
- Weiters ist neben dem Parameter *MaximumSelectionPressure* auch ein Terminieren nach einer bestimmten Anzahl an evaluierten Lösungen interessant, weshalb auch ein zusätzlicher Parameter *MaximumEvaluatedSolutions* eingeführt wurde, der ebenso als Abbruchkriterium für den Algorithmus eingesetzt wird.

Folgender Code-Abschnitt beschreibt das Hinzufügen der *SuccessRatio* als neuen Parameter:

```
1 Parameters.Add(new ValueLookupParameter<DoubleValue>("SuccessRatio",
2    "The ratio of successful to total children that should be achieved.",
3    new DoubleValue(1)));
```

Für jeden Parameter ist dabei der Name, der Datentyp, eine Beschreibung, sowie ein optionaler Standardwert zu definieren. Das Anlegen der weiteren Parameter folgt dem selben Prinzip. Auch die Hauptschleife des Algorithmus benötigt einen Zugriff auf diese Parameter, weshalb die selben Parameter in gleicher Weise für den Operator der Hauptschleife definiert sind. Der Algorithmus übermittelt anschließend beim Anlegen des Operators seine Parameter ebenso der Hauptschleife.

Da in der neuen Algorithmus-Variante die Anzahl der Nachkommen (Parameter *Children*) nicht mehr definiert werden muss, sind in der Algorithmus-Klasse auch sämtliche Prüfungen des Parameters auf gültige Werte zu entfernen, z. B. ist es nicht mehr nötig bei einer Änderung der Populationsgröße sicherzustellen, dass *Children* in der Komma-Variante mindestens so groß wie die Population sein muss.

Die letzte Änderung in der Algorithmus-Klasse betrifft die Konfiguration der Analyzer für spätere Auswertungen. Zusätzlich zum *BestAverageWorstQualityAnalyzer*, der die Qualität der Lösungen in jeder Generation verarbeitet, ist für die Offspring Selection noch der Verlauf des tatsächlichen Selektionsdruckes der Generationen interessant. Für diese Aufgabe ist ein neuer Analyzer „*selectionPressureAnalyzer*“ als zusätzliche Eigenschaft der Klasse definiert:

```
1 [Storable]
2 private ValueAnalyzer selectionPressureAnalyzer;
```

Zur Initialisierung der Analyzer ist in der ursprünglichen Klasse bereits eine Methode *ParameterizeAnalyzers* vorgesehen, die um folgende Zeilen erweitert wird:

```
1 selectionPressureAnalyzer.Name = "SelectionPressure Analyzer";
2 selectionPressureAnalyzer.ResultsParameter.ActualName = "Results";
3 selectionPressureAnalyzer.ValueParameter.ActualName =
4     "SelectionPressure";
5 selectionPressureAnalyzer.ValueParameter.Depth = 0;
6 selectionPressureAnalyzer.ValuesParameter.ActualName =
7     "Selection Pressure History";
```

Der „SelectionPressure Analyzer“ verarbeitet nun die Daten der Variable „SelectionPressure“ und sammelt diese unter dem Namen „Selection Pressure History“.

Erweiterung des Hauptschleifen-Operators

Wie zuvor erwähnt weist der Hauptschleifen-Operator die selben zusätzlichen Parameter *SuccessRatio*, *MaximumSelectionPressure*, *SelectedParents*, sowie *MaximumEvaluatedSolutions* wie der Algorithmus selbst auf. Auch der *Children*-Parameter ist hier wiederum zu entfernen. Zusätzlich zu diesen Änderungen kommt jedoch noch ein weiterer Parameter *CurrentSuccessRatio* hinzu, welcher die globale Variable beschreibt, die während der Ausführung des Algorithmus die aktuell erreichte SuccessRatio aufnimmt. Die *CurrentSuccessRatio* steigt während der Offspring Selection so lange an, bis die gewünschte *SuccessRatio* erreicht wurde.

Alle weiteren Anpassungen der Hauptschleife betreffen das Einführen zusätzlicher Operatoren in den Ablauf der Evolutionsstrategie. Die notwendigen Operatoren umfassen:

- Einen *WeighedParentsQualityComparator*, der eine Lösung mit beliebig vielen Eltern vergleicht, wobei auch eine Gewichtung angegeben werden kann, die den Vergleich in Richtung besserer oder schlechterer Eltern verschiebt. Über das Gewicht kann geregelt werden, dass das Kind beispielsweise nur besser als das schlechteste Elter sein muss. Das Ergebnis der Anwendung dieses Operators im Bezug auf die Offspring Selection ist das Setzen der Variable *SuccessfulOffspring* auf „true“ oder „false“. Zusätzlich zur Ermittlung der Qualität jedes Nachkommens muss dieser Operator eingesetzt werden, um die erfolgreichen Kinder zu ermitteln. Der Operator erwartet, dass bei jeder Kind-Lösung die Eltern als Sub-Scopes im Scope-Tree vorhanden sind. Bei einer Evolutionsstrategie mit Rekombination ist dies der Fall, da vor der Rekombination die jeweiligen Eltern als Sub-Scopes des Kindes eingefügt werden. Für die Durchführung der Mutation ist ein Hinzufügen

des Elters normalerweise jedoch nicht nötig. Aus diesem Grund ist bei einer ES ohne Rekombination ein zusätzlicher Operator notwendig, der vor der Mutation eine Kopie der Lösung (das Elter) als Sub-Scope einfügt. Erst nachdem das Elter in einem Sub-Scope gespeichert wurde, kann die Mutation durchgeführt und die Fitness verglichen werden.

- Die Aufgabe des Kopierens der Eltern-Lösung in einen Sub-Scope übernimmt der *ChildrenCopyCreator*. Dieser Operator wurde neu speziell für diesen Zweck entwickelt.
- Nachdem sämtliche Nachkommen mit den Eltern verglichen wurden, können die Eltern wieder aus den Scopes der Nachkommen entfernt werden. Dies wird durch die Anwendung eines *SubScopesRemover*-Operators durchgeführt.
- Nachdem durch den Einsatz der obigen Operatoren sämtliche Nachkommen erzeugt und mit den Eltern verglichen wurden, kommt der *EvolutionStrategyOffspringSelector* zum Einsatz. Dieser Operator verwaltet temporär in jeder Generation den Pool der nicht erfolgreichen Nachkommen, sowie die neue Population. Sämtliche erzeugten Nachkommen (definiert durch den *SelectedParents*-Parameter) werden überprüft und entsprechend des Wertes der *SuccessfulOffspring* Variable zur Population oder zum Pool hinzugefügt. Sofern eine weitere Iteration der Nachkommens-Erzeugung nötig ist startet dieser Operator diese. Sobald die *SuccessRatio* erreicht wurde übernimmt der *EvolutionStrategyOffspringSelector* das Befüllen der Population mit nicht erfolgreichen Nachkommen. Das Ergebnis des Operators ist eine neue Population in Form von Sub-Scopes, die das Kriterium der *SuccessRatio* erfüllt und wie in der ursprünglichen Algorithmus-Variante weiterverarbeitet werden kann.
- Weiters sind noch zwei *Comparator*-Operatoren nötig, die am Ende der Generation die Bedingungen des maximalen Selektionsdruckes und der maximalen Anzahl evaluierter Lösungen ermitteln. Je nach Ergebnis dieser Vergleiche führt der Einsatz eines *ConditionalBranch*-Operators entweder zu einem Terminieren oder Weiterlaufen des Algorithmus.

Abb. 3.9 zeigt einen Ausschnitt der Operator-Kette der Evolutionsstrategie mit Offspring Selection im Vergleich zur Standard-ES. Die genaue Umsetzung für Operatoren wie Mutation oder Rekombination ist in diesem Ablauf nicht behandelt, entspricht aber der vorhandenen ES-Implementierung. Die zuvor angeführten zusätzlichen Operatoren sind hingegen an den relevanten Stellen eingefügt. Der *SubScopesRemover* direkt nach der Rekombination entfällt in der neuen Variante und verschiebt sich nach den *WeighedParentsQualityComparator*, der die erfolgreichen Kinder ermittelt. Anhand dieser Änderung der Operator-Kette kann eine Offspring Selection erfolgreich in die Evolutionsstrategie integriert werden.

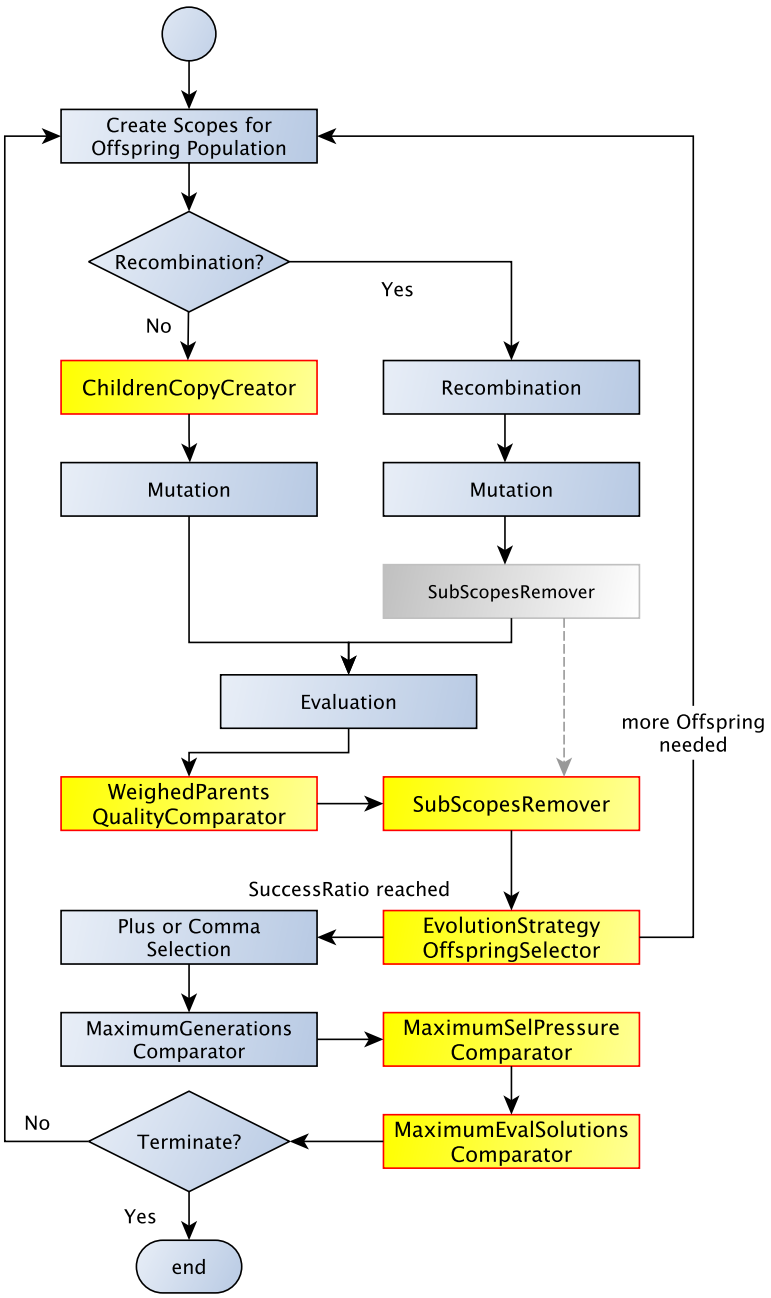

Abbildung 3.9: Änderungen am Ablauf der ES durch Einsatz zusätzlicher Operatoren für die Offspring Selection.

3.2.3 Implementierung der OS-CMA-ES

Das bereits lauffähige HeuristicLab-Plugin einer CMA-ES findet sich im Projekt *HeuristicLab.Algorithms.CMAEvolutionStrategy-3.3*. Das Projekt umfasst nicht nur die Implementierung des Algorithmus, sondern auch die Umsetzung der speziellen Operatoren, die in der CMA-Variante benötigt werden. Zum Beispiel enthält das Projekt für die Rekombination in der CMA-ES einen *EqualweightedRecombinator*, einen *LinearweightedRecombinator* und einen *LogweightedRecombinator*. Das neue Projekt *HeuristicLab.Algorithms.OffspringSelectionCMAEvolutionStrategy-3.3* basiert zwar auf der verfügbaren Implementierung, verzichtet aber auf die Kopie der generellen Operator-Implementierungen und nützt durch einen Verweis auf das Basis-Projekt die bestehenden Operatoren. Somit sind bei einer Erweiterung der möglichen Operatoren für die CMA-ES diese auch direkt in der OS-CMA-ES verfügbar. Im Gegensatz zum Projekt der einfachen Evolutionsstrategie existiert hier keine Aufteilung in eine Algorithmus-Klasse und einen Operator für die Hauptschleife, das Projekt umfasst:

- Die Klasse *OffspringSelectionCMAEvolutionStrategy*, welche den Algorithmus mit all seinen Parametern beschreibt und auch die gesamte Operator-Kette festlegt.
- Einen speziellen Operator *OffspringSelectionCMATerminator*, der analog zum *Terminator* in der Implementierung der CMA-ES sämtliche Abbruchbedingungen prüft.

Die folgenden Abschnitte beschreiben die nötigen Änderungen an diesen Klassen um die Offspring Selection zu integrieren.

Erweiterung der Algorithmus-Klasse

Zunächst sind sämtliche für diese Variante neuen Parameter zu definieren. Dazu zählen:

- Der für die Offspring Selection notwendige Parameter *SuccessRatio*. Die in HeuristicLab implementierte CMA-Variante arbeitet jedoch ohne die Erzeugung eines Überschusses an Nachkommen (λ) durch einen *Children* Parameter. Es gilt hingegen $\mu = \lambda$. Jede Generation erzeugt genau jene μ-Individuen, welche die nächste Population formen. In dieser Variante ist der evolutionäre Fortschritt rein über Adaption der Kovarianzmatrix und die gewählten Rekombinations-Gewichte definiert, die den neuen Mittelwert für die nächste Generation bestimmen. Die OS-CMA Variante sorgt dann dafür, dass diese Population zusätzlich die *SuccessRatio* erfüllt, bevor ein neuer Mittelwert ermittelt und die Kovarianzmatrix adaptiert wird.
- Der Parameter *MaximumSelectionPressure* legt eine Grenze für den Selektionsdruck fest und dient als zusätzliches Abbruchkriterium. Ein

Terminieren anhand des Erreichens einer bestimmten Anzahl an evaluierten Lösungen ist in der CMA-ES bereits umgesetzt und muss demnach nicht hinzugefügt werden.

Neben diesen beiden Parametern benötigt der Algorithmus noch einige zusätzliche globale Variablen, die ebenso als Parameter implementiert sind, jedoch nicht als Algorithmus-Parameter angezeigt werden:

- Die Variable *xMeanQuality* enthält in jeder Generation die Qualität des Mittelwertes, um anschließend die Nachkommen mit der Fitness des Mittelwertes vergleichen zu können.
- Die *SelectionPressure* verwaltet den aktuellen Selektionsdruck jeder Generation. Der Verlauf dieser Variable kann danach über einen zusätzlichen Analyzer visualisiert werden.
- Weiters beschreibt wie auch bei der OS-ES-Hauptschleife der Parameter *CurrentSuccessRatio* die aktuell erreichte SuccessRatio während der Ausführung des Algorithmus.

Das Anlegen dieser Parameter erfolgt nach dem bekannten Prinzip, der einzige Unterschied für eine Definition der globalen Variablen, die nicht als Parameter sichtbar sind, ist der eingesetzte Typ *LookupParameter*:

```
1 Parameters.Add(new LookupParameter<DoubleValue>("xMeanQuality",
2     "The value which represents the quality of the current xMean."));
```

Die Definition eines neuen Analyzers für den Selektionsdruck erfolgt ähnlich zur OS-ES Implementierung. Es muss zunächst der neue Analyzer als Eigenschaft der Klasse angelegt, und anschließend in der *Parameterize*-Methode konfiguriert werden. Diese Erweiterungen reichen aus, um den Algorithmus für eine Offspring Selection vorzubereiten. Die weiteren Änderungen betreffen die zusätzlichen Operatoren und den geänderten Ablauf des Algorithmus.

Für eine Integration der Offspring Selection in die Operator-Kette sind folgende Operatoren notwendig:

- Der *QualityComparator* kann die Qualität zweier Lösungen vergleichen. Im Gegensatz zum *WeighedParentsQualityComparator* erwartet der Operator nicht, dass eine Menge an Eltern-Lösungen als SubScopes der Nachkommen angelegt ist. Durch ihn ist es möglich, die Qualität einer Kind-Lösung mit der globalen Variable *xMeanQuality* zu vergleichen und die Nachkommen als erfolgreich oder nicht erfolgreich zu markieren.
- Für den Schritt der Fitness-Evaluierung des aktuellen Mittelwertes ist ein zusätzlicher *Evaluator* notwendig. Im Ablauf des Algorithmus ist dieser zunächst als *Placeholder*-Operator festgehalten, der bei der Auswahl eines Problems durch den spezifischen Evaluator ersetzt wird.
- Die so ermittelte Lösungsqualität muss anschließend in der zuvor definierten globalen Variable *xMeanQuality* persistiert werden. Zu diesem Zweck wird der *Assigner*-Operator eingesetzt.

- Nach dem Speichern der *xMeanQuality* ist für einen korrekten Ablauf des Algorithmus die temporäre *Quality*-Variable zu löschen, welche durch die Evaluierung automatisch erzeugt wurde. Für diese Aufgabe ist der neu implementierte *Remover*-Operator verantwortlich.
- Dieser zusätzliche Evaluierungsschritt sollte auch in der Anzahl der evaluierten Lösungen berücksichtigt werden, weshalb ein *IntCounter*-Operator einzufügen ist, der die passende Variable erhöht.
- Der Operator *CMAEvolutionStrategyOffspringSelector* führt die Offspring Selection für die CMA-ES durch. Sein Aufbau und seine Funktionsweise sind dem Operator der OS-ES sehr ähnlich. Da jedoch bei der CMA-ES keine Erzeugung von Nachkommen durch Crossover existiert, und jeder Nachkomme vom selben Elter (dem Mittelwert der letzten Population) abstammt, ist der Aufbau des Scope-Trees in dieser Variante anders. Diesen Unterschied muss der Operator berücksichtigen.
- Für die Offspring Selection ist es notwendig, dass auch innerhalb einer Generation oftmals eine zusätzliche Menge an Nachkommen erzeugt wird, bis dass die *SuccessRatio* erreicht ist. Die Individuen werden dabei aus der aktuellen Menge der Lösungen entfernt und entweder im Pool der Nachkommen archiviert oder zur nächsten Population hinzugefügt. Der Operator sieht anschließend vor, dass in einer nächsten Erzeugungs-Iteration neue Scopes für die Kinder angelegt werden. In der CMA Variante ohne Offspring Selection wurde jedoch nur eine als Sub-Scopes dargestellte Population anfangs initialisiert und anschließend immer überschrieben. Für einen korrekten Ablauf ist es deshalb nötig das Generieren der Nachkommens-Scopes in jeder Generation neu durchzuführen, weshalb am Ende des Algorithmus die aktuelle Population über einen *SubScopesRemover* entfernt werden muss.
- Der *OffspringSelectionCMATerminator*, der in einer eigenen Klasse des Projektes speziell implementiert ist, überprüft sämtliche Abbruchbedingungen des Algorithmus. Dieser Operator ersetzt den *Terminator*-Operator der zu Grunde liegenden CMA-ES Implementierung.

Abb. 3.10 zeigt die Änderungen am Ablauf der Operator-Kette bei einer Integration der neuen Operatoren für die Offspring Selection. Die zusätzlichen Operatoren erweitern an den passenden Stellen den Algorithmus und sorgen für die Evaluierung und Speicherung der Fitness des aktuellen Mittelwertes für den Vergleich der Nachkommen mit dem Elter, sowie für die Durchführung der Offspring Selection. Auch der zusätzliche *SubScopesRemover* vor Beginn einer neuen Iteration ist angeführt. Seine Berechtigung wird klar, wenn der aktuelle Ablauf mit der Abfolge der Operatoren in der CMA-Variante verglichen wird. Dort beginnt eine neue Iteration sofort bei der Erzeugung der Nachkommen durch die Mutation und überspringt ein erneutes Anlegen der Scopes.

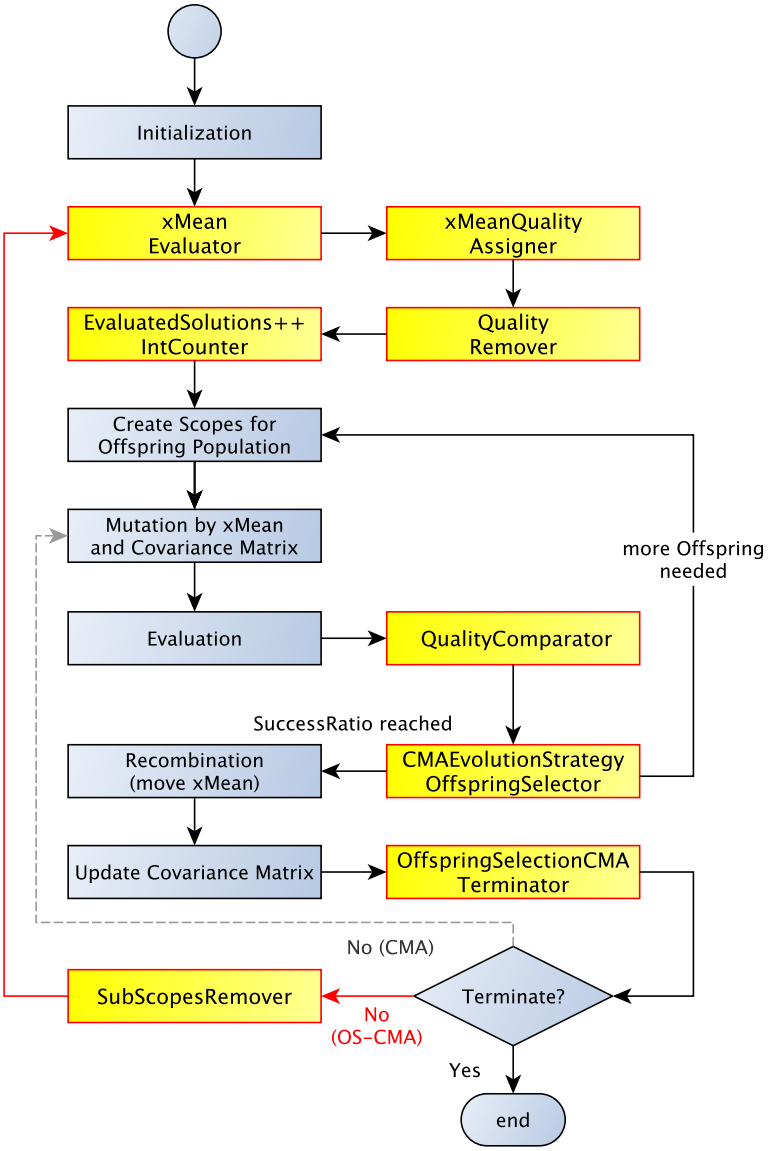

Abbildung 3.10: Der geänderte Ablauf der CMA-ES durch Einsatz zusätzlicher Operatoren für die Offspring Selection.

Aufbau des Terminators

Der *Terminator*-Operator der CMA-ES ist am Ende der Operator-Kette des Algorithmus eingesetzt und prüft sämtliche Abbruchbedingungen des Algorithmus. Diese umfassen das Generationslimit, die Grenze an evaluierten Lösungen, sowie einige weitere durch die Strategieparameter der CMA

festgelegte Kriterien. Der neue Operator *OffspringSelectionCMATerminator* leitet von der Klasse *Terminator* ab, und erweitert diese um die Überprüfung des maximalen Selektionsdruckes. Dies ist durch ein Einfügen der betreffenden Parameter *SelectionPressure* und *MaximumSelectionPressure* gelöst. Das Überschreiben der *Apply*-Methode des Operators erlaubt die Erweiterung der Implementierung des Basis-Terminators um die Prüfung des Selektionsdruckes.

```
1 public override IOperation Apply() {
2   var terminateOp = Terminate != null ? ExecutionContext.CreateOperation
      (Terminate) : null;
3
4   var selPress = SelectionPressureParameter.ActualValue.Value;
5   var maxSelPress = MaximumSelectionPressureParameter.ActualValue.Value;
6   if (selPress >= maxSelPress) return terminateOp;
7
8   return base.Apply();
9 }
```

Durch diese Anpassung und den Einsatz des neuen Terminators am Ende der Operator-Kette ist das Terminieren bei einem Erreichen des maximalen Selektionsdruckes gewährleistet.

Die Umsetzung von Algorithmen in HeuristicLab in der zuvor beschriebenen Weise benötigt zwar eine gewisse Zeit zur Einarbeitung in das Framework, den Klassenaufbau und in die Art der Implementierung mittels Operatoren, Scopes und Parametern, jedoch bietet das Framework danach viele Vorteile. Für spätere Tests sind unter anderem die Verfügbarkeit verschiedenster Problemstellungen, die Analyse-Tools, sowie die Möglichkeit der Pausierung und Persistierung von Testläufen sehr praktisch. Weiters besteht auch die Möglichkeit zur Definition von Experimenten, um unterschiedlichste Parameter-Werte der Algorithmen mit einer ausreichenden Anzahl an Wiederholungen pro Konfiguration zu untersuchen. Das nächste Kapitel widmet sich dem Aufbau der Tests der neuen Algorithmen mit HeuristicLab und den durch die Durchführung erzielten Ergebnissen.

Kapitel 4

Empirische Untersuchung

4.1 Aufbau der Experimente

Der Vorteil von Evolutionsstrategien ist, dass sie gute Lösungen in Form von lokalen Optima relativ schnell finden können. Das Erreichen des globalen Optimums ist hingegen häufig schwierig, da vor dem Finden dieses Optimums der Algorithmus bereits in Richtung eines guten lokalen Optimums konvergiert und danach nur schwer bessere Lösungen findet. Es tritt eine Stagnation im lokalen Optimum ein. Der Vorteil der schnellen Ermittlung einer relativ guten Lösung spielt vor allem in jenen Anwendungsgebieten eine Rolle, die eine sehr zeitaufwändige Fitnessevaluierung jeder Lösung erfordern. Im Bereich der simulations-basierten Optimierung wird der Fitnesswert einer Lösung beispielsweise durch die Simulation eines Problems ermittelt. Solche Berechnungen dauern mitunter sehr lange. Das Ermitteln einer guten Lösung anhand einiger weniger Lösungs-Evaluierungen ist für solche Probleme sehr vorteilhaft. Die durchgeführten Tests der neuen Algorithmen sollen diese möglichst sinnvoll mit den bestehenden Varianten vergleichen. Da die Stärke der Evolutionsstrategie genau in der geringen Anzahl an notwendigen Evaluierungen liegt, wurde für sämtliche Tests zunächst die Qualität bei einer gewissen Anzahl an evaluierten Lösungen als Maßstab für den Vergleich gewählt. Sollten die weiteren Tests anschließend andere Zielsetzungen verfolgen, ist diese in den entsprechenden Abschnitten näher erklärt.

4.1.1 Testen mit HeuristicLab

Alle in diesem Kapitel vorgestellten Tests wurden anhand der Implementierung der Algorithmen in HeuristicLab durchgeführt und analysiert. Einige kurze Testläufe der Algorithmen bei verschiedenen Problemstellungen dienten als Grundlage für die Wahl der passenden Parameter, die anschließend für intensivere Tests herangezogen wurden. Für das Durchführen der Testläufe bietet das HeuristicLab die Möglichkeit der Definition von *Ex-*

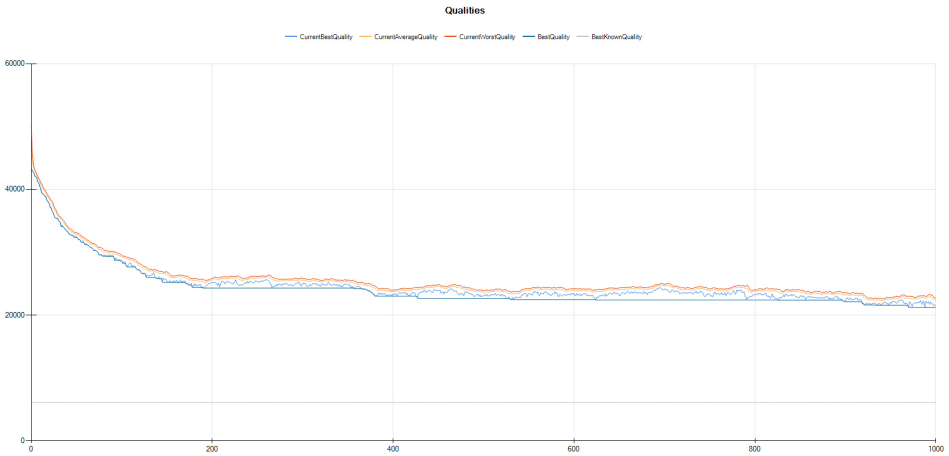

Abbildung 4.1: Visualisierung des Verlaufes der Lösungsqualität in HeuristicLab.

perimenten. Jedes Experiment besteht aus einer Sammlung verschiedener Konfigurationen an Algorithmen oder Problemstellungen, die mit einer fixen Anzahl an Wiederholungen durchgeführt werden. Für die hier vorgestellten Tests kam jeweils ein Experiment pro Algorithmus-Variante und Problemstellung mit fünf Wiederholungen pro Testlauf zum Einsatz. Zur Analyse der Ergebnisse bietet das HeuristicLab verschiedene Tools, von denen hauptsächlich zwei bei der Evaluierung zum Einsatz kamen:

1. Es kann die beste und die durchschnittliche Lösungsqualität aller Wiederholungen eines Testlaufs berechnet werden. Anhand dieser Ergebnisse wird die erreichte Lösungsqualität der Algorithmen verglichen.
2. Der Einsatz von Analyzern in der Algorithmen-Implementierung erlaubt die Visualisierung des Verlaufs von Variablen, wie z. B. Qualität oder Selektionsdruck, über die Generationen hinweg. Diese Verläufe geben Aufschluss über den Fortschritt der Verbesserung der Qualität, die Geschwindigkeit der Konvergenz oder wann ein Algorithmus stagniert. Abb. 4.1 zeigt einen solchen Qualitätsverlauf. Die rote, gelbe und hellblaue Linie zeigen jeweils die schlechteste, durchschnittliche und beste Qualität der Population einer gewissen Generation. Die dunkelblaue Linie zeigt die beste bisher gefundene Qualität, die generell beste Lösung ist durch eine hellgraue Linie dargestellt.

4.1.2 Ausgewählte Probleme

Da Evolutionsstrategien häufig zur Optimierung von Zielfunktionen mit reellwertigen Lösungsvektoren eingesetzt werden, ist ein Test anhand standardisierter Benchmark-Funktionen in verschiedenen Problemdimensionen

sinnvoll. Diese Funktionen stammen von verschiedenen Autoren und werden für die Analyse und den Vergleich von unterschiedlichen Optimierungs-Verfahren eingesetzt. Sie sind in einer Weise konstruiert, die es Algorithmen schwer macht das globale Optimum zu finden und eignen sich so zum Erkennen von möglichen Schwachstellen. Folgende Funktionen wurden für die Tests in dieser Arbeit ausgewählt[1]:

- Die n-dimensionale *Rosenbrock*-Funktion [Ros60]:

$$f(\vec{x}) = \sum_{i=1}^{n-1} 100 \cdot (x_{i+1} - x_i^2)^2 + (1 - x_i)^2 \tag{4.1}$$

  für $-2.048 \leq x(i) \leq 2.048$ mit einem globalen Optimum $f(\vec{x}) = 0$ bei $\vec{x} = (1, 1, 1, \ldots, 1)$.

- Die n-dimensionale *Rastrigin*-Funktion [MSB91]:

$$f(\vec{x}) = 10 \cdot n + \sum_{i=1}^{n} x_i^2 - 10\ cos(2 \cdot \pi \cdot x_i) \tag{4.2}$$

  für $-5.12 \leq x(i) \leq 5.12$ mit einem globalen Optimum $f(\vec{x}) = 0$ bei $\vec{x} = (0, 0, 0, \ldots, 0)$.

- Die n-dimensionale *Schwefel*-Funktion [Sch81]:

$$f(\vec{x}) = 418.982887272433 \cdot n + \sum_{i=1}^{n} -x_i\ sin(\sqrt{|x_i|}) \tag{4.3}$$

  für $-500 \leq x(i) \leq 500$ mit einem globalen Optimum $f(\vec{x}) = 0$ bei $\vec{x} = (420.968746453712, 420.968746453712, \ldots, 420.968746453712)$.

- Die n-dimensionale *Griewank*-Funktion [Gri81]:

$$f(\vec{x}) = \frac{1}{4000} \sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} cos(\frac{x_i}{\sqrt{i}}) + 1 \tag{4.4}$$

  für $-600 \leq x(i) \leq 600$ mit einem globalen Optimum $f(\vec{x}) = 0$ bei $\vec{x} = (0, 0, 0, \ldots, 0)$.

[1] Die Zahlen in den mathematischen Formeln sind in der englischen Komma-Notation angegeben (z. B. 5.12 statt 5,12).

- Die n-dimensionale *Ackley*-Funktion [Ack87][Bäc96]:

$$f(\vec{x}) = 20 + e - 20 \cdot e^{-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}} - e^{\frac{1}{n}\sum_{i=1}^{n} cos(2 \cdot \pi \cdot x_i)} \tag{4.5}$$

für $-32.768 \leq x(i) \leq 32.768$ mit einem globalen Optimum $f(\vec{x}) = 0$ bei $\vec{x} = (0, 0, 0, \ldots, 0)$.

Zusätzlich zu diesen reellwertigen Testfunktionen ist ebenso die Betrachtung einiger kombinatorischer Optimierungsprobleme interessant. Zu diesem Zweck wurden die Algorithmen auf zwei TSP-Benchmarkproblemen (ch130, kroA100[2]), sowie einem CVRP-Problem (E-n101-k14[3]) getestet.

4.2 Untersuchung der OS-ES

Die Evolutionsstrategie kann in einigen unterschiedlichen Varianten betrieben werden. Dazu zählen der optionale Einsatz von Rekombination oder die Art der durchgeführten Selektion mit oder ohne Einbezug der Eltern. Die folgenden Tests vergleichen die Ergebnisse der Evolutionsstrategie ohne Offspring Selection mit denen der neuen OS-ES anhand der gleichen Wahl für die Rekombination und das Selektions-Verfahren.

4.2.1 Ergebnisse ohne Rekombination

Rosenbrock-Funktion

Für die Rosenbrock-Funktion wurde eine (10+50)-ES mit einer (10+1.0)-OS-ES verglichen. Diese Parameterwerte erwiesen sich bei einigen Testläufen für eine Problemdimension von 100 innerhalb 1000 berechneter Generationen als sinnvoll. Tabelle 4.1 zeigt die Ergebnisse der ES, Tabelle 4.2 die entsprechende erreichte Lösungs-Qualität der OS-ES.

Tabelle 4.1: Ergebnisse der (10+50)-ES für die Rosenbrock-Funktion.

Dimension	*Beste Qualität*	*Durchsch. Qualität*	*Eval. Lösungen*
n=20	13,0466	15,8295	250.010
n=100	680,8342	1.064,5199	350.010
n=500	14.598,0586	21.163,7322	490.010
n=1000	69.284,4179	76.286,9836	585.010

[2]Bestandteil der *TSPLIB* der Ruprecht-Karls-Universität Heidelberg, http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/

[3]ein Capacitated Vehicle Routing Problem von Christofides und Eilon [CE69]

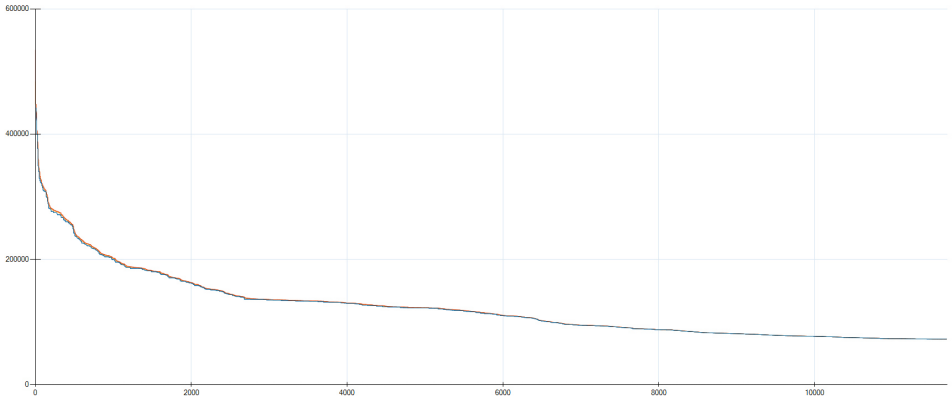

Abbildung 4.2: Qualitätsverlauf der ES für die 1000-dimensionale Rosenbrock-Funktion.

Tabelle 4.2: Ergebnisse der (10+1.0)-OS-ES für die Rosenbrock-Funktion.

Dimension	*Beste Qualität*	*Durchsch. Qualität*	*Eval. Lösungen*
n=20	11,7903	15,1770	250.030
n=100	605,1310	891,3160	350.030
n=500	16.918,8000	21.631,7472	490.030
n=1000	119.071,8099	132.540,2243	585.030

In geringen Problem-Dimensionen konnte die neue OS-ES etwas bessere Ergebnisse als die ES erzielen. Für hohe Problemdimensionen erreichte jedoch die Evolutionsstrategie ohne Offspring Selection eine höhere Lösungsqualität. Die Visualisierung des Qualitätsverlaufes bei einer Problemdimension von 1000 verdeutlicht den Unterschied (Abb. 4.2 und 4.3). Während die ES speziell in den ersten Generationen bereits sehr schnell die Lösungsqualität verbessert, konvergiert die OS-ES weit langsamer und erreicht bei gleicher Anzahl an evaluierten Lösungen eine weniger gute Qualität.

Rastrigin-Funktion

Folgende Tabellen 4.3 und 4.4 zeigen die Ergebnisse für die Rastrigin-Funktion bei einer Problemdimension von bis zu 2000 unter Einsatz einer (10,100)-ES und einer (20,0.7)-OS-ES.

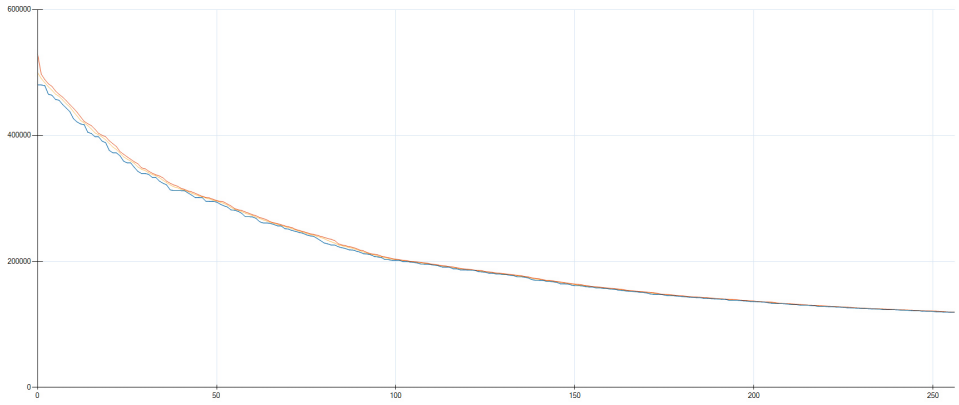

Abbildung 4.3: Qualitätsverlauf der OS-ES für die 1000-dimensionale Rosenbrock-Funktion.

Tabelle 4.3: Ergebnisse der (10,100)-ES für die Rastrigin-Funktion.

Dimension	*Beste Qualität*	*Durchsch. Qualität*	*Eval. Lösungen*
n=20	66,6621	99,7710	500.010
n=100	444,1541	519,1151	700.010
n=500	3.959,6549	4.238,4909	980.010
n=1000	9.575,8919	9.873,1461	1.170.010
n=2000	23.021,6559	23.700,2786	1.400.010

Tabelle 4.4: Ergebnisse der (20,0.7)-OS-ES für die Rastrigin-Funktion.

Dimension	*Beste Qualität*	*Durchsch. Qualität*	*Eval. Lösungen*
n=20	53,7355	78,8022	500.020
n=100	490,8381	555,2990	700.020
n=500	4.026,7815	4.704,9818	980.020
n=1000	13.733,5413	14.416,3747	1.170.020
n=2000	31.734,2130	32.186,78023	1.400.020

Die Ergebnisse unterstreichen die Resultate der Rosenbrock-Funktion. Bei einer Dimensionsgröße von $n = 20$ ermittelt die OS-ES noch gute Lösungen, in höheren Dimensionen wird der Unterschied der erreichten Qualitäten jedoch zunehmend größer. Auch die im folgenden dargestellten Ergebnisse der weiteren Benchmark-Funktionen entsprechen dieser Beobachtung.

Ergebnisse der Schwefel-, Griewank- und Ackley-Funktion

Für den Test anhand der Schwefel-Funktion wurde eine (20,100)-ES mit einer (10,0.7)-OS-ES verglichen:

Tabelle 4.5: Ergebnisse der (20,100)-ES für die Schwefel-Funktion.

Dimension	*Beste Qualität*	*Durchsch. Qualität*	*Eval. Lösungen*
n=20	2.428,0488	2.921,9992	500.020
n=100	16.321,7174	17.821,7649	700.020
n=500	112.630,8635	114.259,4156	980.020
n=1000	251.498,5269	258.541,0286	1.170.020
n=2000	568.536,5850	575.360,2675	1.400.020

Tabelle 4.6: Ergebnisse der (10,0.7)-OS-ES für die Schwefel-Funktion.

Dimension	*Beste Qualität*	*Durchsch. Qualität*	*Eval. Lösungen*
n=20	2.053,0017	3.192,8875	500.050
n=100	16.499,0389	18.126,2569	700.050
n=500	105.505,5094	112.427,4804	980.050
n=1000	322.671,6670	326.511,1125	1.170.050
n=2000	717.981,9362	735.373,1489	1.400.050

Für die Griewank-Funktion wurde die Optimierung von einer (10+100)-ES und einer (20+0.7)-OS-ES durchgeführt:

Tabelle 4.7: Ergebnisse der (10+100)-ES für die Griewank-Funktion.

Dimension	*Beste Qualität*	*Durchsch. Qualität*	*Eval. Lösungen*
n=20	$5,10 \cdot 10^{-2}$	$3,98 \cdot 10^{-1}$	500.010
n=100	9,4155	35,0942	700.010
n=500	1.733,9927	2.055,2837	980.010
n=1000	6.397,6716	6.698,7323	1.170.010
n=2000	21.167,8198	23.566,1836	1.400.010

Tabelle 4.8: Ergebnisse der (20+0.7)-OS-ES für die Griewank-Funktion.

Dimension	*Beste Qualität*	*Durchsch. Qualität*	*Eval. Lösungen*
n=20	$6,40 \cdot 10^{-2}$	$4,74 \cdot 10^{-1}$	500.020
n=100	32,0055	103,7828	700.020
n=500	2.045,0486	2.300,1205	980.020
n=1000	7.976,9450	8.801,0774	1.170.020
n=2000	28.532,5493	31.953,3182	1.400.020

Abschließend kam noch eine (10+100)-ES im Vergleich zu einer (20+0.85)-OS-ES bei der Ackley-Funktion zum Einsatz:

Tabelle 4.9: Ergebnisse der (10+100)-ES für die Ackley-Funktion.

Dimension	*Beste Qualität*	*Durchsch. Qualität*	*Eval. Lösungen*
n=20	2,1697	6,2091	500.010
n=100	14,6881	16,1077	700.010
n=500	19,2942	19,756	980.010
n=1000	20,6673	20,7869	1.170.010
n=2000	21,0044	21,0442	1.400.010

Tabelle 4.10: Ergebnisse der (20+0.85)-OS-ES für die Ackley-Funktion.

Dimension	*Beste Qualität*	*Durchsch. Qualität*	*Eval. Lösungen*
n=20	$1,85 \cdot 10^{-5}$	6,2296	500.020
n=100	9,4143	13,3497	700.020
n=500	19,6987	20,0084	980.020
n=1000	20,8424	20,9189	1.170.020
n=2000	21,0711	21,0939	1.400.020

Zusätzlich zu den Benchmark-Funktionen wurden auch die Ergebnisse der Algorithmen für kombinatorische Optimierungsprobleme bei ungefähr 2.500.000 evaluierten Lösungen ermittelt.

Ergebnisse für TSP und CVRP

Die folgenden Tabellen zeigen die Resultate der (10+100)-ES und einer (10+0.4)-OS-ES für zwei Traveling Salesman Probleme. Das Capacitated Vehicle Routing Problem wurde von einer (10,100)-ES und einer (10,1.0)-OS-ES gelöst:

Tabelle 4.11: Ergebnisse der ES für ausgewählte TSP-Probleme und ein CVRP-Problem.

Problem	*Beste Qualität*	*Durchsch. Qualität*	*Eval. Lösungen*
TSP ch130	1.591,0000	1.781,2000	2.500.010
TSP kroA200	0,0000	533,4000	2.500.010
CVRP e-n101-k14	0,0000	7,2000	2.500.010

Tabelle 4.12: Ergebnisse der OS-ES für die TSP-Probleme und das CVRP-Problem.

Dimension	*Beste Qualität*	*Durchsch. Qualität*	*Eval. Lösungen*
TSP ch130	1.506,0000	2.100,4000	2.500.030
TSP kroA200	0,0000	2.085,6000	2.500.030
CVRP e-n101-k14	0,0000	28,6000	2.500.030

Die Auswertung der Ergebnisse deckt sich auch für diese Probleme mit den Resultaten der Benchmark-Funktionen. Aufgrund der langsameren Konvergenz der OS-ES werden eher schlechtere Lösungsqualitäten erzielt. Eine starke Verbesserung der Qualität in weiteren Generationen ist in beiden Algorithmus-Varianten nicht zu erwarten. Die Evolutionsstrategie erreicht sehr schnell eine gute Lösung und stagniert dann bereits viele Generationen ohne merkliche Verbesserungen zu erzielen. Die OS-ES erreicht ihre Lösung langsamer, weist aber speziell in späten Generationen bereits extrem hohe Selektionsdruck-Werte auf. Der Algorithmus kann also ebenso keine guten Lösungen mehr ermitteln, die eine bessere Fitness als ihre Eltern aufweisen. Ein Unterschied der Wirkung von Offspring Selection in der Komma- und Plus-Variante der Evolutionsstrategie kann in diesen Tests nicht erkannt werden. In beiden Fällen entspricht der Einsatz der Offspring Selection einem leichten Ausbremsen der Evolutionsstrategie. Bei manchen Problemen in geringen Dimensionen kann dieses Verhalten jedoch zu leicht besseren Ergebnissen führen, z. B. bei der Rosenbrock oder der Rastrigin-Funktion bei einer Dimension von $n = 20$.

Ein weiterer wichtiger Aspekt von Evolutionsstrategien ist der zusätzliche Einsatz von Rekombination. Die folgenden Tests widmen sich der Auswirkung von Rekombination auf die Evolutionsstrategie mit und ohne Offspring Selection. Dabei kam jeweils eine Rekombination von zwei Eltern ($\rho = 2$) durch Einsatz eines bestimmten Operators zum Einsatz.

4.2.2 Ergebnisse mit Rekombination

Für die reellwertigen Benchmark-Funktionen erwies sich durch einige kurze Tests der Einsatz eines *Average*-Crossovers, der den neuen Lösungsvektor über den Durchschnitt der Komponenten der Eltern-Vektoren ermittelt, als sehr passend. Eine Ausnahme bildet die Schwefel-Funktion, für die eine *Diskrete Rekombination* zu besseren Ergebnissen führte.

Rosenbrock-Funktion

Im folgenden sind die Ergebnisse einer (20/2+100)-ES im Vergleich zu einer (20/2+0.4)-OS-ES dargestellt:

Tabelle 4.13: Ergebnisse der (20/2+100) -ES für die Rosenbrock-Funktion unter Einsatz eines Average-Crossovers.

Dimension	*Beste Qualität*	*Durchsch. Qualität*	*Eval. Lösungen*
n=20	$9,00 \cdot 10^{-4}$	$7,99 \cdot 10^{-1}$	500.020
n=100	62,6459	65,5622	700.020
n=500	481,7807	490,3841	980.020
n=1000	1.019,0252	1.054,0206	1.170.020

Tabelle 4.14: Ergebnisse der (20/2+0.4)-OS-ES für die Rosenbrock-Funktion unter Einsatz eines Average-Crossovers.

Dimension	*Beste Qualität*	*Durchsch. Qualität*	*Eval. Lösungen*
n=20	$6,50 \cdot 10^{-3}$	$1,24 \cdot 10^{-2}$	500.060
n=100	71,6823	72,7621	700.060
n=500	484,1540	489,7204	980.060
n=1000	1.003,4412	1.022,8139	1.170.060

Die Ergebnisse zeigen generell eine starke Verbesserung der erreichten Qualität gegenüber den Algorithmen ohne Rekombination. Weiters ist es der OS-ES nun auch in höheren Problemdimensionen möglich, etwa gleich gute, wenn nicht leicht bessere Resultate zu erzielen.

Rastrigin-Funktion

Für die Rastrigin-Funktion wurden eine (20/2,100)-ES und eine (10/2,0.55)-OS-ES getestet:

Tabelle 4.15: Ergebnisse der (20/2,100)-ES für die Rastrigin-Funktion mit Rekombination durch einen Average-Crossover.

Dimension	*Beste Qualität*	*Durchsch. Qualität*	*Eval. Lösungen*
n=20	7,9597	10,9445	500.010
n=100	96,5110	113,8232	700.010
n=500	1.046,1368	1.508,6665	980.010
n=1000	10.936,6916	11.138,2186	1.170.010
n=2000	22.102,8128	22.683,3197	1.400.010

Tabelle 4.16: Ergebnisse der (10/2,0.55)-OS-ES für die Rastrigin-Funktion mit Rekombination durch einen Average-Crossover.

Dimension	*Beste Qualität*	*Durchsch. Qualität*	*Eval. Lösungen*
n=20	6,9647	11,5415	500.030
n=100	67,6572	221,4398	700.030
n=500	4.976,9768	5.135,6664	980.030
n=1000	10.981,6203	11.085,8972	1.170.030
n=2000	22.698,8646	23.098,7133	1.400.030

Obwohl auch hier der Einsatz von Rekombination zu besseren Ergebnissen führt, ist die Variante mit Offspring Selection nicht in der Lage, die normale Evolutionsstrategie zu übertreffen. Eine Analyse der Verläufe von Qualität und Selektionsdruck zeigen, dass speziell in frühen Generationen sehr wenig Selektionsdruck ausgeübt wird, da das Ziel der *SuccessRatio* noch relativ leicht erfüllt werden kann.

Schwefel-Funktion

Die (20/2,100)-ES und die (20/2,0.55)-OS-ES der folgenden Tests der Schwefel-Funktion arbeiten mit *Diskreter Rekombination*:

Tabelle 4.17: Ergebnisse der (20/2,100)-ES für die Schwefel-Funktion mit Disktreter Rekombination.

Dimension	*Beste Qualität*	*Durchsch. Qualität*	*Eval. Lösungen*
n=20	236,8767	623,7774	500.020
n=100	2.645,1364	3.818,9092	700.020
n=500	180.525,5511	181.800,4916	980.020
n=1000	380.666,4031	381.920,9306	1.170.020
n=2000	785.985,8225	787.785,0499	1.400.020

Tabelle 4.18: Ergebnisse der (20/2,0.55)-OS-ES für die Schwefel-Funktion mit Diskreter Rekombination.

Dimension	*Beste Qualität*	*Durchsch. Qualität*	*Eval. Lösungen*
n=20	592,1917	1.133,0770	500.060
n=100	4.678,3414	5.248,1094	700.060
n=500	42.982,8983	46.179,3437	980.060
n=1000	358.003,3155	361.491,9957	1.170.060
n=2000	762.430,3591	765.235,2278	1.400.060

In geringen Dimensionen ist es der OS-ES nicht möglich, bessere Ergebnisse als die ES zu erzielen. Beide Algorithmen stagnieren innerhalb der

ungefähr 500.000 evaluierten Lösungen bereits. Hohe Problemdimensionen machen es den Algorithmen jedoch zunehmend schwieriger ein gutes lokales Optimum zu finden. Die Evolutionsstrategie stellt sich sehr schnell auf ein Optimum ein und stagniert dann. Der Qualitätsverlauf zeigt beispielsweise bei Dimension $n = 1000$ bereits nach ca. 800.000 evaluierten Lösungen keine merkliche Verbesserung der Qualität. Die OS-ES hingegen schafft es die Qualität über alle Generationen hinweg langsam immer weiter zu verbessern und erreicht somit innerhalb der 1.170.000 evaluierten Lösungen eine in Summe etwas bessere Qualität als die ES. Die Analyse des Selektionsdruckes zeigt, dass eine weitere Verbesserung durch längere Laufzeit des Algorithmus eventuell noch möglich wäre. Es ist jedoch fraglich, ob die Qualität der Evolutionsstrategie ohne Rekombination und Offspring Selection übertroffen werden kann, welche bei der Schwefel-Funktion in hohen Dimensionen über alle Tests hinweg die besten Ergebnisse lieferte.

Griewank- und Ackley-Funktion

Die Kombination von Rekombination mittels Durchschnittsbildung und dem Einsatz der Offspring Selection erwies sich speziell bei der Griewank- und der Ackley-Testfunktion als sehr effektiv. Im Falle der Griewank-Funktion kamen eine (10/2+100)-ES und eine (40/2+1.0)-OS-ES zum Einsatz.

Tabelle 4.19: Ergebnisse der (10/2+100)-ES für die Griewank-Funktion mit einem Average-Crossover.

Dimension	*Beste Qualität*	*Durchsch. Qualität*	*Eval. Lösungen*
n=20	0,0000	$3,95 \cdot 10^{-3}$	500.010
n=100	$4,90 \cdot 10^{-13}$	$1,13 \cdot 10^{-1}$	700.010
n=500	3,1805	5,2654	980.010
n=1000	65,9563	89,0327	1.170.010
n=2000	636,5047	765,3862	1.400.010

Tabelle 4.20: Ergebnisse der (40/2+1.0)-OS-ES für die Griewank-Funktion mit einem Average-Crossover.

Dimension	*Beste Qualität*	*Durchsch. Qualität*	*Eval. Lösungen*
n=20	0,0000	0,0000	500.040
n=100	0,0000	$2,46 \cdot 10^{-3}$	700.040
n=500	$3,54 \cdot 10^{-6}$	$3,70 \cdot 10^{-2}$	980.040
n=1000	$2,44 \cdot 10^{-2}$	$5,89 \cdot 10^{-2}$	1.170.040
n=2000	$6,72 \cdot 10^{-1}$	$9,16 \cdot 10^{-1}$	1.400.040

Für Problemdimension $n = 20$ konnte bei allen Wiederholungen das globale Optimum gefunden werden, in allen weiteren Dimensionen zumindest

eine Qualität kleiner als Eins. Dies entspricht einer enormen Verbesserung im Vergleich zur ES ohne Offspring Selection. Bei der Ackley-Funktion wurden eine (10/2,100)-ES und eine (40/2,0.85)-OS-ES gewählt:

Tabelle 4.21: Ergebnisse der (10/2,100)-ES für die Ackley-Funktion unter Einsatz eines Average-Crossovers.

Dimension	*Beste Qualität*	*Durchsch. Qualität*	*Eval. Lösungen*
n=20	$3,55 \cdot 10^{-15}$	$3,55 \cdot 10^{-15}$	500.010
n=100	1,3231	2,0328	700.010
n=500	3,8774	4,3568	980.010
n=1000	5,5171	5,9648	1.170.010
n=2000	9,3913	9,9255	1.400.010

Tabelle 4.22: Ergebnisse der (40/2,0.85)-OS-ES für die Ackley-Funktion unter Einsatz eines Average-Crossovers.

Dimension	*Beste Qualität*	*Durchsch. Qualität*	*Eval. Lösungen*
n=20	$3,55 \cdot 10^{-15}$	$3,55 \cdot 10^{-15}$	500.040
n=100	$1,42 \cdot 10^{-14}$	$2,13 \cdot 10^{-14}$	700.040
n=500	$1,71 \cdot 10^{-4}$	$5,41 \cdot 10^{-2}$	980.040
n=1000	$9,04 \cdot 10^{-1}$	$9,22 \cdot 10^{-1}$	1.170.040
n=2000	1,2694	1,3479	1.400.040

Auch hier ist es möglich, durch die Offspring Selection die erreichte Lösungsqualität in sämtlichen Problem-Dimensionen zu verbessern. Im Gegensatz zur Schwefel-Funktion in hohen Dimensionen findet die Optimierung der Griewank- und Ackley-Funktion nicht schleppend über alle Generationen hinweg statt, sondern konvergiert ähnlich der Evolutionsstrategie sehr schnell zu einem Optimum. Die so über Offspring Selection ermittelte Lösung übertrifft jedoch das von der ES ermittelte Optimum.

TSP und CVRP

Für die Durchführung der Rekombination der Traveling Salesman Probleme durch eine (10/2+100)-ES und eine (10/2+0.2)-OS-ES wurde ein ERX-Operator (Edge Recombination Crossover) verwendet. Zur Kreuzung zweier CVRP-Lösungen kam ein einfacher Permutations-Operator nach *Alba und Dorronsoro* zum Einsatz [AD04]. Die Optimierung des CVRP-Problems übernahmen eine (10/2,100)-ES und eine (20/2,0.7)-OS-ES:

Tabelle 4.23: Ergebnisse der ES für die TSP- und CVRP-Probleme mit Rekombination.

Problem	*Beste Qualität*	*Durchsch. Qualität*	*Eval. Lösungen*
TSP ch130	504,0000	818,8000	2.500.010
TSP kroA200	0,0000	965,0000	2.500.010
CVRP e-n101-k14	0,0000	21,4000	2.500.010

Tabelle 4.24: Ergebnisse der OS-ES für die TSP- und CVRP-Probleme mit Rekombination.

Dimension	*Beste Qualität*	*Durchsch. Qualität*	*Eval. Lösungen*
ch130	778,0000	963,4000	2.500.030
kroA200	0,0000	574,2000	2.500.030
CVRP e-n101-k14	0,0000	8,2000	2.500.020

Die Integration einer Rekombination erwies sich für die ES bei dem CVRP-Problem und dem TSP-Problem kroA200 als eher negativ, da die durchschnittliche Lösungsqualität hier höher liegt. Die OS-ES konnte jedoch in allen Fällen von der zusätzlichen Rekombination profitieren. Es ist der OS-ES jedoch nicht möglich, die ES zu übertreffen, da für das kroA200 und das CVRP-Problem die ES ohne Rekombination zuvor bessere Lösungen ermittelte. Eine weitere Verbesserung der Qualität durch eine längere Laufzeit ist wieder unwahrscheinlich, da beide Varianten bereits stagnieren und ein starker Fortschritt der Optimierung nicht mehr möglich ist, außer durch einen zufälligen Glückstreffer.

4.2.3 Rekombination mit multiplen Operatoren

Neben der Auswahl eines fixen Rekombinations-Schemas ist es auch möglich, für jede Rekombination zufällig einen Operator aus einem Pool an möglichen Operatoren zu selektieren. Dies hat den Vorteil, dass manche Probleme, die eventuell im späteren Optimierungsverlauf besser auf einen anderen Rekombinations-Operator ansprechen, noch weiter optimiert werden können. Die Verwendung von multiplen Operatoren führte im Gegensatz zu einfachen Operatoren für die meisten Probleme eher zu schlechteren Ergebnissen.

Ergebnisse der Rosenbrock-, Rastrigin- und Schwefel-Funktion

Die Rosenbrock-Funktion wurde anhand einer (20/2+100)-ES und einer (20/2+0.4)-OS-ES optimiert:

Tabelle 4.25: Ergebnisse der (20/2+100)-ES für die Rosenbrock-Funktion unter Einsatz eines Multi-Operator-Crossovers.

Dimension	*Beste Qualität*	*Durchsch. Qualität*	*Eval. Lösungen*
n=20	$8,63 \cdot 10^{-4}$	$1,88 \cdot 10^{-3}$	500.020
n=100	74,8550	77,5443	700.020
n=500	501,8779	514,7614	980.020
n=1000	1.349,5512	1.413,7105	1.170.020

Tabelle 4.26: Ergebnisse der (20/2+0.4)-OS-ES für die Rosenbrock-Funktion unter Einsatz eines Multi-Operator-Crossovers.

Dimension	*Beste Qualität*	*Durchsch. Qualität*	*Eval. Lösungen*
n=20	$1,04 \cdot 10^{-2}$	$1,51 \cdot 10^{-2}$	500.060
n=100	79,6632	81,4449	700.060
n=500	498,2709	511,7052	980.060
n=1000	1.228,1557	1.375,7207	1.170.060

Die Ergebnisse sind etwas schlechter als jene unter Einsatz des Average-Crossovers, jedoch besser als gänzlich ohne Rekombination. Die OS-ES übertrifft die Qualität der ES nun nur noch in höheren Dimensionen. Für die Rastrigin-Funktion wurden eine (10/2,100)-ES und eine (10/2,0.55)-OS-ES verwendet:

Tabelle 4.27: Ergebnisse der (10/2,100)-ES für die Rastrigin-Funktion mit Rekombination durch verschiedene Crossover-Operatoren.

Dimension	*Beste Qualität*	*Durchsch. Qualität*	*Eval. Lösungen*
n=20	5,9698	11,5415	500.010
n=100	103,4756	116,0121	700.010
n=500	5.352,0984	5.435,3904	980.010
n=1000	10.819,3022	11.186,5072	1.170.010
n=2000	22.827,9411	23.037,9094	1.400.010

Tabelle 4.28: Ergebnisse der (10/2,0.55)-OS-ES für die Rastrigin-Funktion mit Rekombination durch verschiedene Crossover-Operatoren.

Dimension	*Beste Qualität*	*Durchsch. Qualität*	*Eval. Lösungen*
n=20	9,9496	17,1133	500.030
n=100	94,521	114,6192	700.030
n=500	5.252,7748	5.356,3567	980.030
n=1000	11.714,7477	11.845,3991	1.170.030
n=2000	24.893,665	25.606,0853	1.400.030

Auch hier können durch multiple Rekombinations-Operatoren keine besseren Ergebnisse erzielt werden. Die Ergebnisse der OS-ES sind eher etwas schlechter als die der ES, können diese aber eventuell manchmal leicht übertreffen. Große Unterschiede durch eine längere Ausführungszeit sind hier nicht zu erwarten, da der Selektionsdruck gegen Ende der durchgeführten Berechnung bereits relativ hoch ist. Anhand der Schwefel-Funktion wurden eine (10/2,50)-ES und eine (40/2,0.55)-OS-ES verglichen:

Tabelle 4.29: Ergebnisse der (10/2,50)-ES für die Schwefel-Funktion mit Rekombination durch zufällige Operatoren.

Dimension	*Beste Qualität*	*Durchsch. Qualität*	*Eval. Lösungen*
n=20	2.526,8307	2.692,6080	250.010
n=100	10.936,2655	12.851,5789	350.010
n=500	177.077,3837	178.461,9964	490.010
n=1000	382.864,5183	383.725,2190	585.010
n=2000	780.656,9792	789044,4974	700.010

Tabelle 4.30: Ergebnisse der (40/2,0.55)-OS-ES für die Schwefel-Funktion mit Rekombination durch zufällige Operatoren.

Dimension	*Beste Qualität*	*Durchsch. Qualität*	*Eval. Lösungen*
n=20	1.677,9229	2.069,6030	250.040
n=100	11.726,0132	14.122,8425	350.040
n=500	87.559,3650	134.989,3801	490.040
n=1000	225.935,3995	342.643,0206	585.040
n=2000	773.808,9329	780.626,6840	700.040

Wie auch bei einfacher Rekombination übertrifft die OS-ES die Lösungsqualität der ES in höheren Problem-Dimensionen. Die besseren Ergebnisse lieferte jedoch die OS-ES mit einfacher Rekombination, die generell besten Ergebnisse der Schwefel-Funktion in hohen Dimensionen ermittelte die Evolutionsstrategie ohne Rekombination und Offspring Selection.

Griewank-Funktion

Für die Griewank-Funktion kamen eine (40/2+100)-ES und eine (20/2+0.4)-OS-ES zum Einsatz:

Tabelle 4.31: Ergebnisse der (40/2+100)-ES für die Griewank-Funktion mit einem Multi-Operator-Crossover.

Dimension	*Beste Qualität*	*Durchsch. Qualität*	*Eval. Lösungen*
n=20	0,0000	0,0000	500.040
n=100	0,0000	$1,94 \cdot 10^{-2}$	700.040
n=500	$1,31 \cdot 10^{-1}$	$5,44 \cdot 10^{-1}$	980.040
n=1000	3,7904	4,8653	1.170.040
n=2000	85,4860	102,5100	1.400.040

Tabelle 4.32: Ergebnisse der (20/2+0.4)-OS-ES für die Griewank-Funktion mit einem Multi-Operator-Crossover.

Dimension	*Beste Qualität*	*Durchsch. Qualität*	*Eval. Lösungen*
n=20	0,0000	$8,87 \cdot 10^{-3}$	500.100
n=100	$1,15 \cdot 10^{-14}$	$1,08 \cdot 10^{-2}$	700.100
n=500	1,2657	1,6496	980.100
n=1000	24,3682	32,9565	1.170.100
n=2000	629,3811	889,9040	1.400.100

Erstaunlicherweise führt die multiple Rekombination bei der ES anhand der Griewank-Funktion zu den bisher besten Ergebnissen ohne Einsatz von Offspring Selection. Die durch Average-Crossover und Offspring Selection ermittelten Ergebnisse schlagen diese Werte jedoch noch.

Ackley-Funktion

Bei der Ackley-Funktion führt die multiple Rekombination wieder zu schlechteren Ergebnissen, wobei sich jedoch die (40/20,0.85)-OS-ES besser schlägt als die (20/2,100)-ES:

Tabelle 4.33: Ergebnisse der (20/2,100)-ES für die Ackley-Funktion unter Einsatz eines Multi-Operator-Crossovers.

Dimension	*Beste Qualität*	*Durchsch. Qualität*	*Eval. Lösungen*
n=20	$3,55 \cdot 10^{-15}$	$3,55 \cdot 10^{-15}$	500.020
n=100	$5,33 \cdot 10^{-14}$	$8,56 \cdot 10^{-1}$	700.020
n=500	14,6419	15,3532	980.020
n=1000	16,7860	17,0305	1.170.020
n=2000	16,8269	17,0872	1.400.020

Tabelle 4.34: Ergebnisse der (40/2,0.85)-OS-ES für die Ackley-Funktion unter Einsatz eines Multi-Operator-Crossovers.

Dimension	*Beste Qualität*	*Durchsch. Qualität*	*Eval. Lösungen*
n=20	$3,55 \cdot 10^{-15}$	$3,55 \cdot 10^{-15}$	500.040
n=100	$2,13 \cdot 10^{-14}$	$3,27 \cdot 10^{-14}$	700.040
n=500	$4,00 \cdot 10^{-1}$	$7,09 \cdot 10^{-1}$	980.040
n=1000	1,2509	1,4274	1.170.040
n=2000	3,7322	3,8197	1.400.040

Die insgesamt besten Ergebnisse für die Ackley-Funktion konnten durch die OS-ES mit Rekombination durch einen Average-Crossover ermittelt werden.

TSP und CVRP

Auch für die kombinatorischen Optimierungsprobleme wurde der Einsatz multipler Rekombinations-Operatoren getestet. Die Optimierung der TSP-Probleme wurde anhand einer (10/2+100)-ES und einer (10/2+0.2)-OS-ES durchgeführt. Das Capacitated Vehicle Routing Problem löste eine (10/2,100)-ES und eine (20/2,0.7)-OS-ES:

Tabelle 4.35: Ergebnisse der ES für die TSP-Probleme und das CVRP-Problem mit verschiedenen Rekombinations-Operatoren.

Problem	*Beste Qualität*	*Durchsch. Qualität*	*Eval. Lösungen*
TSP ch130	608,0000	853,8000	2.500.010
TSP kroA200	0,0000	986,4000	2.500.010
CVRP e-n101-k14	0,0000	12,4000	2.500.010

Tabelle 4.36: Ergebnisse der OS-ES für die TSP-Probleme und das CVRP-Problem mit verschiedenen Rekombinations-Operatoren.

Dimension	*Beste Qualität*	*Durchsch. Qualität*	*Eval. Lösungen*
TSP ch130	877,0000	1021,8000	2.500.030
TSP kroA200	0,0000	282,0000	2.500.030
CVRP e-n101-k14	0,0000	0,6000	2.500.030

Ohne Offspring Selection erwies sich die multiple Rekombination für die TSP-Probleme als weniger vorteilhaft. Das CVRP-Problem konnte jedoch durchschnittlich besser gelöst werden. Die Offspring Selection konnte die Qualität für das kroA200 und das CVRP-Problem dann noch weiter verbessern. Das TSP-Problem ch130 wurde durch eine ES mit ERX-Rekombination am besten gelöst. Die besten Lösungen für das TSP-Problem kroA200, sowie

für das CVRP-Problem, konnten durch die OS-ES mit multiplen Crossover-Operatoren erzielt werden.

4.2.4 Weitere Experimente

Abgesehen von der Ackley-Funktion wurden alle erfolgreichen Tests der Benchmark-Funktionen durch eine OS-ES mit Plus-Selektion erzielt. Einige kurze Experimente zeigten, dass durch den Einsatz der Plus-Selektion für die Rastrigin-Funktion eventuell bessere Lösungen erreicht werden können. Die besten Resultate konnten bisher für die Griewank- und die Ackley-Funktion erzielt werden, wobei die Ackley-Funktion nicht ganz so stark von der Offspring Selection profitierte. Eventuell könnte eine Plus-Selektion wie bei der Griewank-Funktion auch für die Ackley-Funktion vorteilhaft sein. Für die Schwefel-Funktion erwies sich die Verwendung der Plus-Selektion als weniger passend.

Plus-Selektion für die Rastrigin- und Ackley-Funktion

Im folgenden sind die Ergebnisse der Rastrigin- und Ackley-Funktion unter Einsatz einer Plus-Selektion bei gleicher Anzahl an evaluierten Lösungen angeführt. Die Rastrigin-Funktion wurde dabei von einer (40/2+100)-ES und einer (40/2+0.2)-OS-ES unter Einsatz eines Average-Crossovers optimiert:

Tabelle 4.37: Ergebnisse der (40/2+100)-ES für die Rastrigin-Funktion mit Rekombination durch einen Average-Crossover.

Dimension	*Beste Qualität*	*Durchsch. Qualität*	*Eval. Lösungen*
n=20	4,9748	7,5617	500.040
n=100	69,6471	592,8804	700.040
n=500	4.677,2092	4.704,5858	980.040
n=1000	9.708,4941	9.821,3278	1.170.040
n=2000	20.171,5191	20.388,1573	1.400.040

Tabelle 4.38: Ergebnisse der (40/2+0.2)-OS-ES für die Rastrigin-Funktion mit Rekombination durch einen Average-Crossover.

Dimension	*Beste Qualität*	*Durchsch. Qualität*	*Eval. Lösungen*
n=20	3,9798	6,9647	500.040
n=100	739,7991	753,1174	700.040
n=500	4.712,1159	4.734,6611	980.040
n=1000	9.934,0729	9.967,7666	1.170.040
n=2000	20.019,1209	20.430,5434	1.400.040

Die Ergebnisse zeigen, dass die Plus-Selektion in beiden Algorithmus-Varianten eine bessere Qualität erreicht als die zuvor eingesetzte Komma-

Selektion. Durch die Offspring Selection können jedoch auch hier, mit Ausnahme der Dimension $n = 20$, keine besseren Ergebnisse erzielt werden. Die OS-ES ist demnach zur Optimierung der Rastrigin-Funktion nicht gut geeignet. Für die Ackley-Funktion wurde eine (40/2+100)-ES und eine (40/2+0.4)-OS-ES gewählt:

Tabelle 4.39: Ergebnisse der (40/2+100)-ES für die Ackley-Funktion unter Einsatz eines Average-Crossovers.

Dimension	*Beste Qualität*	*Durchsch. Qualität*	*Eval. Lösungen*
n=20	$3,55 \cdot 10^{-15}$	$3,55 \cdot 10^{-15}$	500.040
n=100	$3,20 \cdot 10^{-14}$	$6,96 \cdot 10^{-14}$	700.040
n=500	1,1672	1,2831	980.040
n=1000	1,7474	1,9262	1.170.040
n=2000	2,5772	2,6376	1.400.040

Tabelle 4.40: Ergebnisse der (40/2+0.4)-OS-ES für die Ackley-Funktion unter Einsatz eines Average-Crossovers.

Dimension	*Beste Qualität*	*Durchsch. Qualität*	*Eval. Lösungen*
n=20	$3,55 \cdot 10^{-15}$	$3,55 \cdot 10^{-15}$	500.040
n=100	$2,84 \cdot 10^{-14}$	$5,90 \cdot 10^{-14}$	700.040
n=500	0,9853	1,1291	980.040
n=1000	1,3452	1,5299	1.170.040
n=2000	1,9785	2,0073	1.400.040

Die Plus-Selektion stellt für die ES ohne Offspring Selection eine Verbesserung der Ergebnisse dar. Die Resultate der OS-ES übertreffen diese dann noch. Die generell besten Ergebnisse für die Ackley-Funktion wurden jedoch durch die OS-ES mit Komma-Selektion ermittelt. In Anbetracht aller Experimente ist die Wahl eines passenden Selektions-Mechanismus zwar sehr wichtig, ob eine Verbesserung der Lösungsqualität durch Offspring Selection möglich ist, hängt aber eher von dem zu lösenden Optimierungsproblem ab.

Nähere Betrachtung der Schwefel-Funktion

Der Einsatz von Offspring Selection konnte die Resultate der ES für die Schwefel-Funktion bisher in keiner der verschiedenen Dimensionsgrößen übertreffen. In kleinen Dimensionen erwies sich die ES mit Rekombination als effektiv, höhere Problemdimensionen wurden von der ES ohne Rekombination am besten gelöst. Für die Dimensionsgrößen $n = 500$ und $n = 1000$ konnte jedoch die OS-ES mit Rekombination manchmal innerhalb der Ausführungszeit ein lokales Optimum finden, das eine bessere Qualität aufweist. Eine Analyse des Selektionsdruckes zeigt, dass in diesen Dimensionen spe-

ziell auch durch einen Einsatz von multiplen Rekombinations-Operatoren bei allen Durchläufen noch Verbesserungspotential durch eine längere Laufzeit besteht. Während die ES bereits bei einem Optimum stagniert, schafft die OS-ES auch in späten Generationen noch einen weiteren Fortschritt. Die folgenden Experimente widmen sich dem Test der OS-ES mit multiplen Rekombinations-Operatoren für die Schwefel-Funktion anhand einer größeren Menge an evaluierten Lösungen im Vergleich zur Standard-ES ohne Rekombination. Dabei kamen die selben Algorithmus-Parameter wie bei den ersten Tests zum Einsatz:

Tabelle 4.41: Ergebnisse der (20,100)-ES für die Schwefel-Funktion.

Dimension	*Beste Qualität*	*Durchsch. Qualität*	*Eval. Lösungen*
n=20	1.756,8571	2.736,8209	1.000.020
n=100	16.014,8354	17.256,1606	1.400.020
n=500	103.321,8152	108.302,8960	1.960.020
n=1000	242.890,0731	245.717,8423	2.340.020
n=2000	562.426,8207	564.105,1532	2.800.020

Tabelle 4.42: Ergebnisse der (40/2,0.55)-OS-ES für die Schwefel-Funktion mit Rekombination durch zufällige Operatoren.

Dimension	*Beste Qualität*	*Durchsch. Qualität*	*Eval. Lösungen*
n=20	1.500,2461	1.863,4729	1.000.040
n=100	14.292,9804	14.889,0442	1.400.040
n=500	78.236,2711	83.554,9278	1.960.040
n=1000	173.115,7425	216.981,5297	2.340.040
n=2000	651.584,7558	717.980,3997	2.800.040

Die durch eine längere Laufzeit ermittelten Lösungen für die ES sind dabei nur unwesentlich besser, da der Algorithmus bereits zuvor schon zu einem lokalen Optimum konvergierte und nur noch schwer bessere Lösungen erreicht. Für die OS-ES zeigt sich durch die längere Laufzeit eine stärkere Verbesserung der Qualität. In Problemdimension $n = 500$ und $n = 1000$ schafft es der Algorithmus nun zu einem lokalen Optimum zu konvergieren, das sogar die bisher beste Lösung der Standard-ES ohne Rekombination übertrifft. Für die Dimensionsgröße $n = 2000$ waren die ca. 2.800.000 evaluierten Lösungen noch nicht ausreichend, um ein solches lokales Optimum zu finden. Es ist jedoch anzunehmen, dass durch eine noch längere Laufzeit hier ebenso eine bessere Lösung gefunden werden kann. Für kleinere Problemdimensionen erreicht nach wie vor die ES mit diskreter Rekombination die besten Resultate.

4.2.5 Evaluation der Ergebnisse

Die Offspring Selection ermöglicht in vielen Fällen eine Verbesserung der erreichten Lösungsqualität im Gegensatz zur Evolutionsstrategie ohne selbstadaptiver Steuerung des Selektionsdruckes, vor allem unter Einsatz von Rekombination. Ohne Rekombination führt die Offspring Selection meist zu einem langsameren Konvergenzverhalten und kann speziell in höheren Problem-Dimensionen die Ergebnisse der ES nicht übertreffen. In kleinen Problemdimensionen kann die OS-ES auch ohne Rekombination eher mithalten oder teilweise sogar etwas bessere Qualitäten erreichen. Dies ist z. B. bei der Rosenbrock oder Rastrigin-Funktion für die Dimensionsgröße $n = 20$ der Fall.

Die Einführung von Rekombination ermöglicht, mit Ausnahme der Schwefel-Funktion, durchwegs bessere Ergebnisse bei einer gleichen Anzahl an Lösungs-Evaluierungen. In hohen Dimensionen der Schwefel-Funktion erreichte die Evolutionsstrategie ohne Rekombination bessere Ergebnisse. Bei sämtlichen anderen Testfällen verbessert die Rekombination die Lösungsqualität, wobei sich nun auch in hohen Problemdimensionen die Integration von Offspring Selection positiv auswirkt und ähnliche oder sogar bessere Ergebnisse produziert. Für die Rosenbrock, die Griewank und die Ackley-Funktion können in nahezu allen Dimensionen bessere Resultate erzielt werden. Speziell die Optimierung der Griewank-Funktion profitiert sehr stark von der Kombination aus Offspring Selektion und Rekombination, denn sogar bei 2000 Dimensionen wird eine Qualität kleiner als Eins erreicht. Für die Ackley-Funktion ist dies bis zu einer Dimensionsgröße von 1000 möglich. Auch für die kombinatorischen Probleme kann die Rekombination sinnvoll sein, vor allem wenn auch Offspring Selection zum Einsatz kommt. Sämtliche kombinatorischen Probleme lassen sich bei Offspring Selection mit Rekombination besser lösen als ohne Rekombination. Bei der normalen Evolutionsstrategie hat die Rekombination für die Probleme kroA200 und E-n101-k14 eine negative Auswirkung auf die durchschnittliche Qualität, die Offspring Selection hebt diesen negativen Effekt wieder etwas auf. In Summe erreicht jedoch die Evolutionsstrategie ohne Offspring Selection leicht bessere Ergebnisse für diese beiden Probleme.

Der Einsatz von multiplen Crossover-Operatoren anstelle eines einzigen führte in den meisten Fällen zu schlechteren Ergebnissen, wobei trotzdem die OS-ES äquivalent zu den Ergebnissen bei einfacher Rekombination etwas bessere Resultate als die ES erzielt. Eine Ausnahme bildet die Schwefel-Funktion, da hier durch multiple Operatoren zumindest bei einer längeren Ausführungszeit in höheren Dimensionen die besten Ergebnisse erzielt werden können. Für die kombinatorischen Probleme erwies sich eine Rekombination mit verschiedenen Operatoren für die Probleme kroA200 und E-n101-k14 als sinnvoll. Die beste Durchschnitts-Qualität konnte für diese Probleme durch die OS-ES unter Einsatz multipler Operatoren erzielt

werden. Das TSP-Problem ch130 wurde am besten von der normalen Evolutionsstrategie mit ERX-Rekombination gelöst.

Es kann zusammengefasst werden, dass in vielen Fällen die Kombination von Rekombination und Offspring Selection zu sehr guten Ergebnissen führt, wobei jedoch eventuell eine etwas höhere Anzahl an Lösungs-Evaluierungen nötig ist. Für manche Probleme kann auch der Einsatz von multiplen Crossover-Operatoren sinnvoll sein. Beispielsweise konnten durch die richtige Parameter- und Operator-Wahl für die Probleme Rosenbrock, Griewank, Ackley, TSP kroA200 und CVRP e-n101-k14 über den Einsatz der selbstadaptiven Selektionsdruck-Steuerung die in Summe besten Resultate erzielt werden. Auch für die Schwefel-Funktion konnte die erreichte Lösungsqualität in höheren Dimensionen verbessert werden, wobei dies aber eine größere Anzahl an Lösungs-Evaluierungen benötigte. Ohne Rekombination ist der Einsatz von Offspring Selection eher weniger sinnvoll und erzielt bei einer langsameren Konvergenz-Geschwindigkeit im besten Fall ähnlich gute bis nur leicht bessere Ergebnisse. Ein Unterschied zwischen der Komma- und der Plus-Variante der Evolutionsstrategie unter Verwendung der Offspring Selection konnte nicht erkannt werden. Sofern die Offspring Selection mit einem bestimmten Crossover-Operator für das gewählte Problem sinnvoll ist, gilt dies für beide Selektions-Verfahren. Eine passende Parameterwahl ist jedoch für die Komma-Variante etwas leichter, da sich hier ein Wert von $SuccessRatio \approx 0.7$ in vielen Fällen als sinnvoll erwies.

4.3 Untersuchung der OS-CMA-ES

4.3.1 Aufbau der Tests

Auch die Experimente zur Untersuchung der CMA-ES mit Offspring Selection wurden unter Einsatz der Implementierung in HeuristicLab durchgeführt. Da die CMA-ES jedoch eine spezielle Form der Evolutionsstrategie darstellt, die Mutation über den Mittelwert der letzten Population und einer Kovarianzmatrix realisiert, können damit nicht sämtliche Probleme bearbeitet werden. Es ist nicht möglich, für kombinatorische Optimierungsprobleme einen Mittelwert und eine Kovarianzmatrix zu ermitteln, weshalb diese Algorithmus-Variante nur auf den zuvor vorgestellten reellwertigen Benchmark-Zielfunktionen getestet wurde. Weiters wurden diese Probleme nicht in sehr hohen Dimensionen bearbeitet, da bei hoher Dimensionsgröße die am Ende jeder Generation durchgeführte Adaption der Kovarianzmatrix und die Anpassung der Strategieparameter sehr rechen- und zeitintensiv sind. Selbst bei einer Dimension von $n = 500$ und nur wenigen Generationen kann die Optimierung bereits mehrere Stunden in Anspruch nehmen, weshalb hier nur sehr wenige evaluierte Lösungen als Grenze gesetzt wurden. Eine Optimierung mit höheren Dimensionsgrößen wurde nicht durchgeführt. Die Wahl einer vorteilhaften Populationsgröße, sowie der eingesetzte

Rekombinations-Operator wurden wieder anhand einiger kurzer Testläufe ausgewählt.

4.3.2 Ergebnisse

Rosenbrock-Funktion

Die Rosenbrock-Funktion wurde anhand einer CMA-ES mit $\mu = 10$ und einer logarithmisch-gewichteten Rekombination optimiert. Tabelle 4.43 zeigt die Resultate der CMA-ES ohne Offspring Selection. Für die OS-Variante kam eine SuccessRatio von 0,1 zum Einsatz (Tabelle 4.44).

Tabelle 4.43: Ergebnisse der CMA-ES für die Rosenbrock-Funktion.

Dimension	*Beste Qualität*	*Durchsch. Qualität*	*Eval. Lösungen*
n=20	$2,13 \cdot 10^{-28}$	$2,98 \cdot 10^{-28}$	500.000
n=100	$1,07 \cdot 10^{-26}$	$7,97 \cdot 10^{-1}$	700.000
n=500	495,8189	533,0588	25.000

Tabelle 4.44: Ergebnisse der OS-CMA-ES für die Rosenbrock-Funktion.

Dimension	*Beste Qualität*	*Durchsch. Qualität*	*Eval. Lösungen*
n=20	88,1206	107,2130	500.000
n=100	2262,3838	2973,7259	700.000
n=500	184.284,2441	201.355,9702	25.000

Die Ergebnisse der CMA-ES mit Offspring Selection sind sehr viel schlechter als die der normalen CMA-ES. Der Grund dafür kann durch einen Vergleich der Qualitätskurven in Abb. 4.4 und Abb. 4.5 ermittelt werden. Die CMA-ES schafft es bereits in den ersten 250 Generationen (ca. 2500 evaluierte Lösungen) die Qualität enorm zu verbessern und erreicht dann nach ca. 450.000 evaluierten Lösungen Werte nahe bei Null. Die OS-CMA-ES berechnet insgesamt nur zehn Generationen bis die Grenze der evaluierten Lösungen erreicht ist. Ein Blick auf den Verlauf des Selektionsdruckes zeigt, dass die *SuccessRatio* in den ersten sieben Generationen sehr leicht erreicht werden konnte, danach jedoch immer schwieriger. In der neunten Generation mussten bereits über 50.000 Lösungen erzeugt werden, um nur eine erfolgreiche Lösung zu generieren. In der zehnten Generation wurde die Berechnung nach über 600.000 evaluierten Lösungen schließlich abgebrochen, da das Limit erreicht wurde. Der Algorithmus befand sich also an einem Punkt, an dem keine besseren Lösungen anhand des Mittelpunktes und der Kovarianzmatrix mehr erreicht werden konnten. Durch ein Anpassen der Kovarianzmatrix und der Strategieparameter würde die CMA-ES stärker explorativ weitersuchen. Die Offspring Selection verhindert jedoch

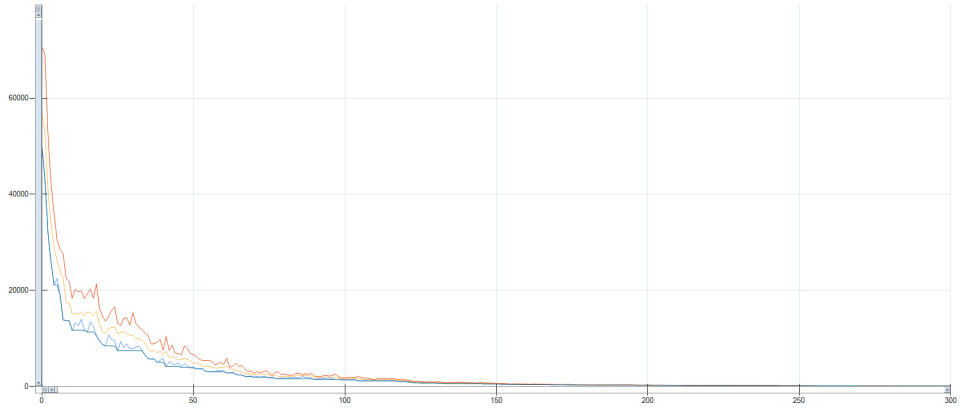

Abbildung 4.4: Qualitätsverlauf der CMA-ES für die 100-dimensionale Rosenbrock-Funktion, der Auschnitt zeigt die Generationen 0 bis 300.

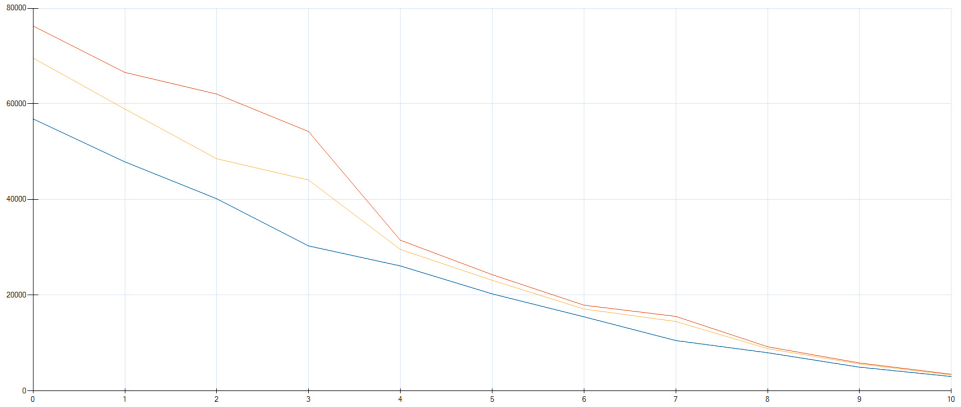

Abbildung 4.5: Qualitätsverlauf der OS-CMA-ES für die 100-dimensionale Rosenbrock-Funktion.

die Anpassung der Strategieparameter und der Kovarianzmatrix bevor die *SuccessRatio* erfüllt ist. Die Folge davon ist ein Stagnieren des Algorithmus. Dieses Verhalten betrifft im speziellen die Rosenbrock-Funktion. Doch auch bei den weiteren Testfunktionen ist zu beobachten, dass der Algorithmus durch die Offspring Selection eher ausgebremst wird und schlechtere Ergebnisse liefert.

Rastrigin-, Schwefel-, Griewank- und Ackley-Funktion

Die Rastrigin-Funktion wurde anhand einer CMA-ES mit $\mu = 20$ und linear-gewichteter Rekombination, sowie einer OS-CMA-ES mit $\mu = 20$, gleichmäßig-gewichteter Rekombination und $SuccessRatio = 0.55$ getestet:

Tabelle 4.45: Ergebnisse der CMA-ES für die Rastrigin-Funktion.

Dimension	*Beste Qualität*	*Durchsch. Qualität*	*Eval. Lösungen*
n=20	5,9698	10,7456	500.000
n=100	71,6370	101,4857	700.000
n=500	4.433,6512	5.375,5164	25.000

Tabelle 4.46: Ergebnisse der OS-CMA-ES für die Rastrigin-Funktion.

Dimension	*Beste Qualität*	*Durchsch. Qualität*	*Eval. Lösungen*
n=20	39,3024	58,4043	500.000
n=100	744,4716	879,4295	700.000
n=500	7.681,0548	7.812,1940	25.000

Bei der Schwefel-Funktion kamen eine CMA-ES mit einer Populationsgröße von $\mu = 40$ und logarithmisch-gewichteter Rekombination, sowie eine $(10, 0.2)$-OS-CMA-ES mit linear-gewichteter Rekombination zum Einsatz:

Tabelle 4.47: Ergebnisse der CMA-ES für die Schwefel-Funktion.

Dimension	*Beste Qualität*	*Durchsch. Qualität*	*Eval. Lösungen*
n=20	2.844,1108	3.471,9115	500.000
n=100	17.449,2758	18.6666,1292	700.000
n=500	98.426,7301	99.756, 5135	25.000

Tabelle 4.48: Ergebnisse der OS-CMA-ES für die Schwefel-Funktion.

Dimension	*Beste Qualität*	*Durchsch. Qualität*	*Eval. Lösungen*
n=20	3.377, 3700	3.909,8486	500.000
n=100	20.001,8512	20.800,8918	700.000
n=500	114.344,4612	119.081,6507	25.000

Die Ergebnisse der Griewank-Funktion wurden durch eine CMA-ES mit $\mu = 40$ und linear-gewichteter Rekombination erzielt. Für die Variante mit Offspring Selection wurden die Parameter $\mu = 10$ und $SuccessRatio = 0.2$ bei linear-gewichteter Rekombination gewählt:

Tabelle 4.49: Ergebnisse der CMA-ES für die Griewank-Funktion.

Dimension	*Beste Qualität*	*Durchsch. Qualität*	*Eval. Lösungen*
n=20	0,0000	0,0000	300.000
n=100	0,0000	0,0000	700.000
n=500	17,1531	21,8040	25.000

Tabelle 4.50: Ergebnisse der OS-CMA-ES für die Griewank-Funktion.

Dimension	*Beste Qualität*	*Durchsch. Qualität*	*Eval. Lösungen*
n=20	8,5718	12,5779	500.000
n=100	82,3140	85,0914	700.000
n=500	960,4716	1.045,8641	25.000

Die Ackley-Funktion wurde jeweils mit einer Populationsgröße von 40 und gleichmäßig-gewichteter Rekombination gelöst. Die OS-CMA-ES wurde mit $SuccessRatio = 0.2$ betrieben:

Tabelle 4.51: Ergebnisse der CMA-ES für die Ackley-Funktion.

Dimension	*Beste Qualität*	*Durchsch. Qualität*	*Eval. Lösungen*
n=20	$3,55 \cdot 10^{-15}$	7,7816	300.000
n=100	$1,42 \cdot 10^{-14}$	11,7070	700.000
n=500	17,1531	21,8040	25.000

Tabelle 4.52: Ergebnisse der OS-CMA-ES für die Ackley-Funktion.

Dimension	*Beste Qualität*	*Durchsch. Qualität*	*Eval. Lösungen*
n=20	15,3557	19,1058	500.000
n=100	15,9346	18,9712	700.000
n=500	21,0900	21,1170	25.000

Sämtliche Testfunktion zeigen bessere Ergebnisse ohne den Einsatz von Offspring Selection. Diese sind wahrscheinlich durch eine bessere Anpassung des Algorithmus an den aktuellen Suchraum durch die Adaption der Kovarianzmatrix und der Strategieparameter am Ende jeder Generation zu erklären. Vor allem wenn nur noch schwer bessere Ergebnisse erzielt werden können, ist die Änderung zu einem eher explorativen Suchverhalten durch die Anpassung der Kovarianzmatrix an die aktuellen Gegebenheiten sehr wichtig. Die Offspring Selection verhindert diese Anpassung jedoch.

4.3.3 Evaluation der OS-CMA-ES

Die Offspring Selection in der CMA-Evolutionsstrategie erscheint anhand der durchgeführten Tests als nicht sinnvoll. Sie bremst den Fortschritt der Evolutionsstrategie eher aus, oder führt im schlimmsten Fall sogar zu einer Stagnation des Algorithmus:

- Sofern relativ leicht gute Lösungen erreicht werden können, ist es sehr einfach die *SuccessRatio* zu erfüllen. Falls die Kovarianzmatrix und Strategieparameter so gewählt sind, dass der Algorithmus auf ein Optimum hinarbeitet, ist nahezu kein zusätzlicher Aufwand nötig, um genug erfolgreiche Kinder zu erzeugen. In diesem Fall konvergiert das Verfahren ähnlich schnell zur Variante ohne Offspring Selection.
- Sobald es schwieriger wird bessere Kinder zu erzeugen, führt die Anpassung der Kovarianzmatrix innerhalb weniger Generationen zu passenden Werten, die ermöglichen anschließend bessere Lösungen zu finden oder aus einem lokalen Optimum zu gelangen, um danach wieder eher zielgerichtet zu arbeiten. Die Offspring Selection verhindert dies, da sie in diesem Fall die weitere Erzeugung von Kindern erzwingt, bis dass die SuccessRatio erreicht ist. Bei einem lokalen Optimum ist es unwahrscheinlich, dass die zuvor stark zielgerichtete Kovarianzmatrix zufällig aus diesem herausfindet. Wenn beispielsweise zwanzig Iterationen der Offspring Selection nötig sind, um die SuccessRatio zu erfüllen, hätte die normale CMA-ES bereits zwanzig Generationen berechnet, in denen wieder stärker explorativ gesucht wurde, und eventuell bereits weiterer Fortschritt erzielt werden konnte. Im besten Fall kann die OS-CMA-ES nach wenigen Iterationen wieder die Kovarianzmatrix anpassen und weitersuchen, im schlimmsten Fall stagniert der Algorithmus bei dem lokalen Optimum.

Durch dieses Verhalten ist es der OS-CMA-ES nicht möglich, die normale CMA-ES zu übertreffen. Je mehr zusätzliche Lösungen die Offspring-Selection benötigt, desto mehr Zeit geht verloren, in der das Verfahren durch die Anpassung der Matrix schneller vorankommen würde. Eine Integration der Anpassung der Kovarianzmatrix in den Zyklus der Offspring Selection würde im Endeffekt die Offspring Selection ausschalten. Denn das Ziel der Offspring Selektion ist es, die Beschaffenheit der Population zu beeinflussen, bevor solche Anpassungen durchgeführt werden. Es muss trotzdem festgehalten werden, dass die Ergebnisse der CMA-ES ohne Offspring Selection teilweise sehr gut sind und die Resultate der ES übertreffen. Dies gilt für die Rosenbrock, die Rastrigin und die Griewank-Funktion in niedrigen Dimensionen. Höhere Dimensionen erfordern für die CMA-ES durch die Adaption der Kovarianzmatrix sehr zeitintensive Berechnungen und können über die OS-ES besser gelöst werden. Ein weiterer Nachteil ist, dass die CMA-ES nur auf reellwertige Probleme angewandt werden kann.

Kapitel 5

Zusammenfassung

Diese Arbeit widmete sich nach einer umfassenden Einführung, welche Evolutionäre Algorithmen, Evolutionsstrategien und Offspring Selection behandelte, der Implementierung einer selbstadaptiven Selektionsdrucksteuerung für die Evolutionsstrategie. Durch eine Integration des Konzeptes der Offspring Selection wurden zwei neue Algorithmen, die $(\mu^{+}_{,}SuccessRatio)$-OS-ES und die OS-CMA-ES, entwickelt und in Form von HeuristicLab-Plugins implementiert. Anhand einiger Tests konnten anschließend das Verhalten und die erzielte Lösungsqualität dieser Algorithmen untersucht werden. Dabei wurden die reellwertigen Benchmark-Funktionen Rosenbrock, Rastrigin, Griewank und Ackley in verschiedenen Problemdimensionen bearbeitet. Die OS-ES wurde zusätzlich noch zur Lösung des CVRP-Problems E-n101-k14 und der TSP-Probleme ch130 und kroA200 eingesetzt. Für die CMA-ES erwies sich die Offspring Selection als eher ungünstig, da diese einer regelmäßigen Anpassung der Kovarianzmatrix und der Strategieparameter im Wege steht. Die Folge ist eine langsamere Konvergenz-Geschwindigkeit oder im schlimmsten Fall sogar ein Stagnieren des Algorithmus. Die OS-ES kann jedoch vor allem unter Einbezug von Rekombination in vielen Fällen bessere Ergebnisse erzielen, wobei aber eventuell eine höhere Anzahl an Lösungs-Evaluierungen notwendig ist. Die Griewank- und die Ackley-Funktion profitierten am meisten von der selbstadaptiven Selektionsdruck-Steuerung, jedoch konnten ebenso für die Funktionen Rosenbrock und Schwefel teilweise bessere Resultate erzielt werden. Auch für die Probleme kroA200 und E-n101-k14 wurde eine bessere Durchschnitts-Qualität aller Testläufe erreicht. Weiteres Potential besteht hauptsächlich durch eine mögliche Weiterentwicklung der OS-ES in einer Form, die sich die Information des aktuell bestehenden Selektionsdruckes zunutze macht. Beispielsweise könnten die Strategieparameter am Ende jeder Generation anhand des Selektionsdruckes dynamisch angepasst werden. Die Anwendung einer selbstadaptiven Selektionsdrucksteuerung in Evolutionsstrategien mit mehreren parallelen Populationen könnte ebenso noch interessant sein.

Quellenverzeichnis

Literatur

[ABC07] David L. Applegate, Robert E. Bixby und William J. Cook. *The Traveling Salesman Problem: A Computational Study*. Princeton Univ Pr, 2007 (siehe S. 5).

[Ack87] David H. Ackley. „A Connectionist Machine for Genetic Hillclimbing“. In: *The Springer International Series in Engineering and Computer Science* 28 (1987) (siehe S. 65).

[AD04] Enrique Alba und Bernabé Dorronsoro. „Solving the Vehicle Routing Problem by Using Cellular Genetic Algorithms“. In: *Evolutionary Computation in Combinatorial Optimization, Lecture Notes in Computer Science*. (EvoCOP 2004). Hrsg. von J. Gottlieb und G. R. Raidl. Bd. 3004. Coimbra, Portugal: Springer, 2004, S. 11–20 (siehe S. 74).

[Aff01] Michael Affenzeller. „Transferring the concept of selective pressure from from evolutionary strategies to genetic algorithms“. In: *Proceedings of the of 14th International Conference on System Science*. Bd. 2. 2001, S. 346–353 (siehe S. 35).

[Aff05] Michael Affenzeller. *Population Genetics and Evolutionary Computation: Theoretical and Practical Aspects*. Trauner Verlag, 2005 (siehe S. 37, 47).

[Ahm07] Zakir H. Ahmed. „Genetic Algorithm for the Traveling Salesman Problem using Sequential Constructive Crossover Operator“. In: *International Journal of Biometrics & Bioinformatics (IJBB)* 3.6 (2007), S. 96–105 (siehe S. 10).

[Bäc96] Thomas Bäck. *Evolutionary Algorithms in Theory and Practice*. Oxford Univ Pr, 1996 (siehe S. 8, 19, 65).

[BFK13] Thomas Bäck, Christophe Foussette und Peter Krause. *Contemporary Evolution Strategies*. Springer, 2013 (siehe S. 22, 28).

[BS02] Hans-Georg Beyer und Hans-Paul Schwefel. „Evolution strategies – A comprehensive introduction“. In: *Natural Computing* 1.1 (2002), S. 3–52 (siehe S. 12, 17).

[CE69] N. Christofides und S. Eilon. „An Algorithm for the Vehicle-dispatching Problem“. In: *Operational Research Quarterly* 20.3 (Sep. 1969) (siehe S. 65).

[Čer85] V. Černý. „Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm“. In: *Journal of Optimization Theory and Applications* 45.1 (Jan. 1985), S. 41–51 (siehe S. 36).

[Coo11] William J. Cook. *In Pursuit of the Traveling Salesman: Mathematics at the Limits of Computation*. Princeton University Press, 2011 (siehe S. 5).

[Dar06] Charles Darwin. *On the Origin of Species by Means of Natural Selection*. Dover Pubn Inc, 2006 (siehe S. 3).

[ECS89] Larry J. Eshelman, Richard A. Caruana und J. David Schaffer. „Biases in the crossover landscape“. In: *Proceedings of the third international conference on Genetic algorithms*. (George Mason University, Virginia). Hrsg. von J. David Schaffer. San Francisco: Morgan Kaufmann Publishers Inc., Juni 1989, S. 10–19 (siehe S. 15).

[ES98] Ágoston E. Eiben und C. A. Schippers. „On Evolutionary Exploration and Exploitation“. In: *Fundamenta Informaticae* 35.1-4 (1998), S. 35–50 (siehe S. 15).

[Fog66] Lawrence J. Fogel. *Artificial Intelligence through Simulated Evolution*. John Wiley & Sons, 1966 (siehe S. 15).

[GKK13] Ingrid Gerdes, Frank Klawonn und Rudolf Kruse. *Evolutionäre Algorithmen: Genetische Algorithmen - Strategien Und Optimierungsverfahren - Beispielanwendungen*. 2004. Aufl. Springer, 2013 (siehe S. 1, 8, 12).

[Gol89] David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison Wesley Pub Co Inc, 1989 (siehe S. 9, 32).

[Gri81] A. O. Griewank. „Generalized descent for global optimization“. In: *Journal of Optimization Theory and Applications* 34 (Mai 1981), S. 11–39 (siehe S. 64).

[HAA15] Nikolaus Hansen, Dirk V. Arnold und Anne Auger. „Evolution Strategies, Version April 2013“. In: *Handbook of Computational Intelligence*. Hrsg. von Janusz Kacprzyk und Witold Pedrycz. Springer, Veröffentlichung geplant in 2015 (siehe S. 28).

[Han07] Nikolaus Hansen. *The CMA Evolution Strategy: A Tutorial*. Techn. Ber. Aug. 2007. URL: https://www.lri.fr/~hansen/cmatutorial.pdf (siehe S. 26, 28).

[Han98] Nikolaus Hansen. *Verallgemeinerte individuelle Schrittweitenregelung in der Evolutionsstrategie: Eine Untersuchung zur entstochastisierten, koordinatensystemunabhängigen Adaptation der Mutationsverteilung*. Mensch & Buch, 1998 (siehe S. 21, 24, 28).

[HE06] Joachim Hartung und Bärbel Elpelt. *Multivariate Statistik*. 7. Aufl. Oldenbourg Wissenschaftsverlag, 2006 (siehe S. 21).

[HOG95] Nikolaus Hansen, Andreas Ostermeier und Andreas Gawelczyk. „On the adaptation of arbitrary normal mutation distributions in evolution strategies: The generating set adaptation“. In: *Proceedings of the of 6th International Conference on Genetic Algorithms*. 1995, S. 57–64 (siehe S. 17).

[Hol75] John H. Holland. *Adaptation in Natural and Artificial Systems*. The Mit Press, 1975 (siehe S. 12, 15, 32).

[Hol92a] John H. Holland. In: *Spektrum der Wissenschaft* (Sep. 1992), S. 44–51 (siehe S. 3).

[Hol92b] John H. Holland. „Genetic Algorithms“. In: *Scientific American* 267.1 (1992) (siehe S. 9, 15).

[JJ03] Konrad Jacobs und Dieter Jungnickel. *Einführung in die Kombinatorik*. 2. Aufl. Gruyter, 2003 (siehe S. 5).

[KGV83] S. Kirkpatrick, C. D. Gelatt und M. P. Vecchi. „Optimization by Simulated Annealing“. In: *Science* 220.4598 (Mai 1983), S. 671–680. URL: http://www.jstor.org/discover/10.2307/1690046 (siehe S. 36).

[KKK12] Naven Kumar, Karambir und Rajiv Kumar. „A Comparative Analysis of PMX, CX and OX Crosover operators for solving Traveling Salesman Problem“. In: *International Journal of Latest Research in Science and Technology* 1.2 (2012), S. 98–101 (siehe S. 10).

[Koz93] John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Mit Pr, 1993 (siehe S. 11, 15).

[LP02] William B. Langdon und Riccardo Poli. *Foundations of Genetic Programming*. first edition. Springer, 2002 (siehe S. 12).

[MB07] Silja Meyer-Nieberg und Hans-Georg Beyer. „Self-adaptation in evolutionary algorithms“. In: *Parameter Setting in Evolutionary Algorithms, Studies in Computational Intelligence*. Hrsg. von Fernando G. Lobo, Cláudio F. Lima und Zbigniew Michalewicz. Bd. 54. Berlin: Springer-Verlag Gmbh, 2007 (siehe S. 17).

[Mic98] Zbigniew Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer, 1998 (siehe S. 10, 34).

[MSB91] H. Mühlenbein, D. Schomisch und J. Born. „The Parallel Genetic Algorithm as Function Optimizer“. In: *Parallel Computing* 17.6-7 (1991), S. 619–632 (siehe S. 64).

[Nis97] Volker Nissen. *Einführung in Evolutionäre Algorithmen*. Vieweg+Teubner Verlag, 1997 (siehe S. 1, 8, 10, 15, 19, 23, 30).

[OGH94] Andreas Ostermeier, Andreas Gawelczyk und Nikolaus Hansen. „A derandomized approach to self-adaptation of evolution strategies“. In: *Evolutionary Computatio* 2 (4 1994), S. 369–380 (siehe S. 24).

[Rec73] Ingo Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. frommann-holzboog, 1973 (siehe S. 15, 16).

[RK14] Kanchan Rani und Vikas Kumar. „Solving Travelling Salesman Problem using Genetic Algorithm based on Heuristic Crossover and Mutation Operator“. In: *IMPACT: International Journal of Research in Engineering & Technology* 2.2 (2014), S. 27–34 (siehe S. 10).

[Ros60] H. H. Rosenbrock. „An Automatic Method for Finding the Greatest or Least Value of a Function“. In: *Computer Journal* 3 (1960), S. 175–184 (siehe S. 64).

[Sch74] Hans-Paul Schwefel. *Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie*. Birkhäuser, 1974 (siehe S. 15).

[Sch81] Hans-Paul Schwefel. „Numerical optimization of computer models“. In: (1981) (siehe S. 64).

[Scr13] Luca Scrucca. „GA: A Package for Genetic Algorithms in R“. In: *Journal of Statistical Software* 53 (4 Apr. 2013), S. 1–37. URL: http://www.jstatsoft.org/v53/i04/ (siehe S. 32).

[SHF94] Eberhard Schöneburg, Frank Heinzmann und Sven Feddersen. *Genetische Algorithmen und Evolutionstrategien*. Addison-Wesley Verlag, 1994 (siehe S. 35).

[SR95] Hans-Paul Schwefel und Günter Rudolph. „Contemporary evolution strategies“. In: *Advances in Artificial Life, Lecture Notes in Computer Science* 929 (1995), S. 891–907 (siehe S. 22).

[Tro12] Andrew Troelsen. *Pro C# 5.0 and the .NET 4.5 Framework*. Apress, 2012 (siehe S. 48).

[Üço02] Göktürk Üçoluk. „Genetic Algorithm Solution of the TSP Avoiding Special Crossover and Mutation“. In: *TSI Press* (2002) (siehe S. 10).

[Wei02] Karsten Weicker. *Evolutionäre Algorithmen*. Vieweg+Teubner Verlag, 2002 (siehe S. 3, 6).

[Yan08] Xin-she Yang. *Introduction To Computational Mathematics*. World Scientific Publishing Company, 2008 (siehe S. 5).

Online-Quellen

[Wei09] Thomas Weise. *Global Optimization Algorithms – Theory and Application*. Version 2. 26. Juni 2009. URL: http://www.it-weise.de/ (siehe S. 11, 14).