Fachbereich Informatik
Fachgebiet Neuroinformatik

Implementierung von Partikel-Schwarm-Optimierung und Vergleich mit einer Evolutionsstrategie

Schriftliche Prüfungsarbeit
für die Bachelor-Prüfung des Studiengangs Angewandte Informatik
an der Ruhr-Universität Bochum

vorgelegt von
Melchior, Jan

Prüfer 1: Würtz, Dr. Rolf P.
Prüfer 2: Igel, Prof. Dr. Christian

Abgabe: 17.09.2008

Erklärung

Ich erkläre, dass das Thema dieser Arbeit nicht identisch ist mit dem Thema einer von mir bereits für ein anderes Examen eingereichten Arbeit.

Ich erkläre weiterhin, dass ich die Arbeit nicht bereits an einer anderen Hochschule zur Erlangung eines akademischen Grades eingereicht habe.

Ich versichere, dass ich die Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen benutzt habe. Die Stellen der Arbeit, die anderen Werken dem Wortlaut oder dem Sinn nach entnommen sind, habe ich unter Angabe der Quellen der Entlehnung kenntlich gemacht. Dies gilt sinngemäß auch für gelieferte Zeichnungen, Skizzen und bildliche Darstellungen und dergleichen.

_____ _____

Datum Unterschrift

Zusammenfassung

Ziel dieser Arbeit ist die Beschreibung der Partikel-Schwarm-Optimierung, sowie deren objektorientierte Implementierung. Des weiteren soll ein hinreichend objektiver Vergleich mit einer der zur Zeit besten Evolutionsstrategien aufgezeigt werden.

Bei der Implementierung wurde Wert gelegt auf einen wiederverwendbaren, leicht verständlichen Quellcode, der es dem Leser ermöglichen soll, mit geringem Aufwand eigene Varianten und Konzepte des Partikel-Schwarm-Algorithmus umzusetzen.

Die Partikel-Schwarm-Optimierung zählt zu den evolutionären Algorithmen und setzt Konzepte der Schwarm-Intelligenz um. Im ersten Kapitel werden diese Begrifflichkeiten erläutert, um jedem Leser den Einstieg in diesen Themenbereich zu ermöglichen und ein fundiertes Verständnis für den Ursprung des Algorithmus zu erreichen. Das zweite Kapitel beschreibt diesen Algorithmus und nimmt Bezug auf diverse Veränderungen, die seit Beginn der Partikel-Schwarm-Optimierung (1995) untersucht wurden. Neben den Konvergenzkriterien, Zufallsvariablen und Nachbarschaftstopologien werden anschließend im dritten Kapitel adaptive Versionen des Algorithmus vorgestellt. Dies umfasst auch zwei eigene Versionen. So zum einen die „Mean-Based-PSO“, welche das gewichtete Mittel des Schwarms nutzt und zum anderen die „Escape-PSO“, welche die lokale Konvergenz besser verhindern können soll.

Kapitel vier beschreibt die Implementierung anhand von UML-Diagrammen.

Im letzten Kapitel werden schließlich verschiedene Varianten des Algorithmus auf ihre Brauchbarkeit untersucht. Im Anschluss wird ein Vergleich mit der „Covariance-Matrix-Adapation Evolution Strategy“ (CMA-ES) angestrebt. Hierfür werden die Ergebnisse der Verfahren auf neun Benchmarkfunktionen verglichen.

Inhaltsverzeichnis

Erklärung II

Zusammenfassung III

Inhaltsverzeichnis IV

1 Theoretische Grundlagen 1

1.1 Optimierung 1

1.2 Optimierung in der Natur 3

1.3 Die biologische Evolution 4

1.4 Evolutionäre Algorithmen 5

1.5 Intelligenz und Verstand 7

1.6 Die Schwarm-Intelligenz 8

1.7 Emergenz 9

2 Die Partikel-Schwarm-Optimierung 10

2.1 Von Schwärmen zum Optimierungsverfahren 10

2.2 Der Grundalgorithmus 11

2.2.1 Pseudo-Code 14

2.2.2 Anpassung für binäre Probleme 15

2.3 Zufallsvariablen 16

2.3.1 Rechteck-Gleichverteilung 16

2.3.2 Kugel-Gleichverteilung 17

2.3.3 Normalverteilung 18

2.3.4 Pivot-Methode 19

2.4 Beschränkung des Schwarms 20

2.4.1 Beschränkung auf den Definitionsbereich 20

2.4.2 Beschränkung der Geschwindigkeit 21

2.4.3 Konvergenz-Methoden 21

2.4.3.1 Die „Inertia weight“ 21

2.4.3.2 Die „Global Local Best Inertia weight“ 22

2.4.3.3 Der „Constriction Factor“ 23

2.5 Nachbarschaftsbeziehungen 24

2.5.1 Ring-Topologie 24

2.5.2 Random-Topologie 25

2.5.3 Wheel-Topologie 25

2.5.4 Neumann-Topologie 26

2.5.5 Distanz-Topologie 27

2.5.6 Dynamische-Topologien 27

2.5.7 Memory-Swarm-Topologie 28

3 Adaptive Erweiterungen 30

3.1 Landscape Adaptive PSO 30

3.1.1 „Crossing over“ 30

3.1.2 „Distribution vector“ 31
3.1.3 LA-PSO Pseudo-Code 33
3.2 Fully Informed Particle-Swarm 34
3.3 Mean-Based-PSO 34
3.4 ESC-PSO 37
3.5 Tribes 38
4 Implementierung 40
4.1 Die Klasse Particle 40
4.2 Die Klasse Swarm 41
4.3 Die Schnittstelle IDistribution 42
4.4 Die Schnittstelle ITopology 43
4.5 Die Schnittstelle IFitness 43
4.6 Die abstrakte Klasse Function 44
4.7 Die abstrakte Klasse Pso 44
5 Auswertung 46
5.1 Convariance-Matrix-Adaptation Evolution Strategy 46
5.2 Vergleichsbasis 47
5.3 Benchmark Funktionen 48
5.3.1 Unimodale Funktionen 48
5.3.2 Multimodale Funktionen 50
5.4 Optimierungsvergleiche 53
5.4.1 PSO-Parameterwahl 53
5.4.2 Vergleich mit der Evolutions Strategie „CMA-ES“ 57
5.5 Fazit 62
Literaturverzeichnis 63
Abbildungsverzeichnis 66
Formelverzeichnis 68
Anhang (CD) 70

1 Theoretische Grundlagen

In diesem Kapitel sollen die Grundlagen vermitteltet werden, auf denen die Partikel-Schwarm-Optimierung basiert.

Zunächst wird definiert, was allgemein unter Optimierung verstanden werden kann und welche Probleme bei klassischen Optimierungsverfahren auftreten können. Aufgrund der auftretenden Probleme werden neue Verfahrensweisen nötig. Hier bietet die Natur mit der Evolution und der Intelligenz zwei Optimierungsstrategien, die für den Partikel-Schwarm-Algorithmus von Bedeutung sind. Daher wird zunächst die grundlegende Funktionsweise der biologischen Evolution beschrieben, die als Vorbild für die im Anschluss dargestellten Evolutionären Algorithmen dient. Bevor der Begriff der Schwarm-Intelligenz erklärt werden soll, wird zunächst der Zusammenhang zwischen Intelligenz und Kommunikation sowie der Einfluss von evolutionären Prozessen in der Informationsverarbeitung aufgezeigt. Nach der Erläuterung des Begriffs Emergenz und dessen Bedeutung für die Schwarm-Intelligenz wird im zweiten Kapitel hierauf aufbauend der Partikel-Schwarm-Algorithmus beschrieben.

1.1 Optimierung

In den verschiedenen Disziplinen steht man vor dem Problem, die optimale Kombination von Parametern bestimmen zu wollen. Sucht man beispielsweise die effizienteste Einstellung einer Maschine, bewertet man die Parametereinstellungen gemäß einer gewählten Güte wie Kosten, Ausfallsicherheit oder auch Effizienz allgemein. Hat eine Maschine 10 Parameter, die jeweils 100 Einstellungen annehmen können, so steht man vor der unmöglichen Aufgabe, 10^{100} Möglichkeiten ausprobieren zu müssen, um ein Optimum eindeutig zu identifizieren. Man ist jedoch in der Lage, auf Basis von Erfahrung und statistischen Auswertungen die Abhängigkeit der Güte von den Parametern durch eine Funktion *f : Parameter* → *Güte* zu modellieren. Ein Teilgebiet der angewandten Mathematik - das „Operations Research“[1] - beschäftigt sich insbesondere mit der bestmöglichen Modellierung des Problems, um später mathematische Optimierungsverfahren anwenden zu können. Eine hinreichend genaue Beschreibung der Realität des Modells ist dabei essentiell, damit das ermittelte Optimum des Modells auch das reale Optimum darstellt. Da das Modell eine Funktion ist, sucht man das globale Optimum. Dieses kann je nach Bedeutung der Güte zum Beispiel ein Minimum an Kosten oder auch ein Maximum an Effizienz sein. Für das Optimieren von Funktionen stehen uns aus der Mathematik verschiedene Verfahren zur Verfügung: Ist die Funktion linear, kann man beispielsweise das „Simplex Verfahren“[2] anwenden, um das Optimum eindeutig zu berechnen. Für nicht lineare

[1] (auch: Unternehmensforschung) wissenschaftliche Disziplin die sich mit Verfahren zur Vorbereitung von Entscheidungen bei Führung von Betrieben, Verwaltung und anderen Organisationen durch Anwendung mathematischer Methoden und Modelle befasst [26]

[2] (auch Simplex-Algorithmus) ist ein Optimierungsverfahren der Numerik zur Lösung linearer Optimierungsprobleme. [26]

Funktionen ist aus der Analysis bekannt, dass die Extremstellen nur an den Punkten liegen können, an denen der Gradient der Nullvektor ist. Um diese Punkte aufzufinden nutzen gradientenbasierte Verfahren den Umstand aus, dass der Gradient in Richtung des steilsten Anstiegs zeigt und somit der negierte Gradient in Richtung des stärksten Gefälles. So kann man durch schrittweises Folgen des Gradienten oder negierten Gradienten in ein Extremum konvergieren. Grafik 1.1 veranschaulicht dies für zwei Funktionen *f(x)* und *f(x,y),* in denen der negierte Gradient, hier in hellblau angedeutet, in Richtung des Minimums zeigt.

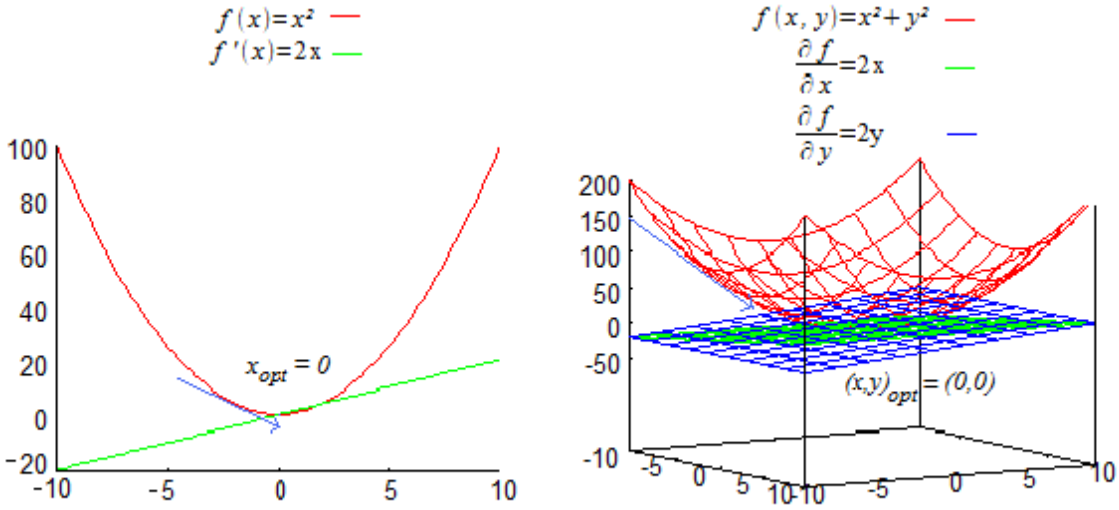
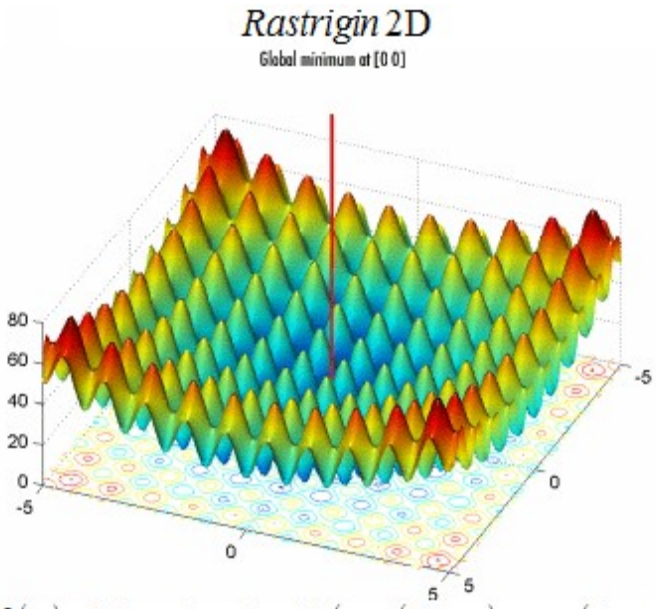


Abbildung 1.1: Visualisierung konvexer Funktionsoptimierung durch Verwendung des Gradienten

Gradientenverfahren sind schnell. Sie bürgen aber die Gefahr bei multimodalen[1] Funktionen in ein lokales Extremum zu konvergieren. Hier bieten Verfahren wie der „Hillclimbing Algorithmus“[2] eine gewisse Alternative. Diese zielen auf eine zufällige Reinitialisierung beim Erreichen eines Extremums ab, mit der Absicht, bei einem der Durchläufe das tatsächliche Optimum zu erreichen. Leider kann auch hier nicht ausgeschlossen werden, dass es sich um ein lokales Extremum handelt. Dies kann durch starke Multimodalität sehr wahrscheinlich werden, wie dies Grafik 1.2 beispielhaft zeigt. Erschwerend kommt das Problem von nicht differenzierbaren oder nicht stetig differenzierbaren Funktionen hinzu. So besteht zwar die Möglichkeit, den Gradienten über Näherungsverfahren zu approximieren. Allerdings ist dies meist sehr aufwendig, und das Problem der lokalen Extrema besteht auch hier weiterhin.

[1] In diesem Zusammenhang sind Funktionen mit mehr als einem lokalen Optimum gemeint.

[2] Optimierungsverfahren welches von einem gegebenen Startpunkt aus durch Gradientenauswertung in Richtung des Optimum konvergiert.



$$f(x)=20+x^2+y^2-10(\cos(2\pi x)+\cos(2\pi y))$$

Abbildung 1.2: Beispielabbildung einer Multimodalen Funktion

Zusammenfassend darf festgestellt werden, dass klassische Optimierungsverfahren immer dann an ihre Grenzen stoßen, wenn die Funktion multimodal oder nicht differenzierbar ist. Hieraus ergibt sich zwangsläufig die Notwendigkeit, neue Techniken zu entwickeln.

1.2 Optimierung in der Natur

Blickt man in die Natur, so kann man überall Lebewesen entdecken, die sich im Laufe der Evolution an ihren Lebensraum angepasst haben. Der Prozess der biologischen Evolution ist dabei hochkomplex mit einer unüberschaubaren Anzahl von Kausalzusammenhängen. Glücklicherweise ist der Mensch in der Lage, komplexe Sachverhalte zu abstrahieren. Primär lassen sich drei Operationen identifizieren, die in der Lage sind, Lebewesen an ihre Umgebung anzupassen und somit ihre Überlebenschancen verbessern. Es sind dies: Rekombination, Mutation und Selektion. Sie beeinflussen die Genome der Lebewesen, so dass langfristig über Generationen gesehen sich die „besten Individuen“ im Sinne der Evolution durchsetzen. So kann man also Evolution als Optimierungsprozess auffassen, welcher die Lebewesen im Bezug auf ihrem Umwelt optimiert.

Menschen und höher entwickelte Tiere sind jedoch auch in der Lage, sich schnell an neue Situationen anzupassen. So benutzen sie ihren Verstand („Mind“), um beispielsweise Techniken zur Futtererschließung oder Jagd zu erlernen, die ihnen einen Vorteil verschaffen.

Im Buch „Swarm Intelligence“ von Kennedy & Eberhardt [1] werden die zwei adaptiven Prozesse „Evolution“ und „Mind“ nach der Bezeichnung von Gregor Bateson als “The two great stochastic Systems“ beschrieben.

Die Bezeichnung als stochastische Systeme rührt von der Beobachtung her, dass der Zufall eine entscheidende Rolle spielt, um neue Bereiche zu erschließen. Die Evolution wird unter anderem auch durch zufällige Mutationen beeinflusst, welche völlig neue Ausprägungen von Fähigkeiten ermöglichen. Genauso wird auch unser Denken von zufälligen Prozessen beeinflusst, ohne die Kreativität und Phantasie nicht existieren würden. Im Folgenden möchte ich beide Systeme detaillierter vorstellen. Ziel soll sein, besser zu verstehen, wie die Natur „optimiert“, um diese Strategie auf allgemeine Probleme aller Disziplinen übertragen zu können.

1.3 Die biologische Evolution

Die biologische Evolution basiert auf der Veränderung des Erbgutes primär durch die drei Operationen Mutation, Rekombination und Selektion. Dadurch stehen alle Lebewesen in einem ständigen „Optimierungsprozess“ der Anpassung an ihre dynamische Umwelt.

Die Rekombination ermöglicht eine Neukombination der Merkmale zweier Elternindividuen, deren Gene sich im Sinne der Evolution als „gut“ erwiesen haben, da diese das fortpflanzungsfähige Alter erreicht haben. Grundlegend besteht die Rekombination aus dem „Crossing over“ zweier homologer Chromsomen, wobei ein Stück der Chromatiden bei der sexuellen Fortpflanzung ausgetauscht wird. Nachkommen erhalten dadurch mütterliche sowie väterliche Merkmale. Dieser Vorgang sorgt mit dafür, dass neue Merkmalskombinationen bei den sich geschlechtlich fortpflanzenden Lebewesen entstehen. Auch wenn die Anzahl der Rekombinationsmöglichkeiten sehr groß ist, so ist die Rekombination doch auf die Genome der beiden Elternteile beschränkt. Da alle hochentwickelten Lebensformen ursprünglich von Einzellern abstammen, wird deutlich das diese nicht durch den alleinigen Einfluss der Rekombination möglich gewesen wäre.

Die Mutation ist ein häufig auftretender spontaner Prozess der Natur, welcher im Normalfall vom Organismus wieder „repariert“ wird. Ist dies nicht der Fall, dann ermöglicht die Mutation die zufällige Änderung von Genen und somit auch die Möglichkeit, völlig neue Spezies hervorzubringen oder auch, den Organismus zu schädigen. Während die Rekombination also eine Neukombination innerhalb der Gene darstellt, ist die Mutation ungerichtet und kann eine Verbesserung ebenso wie eine Verschlechterung hervorrufen.

Die Selektion setzt das Prinzip „survival of the fittest“ durch, bei dem besser angepasste Individuen sich gegen andere Individuen der Population durchsetzen. Dies kann bedeuten, dass Individuen nicht in der Lage sind, ihren Fressfeinden zu ent-

kommen, Futter oder Sexualpartner zu finden und somit in keinem der Fälle ihr Genom an Nachkommen weitergeben können. In der Natur lässt sich der Erfolg eines Lebewesens - also seine „Fitness“- nur indirekt an dessen Erfolg messen sich zu reproduzieren. Dies lässt also auch eine Aussage über die Qualität eines Genoms zu. Betrachtet man den Prozess der Evolution vereinfacht, so kann man nun diesen Prozess simulieren und für Problemlösungen technischer Probleme verwenden.

1.4 Evolutionäre Algorithmen

Die Evolutionären Algorithmen beschreiben eine Klasse von Algorithmen die das Konzept der biologischen Evolution simulieren um so rechnergestützte Optimierung zu ermöglichen.

Diese Algorithmen sind sogenannte Meta-Heuristiken. Dies sind Optimierungsverfahren, die nicht zwingend das globale Optimum erreichen, sondern meist mit einer hinreichend genauen Abbruchbedingung arbeiten. Es ist ein Vorteil dieser Verfahren, dass sie in der Lage sind, auch Probleme mit exponentieller Laufzeit hinreichend genau zu optimieren. Evolutionäre Algorithmen simulieren die biologische Evolution auf einer Population von K Individuen, welche jeweils eine kodierte Lösung des zu optimierenden Problems darstellen. Abbildung 1.3 zeigt den grundlegenden schematischen Ablauf eines solchen Algorithmus.

Zu Beginn wird eine Population von K Individuen meist mit zufälligen Werten des Lösungsraumes initialisiert und anhand der Fitnessfunktion bewertet. Wie zuvor erklärt beschreibt die Fitnessfunktion dabei den Zusammenhang zwischen Individuum und Fitness des Problems hinreichend genau.

Zu Beginn einer jeden Iteration wird entschieden, welche Individuen als Eltern die nächste Generation bilden. Hier gibt es diverse „Evolutionsstrategien“, die im Anschluss angesprochen werden sollen. Häufig bilden jedoch 50% der besten Individuen der aktuellen Generation die Eltern der Folgegeneration. Die Elternindividuen werden nun rekombiniert oder in manchen Strategien auch geklont und anschließend mutiert, indem die Werte mit einer gewählten Wahrscheinlichkeit leicht variiert werden. Die Mutation spielt eine entscheidende Rolle in allen evolutionären Prozessen und wird meist durch eine Normalverteilung beschrieben. Hierbei haben geringfügige Änderungen eine sehr hohe Wahrscheinlichkeit, während große Änderungen eher selten auftreten. Im Anschluss werden die Individuen nach einem Schema selektiert, wodurch die verbleibenden Individuen die Folgegeneration bilden.

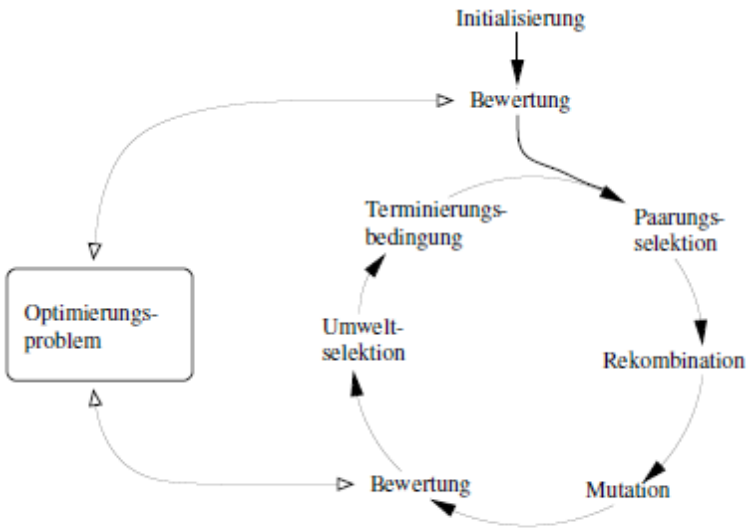


Abbildung 1.3: Optimierungsschemata evolutionärer Algorithmen

Evolutionsstrategien sind verschiedene Varianten von evolutionären Algorithmen und bilden die biologische Evolution in unterschiedlichen Komplexitätsstufen ab. Hier soll ein kurzer Überblick über die diversen Verfahren vermittelt werden. Eine detaillierte Beschreibung mit Verweisen zu Sekundärliteratur bietet Quelle [16] sowie Quelle [17].

Die (1 + 1) – Evolutionsstrategie stellt die einfachste Variante dar, in der die gesamte Population durch ein Individuum repräsentiert wird. Das Individuum wird geklont und mutiert, wodurch im Anschluss die Folgegeneration aus dem besseren der beiden Individuen gebildet wird.

Die $(\mu + \lambda)$ – Evolutionsstrategie ist die Verallgemeinerung der (1 + 1) Variante, wobei die von den μ Elternindividuen erzeugten λ Klone wiederum mutiert werden. Aus der entstanden Population der Größe $\mu + \lambda$ werden die besten μ Individuen ausgewählt und bilden die Folgegeneration. Man beachte, dass bei diesem und dem vorangegangenen Verfahren ein Individuum der Elterngeneration stets auch die Folgegeneration bilden kann und somit im übertragenen Sinne die Möglichkeit hat, ewig zu Leben.

Die (μ , λ) – Evolutionsstrategie verhindert den Effekt des ewigen Lebens, indem die Folgegeneration stets ausschließlich aus den mutierten Klonen gebildet wird. Hier ist die Gefahr groß, dass alle Klone eine schlechtere Fitness haben könnten als ihre Eltern, wodurch die Optimierung dann einen Rückschritt machen würde.

Die $(\mu / p \# \lambda)$ – Evolutionsstrategie führt die Rekombination ein, bei der ein jeder Nachkomme λ der Elterngeneration aus p Eltern rekombiniert wird. Das Zeichen # steht hier als Variable für die $(\mu + \lambda)$ – oder (μ , λ) – Evolutionsstrategie, da nach der Rekombination nun entsprechend der gewählten Variante verfahren wird.

1.5 Intelligenz und Verstand

Neben der Evolution gilt nach [1] auch den Verstand als adaptives System.

Kennedy & Eberhard definieren Intelligenz mit „The quality of good mind“[1], also als Qualität des Verstandes. Intuitiv sieht man ein Individuum als intelligenter an, je besser es in der Lage ist, Probleme zu lösen und sich an neue Gegebenheiten anzupassen. Betrachtet man das Gehirn als Sitz der Intelligenz, so resultiert die Intelligenz aus der Kommunikation der Nervenzellen untereinander, denn jede Zelle allein kann nicht viel mehr als ankommende Reize zu verrechnen und das Resultat als Aktionspotenzial weiterzuleiten. Dennoch ist unser Gehirn in der Lage, ein hohes Maß an Intelligenz hervorzubringen. Intelligenz kann also als Resultat einfacher kommunikativer Prozesse verstanden werden.

Dawkin[6][1] stellte ein Modell auf, welches mentale Prozesse und Entwicklungen den gleichen Einflüssen durch Mutation, Rekombination und Selektion unterlegen sieht wie die Evolution. Er bezeichnet Gedanken und Ideen als Memes, vom griechischen Wort „mimem“ = imitieren und beschränkt deren Existenz nicht nur auf das Gehirn. Memes können diverse Formen der Kommunikation nutzen, um sich zu reproduzieren. So kann eine Idee vom Buch zum Computer und wiederum zu Gehirnen gelangen. Man beachte, dass die Kommunikation die entscheidende Grundlage für die Reproduktion darstellt.

Memes verbreiten sich über Imitation. Dabei kann das entstehende Meme dem Einfluss mehrerer Memes unterliegen und so eine Rekombination derer darstellen. So lernt man und erzeugt somit ein neues Meme, indem man eine oder auch mehrere Personen und deren Memes imitiert und rekombiniert. Da jedes Meme, das heißt jeder Gedanke, von keiner Person je gleich interpretiert wird, unterliegt ein Meme bei der Weitergabe durch Imitation dem Einfluss der Mutation. Im Sinne des Prinzips „Survival of the Fittest“ verbreiten sich Memes, die eine hohe Akzeptanz in der Population erlangen sehr stark, während Memes schließlich aussterben, wenn sie keine Plattform wie Bücher, Gehirne, etc. mehr haben, von der sie sich reproduzieren können.

Je schneller die Kommunikation ist, desto rasanter verläuft die Evolution von Memes. Mit der Erfindung der Schrift wurde eine neue Plattform geschaffen, von der aus sie ein weit größeres Potential hatte, sich zu reproduzieren als durch mündliche Überlieferung. Der gleiche Prozess fand mit dem Beginn des Informationszeitalters statt, indem Distanzen bei der Verbreitung von Information fast keine Rolle mehr spielten. Das Internet als riesiges Informationsnetzwerk bot im Grunde jedem Meme eine Plattform. Dies führte aber zu dem Problem, dass die Qualität der Memes nicht mehr zwangsläufig durch die Gesellschaft validiert wird. Daher ist es wichtig, Informationen aus dem Internet zu prüfen. Im Hinblick auf Optimierung in der Natur optimieren sich mentale Prozess wie Ideen nicht nur innerhalb des Verstandes eines Individuums, sondern erfahren eine evolutionäre Entwicklung innerhalb der Population.

1.6 Die Schwarm-Intelligenz

Man hat Intelligenz verstanden als Fähigkeit und Qualität, den Verstand zu benutzen und sieht Informationen bei ihrer Verbreitung den gleichen oder ähnlichen Operationen unterliegen, wie die der Evolution. Allgemein formuliert ist der Verstand ein System aus Einzelkomponenten, die in der Lage sind, durch Kommunikation Intelligenz hervorzubringen.

Überträgt man diese Vorstellung auf Schwärme, versteht man hierunter die Fähigkeit und Qualität eines Schwarms (einer Gruppe von Individuen) „seinen" Verstand zu benutzen. Man betrachtet hier gewollt nicht die Intelligenz der Einzelindividuen, da die Intelligenz des Schwarms nicht mit der Summe der Intelligenz seiner Individuen gleichzusetzen ist. Der Schwarm ist in der Lage, auch solche Probleme effizient zu lösen, deren Lösung die Fähigkeiten der Einzelindividuen übersteigt. Betrachtet man einen Fischschwarm, so kann man kein Tier ausmachen, welches eine Führungsposition einnimmt. Vielmehr scheint der Schwarm als einzelnes Individuum zu agieren. Trotzdem findet der Schwarm neue Futterquellen oder kann seinen Fressfeinden ausweichen. Viele Jahre vermuteten Verhaltensforscher und Biologen ein noch nicht entdecktes zentrales Steuerungskonzept. Bis es Graig Reynol 1987 gelang, mittels eines einfachen Modells namens Boid ohne eine zentrale Steuerung das Flugverhalten eines Vogelschwarms realistisch zu simulieren [1]. Dies erfolgte auf Basis von drei einfachen Regeln, die jeder Vogel in dieser Reihenfolge zu beachten hatte.

1. Weiche Artgenossen und Hindernissen aus
2. Versuche, mit der gleichen Geschwindigkeit wie dein Nachbar zu fliegen
3. Versuche, näher an das Zentrum des Schwarms heran zu kommen

Erweitert man das Modell um eine vierte Regel, so wird die Fortbewegung des Schwarms als Kollektiv in Richtung eines Ziels nachvollziehbar.

4. Versuche, dein angestrebtes Ziel zu erreichen

In der Simulation wird den Einzelindividuen meist ein globales Ziel vorgegeben. Realistischer ist es jedoch, den Schwarm beispielsweise nach Futter suchen zu lassen.

Entdeckt ein Tier eine Futterquelle, so orientiert es sich etwas weniger an seinen Nachbarn und richtet seine Bewegungen teilweise auf das Ziel aus. Alle anderen Tiere, die kein Ziel ausmachen, orientieren sich also durch die obigen drei Regeln auch an der Richtungsänderung des besagten Tieres. Endecken nun mehrere Tiere diese Futterquelle durch Zufall oder weil die leichte Richtungsänderung sie dazu veranlasst, so verlagert sich die Gesamtrichtung des Schwarms immer mehr in diese Richtung. Aus der Sicht des einzelnen Tieres sind die Ziele Fressen und Sicherheit. Dabei strebt aber kein Tier an, dass sich der gesamte Schwarm zum Ziel bewegt, sondern dies resultiert unbewusst aus den oben genannten Verhaltensweisen der einzelnen

Tiere. Der Reiz einer Futterquelle veranlasst das Tier zur Richtungsänderung, wobei es allerdings den Schwarm aufgrund der Gefahr nicht allzu weit verlässt.

1.7 Emergenz

Diesen Effekt, dass sich aus dem Zusammenspiel von einzelnen Elementen eine geordnete Struktur ergibt, die weder in den Operationen direkt, noch in den Elementen enthalten ist, nennt man Emergenz.

Emergenz bedeutet also, dass ein Kollektiv in der Lage ist, ein Problem zu lösen, ohne dass die Definition des Problems oder eine Lösungsstrategie implizit vorliegt.

Ein Fischschwarm bietet durch das Kollektiv eine hohe Sicherheit gegenüber Fressfeinden, was dem Einzeltier jedoch nicht bewusst ist. Vielmehr versucht jedes einzelne Tier, sich im Schwarm zu verstecken.

In diesem Sinne unterliegt auch unser Gehirn per Definition dem emergenten Effekt. denn die Intelligenz resultiert aus der Kommunikation, ohne dass Strategien zur Problemlösung vorliegen müssen. Das Resultat ist eine hochgradig komplexe und adaptive Struktur, die Probleme lösen kann.

Axelrod[1] stellte ein Modell auf, welches die Informationsverbreitung in einer Gesellschaft simulieren sollte. Das Modell besteht aus einem zweidimensionalen Feld, dessen Einträge N Zahlen von Null bis Neun enthalten, die potentielle Lösungen eines Problems darstellen. Axelrod zeigte, dass sich durch einfache Operationen sogar NP-Harte Probleme (allerdings in niedrigen Dimensionen) wie beispielsweise das Travelling Salesmen Problem[1] meta-heuristisch optimieren lassen. Hierfür wird eine Fitnessfunktion aufgestellt, die die Fitness des Eintrags berechnet. In jeder Iteration, in der alle Einträge durchlaufen werden, wird nun der Fitnesswert des aktuellen Wertes mit dem seiner Nachbarn verglichen. Derjenige Nachbar, der den besten Fitnesswert hat, wird ausgewählt, und es wird zufällig eine Position im Eintrag gewählt und dessen Wert übernommen. Der Algorithmus fand in Bezug auf das TSP immer eine Lösung, die entweder das Optimum war oder sehr nahe heran reichte. Auch hier lassen sich klare Parallelen zu den evolutionären Algorithmen identifizieren. Allerdings fehlt die Selektion vollständig, und die Informationsweitergabe geschieht nur über eine festgelegte Nachbarschaft.

[1] Travelling Salesmen Problem – kombinatorisches Optmierungsproblem. Die Aufgabe besteht darin, eine Reihenfolge für den Besuch mehrerer Orte so zu wählen, dass die gesamte Reisestrecke des Handlungsreisenden nach der Rückkehr zum Ausgangsort möglichst kurz ist. [3]

2 Die Partikel-Schwarm-Optimierung

Das folgende Kapitel beschreibt zunächst den Grundalgorithmus und zeigt dabei die Zugehörigkeit der PSO zu den Evolutionären Algorithmen und der Ähnlichkeit zum „Boid-Modell“ auf. Im Anschluss werden verschiedene Adaptionen vorgestellt die sich primäre auf die Zufallsvariablen, Beschränkungskriterien und die Topologien beziehen, die von der PSO genutzt werden.

2.1 Von Schwärmen zum Optimierungsverfahren

Die Partikel Schwarm Optimierung, kurz PSO, besitzt starke Ähnlichkeit mit dem zuvor beschriebenen „Boid Modell“, da die PSO ebenfalls aus einer Gruppe von Individuen besteht, die durch einfache kommunikative Prozesse innerhalb ihrer Nachbarschaft beeinflusst werden. Der Unterschied besteht darin, dass der Schwarm und dessen Emergenz nicht mehr für die Simulation des Vogelfluges benutzt wird, sondern dazu, das Optimum einer Fitnessfunktion zu bestimmen.

Kennedy & Eberhardt erklären in ihrem Buch „Swarm Intelligence“ ein Modell, welches grundlegend die Schwarmintelligenz beschreibt, die auf drei einfachen Operationen basiert. Sie fassen Anpassung in einer Population und somit Optimierung als Resultat von Evaluation, Vergleich und Imitation auf. Dabei betrachten sie den Schwarm in Hinblick auf zwei Komponenten - nämlich die „low-level component“, die die Verhaltensweisen und kommunikativen Prozesse der Einzelindividuen beschreibt und die resultierende „high-level component“, welche die komplexen Strukturen und Ordnungen beschreibt, in diesem Fall die emergente resultierende Strategie ein Optimum zu finden.

Evaluation stellt die Voraussetzung einer jeden zielgerichteten Aktion dar, denn ohne die Möglichkeit den momentanen Zustand bewerten zu können, kann auch nicht zielgerichtet im Sinne einer Verbesserung gehandelt werden. Höher entwickelte Lebensformen sind in der Lage, auf Basis von Erfahrung und Wissen ihren Zustand zu bewerten. Zustände können in diesem Zusammenhang diverse Ausprägungen sein wie Positionen, Verhaltensweisen oder auch Ideen. Im PSO-Algorithmus wird die Evaluation durch die Auswertung der Fitnessfunktion bereitgestellt.

Der Vergleich ist die zweite wichtige Operation. So sind höher entwickelte Lebensformen nicht nur in der Lage, ihren momentanen Zustand mit historischen Zuständen zu vergleichen, sondern können durch Kommunikation zusätzlich das Wissen anderer Individuen nutzen und mit dem ihrigen vergleichen. In der PSO kennt jeder Partikel seine beste jeweils erreicht Position und kommuniziert über eine festgelegte Nachbarschaft mit anderen Partikeln, an deren bester Position es sich dann orientiert.

Die dritte Operation beschreibt die „Imitation“, womit gemeint ist, dass Lebewesen nun aus der Evaluation und dem Vergleich Konsequenzen ziehen, indem sie „gute

Verhaltensweisen“ imitieren. Da Interpretation immer subjektiv ist, unterliegt sie stochastischen Einflüssen in der Form, dass die resultierende Imitation kein deterministischer Prozess ist.

Die Parallelen zu evolutionären Prozessen finden sich in der Imitation durch die Mutation und Rekombination wieder. Die Selektion entspricht der Tatsache, dass schlechte Informanten nicht imitiert werden. Auch im „Boid Modell“ finden sich diese Operationen wieder, wo jeder Vogel seine Position sicherlich hinsichtlich der Sicherheit bewertet, seine Position mit der seiner Nachbarschaft vergleicht und schließlich die Geschwindigkeit und Richtung seiner Nachbarn imitiert.

2.2 Der Grundalgorithmus

Die PSO wendet diese zuvor genannten drei Operationen auf einen Partikel-Schwarm an, um den resultierenden emergenten Effekt hinsichtlich der Optimierung zu nutzen. In Anlehnung an das „Boid Modell“ unterstützt es das Verständnis des Algorithmus, wenn man sich den Schwarm als natürlichen Vogel,- oder Fischschwarm vorstellt, der sich mit bestimmten Geschwindigkeiten fortbewegt.

Dieser Schwarm besteht aus N Partikeln, deren Positionsvektor x_i jeweils eine potentielle Lösung des Optimierungsproblems darstellt.

Jeder Partikel ist in der Lage, seine Position durch Einsetzen des Positionsvektors in die D-dimensionale $Fitness(\ x^D\ )$ zu evaluieren und erinnert sich an die beste jemals erreichte Position p_i und den zugehörigen Fitnesswert $best$.

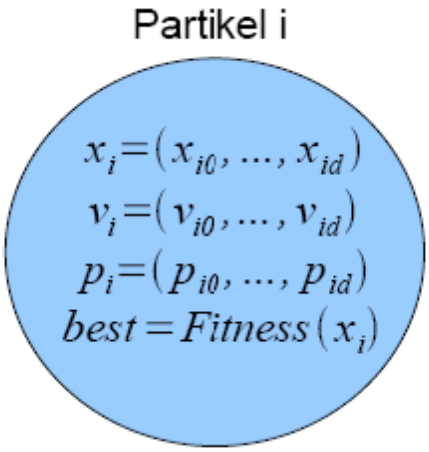


Abbildung 2.1: Aufbau eines Partikels der PSO

Ziel ist es, die optimale Position zu finden, die durch das globale Minimum/Maximum der Fitnessfunktion repräsentiert wird. In folgenden wird stets das globale Minimum gesucht da ein häufiges Optimierungsproblem darin besteht eine Fehlerfunktion zu minimieren, durch einfache Anpassung ist der Algorithmus jedoch auf die Maximierung veränderbar.

Jeder Partikel hat einen Geschwindigkeitsvektor $v_{i,}$, durch dessen Adaption in Verbindung mit der Position das Optimum identifiziert werden soll.

Ausgehend von der Beobachtung, dass bei der Entscheidungsfindung die eigene Erfahrung und die der kommunikativen Nachbarschaft Einfluss nimmt, geht man davon aus, dass jeder Partikel sich an seiner eigenen besten jemals erreichten Position p_i und der besten erreichten Position einer definierten Nachbarschaft des Partikels orientiert. Die Kommunikation mit einer festgelegten Nachbarschaft weißt Ähnlichkeit mit dem „Axelrod's Culture Model“ auf.

Betrachtet man den einfachsten Fall, besteht die Nachbarschaft aus allen Partikeln. Somit ist der beste Nachbar gleichzeitig der beste Partikel des gesamten Schwarms p_g.

Jeder Partikel ist nun in der Lage, seine Position zu evaluieren, kann diese mit seinen Nachbarn vergleichen und durch eine entsprechende Geschwindigkeitsänderung imitieren.

Für die Berechnung zum Zeitpunkt t wird die alte Geschwindigkeit $v_i\ (t - 1)$ mit den Differenzvektoren $p_i – x_i(t - 1)$ und $p_g – x_i(t – 1)$ addiert, um die neue Geschwindigkeit $v_i\ (t)$ zu berechnen.

$$v_i(t) = v_i(t-1) + (p_i - x_i(t-1)) + (p_g - x_i(t-1))$$

Formel 1: Statische Geschwindigkeitsadaption

Diese Berechnung führt durch die gleich gewichtete Vektoraddition zum grünen Kreis in Abbildung 2.2. Jedoch führt dies zu dem Problem, dass man in manchen Situationen nicht weiß, ob p_i oder p_g die bessere Richtung ist. Der bessere Fitnesswert ist hier nur eine lokale Orientierung, wobei das globale Optimum in Richtung der schlechteren Fitness liegen kann. Um dem Verfahren die Möglichkeit zu bieten, den lokalen Optima zu entkommen, gewichtet man beide Einflüsse mit den Zufallsvariablen ϕ_1 und $\phi_{2,}$ die sich aus den beiden Lernfaktoren c_1 und $c_{2,}$ sowie zwei Zufallsvariablen im Bereich [0,1] zusammensetzen wie Formel 2 zu entnehmen ist.

Diese stellt im übertragen Sinne die Mutation und Rekombination evolutionärer Algorithmen dar.

$$\varphi_1 = c_1\, random(0,1)$$
$$\varphi_2 = c_2\, random(0,1)$$

Formel 2: Variable Lernfaktoren

Die Berechnung der Formel 3 führt nun zu einem zufälligen Punkt in dem in Abbildung 2.2 hellgrün markiertem Lösungsraum.

$$v_i(t) = v_i(t-1) + \varphi_1(p_i - x_i(t-1)) + \varphi_2(p_g - x_i(t-1))$$
$$\varphi_1 + \varphi_2 = 4.0$$
$$x_i(t) = x_i(t-1) + v_i(t)$$

Formel 3: Geschwindigkeitsadaption der klassischen PSO

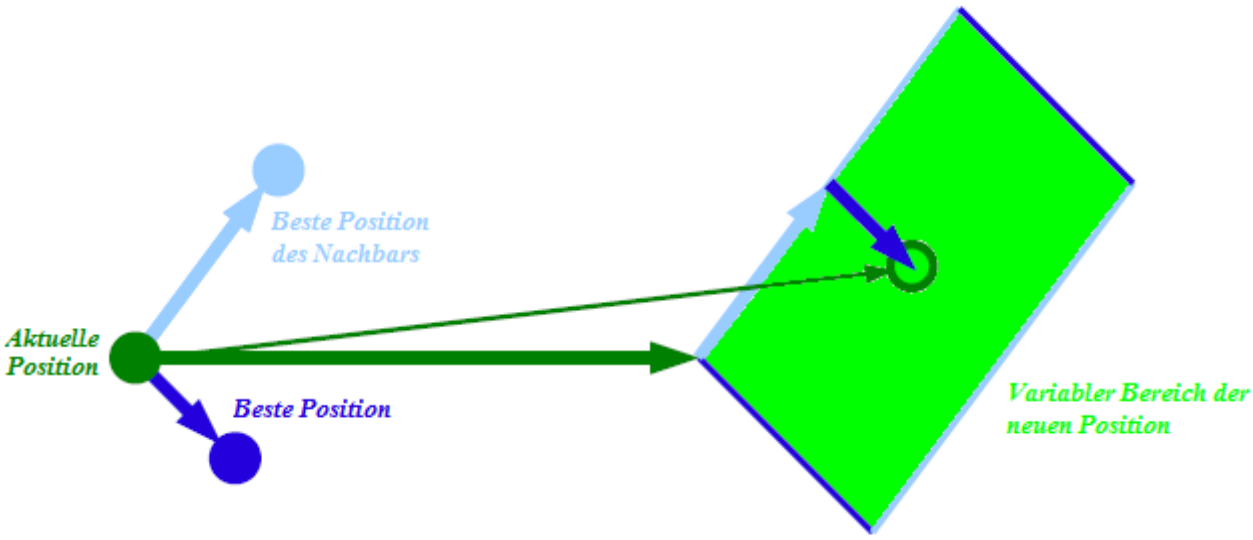


Abbildung 2.2: Grafische Visualisierung der Partikelbewegung

Diese zwei Zeilen von Formel 3 stellen den Kern der gesamten PSO dar und werden in jeder Iteration für jeden Partikel berechnet.

Um den Funktionsraum erfolgreich erkunden zu können, benötigt unser Schwarm eine Mindestanzahl von Partikeln. Zu viele Partikel führen zur redundanten Informationsauswertung, was zu verhindern ist. Als sinnvoll haben sich nach [1][2] basierend auf empirischen Studien 20-40 Partikel für einen Großteil aller Probleme erwiesen.

Diese Partikel müssen vor der ersten Iteration zufällig im Problemraum initialisiert werden und bekommen eine ebenfalls zufällige Geschwindigkeit im Bereich [-Vmax,Vmax] zugewiesen, wobei Vmax ein festgelegt Höchstgeschwindigkeit dartsellt .

2.2.1 Pseudo-Code

Der PSO-Algorithmus berechnet nach der Initialisierung in jeder Iteration zunächst den Fitnesswert jedes Partikels und merkt sich die aktuelle Position und den Fitnesswert, falls dieser besser ist als der historisch Beste. Im Anschluss wird für jeden Partikel der beste Nachbar bestimmt (im Falle der Gesamtnachbarschaft wird nur einmal der beste Partikel des gesamten Schwarms bestimmt). Anschließend wird die Geschwindigkeit entsprechend der Formel 3 neu berechnet. Der neu berechnete Geschwindigkeitsvektor wird nun noch auf Vmax beschränkt, was verhindert, dass die Schritte der Partikel zu groß werden. Hierauf wird im nächsten Abschnitt genauer eingegangen.

```
LOOP UNTIL Kriterium                          // Übliche Abbruchkriterien sind
   FOR(i=1 to Partikelanzahl)                 // Interationsanzahl oder Fitnesswert
      f=Fitness(p_i)                          // Schranke
      IF(f > best_i) THEN DO                  // Die Fitness alle Partikel wir zunächst
         FOR(d=1 TO Dimension)                // bestimmt
            p_id=x_id
            best_i=f
         NEXT d
      END DO
   NEXT i

                                              // Der beste Nachbar eines Partikels einer
   FOR(i=1 to Partikelanzahl)                 // definierten Nachbarschaftsbeziehung
      g=i                                     // wird ermittelt
      FOR(j=Nachbarschafts_i)
         IF(best_j > best_g) THEN DO
            g=j
         END DO
      NEXT j

      FOR(d=1 TO Dimension)                   // Die Geschwindigkeiten werden unter der
         v_id=v_id+φ1(p_id-x_id)+φ2(p_gd-x_id) // Information des besten des
         IF(v_id > Vmax) THEN DO              // aktuellen Partikels und des besten
            v_id=Vmax                         // Nachbarpartikels neu berechnet
         END DO                               // Geschwindigkeitsbegrenzung auf Werte
         IF(v_id < -Vmax) THEN DO             // aus [-Vmax,Vmax]
            v_id=-Vmax
         END DO
         x_id=x_id+v_id                       // Update der Partikelposition
      NEXT d
   NEXT i
END LOOP
```

Abbilung 2.3: Der Partikel-Schwarm-Grundalgorithmus als Pseudo-Code

2.2.2 Anpassung für binäre Probleme

Ein Vorteil des PSO Algorithmus ist der, dass er durch eine kleine Änderung auch in der Lage ist, Probleme zu lösen, die binär formulierbar sind wie das „Travelling Sales Man Problem“. Hierfür werden potentielle Lösungen als Binärvektor codiert. Vor der eigentlichen Positionsadaption, deren Einträge jetzt nur noch den Wert Null oder Eins annehmen kann, wird mittels Anwendung der Sigmoidfunktion auf die Geschwindigkeit entschieden, ob der Wert zu Eins oder zu Null tendiert. Vmax kann beispielsweise auf 4.0 gesetzt werden[1], so dass s(Vmax) = 0.018 ergibt.

$$s(v_{id}) = \frac{1}{(1 - \exp(-v_{id}))}$$

Formel 4: Die Sigmoidfunktion

Der Algorithmus verändert sich wie folgt:

```
                    :
FOR(d=1 TO Dimension)          // Die Geschwindigkeiten werden unter der
  v_id = v_id + φ1(p_id - x_id) + φ2(p_gd - x_id)   // Information des besten des
  IF (v_id >> Vmax) THEN DO    // aktuellen Partikels und des besten
    v_id = Vmax                // Nachbarpartikels neu berechnet
  END DO                       // Geschwindigkeitsbegrenzung auf Werte
  IF (v_id << -Vmax) THEN DO   // aus [-Vmax,Vmax]
    v_id = -Vmax
  END DO
  IF (random(0,1) < s(v_id)) THEN DO   // Update der Partikelposition
    x_id = 1;
  ELSE
    x_id = 0;
  END DO
NEXT d
                    :
```

Abbildung 2.4: Der PSO-Grundalgorithmus als Pseudo-Code für binäre Probleme

Auf die binäre Version des PSO wird in dieser Arbeit nicht weiter eingegangen, da hier der Vergleich der PSO für kontinuierliche Probleme im Fokus steht. Für weiter führende Studien und zu einem Vergleich mit genetischen Algorithmen kann Quelle [1] genutzt werden.

2.3 Zufallsvariablen

Der Einfluss des Zufalls in der PSO ist entscheidend für den Erfolg, denn der Zufall kann die Konvergenz in ein lokales Optimum verhindern. Hinzu kommt, dass die zufällige Erkundung des Funktionsraums von diesen Variablen abhängt und somit die Qualität der Zufallszahlen essentiell ist. Bei der Implementierung ist daher darauf zu achten, dass der verwendete Zufallszahlgenerator qualitativ hochwertige Zufallszahlen im geforderten Bereich liefert.

2.3.1 Rechteck-Gleichverteilung

Die bisher verwendeten Zufallszahlen ϕ_1 und ϕ_2 sind gleich verteilt und spannen jeweils einen D-dimensionalen rechteckigen Lösungsraum auf. Der Lösungsraum ergibt sich aus der Summation der beiden Zufallszahlen, die jeweils aus einem D-Würfel mit der Kantenlänge c_i stammen. Bei der Summation der beiden gleich verteilten Zufallsvariablen kann nur dann weiterhin von einer Gleichverteilung ausgegangen werden, wenn $c_1 = c_2$ gilt. Die Variablen stammen folglich aus der gleichen Verteilung. Da in der Regel jedoch gilt $c_1 \sim c_2$, kann dies vernachlässigt werden. Jedoch kommt die Frage auf, inwiefern die Eckpunkte dieser Verteilung von Bedeutung sind. Da keine Informationen bekannt sind, welche Richtung bevorzugt zu wählen ist, ist die Signifikanz der Eck-Bereiche nach[2] fraglich. In Grafik [2.2] - hier erneut abgebildet - ist dies anschaulich für D=2 gezeigt. In diesem Fall wurde $c_1=c_2=2.0$ gewählt, so dass der hellgrüne aufgespannte Raum sich aus den doppelte Längen der beiden blauen Vektoren ergibt.

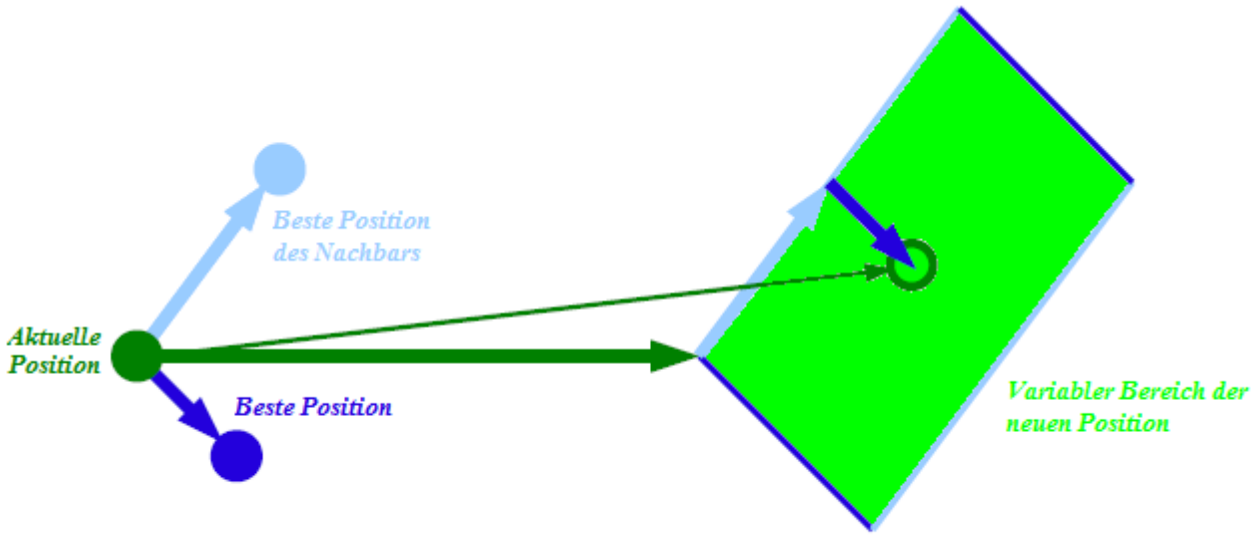


Abbildung 2.2: Grafische Visualisierung der Partikelbewegung

2.3.2 Kugel-Gleichverteilung

Die Idee hierbei ist, die Zufallsvariablen gleichmäßig in einer D-Kugel statt in einem D-Würfel zu verteilen, um so das zuvor erwähnte Problem der Signifikanz der Ecken zu eliminieren.

Der Radius wird so gewählt, dass die D-dimensionale Kugel das gleiche Volumen besitzt wie das D-dimensionale Rechteck. Die Zufallsvariablen setzen sich nach Formel 2 zusammen aus einem Lernfaktor und einer solchen Zufallsvariablen, die ein Volumen von 1 aufspannt. Ziel ist es nun, in Abhängigkeit der Dimension eine Zahl aus einer D-dimensionalen Kugel mit dem Volumen 1 zu ziehen.

Das Volumen einer D-dimensionalen Kugel mit Radius r berechnet sich wie folgt:

$$V_i = \frac{\pi^{D/2}}{D/2} r^D$$

Formel 5: Volumen eine D-dimensionalen Kugel

So dass sich nach Umformung der Gleichung mit $V_i = 1$ die folgende Berechnung für den Radius ergibt:

$$\text{Für } D \text{ gerade:} \quad r = \frac{(D/2)!^{1/D}}{\sqrt{\pi}} \qquad \text{Für } D \text{ ungerade:} \quad r = \frac{D!^{(1/D)} \pi^{1/2D}}{2\sqrt{\pi}}$$

Formel 6: Radiusberechnung einer D-dimensionalen Kugel mit dem Volumen 1

Man kann nun einen solchen Punkt innerhalb dieser D-Kugel berechnen, in dem man die Richtung eines Punktes durch eine Normalverteilung ermittelt und den Radius zufällig gleich verteilt wählt. Die genaue Berechnung ist dem Quellcode zu entnehmen.

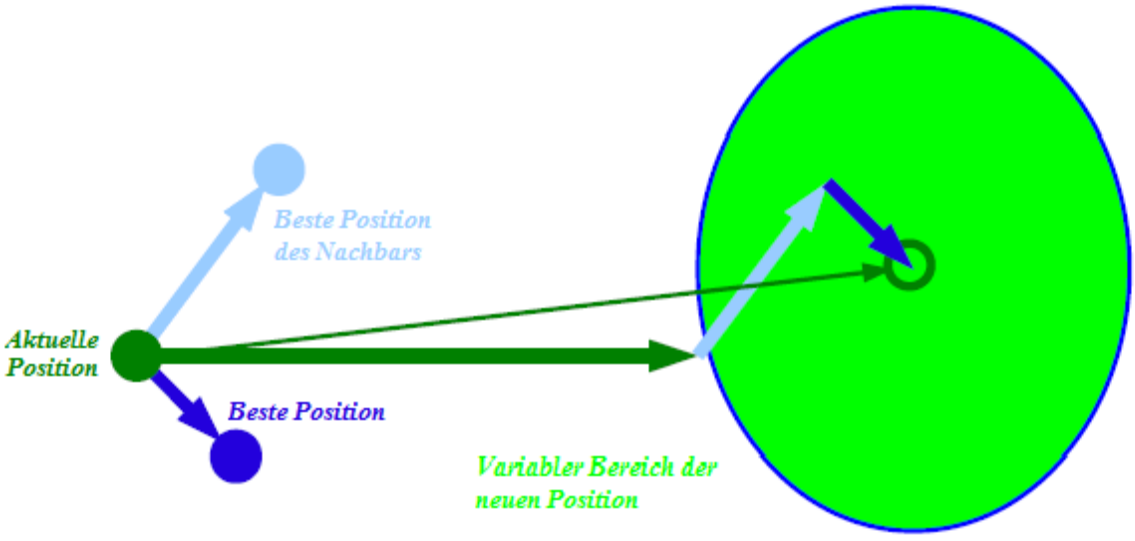


Abbildung 2.5: Grafische darstellung der Partikelbewegungung unter Nutzung einer Zufallsvariable einer D-Kugel

2.3.3 Normalverteilung

Sehr viele natürliche Prozesse laufen normal verteilt ab, was bedeutet, dass die Wahrscheinlichkeit einer größeren Änderung unproportional abnimmt.
Allgemein hat die Normalverteilung die folgende Dichtefunktion:

$$N(x)=\frac{1}{\sigma\sqrt{2\pi}}\exp\left(-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right)$$

Formel 7: Dirchtefunktion der Normalverteilung

Wählt man die folgenden Werte der Standartabweichung μ und der Varianz σ dann liegen 95% der Zufallsvariablen ϕ_1 und ϕ_2 im Bereich $[0,c_i]$ so dass der Wertebereich ähnlich dem der zuvor genannten Verteilungen ist.

$$\mu=c_i$$
$$\sigma=\frac{c_i}{4}$$

Abbildung 2.6: Mean und Varianz bei Verwendung der Normalverteilung

Maurice Clerc [2] merkt an, dass die Verteilung für binäre Probleme effektiv ist, jedoch für kontinuierliche Probleme relativ schlechte Ergebnisse liefert. Grafik [2.7] visualisiert die Verteilung nochmals für den zweidimensionalen Fall. Dabei ist zu beachten, dass die Normalverteilung im Wertebereich nicht beschränkt ist, wie in der Grafik vielleicht anzunehmen wäre, sondern dass der blaue Rand lediglich das 95%-quantil markiert.

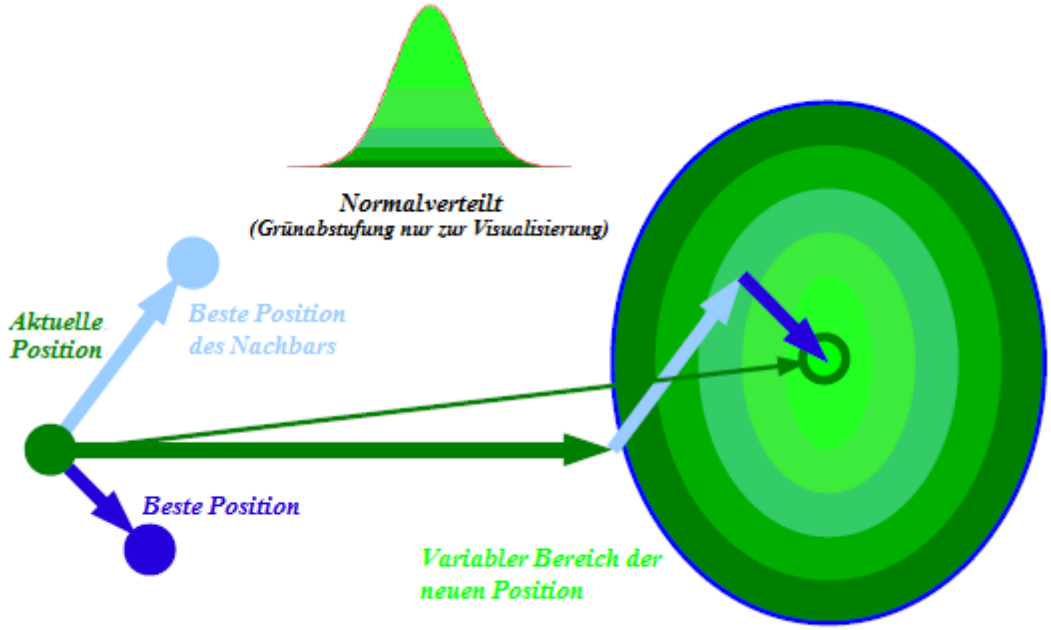


Abbildung 2.7: Grafische darstellung der Partikelbewegungung unter Nutzung einer Zufallsvariable der Normalverteilung

2.3.4 Pivot-Methode

Maurice Clerc [2] führt eine weitere Möglichkeit auf, bei der eine Zufallsverteilung für das PSO-Verfahren genutzt wird. Da diese Methode stark von der ursprünglichen Version abweicht, weil sie komplett ohne Geschwindigkeit auskommt, wird dieses Verfahren hier nur kurz beschrieben. Für eine genauere Einführung in diese Methode wird auf [2] verwiesen.

Die Idee bei dieser Methode ist es, jeweils eine D-Kugel-Verteilung (oder auch D-Kugel-, D-Normalverteilung) mit dem Mittelpunkt der eigenen besten Position *o* und der besten Position des besten Nachbarn *g* zu erzeugen. Dabei entspricht der Radius beider Verteilungen der Distanz zwischen beiden Punkten. Aus beiden Verteilungen wird jeweils ein zufälliger Punkt gewählt und entsprechend seiner Signifikanz bezüglich des Problems gewichtet (siehe Formel 8). Die Summe beider Punkte - wiederum gewichtet mit den Lernfaktoren c_1 und c_2 - ergibt schließlich den neuen Punkt. Formel 8 illu-striert dies, indem die beiden Punkte in direkte Abhängigkeit zum Fitnesswert gewählt werden. Die neue Position des Partikels entsteht also ohne Geschwindigkeitsadaption lediglich aus der vom Zufall beeinflussten Rekombination von *o* und *g*.

$$w_1 = \frac{Fitness(o)}{Fitness(o) + Fitness(g)}$$

$$w_2 = \frac{Fitness(g)}{Fitness(g) + Fitness(o)}$$

$$x \Leftarrow c_1 w_1 Random(Ellipsoid(o)) + c_2 w_2 Random(Ellipsoid(g))$$

Formel 8: Berechnung der neue Partikelposition durch die Pivot-Methode

2.4 Beschränkung des Schwarms

Die Berechnung der Geschwindigkeit eines jeden Partikels wird durch vom Zufall beeinflusstes Aufsummieren erreicht. Daraus resultiert, dass sich die Partikel unkontrolliert bewegen. So kann der gewünschte Zustand, dass die Partikel zyklisch auf einen Punkt konvergieren, ebenso eintreten, wie die oszillierende Entfernung von diesem. In diesem Zustand pendeln die Partikel um einen Punkt abhängig vom historischen eigenen und dem besten der Nachbarschaft - und zwar mit immer weiter werdenden Bahnen [1]. Die Partikel verlassen so meist schnell den Definitionsbereich und sind somit für das Problem kontraproduktiv. Es ist demnach nötig, die Partikel zu beschränken d.h. sie zu veranlassen, sich im sinnvollen Teil des Definitionsbereiches zu bewegen, in dem sie schließlich auch zum Optimum konvergieren sollten.

2.4.1 Beschränkung auf den Definitionsbereich

Es kommt während des Algorithmus häufig vor, dass Partikel den Definitionsbereich verlassen und somit nicht mehr konstruktiv zur Identifizierung des Optimums beitragen. Eine Möglichkeit den Partikel zu veranlassen, den Definitionsbereich aufzusuchen, besteht darin, den Fitnesswert aller derjenigen Partikel auf den schlechtesten Wert zu setzen, die in mindestens einer Dimension den Definitionsbereich verlassen haben. Allerdings benötigt der Partikel, beeinflusst von besseren Partikeln innerhalb des Definitionsbereiches, meist mehre Iterationen, um den Definitionsbereich wieder zu betreten. Daher ist es sinnvoll, die Position des Partikels auf die Grenze des Definitionsbereiches zu verlagern und die Geschwindigkeit auf Null zu setzen.

$$x_d \notin [Dmin_d, Dmax_d] \Rightarrow \begin{array}{r} v_d = 0 \\ x_d < Dmin_d \Rightarrow x_d = Dmin_d \\ x_d > Dmax_d \Rightarrow x_d = Dmax_d \end{array}$$

Formel 9: Beschränkung der Partikel auf den Definitionsbereich

Eine weitere Methode, die ohne das Setzen der Geschwindigkeit auskommt, wird in [15] vorgestellt. Hier wird die Position x_d beim Verlassen des Definitionsbereiches auf einen zufälligen Wert innerhalb des Definitionsbereiches gesetzt. Allerdings muss das Verlassen einer Dimension nicht unbedingt bedeuten, dass dies nicht die richtige Richtung gewesen ist, sondern nur die Geschwindigkeit zu hoch war.

$$x_d \notin [Dmin_d, Dmax_d] \Rightarrow \begin{array}{l} x_d < Dmin_d \Rightarrow x_d = Dmin_d + random(0, Dmax_d - Dmin_d) \\ x_d > Dmax_d \Rightarrow x_d = Dmax_d - random(0, Dmax_d - Dmin_d) \end{array}$$

Formel 10: Zufälliges Setzen der Partikel-Dimension beim Verlassen des Def. Bereichs

2.4.2 Beschränkung der Geschwindigkeit

Die erste und einfachste Möglichkeit, die Partikel in ihrer Geschwindigkeit zu beeinflussen, ist die bereits im Algorithmus beschriebene Geschwindigkeitsbegrenzung durch Vmax. Diese gewählte Konstante beschränkt die Partikel in ihrem Bewegungsfreiraum und sorgt so für eine Suche innerhalb des Definitionsbereichs.

$$IF\,(v_{id} > Vmax_d)\;\; THEN$$
$$v_{id} = Vmax_d$$
$$IF\,(v_{id} < -Vmax_d)\;\; THEN$$
$$v_{id} = -Vmax_d$$

Formel 11: Geschwindigkeitsbeschränkung durch Vmax

Die Wahl von Vmax beeinflusst dabei die Konvergenzgeschwindigkeit. Ist Vmax zu groß, so bewegen sich die Partikel zu oft außerhalb des Definitionsbereichs. Ist Vmax zu klein, können die Partikel nicht mehr aus lokalen Optima entweichen. Kennedy & Eberhardt[1] empfehlen aus diesem Grund, in [1][9] in jeder Dimension Vmax als Hälfte des jeweiligen Definitionsbereiches zu wählen.

$$Vmax_d = \frac{|(Dmin_d)| + |(Dmax_d)|}{2}$$

Formel 12: Sinnvolle Wahl von Vmax

2.4.3 Konvergenz-Methoden

Es ist sinnvoll, den gesamten Schwarm im Verlauf der Optimierung in ein identifiziertes Optimum konvergieren zu lassen, so dass dieser Punkt zum Ende hin exakter bestimmt wird. Um die Konvergenz des Schwarms zu garantieren, d.h. von der explorativen Suche zur lokalen Optimierung überzugehen, sind verschiedene Methoden untersucht worden, die im folgenden Abschnitt beschrieben werden.

2.4.3.1 Die „Inertia weight“

Kennedy & Shi [1][9] studierten ein von ihnen entwickeltes Konvergenzkriterium namens „Inertia Weight“, was übersetzt soviel heißt wie Trägheitsgewicht. Dieses Gewicht α wird im Verlaufe der Iterationen reduziert, so dass der Einfluss der alten Geschwindigkeit reduziert wird und eine Bewegung in Richtung der beiden besten Po-

sitionen verstärkt wird. Der resultierende Effekt ist bei sinnvoller Wahl von α der, dass die Partikel konvergieren und zusätzlich kein Vmax mehr benötigt wird. So konvergieren die Partikel mit der Zeit gegen ein vom Schwarm identifiziertes Optimum.

$$v_{id} = \alpha v_{id} + \varphi_1 (p_{id} - x_{id}) + \varphi_2 (p_{gd} - x_{id})$$

Formel 13: Die "inertia weight" Konvergenz-Methode

Die Idee von Kennedy & Shi war es, den Algorithmus zu Beginn explorativ suchen zu lassen und im Verlaufe der Zeit den Bereich der besten gefunden Punktes näher zu untersuchen. Eine lineare Reduzierung von α von 0.9 auf 0.4 über eine festgelegte Iterationsanzahl hat sich nach [09][15] als sinnvoll erwiesen und garantiert die Konvergenz der Partikel, indem der Einfluss der historischen Geschwindigkeit reduziert wird. Allerdings kann nicht gesagt werden, wie lange die explorative Phase dauern sollte. Daher ist es schwierig, diejenige optimale Iterationsanzahl zu wählen, in der der Faktor verringert wird.

2.4.3.2 Die „Global Local Best Inertia weight“

Wenn man das PSO-Verfahren anwendet und als Abbruchkriterium nicht eine festgelegte Iterationsanzahl ansetzt, sondern das Unterschreiten eines Fitnesswertes, so kann man keine Aussagen über die Iterationsanzahl treffen und wählt die inertial weight eventuell schlecht. In [10] wird eine neue Variante der inertial weight vorgestellt, die auf Basis des Quotienten des global besten Wertes und des lokal besten Wertes des aktuellen Partikels die inertial weight berechnet. Der Vorteil dabei ist, dass der Faktor nicht mehr von der Iteration abhängt und individuell für jeden Partikel entschieden wird, ob explorativ oder lokal gesucht werden soll.

$$\alpha_i = 1.1 - \left(\frac{GlobalBest}{LocalBest_i}\right)$$

Formel 14: Die "global-local-inertia wieght" Konvergenz-Methode

2.4.3.3 Der „Constriction Factor“

Ein weiteres Konvergenzkriterium ist der „Constriction Koeffizient“ von Maurice Clerc [2]. Dieser hat den Vorteil , dass er als Konstante gewählt werden kann und wie folgt Einfluss nimmt:

$$v_{id} - \chi(v_{id} + \varphi_1(p_{id} - x_{id}) + \varphi_2(p_{gd} - x_{id}))$$

Formel 15: Berechnugn der Geschwindigkeit mit "Constriction Factor"

Wobei der Faktor wie folgt berechnet wird:

$$\chi = \frac{2\kappa}{\left(\varphi - 2 + \sqrt{\varphi^2 - 4\varphi}\right)}$$

Formel 16: Berechnugn des "Constriction Factors"

Die Wahl von k und $\varphi = \max(\varphi_1 + \varphi) = c_1 + c_2$ garantiert hierbei die Konvergenzsicherheit des Algorithmus, in dem sichergestellt wird, dass die Eigenwerte der Geschwindigkeitsvektoren kleiner 1 sind und diese somit nicht gestreckt werden. In [2] wird gezeigt, dass die inertial weight und der constriction Factor bei entsprechender Wahl der Werte mathematisch übereinstimmen. Im Allgemeinen hat sich nach [19][2] zum Beispiel die Wahl von k=1 und $\varphi = 4.1$ in Formel 16 als sinnvoll erwiesen, wodurch man $\chi \sim 0.73$ erhält.

Kennedy und Eberhardt schlagen vor, zusätzlich eine Geschwindigkeitsbegrenzung Vmax zu verwenden. So werden insbesondere zu Beginn des Verfahrens die Schrittweite der Partikel in sinnvollen Größen gehalten

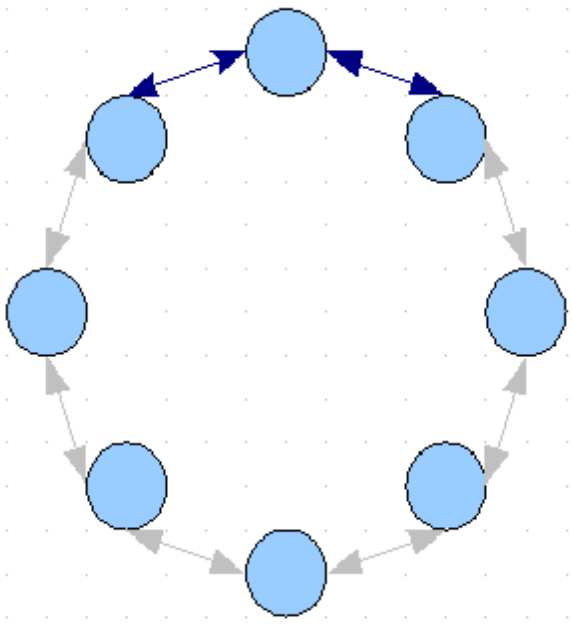
2.5 Nachbarschaftsbeziehungen

In der ursprünglichen PSO wurde jeder Partikel neben seiner historisch besten Position auch von der historisch besten Position des gesamten Schwarms informiert. Diese gBest-Methode (global Best) veranlasst den Schwarm, schnell ein identifiziertes Optimum aufzusuchen und birgt somit die Gefahr, in ein lokales Optimum zu konvergieren. Im Kontrast zur gBest-Methode entwickelten K&E [1] die lBest-Methode (local best), bei der der aktuelle Partikel sich neben seiner historisch besten Position auf die historisch beste Position einer gewählten Nachbarschaft bezieht. Die lBest-Methode bietet dem Schwarm mehr Sicherheit, das globale Optimum zu finden, da sich die Informationen des global besten Partikels langsamer im Schwarm verteilen. Mit kleineren Nachbarschaften jedoch sinkt auch die Konvergenzgeschwindigkeit.

2.5.1 Ring-Topologie

Die Ring-Topologie verbindet die Partikel zu einem Ring, in dem jeder Partikel p_i k Nachbarn hat, aus denen der beste Partikel ausgewählt wird, wobei p_i zu dieser Nachbarschaft selbst hinzuzählt. Diese Topologie ist die lBest-Methode bzw. - wenn die Nachbarschaftsgröße k gleich der Schwarmgröße ist - die gBest-Methode.

Eine Nachbarschaftsgröße von K=3 hat sich als sinnvoll erwiesen, insbesondere wenn man keine Informationen über die Fitnessfunktion hat. So wählt das aktuelle Partikel das beste Partikel aus seinem rechten und linken Nachbarn sowie sich selbst als Informanten aus.



Ring-Topologie

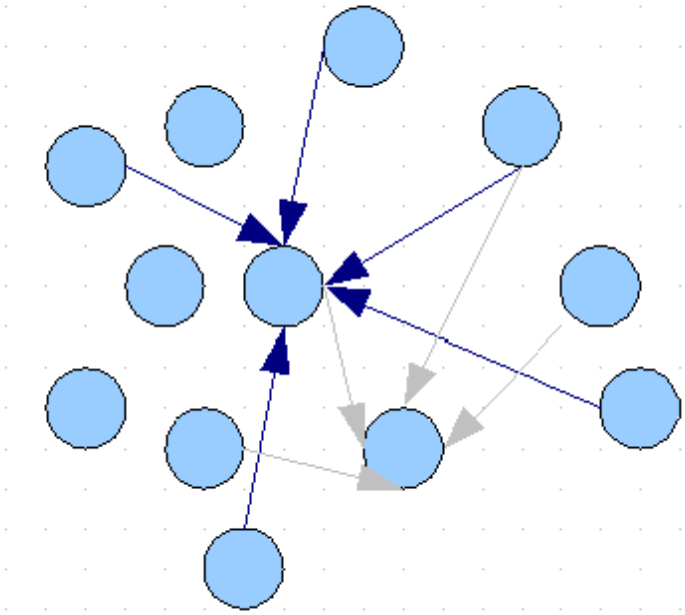
Abbildung 2.9: Die Ring-Nachbarschaftsbeziehung

2.5.2 Random-Topologie

In Anlehnung an die Ring - Topologie kann es sinnvoll sein, die k Nachbarn zufällig im Kontrast zu einer fixen Topologie auszuwählen. Ein Vorteil besteht in der Tatsache, dass jeder Partikel die Chance hat, in diesem Iterationsschritt direkt durch den momentan besten Partikel informiert zu werden. Im Kontrast zu den fixen Topologien verbreitet sich die Information nicht kontinuierlich durch den Graphen, sondern ist durch den Zufall beeinflusst. Der Schwarm ist weniger anfällig für lokale Minima.

Die Zufällige Nachbarschaft hat sich nach [2] als robust erwiesen was Kapitel 5 bestätigt.

$$p_{lbest(k=3)} = best(p_{r1}, p_i, p_{r2})$$
$$r1, r2 = rand(0..n)$$
$$r1 \neq r2$$



Random-Topologie

Abbildung 10: Die Zufall-Nachbarschaftsbeziehung

2.5.3 Wheel-Topologie

Die auch unter dem Namen Queen- oder Star-Topologie bekannte Nachbarschaftsbeziehung hat ein Partikel, welcher nach der gBest-Methode seine Geschwindigkeit adaptiert. Alle anderen Partikel orientieren sich an diesem Partikel. Die Idee dabei ist, einem Partikel eine Rolle als „Führungsposition“ zuzuweisen, dem die anderen Partikel langsam folgen.

$$p_{informer} = gbest$$
$$P_i = best(p_{informer}, p_i)$$

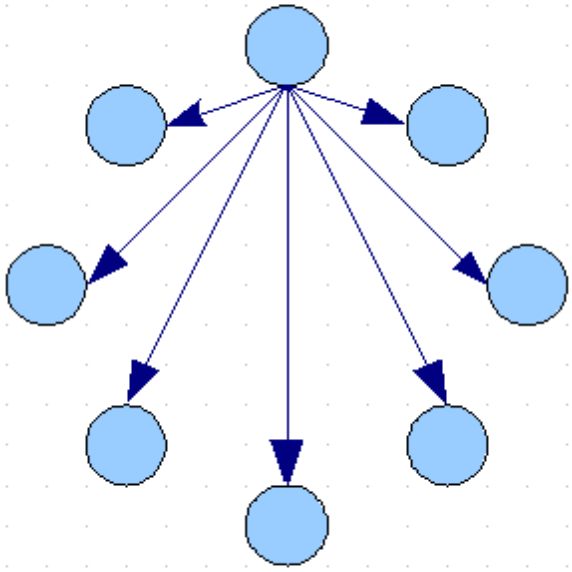


Abbildung 11: Die Wheel-Nachbarschaftsbeziehung

2.5.4 Neumann-Topologie

In der „von Neumann“ Nachbarschaft sind die Partikel in einem quadratischen Raster der Größe $\sqrt{N} * \sqrt{N}$ angeordnet. Jeder Partikel wählt als Informant aus seinen unmittelbaren Nachbarn über, unter, links und rechts von ihm denjenigen mit dem besten Fitnesswert oder aber sich selbst, wenn diese Nachbarn einen geringeren

$$p_g = BEST(p_{MOD(i-1,N)}, p_{MOD(i+1,N)}, p_i, p_{MOD(i+\sqrt{N},N)}, p_{MOD(i-\sqrt{N},N)})$$

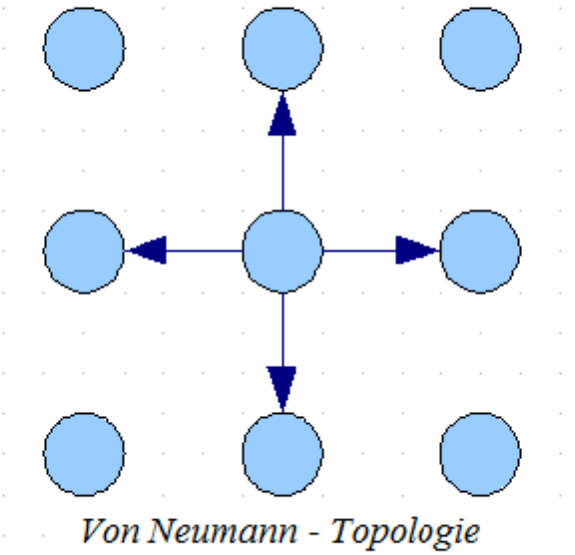


Abbildung 12: Die Von-Neumann-Nachbarschaftsbeziehung

Fitnesswert aufweisen.

Das Muster muss nicht zwangsläufig quadratisch sein, jedoch wird so eine gleichmäßige Informationsverbreitung erreicht. In dieser Topologie kann sich die Information über zwei Dimensionen verbreiten. Die von Neumann-Topologie liegt somit in Konvergenz und Sicherheit zwischen dem Gbest- und Lbest-Modell und bietet somit eine ausgewogene Kombination aus Konvergenzgeschwindigkeit und Sicherheit.

2.5.5 Distanz-Topologie

Obwohl eine Nachbarschaftsbeziehung, die auf der Entfernung der Partikel basiert, wohl der Informationsverbreitung eines Vogelschwarms am nächsten kommt, ist mir keine Arbeit bekannt, die solche Topologien untersucht. Insbesondere in Kombination mit fixen Topologien kann die Distanz für die Partikel ein weiteres Entscheidungskriterium darstellen.

Man definiert eine Distanzfunktion für zwei Partikel p_1 und p_2. In diesem Fall beispielsweise die euklidische Distanz:

$$\delta(p_1, p_2) = \sqrt{\sum_{d=1}^{D} (p_{1d} - p_{2d})^2}$$

Formel 15: Euklidische Distanz

Die Idee dabei ist, nun jedem Partikel verstärkt die Möglichkeit zur Erkundung seiner nächsten Umgebung zu geben. Man bestimmt also zunächst die Partikel des Schwarms oder einer fixen Nachbarschaft, die besser sind als der aktuelle Partikel. Als Nachbar wird nun nicht direkt der beste Partikel gewählt, sondern derjenige Partikel, welcher eine bessere Fitness aufweist und die geringste Distanz zu dem aktuellen Partikel hat.

Der Berechnungsaufwand der euklidischen Distanz ist ungleich höher als der bei fixen Nachbarschaften, erweist sich aber ebenfalls als eine brauchbare Nachbarschaftsbeziehung wie Kapitel 5 zu entnehmen ist.

2.5.6 Dynamische-Topologien

Je mehr Informanten man pro Partikel wählt, desto höher ist die Konvergenzgeschwindigkeit, allerdings auch die Gefahr, in ein lokales Optimum zu konvergieren. So liegt es nicht fern zu versuchen, die jeweils passende Informantenanzahl und Struktur zu wählen.

Eine Möglichkeit wird in [11] untersucht, indem gBest, lBest(3) und von Neumann Nachbarschaften kombiniert werden. Dies geschieht mit dem Ziel, stets die optimale Topolgie zu wählen und somit ein Optimum aus Sicherheit und Konvergenzge-

schwindigkeit zu erlangen. Allerdings setzt der Autor diese Idee unzureichend um, denn er vergleicht nicht die Fitnessauswertungen pro Iteration, sondern die Iterationszahl. Der hybride Ansatz hat drei mal so viele Fitnessauswertungen, da der Autor zunächst jede Topologie berechnet, dann den Schwarm temporär mittels der jeweiligen Topologie „updatet“ und schließlich entscheidet, welche der Topologien die beste ist.

Geht man davon aus, dass keine Informationen über die Struktur der Fitnessfunktion zur Verfügung stehen, so ist es sehr schwierig, eine Aussage über den Informationsbedarf eines Partikels zu treffen; denn man weiß ja nicht, ob in der Nähe eines Partikels mit schlechter Fitness eventuell das Optimum liegt.

Mit der Absicht, stets die beste Nachbarschaftsgröße auszuwählen, kann diese kontinuierlich vergrößert werden, um zu Beginn den Schwarm explorativer suchen zu lassen und gegen Ende die Konvergenz gegen das identifizierte Optimum zu erhöhen. Eine lineare Erhöhung (grüne Kurve) der Nachbarschaftsgröße von 1 auf n-1 über die Iterationszahl I erwies sich als sinnvoll. Die Kurve kann jedoch positiv oder negativ gekrümmt werden, womit sich das Verhalten wieder dem lBest bzw. gBest annähert. Diese Form der Nachbarschaft erwies sich als ebenso sicher wie die lBest(k=3) Methode und ebenso schnell wie die gBest-Methode. Die diesbezüglichen Beobachtungen wurden in [14] bestätigt.

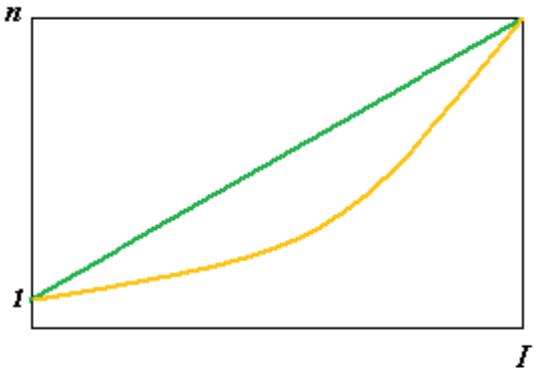


Abbildung 13: Nachbarschaftsgrößen-Verlauf dynamischer Nachbarschaften

2.5.7 Memory-Swarm-Topologie

Maurice Clerc behandelt in seinem Buch [2] eine Variante der Topologie, in der er den Schwarm in „Memories“ und „Informer“ unterteilt. Die Idee dabei ist, die sich schnell und explorativ bewegenden „Informer“ als Informanten für die trägen „Memories“ zu nutzen. Ein Problem, warum diese Variante im klassischen PSO-Verfahren keine Anwendung findet, ist die Tatsache, dass die Schwärme sehr groß werden. Eine adaptive Version von PSO namens TRIBES von Maurice Clerc [2] nutzt diese

Topologie, wobei das Verfahren mit wesentlich kleineren Schwärmen auskommt, da die Schwarmgröße über einen Selektionsprozess angepasst wird.

Der TRIBES Algorithmus wird später nur schematisch aufgezeigt, denn die detaillierte Behandlung und Implementierung würde den Umfang dieser Arbeit sprengen.

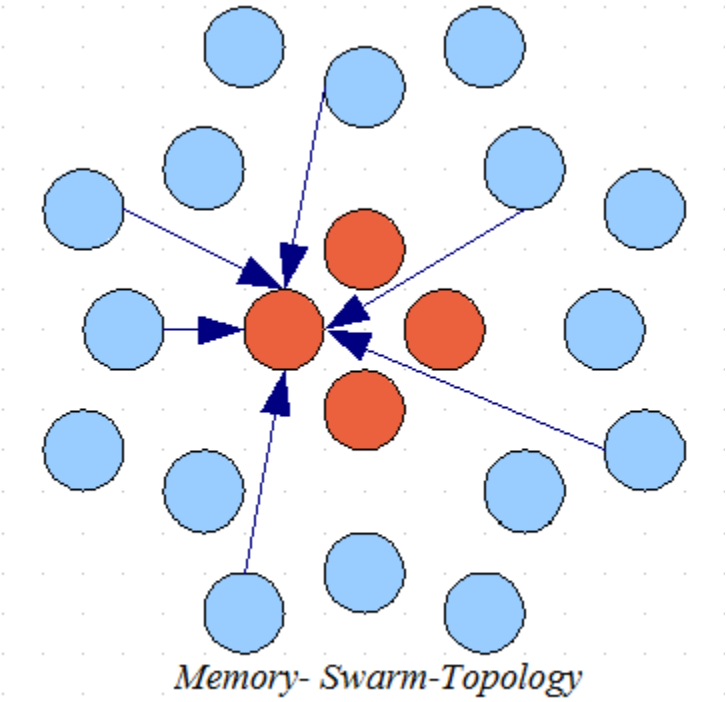


Abbildung 14: Memory-Swarm-Nachbarschaftsbeziehung

3 Adaptive Erweiterungen

Ein Problem des PSO-Verfahrens ist, dass es nicht adaptiv ist. Dies bedeutet, dass die Wahl der Varianten, Parameter und Topologien stark vom Problem abhängt. In vielen Fällen hat man jedoch keine Information über die Struktur der Fitnessfunktion und kann somit keine Aussage über die optimale Wahl der Parameter treffen.

So wäre es von Vorteil, wenn das Verfahren so erweitert werden könnte, dass es sich während der Optimierung individuell dem Problem anpasst.

In [15] wird eine adaptive Version der PSO „LA-PSO“ präsentiert, das auf zwei Variationen basiert.

Aus der Beobachtung resultierend, dass das Zentrum der Population zentrale Bedeutung für evolutionäre Algorithmen hat, werden der „Fully-Informed Particle Swarm“ und eine eigene Variante namens MB-PSO vorgestellt.

Da der Schwarm in multimodalen Funktionen häufig in lokale Optima konvergiert, entstand ein neuer Ansatz namens ESC-PSO.

Im Anschluss wird aus Umfangsgründen kurz das Verfahren „Tribes“[2] von Maurice Clerc beschrieben, welches auf PSO basiert.

3.1 Landscape Adaptive PSO

Die Autoren versuchen das PSO-Verfahren zu erweitern, indem sie zwei zusätzliche Möglichkeiten bieten, lokalen Minima zu entkommen.

Mittels „Crossing Over“ , sowie einer Einschränkung des Suchbereichs durch einen „Distribution-Vector“ soll so die Effizienz des Verfahrens erhöht werden.

3.1.1 „Crossing over“

Im LAPSO wird dem Partikel durch einen ähnlichen Einfluss des Zufalls die Möglichkeit gegeben, ein „Crossing Over“ durchzuführen, indem nicht die Differenz zur aktuellen Position gebildet wird, sondern die Summe. Die neue Position wird also nicht beschränkt und liegt in der Verlängerung der aktuellen Position.

$$v_{id} = v_{id} + \varphi_1 (p_{id} + x_{id}) + \varphi_2 (p_{gd} + x_{id})$$

Formel 16: "Crossing Over" im LAPSO

3.1.2 „Distribution vector“

Die Autoren merken an, dass der Originalalgorithmus insbesondere bei unsymmetrischen Fitnessfunktionen nicht auf diese Asymmetrie bei der Beschränkung eingeht, sondern alle Dimension gleich beschränkt. Betrachtet man eine asymmetrische Funktion wie in Grafik [15], so sollte es sinnvoll sein, Dimensionen, in denen die Partikel über einen großen Bereich verteilt liegen, stärker zu beschränken als diejenigen, in denen die Partikel dicht zusammen liegen.

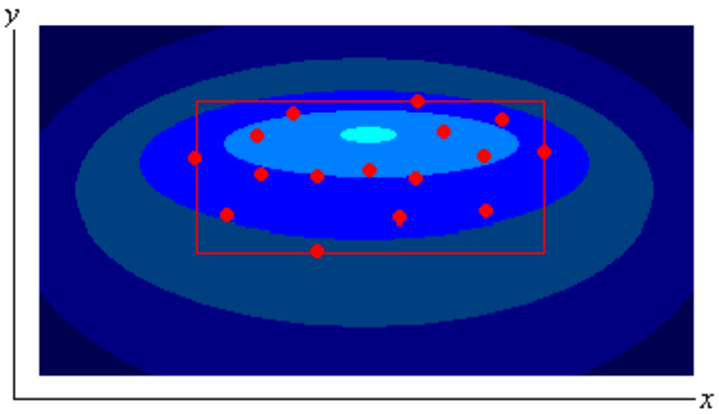


Abbildung 15: Darstellung des „Distribution vectors"

Die Autoren führen einen Distributionsvektor *DV* ein, der diese adaptierte Beschränkung für jede Dimension berechnet, indem zunächst der maximale Abstand innerhalb einer Dimension zwischen allen Partikeln berechnet wird. Danach wird das Verhältnis auf den Abstandsbetrag normiert, so dass der Vektor nur Werte zwischen Null und Eins annimmt.

$$DV_d = \frac{max_{n=1}^{N}(x_{nd}) - min_{n=1}^{N}(x_{nd})}{\left|(max_{n=1}^{N}(x_{nd}))\right| + \left|(min_{n=1}^{N}(x_{nd}))\right|}$$

Abbildung 16: Berechnung des „Distribution vectors" in LAPSO

Die Autoren beschreiben außerdem, das Werte nahe an Eins liegend einen Hinweis auf weite durchschnittliche Entfernungen zwischen den Partikeln in dieser Dimension anzeigen. Kleinere Wert weisen demnach auf kleine Entfernungen hin.

Entsprechend dem „Constriction Factor“ beschränkt der „Distribution Vector“ nun die Geschwindigkeit wie folgt:

$$v_{id} = DV_d \left(v_{id} + \varphi_1 (p_{id} - x_{id}) + \varphi_2 (p_{gd} - x_{id}) \right)$$

Abbildung 17: Geschwindigkeitsberechung in LAPSO

Befinden sich die Partikel in einem lokalen Optimum, so wird der Vektor sehr klein, wodurch die Möglichkeit gegeben werden muss, aus diesem lokalen Optimum zu entkommen. Ebenso muss ein divergierender Zustand des Schwarms vermieden werden. Gemäß der Betrachtung der „inertial weight“ und des „Constriction factor“ wählen die Autoren $Dv_{max} = 0.95$ und $Dv_{min} = 0.4$. Der Wert des Vektors wir beim Überschreiten auf die untere Schranke, beim Unterschreiten auf die obere Schranke gesetzt.

$$DV_d = \begin{cases} Dv_{min} : & DV_d > Dv_{max} \\ Dv_{max} : & DV_d < Dv_{min} \\ DV_d : & sonst \end{cases}$$

Abbildung 18: Beschränkung des Distribution Vectors

An dieser Stelle soll bereits angemerkt werden, dass der Schwarm meist relativ gleichmäßig verteilt ist. So ergibt die Berechnung des DV sehr häufig 1, was zur Verwendung des Faktors 0.4 führt. Es ist anzunehmen, dass das Verfahren vom Nullpunkt befangen ist; denn sei max = 10 und min= -10 ,so ist der DV 1 → 0.4 , da der Schwarm symmetrisch in Bezug auf den Nullpunkt liegt.

Ist max = 10 und min= 5 oder max = -5 und min= -10, so ist der DV 1/3 → 0.95.

Somit wird der Schwarm in der aktuellen Dimension immer dann stark beschränkt, wenn die Ausdehnung in positiver und negativer Richtung gleich ist. In allen anderen Fällen findet eine geringe Beschränkung statt. Bei einer symmetrischen Initialisierung im Funktionsraum ist daher anzunehmen, dass das Verfahren schnell gegen den Nullpunkt konvergiert. Verschiebt man das Optimum, so sollte das Verfahren jedoch nicht in der Lage sein, dieses zu finden.

3.1.3 LA-PSO Pseudo-Code

```
LOOP UNTIL Kriterium
  FOR(i=1 to Partikelanzahl)
    f = Fitness(p_i)
    IF(f > best_i) THEN DO
      FOR(d=1 TO Dimension)
        p_id = x_id
        best_i = f
      NEXT d
    END DO
  NEXT i
  FOR(i=1 to Partikelanzahl)
    g = i
    FOR(j = Nachbarschafts_i)
      IF(best_j > best_g) THEN DO
        g = j
      END DO
    NEXT j
    CALCULATE DV
    r = random(0,1)
    FOR(d=1 TO Dimension)
      IF(r < 0.1) THEN DO
        v_id = v_id + φ1(p_id + x_id) + φ2(p_gd + x_id)
      ELSE
        v_id = DV_d(v_id + φ1(p_id - x_id) + φ2(p_gd - x_id))
      x_id = x_id + v_id
      IF(x_id > Dmax) THEN DO
        x_id = Dmax - random(0.1)(Dmax - Dmin)
      END DO
      IF(x_id < Dmin) THEN DO
        x_id = Dmin + random(0.1)(Dmax - Dmin)
      END DO
    NEXT d
  NEXT i
END LOOP
```

Abbildung 16: Landscape Adaptive Pseudo-Code

3.2 Fully Informed Particle-Swarm

In den bereits vorgestellten Versionen der Partikel-Schwarm-Optimierung orientierten sich die Partikel stets an ihrer eigenen besten Position und an der besten Position des besten Nachbarn.

Diese selektive Orientierung hat jedoch den Nachteil, dass sie ausschließlich die Informationen von zwei Partikeln nutzt, obwohl nicht ausgeschlossen werden kann, dass die Informationen der anderen Partikel nicht ebenfalls von Bedeutung sind.

Kennedy, Mendes und Neves untersuchten in [12] eine Methode names „Fully-Informed-Particle-Swarm“ - kurz „FIPS“ - in dem jeder Partikel sich nicht an seinem besten Nachbarn orientiert, sondern ein gewichtetes Mittel aus all seinen Nachbarn und sich selbst nutzt. Die Geschwindigkeitsadaption sieht demnach wie folgt aus -wobei P_{md} ein gewichtetes Mittel aus P_i und dessen gesamter Nachbarschaft darstellt:

$$\begin{aligned} \varphi &= \varphi_1 + \varphi_2 \\ v_{id} &= v_{id} + \varphi(p_{md} - x_{id}) \\ x_{id} &= x_{id} + v_{id} \end{aligned}$$

Formel 19 : FIPS-Geschwindigkeitsberechnung

Eine Möglichkeit bestünde darin, das gewichtete arithmetische Mittel der Nachbarschaftspartikel zu bilden. Aber auch eine gewichtete Verrechnung von Fitnesswert und euklidischer Distanz stellt eine Möglichkeit dar. Eine detaillierte Beschreibung dieser Version ist Quelle [12] zu entnehmen.

3.3 Mean-Based-PSO

Evolutionäre Algorithmen nutzen die Gesamtinformationen der Population, indem sie meist ein gewichtetes Mittel der Population bilden und anschließend um diesen Punkt normalverteilt die mutierten Individuen der Folgegeneration bilden.

So kann das gewichtete Mittel des Schwarmes als Punkt verstanden werden, gegen den dieser in der aktuellen Iteration konvergiert. Allerdings muss dieser Punkt nicht zwangsläufig einen positiven Einfluss auf den gesamten Schwarm haben. Schlimmer noch - er kann den Schwarm sogar im Sinne der Optimierung fehl leiten. Es ist daher fraglich,ob das generelle Folgen dieses Punktes sinnvoll ist.

Meine Idee besteht nun darin, dem Schwarm lediglich die Möglichkeit zu geben, seinem gewichteten Mittel zu folgen. Hierfür wird stets die Position der Partikel mit der schlechtesten Fitness durch das gewichtete Mittel des Schwarms ersetzt. Hierzu wird die Geschwindigkeit des Partikels auf Null gesetzt, und die historisch beste Position

bleibt unverändert. So bleibt dem Partikel die Möglichkeit offen, diesen Punkt wieder anzustreben, falls das gewichtete Mittel schlechter sein sollte.

Es ergibt sich folgende Berechnung des linear gewichteten Mittels P_{mean} .

$$P_{mean} = \frac{2}{n\,(n+1)}\left(n\ \vec{P}_0 + (n-1)\ \vec{P}_1 + \ldots + 2\ \vec{P}_{n-2} + 1\ \vec{P}_{n-1} \right)$$

$$Fitness(\vec{P}_i) \leq Fitness(\vec{P}_{i+1}) \ \forall\ \vec{P}_i$$
$$i \in \mathbb{N} \ \wedge\ 0 \leq i < n$$

Formel 20 : Die Berechnung des gewichteten arithmetischen Mittels

Die Partikel sind jeweils mit ihren Gewichten markiert, die entsprechend Formel 20 umso größer sind, je kleiner der Fitnesswert ist.

Allerdings soll auch hier angemerkt werden, dass das Verfahren vermutlich von der Symmetrie der Funktion abhängt. Je stärker das Optimum der Funktion verschoben ist, desto weniger signifikant ist der gemittelte Schwarm.

Im Beispiel von Abbildung 3.1 wird zunächst der schlechteste Partikel durch das gewichtete Mittel ersetzt. In diesem Fall orientieren sich die Partikel am gewichteten Mittel - hier mit M markiert - da dieses den besten Fitness wert besitzt.

$$\vec{P}_{mean} = \frac{1}{10} * \left(4 * \begin{pmatrix} 14 \\ 11 \end{pmatrix} + 3 * \begin{pmatrix} 7 \\ 15 \end{pmatrix} + 2 * \begin{pmatrix} 2 \\ 10 \end{pmatrix} + 1 * \begin{pmatrix} 18 \\ 17 \end{pmatrix} \right) = \begin{pmatrix} 9.9 \\ 12.6 \end{pmatrix}$$

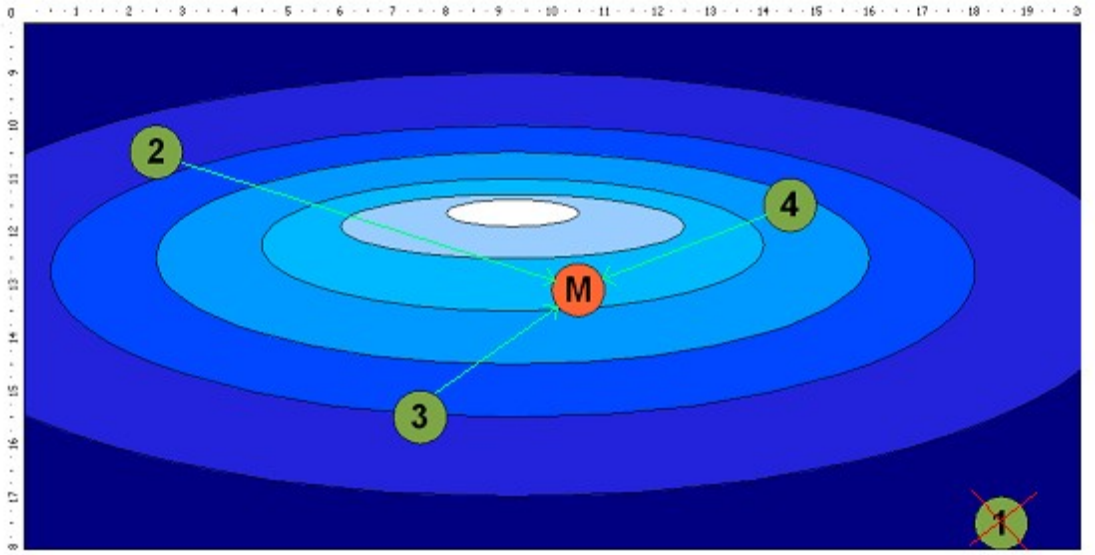


Abbildung 3.1 : Partikel folgen dem gewichteten Mittel

Angenommen, das gewichtete Mittel besitzt nicht den besten Fitnesswert der Nachbarschaft, so wählen die Partikel entsprechend des klassischen Ansatzes den besten Partikel als Orientierung, wie Grafik 3.2 zu entnehmen ist.

$$\vec{P}_{mean} = \frac{1}{10} * \left(4 * \begin{pmatrix} 11 \\ 11 \end{pmatrix} + 3 * \begin{pmatrix} 14 \\ 14 \end{pmatrix} + 2 * \begin{pmatrix} 1 \\ 15 \end{pmatrix} + 1 * \begin{pmatrix} 18 \\ 9 \end{pmatrix} \right) = \begin{pmatrix} 10.6 \\ 12.5 \end{pmatrix}$$

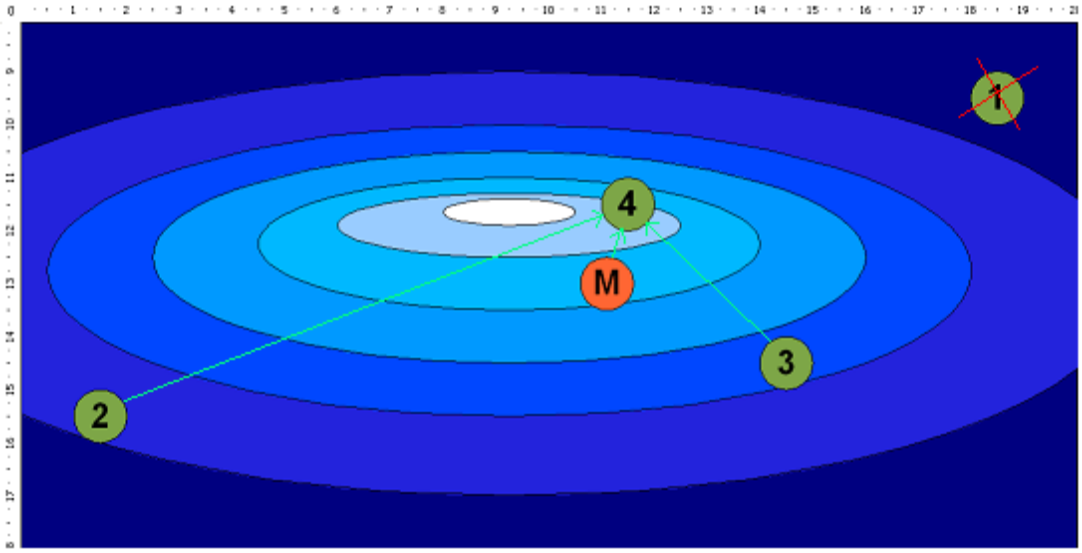


Abbildung 3.2: Partikel folgen nicht dem gewichteten Mittel

Der Pseudocode wird hier verkürzt abgebildet, da er sich lediglich durch die Berechnung des Mittels und dem Ersatz des schlechtesten Partikels vom Pseudo-Code in 2.2.1. unterscheidet. Für die Sortierung wurde der „Bubble-Sort"-Algorithmus [3] verwendet.

```
For Iterations'
  Update(Swarm);
  SortByFitness(Swarm);
  CalculateMean(Swarm);
  Foreach Particle in Swarm
    GetBestNeighbor(Particle);
    UpdateVelocity(Particle);
    UpdatePosition(Particle);
  endFor;
  ReplaceWorstByMean();
endFor;
```

Abbildung 3.3: Mean-Based-PSO Pseudo Code

3.4 ESC-PSO

Mit der ESC-PSO (Escape-SPO) stelle ich eine Erweiterung der klassischen PSO vor, die in der Lage ist, auch aus lokalen Minima stark multimodaler Funktionen zu entkommen.

Betrachtet man das Verhalten der Partikel während der Optimierung multimodaler Funktionen, so konvergiert der gesamte Schwarm irgendwann meist in ein lokales Optimum. Es wäre erstrebenswert, solchen Minima entkommen zu können oder auch präventiv, die Konvergenz in solche frühzeitig zu verhindern.

Ausgehend von dieser Überlegung stellt sich die Frage, wie man die Konvergenz des Schwarms erkennen kann. Dies wäre möglich, sofern man den tatsächlichen Gradienten der Fitnessfunktion berechnen könnte. Da aber die Fitnessfunktion in der Regel nicht differenzierbar ist, ist man auf eine Approximation angewiesen. Hierzu lässt sich zum Beispiel ein neuer Parameter bilden aus dem Verhältnis der besten Fitness des Schwarms der letzen Iteration und der Fitness der aktuellen Iteration. Dieser Parameter beschreibt zwar die aktuelle Fitnessverbesserung und lässt sich somit als Gradient auffassen; wegen der oben beschriebenen Approximation will ich hierfür aber die Bezeichnung „Pseudo-Gradient“ wählen. Konvergiert der Gradient gegen Null, so heißt dies jedoch noch nicht, dass auch der Schwarm konvergiert; denn Partikel können sich längere Zeit in schlechteren Regionen aufhalten und schließlich doch eine bessere Position ausfindig machen. Der „Pseudo-Gradient“ berechnet sich demnach wie folgt:

$$grad = 1 - \frac{Fitness(bestParticle_{i-1})}{Fitness(bestParticle_i)}$$

Formel 21 *:* Berechung des Pseudo-Gradienten

Der zweite entscheidende Faktor ist der des aufgespannten Raums des gesamten Schwarms, wie dieser in Abbildung 15 zu sehen ist. Konvergiert das Volumen dieses Raumes mit dem Gradienten gegen Null, so können die Partikel den Bereich nicht mehr verlassen. Die Spannweite des Schwarms berechnet sich demnach wie folgt:

$$Span_d = \frac{MAX(particlePos_{id}) - MIN(particlePos_{id})}{MAX_d - MIN_d}$$
$$i \in [0...N] \wedge d \in [0...D]$$

Formel 22 *:* Berechung der Spannweite

Eine solche Konvergenz lässt sich frühzeitig erkennen, wenn der Schwarm einen bestimmten Gradienten- und Volumenwert unterschreitet. Die Grenze des Gradienten, bei dessen Unterschreiten Konvergenz angenommen wird, hängt direkt von der Struktur der Funktion ab. So sollte dieser Wert für steile Funktionen sicherlich größer

gewählt werden als für flache Funktionen. Eine Verbesserung der Fitness von einem Zehntausendstel hat sich dabei als relativ robust erwiesen.

$$MaxGrad \quad 0.0001$$

Formel 23 *:* Pseudo-Gradienten-Schranke

Für die Begrenzung des Volumens wird der Definitionsbereich mit dem „Pseudo-Gradienten“ multipliziert, so dass der maximal aufgespannte Raum auch vom Gradientenwert abhängt. Die Entscheidung über das Vorliegen konvergenten Verhaltens erfolgt demnach über Formel 24.

$$grad < MAXGrad \quad \wedge \quad Span_d < (MAX_d - MIN_d) * grad$$

Formel 24 *:* Reinitialisierungsauswahl

In diesem Fall werden die Partikel nach Formel 25 normalverteilt über den gesamten Definitionsbereich reinitialisiert.

$$particle_{xd} = Normalverteilung\,(particle_{xd}, (MAX_d - MIN_d)/2.0)$$

Formel 25*:* Berechung der reinitialisierten Partikelposition

Die hier vorgestellte Erweiterung ist die erste Umsetzung des Prinzips. So wäre es sinnvoll, den Gradienten auf Basis vieler Iterationen zu bestimmen. Hinzu kommt, dass auch jeder Partikel einen Fitnessverlauf aufweist. So könnte die Reinitialisierung selektiv anhand der einzelnen „Pseudo-Gradienten“ der Partikel erfolgen. Auch die Wahl der Grenzen sollte eventuell adaptiv anhand des globalen Fitnessverlaufs gelernt werden.

3.5 Tribes

Dieser Algorithmus basiert auf dem PSO-Algorithmus und nutzt zusätzlich die Methode von „Sub-Schwärmen“, was bedeutet, dass es nicht einen Schwarm gibt, sondern unter Umständen viele Schwärme. Während der Optimierung wird mittels Verhaltensregeln Größe und Anzahl der Schwärme angepasst. Neben der Möglichkeit von mehreren Schwärmen wird zusätzlich in Abhängigkeit des Fitnessverlaufs der einzelnen Stämme eine Aktualisierungstrategie gewählt.

Die Methode beginnt mit einem zufällig initialisierten Partikel, welcher den ersten Stamm darstellt. Jeder Stamm(„Tribe“) hat in Bezug auf die Fitness der zugehörigen Partikel einen Status (gut, neutral, schlecht). Ebenso hat auch jeder Partikel einen Status (gut, schlecht).

In jeder Iteration werden Partikel zu Stämmen hinzugefügt, oder es werden die schlechtesten Partikel *P* der Stämme entfernt, je nachdem wie der Status der einzelnen Stämme und Partikel ist.

Auch ganze Stämme werden mit dem Ausscheiden des letzten Partikels entfernt oder beim Hinzufügen und Überschreiten einer Maximalanzahl geteilt.

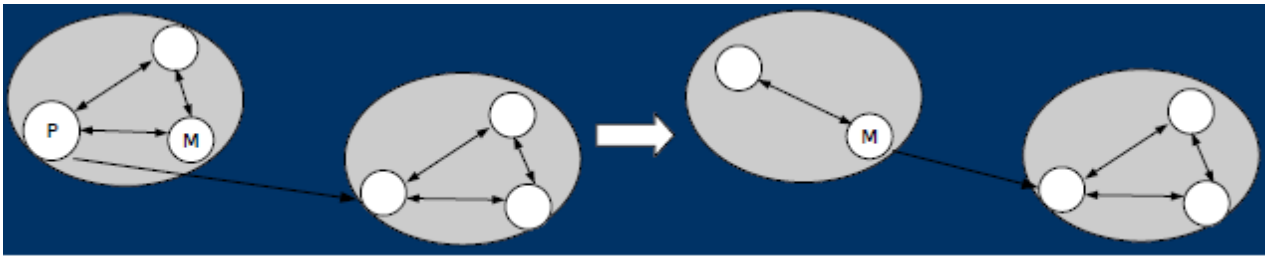


Abbildung 17: Tribes – Schwärme

Je nachdem wie die Statusveränderung zwischen zwei Iterationen der Stämme ausfällt wird eine Aktualsierungsstrategie der Partikel gewählt. In Grafik 1.8 steht ein „+“ für gut, „-“ für schlecht und „=“ für neutral, und die Klammern kennzeichnen die jeweilige Iteration.

Gathered statuses	*Strategy of displacement*
(= +) (+ +)	local by independent gaussians
(+ =) (- +)	disturbed pivot
(- -) (= -) (+ -) (- =) (= =)	pivot

Abbildung 18: Tribes – Aktualisierungsregeln

Allgemein betrachtet entscheidet die Aktualisierungsmethode über den Grad der Lokalität der Suche des Stammes. So steht die erste Zeile der Grafik 18 für eine Verbesserung des Schwarms ,was zu einer lokalen Suche führt. Dagegen führt eine Stagnation zu einer leicht erweiterten und eine Verschlechterung zu einer „globaleren“ Suche.

Eine Abbildung des gesamten Algorithmus ist Quelle [20] zu entnehmen.

Ein detaillierte Beschreibung und Analyse kann in [2] nachgelesen werden.

Quelle [21] bietet das Verfahren in C++ und Java implementiert.

4 Implementierung

Ziel der Implementierung ist es, eine Klassenbibliothek bereitzustellen, die den Entwicklern viel Freiraum beim Experimentieren mit PSO-Algorithmen lässt und dabei den Programmieraufwand möglichst gering hält.

Es wurde ein objektorientierter Ansatz gewählt ,der die Struktur des Codes und somit die Übersichtlichkeit durch das Klassenkonzept verstärkt. Ein ausführlich dokumentierter Quellcode erhöht zusätzlich die Verständlichkeit. Das „Geheimnisprinzip“ der Klassen verringert das Risiko versehentlicher Datenmanipulation und somit das Auftreten von schwer lokalisierbaren Fehlern. Die Möglichkeit der Vererbung gestattet es, den Aufwand des Programmierens gering zu halten und bestehende Konzepte leicht den eigenen Anforderungen anzupassen.

Häufig in Abhängigkeit der Problemstellung variierende Einflüsse wurden über das Konzept der Schnittstellen implementiert. Hierzu gehören die Fitnessfunktion, Nachbarschaftsbeziehung, sowie Zufallsvariable. Dadurch wird ein komfortabler Austausch durch eigene Varianten ermöglicht.

Für die Implementierung wurde die Programmiersprache C# mit der Entwicklungsumgebung Microsoft Visual Studio 2005 verwendet, da diese ein hohes Maß an Portabilität aufweist. Durch die große Ähnlichkeit der C#-Syntax zur JAVA-Syntax kann der Quellcode unter Einsatz von CASE-Tools direkt in JAVA-Code überführt werden. Zusätzlich besteht die Möglichkeit, den Quellcode über DLL-Dateien in andere Hochsprachen einzubetten.

Im folgenden wird der grundlegende Aufbau der Klassenbibliothek beschrieben. Eine detaillierte Beschreibung im Quelltext zu entnehmen.

4.1 Die Klasse Particle

Die Klasse „Particle“ bietet die Möglichkeit, Partikel-Objekte zu erstellen, die die vom PSO-Algorithmus benötigten Attribute besitzen. Wie Grafik 4.1 zu entnehmen ist, besitzt die Klasse fünf Attribute, welche die Position, die Geschwindigkeit, den aktuellen Fitnesswert und die beste jemals erreichte Position mit zugehörigem Fitness-Wert speichern.

Alle Attribute können gesetzt und abgefragt werden. Dabei können die Attributwerte der Felder durch Übergeben eines neuen Feldes komplett ersetzt oder auch einzelne Werte innerhalb eines Feldes verändert werden. Durch die Funktion *„getDimensi-*

on()“ kann diejenige Dimension der Fitnessfunktion ermittelt werden, die mit der Länge der Positionsfelder des Partikels übereinstimmt.

Die Funktion *„update()“* berechnet mit einem übergebenen Objekt vom Typ *„IFitness“* (siehe Kapitel 4.5) den Fitnesswert des Partikels, setzt diesen Wert und aktualisiert gegebenenfalls den besten je erreichten Wert und die zugehörige Position.

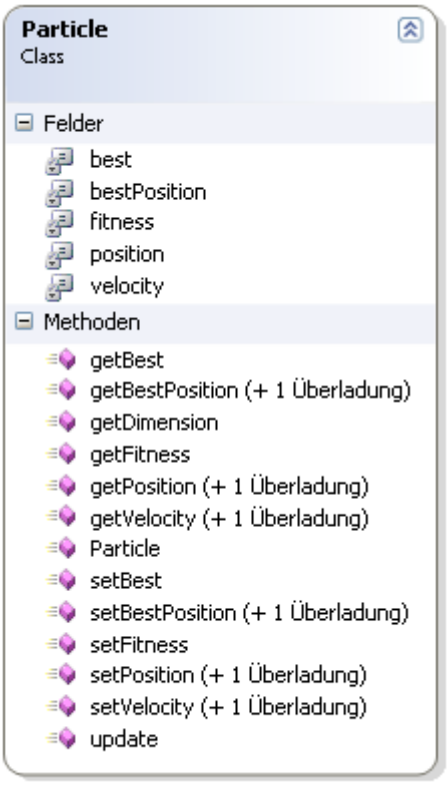


Abbildung 4.1: Die Klasse "Particle"

4.2 Die Klasse Swarm

Die Klasse *„Swarm“* stellt eine Containerklasse dar, die Objekte vom Typ *„Particle“* aufnimmt und diese im Sinne des PSO-Algorithmus als Schwarm verwaltet. Beim Aufruf des Konstruktors muss die Anzahl der Partikel angegeben werden, die der Schwarm enthalten soll, ferner die Fitness-funktion vom Typ *„IFitness“* (siehe Kapitel 4.5). Die Partikel werden unter Verwendung der Fitnessfunktion auf zufälligen Positionen und mit zufälligen Geschwindigkeiten im Definitionsbereich der Funktion entsprechend Kapitel 2.4.1 initialisiert.

Die Klasse speichert stets den besten jemals erreichten Wert des gesamten Schwarms und die Referenz auf das zugehörige Partikel, so dass dieses bei Bedarf schnell ermittelte werden kann.

Zusätzlich implementiert die Klasse die Schnittstelle *„IEnumerable“*. So kann komfortable durch Objekt-Indizierung über den Schwarm iteriert werden. Um ungewolltes

Überschreiben von Partikelreferenzen zu vermeiden, ist die Zuweisung über die Indizierung jedoch verhindert. Der Index eines Partikels kann über die Methode *„indexOf()“* abgefragt werden.

Die Funktion *„update()“* nimmt ein Funktionsobjekt vom Typ *„IFitness“* entgegen und aktualisiert jedes Partikel durch Aufruf der *„update()“*-Funktion eines jeden Partikels. Sie speichert gleichzeitig den besten Partikel und dessen Wert in den beiden Variablen *„best“* und *„particle“*.

Da manche PSO-Algorithmen Selektion verwenden, besteht die Möglichkeit, Partikel mit den Methoden *„removeParticle()“* zu löschen oder durch *„addParticle()“* hinzuzufügen.

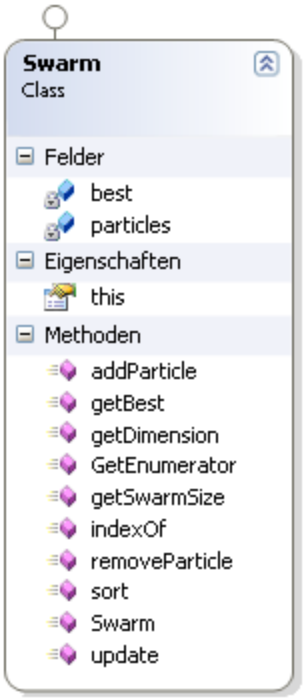


Abbildung 4.2: Die Klasse "Swarm"

4.3 Die Schnittstelle IDistribution

Die Schnittstelle *„IDistribution“* garantiert den identischen Zugriff auf Zufallsvariablen verschiedener Verteilungen. Die Funktion *„calculateRandomVector“* gibt einen Vektor mit Zufallsvariablen für jede Dimension zurück.

Abbildung 4.3: Die Schnittstelle "IDistribution"

Die Klassenbibliothek stellt mit der Klasse *„Hypercube“*, *„Hypersphерical“* und *„Gaussian“* die in Abschnitt 2.2 beschriebenen Zufallsverteilungen bereit.

4.4 Die Schnittstelle ITopology

Die Schnittstelle *„ITopology“* garantiert den identischen Zugriff auf verschiedene Nachbarschaftstopologien, wie sie in Abschnitt 2.3 beschrieben werden. Die Methode *„getBestNeighbor()“* nimmt die Referenz eines Partikels entgegen und ermittelt den Partikel in der Nachbarschaft des aktuellen Partikels, welcher den besten historischen Fitnesswert besitzt.

Da sich die Topologie stets auf den Schwarm bezieht, muss dieser dem Topologie Objekt bekannt sein. Alle Topologien des Kapitels 2.4 mit Ausnahme der „Memory-Swarms“ stehen in der Klassenbibliothek für die Verwendung bereit.

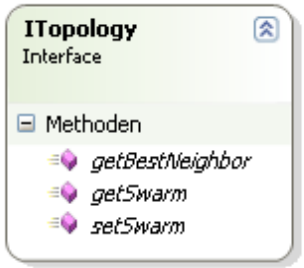


Abbildung 4.4: Die Schnittstelle "ITopology"

4.5 Die Schnittstelle IFitness

Die Schnittstelle *„IFitness“* stellt sicher, dass jede Klasse, die eine Fitnessfunktion repräsentiert, drei essentielle Methoden bereitstellt.

Der Methode *„calculateFitness()“* wird eine Partikel-Referenz übergeben, deren Positionswerte entsprechend zur Berechnung des Fitnesswertes genutzt werden. Die Methode gibt anschließend diese Referenz zurück.

Die Fitness-Funktion sollte durch ihren Definitionsbereich eingeschränkt werden, der dann in jeder Dimension über die beiden Methoden *„getMinDomain()“* und *„getMaxDomain()“* abgefragt werden kann.

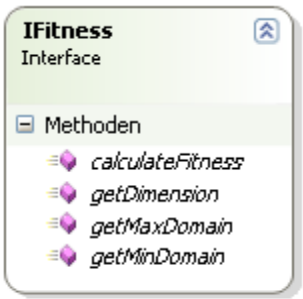


Abbildung 4.5: Die Schnittstelle "IFitness"

4.6 Die abstrakte Klasse Function

Die Klasse bietet ein abstraktes Konzept, welches eine sinnvolle Vorimplementierung der von der Schnittstelle „IFitness“ geforderten Methoden bereitstellt, so dass lediglich die Methode *„calculateFitness“* zur eigentlichen Berechnung implementiert werden muss. Die in Kapitel 5 beschriebenen Benchmarkfunktionen sind in der Bibliothek enthalten und von dieser Klasse abgeleitet.

Die Methode „particleOutside()“ überprüft, ob sich ein Partikel in allen Dimensionen innerhalb des Definitionsbereichs befindet. Ist dies nicht der Fall, so gibt sie TRUE zurück.

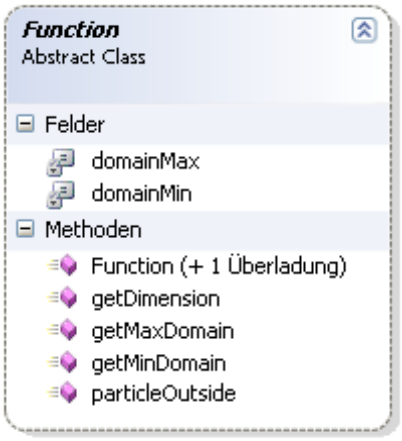


Abbildung 4.6: Die Klasse "Function"

4.7 Die abstrakte Klasse Pso

Die Klasse Pso bietet ein Grundkonzept von Attributen und Operationen, die in den meisten PSO Varianten benötigt werden. Neben den beiden Lernfaktoren c_1 und c_2 sollte jeder PSO Version die Funktion , der Schwarm, die Nachbarschaftsbeziehung, sowie die Zufallszahlverteilung bekannt sein. Neben der Möglichkeit diese Parameter und Referenzen zu setzen und abzufragen, existiert noch die Möglichkeit, den aktuell besten Wert des Schwarms abzufragen. Die „Sigmoid-Funktion“ kommt entsprechend zum Einsatz, wenn ein binäres codiertes Problem optimiert werden soll. Ebenso wie die Klasse *„Function“* kann von der Klasse *„Pso“* kein Objekt erzeugt werden, da sie nur ein Grundgerüst für eine PSO -Implementierung darstellt. Die Bibliothek stellt jedoch die in Abschnitt 2 und 3 beschriebenen Varianten bereit, deren Konstruktoren sich hauptsächlich in den Konvergenz-Parametern aus Abschnitt 2.2 unterscheiden. Wie zuvor bereits angemerkt, ist eine detaillierte Beschreibung de Kommentaren des Quelltextes zu entnehmen. Die Beschreibung der verschiedenen Version ist dem UML-Diagramm und Quellcode auf der im Anhang beiliegenden CD zu entnehmen.

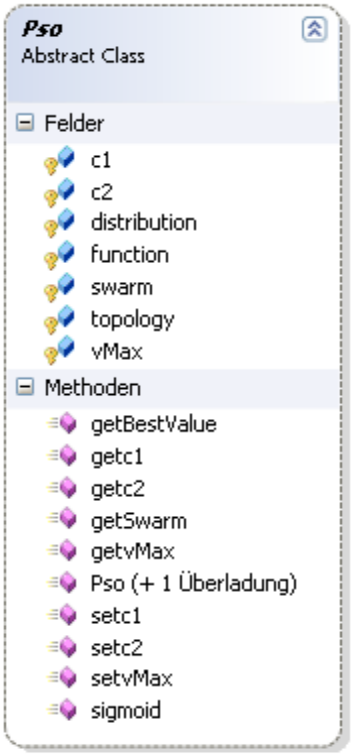


Abbildung 4.7: Die Klasse "Pso"

5 Auswertung

In diesem Kapitel soll zunächst die Brauchbarkeit der verschiedenen PSO-Varianten überprüft werden. Im Anschluss soll ein möglichst objektiver Vergleich mit einer der zurzeit besten Evolutions-Strategien namens „Covariance Matrix Adaptation - Evolution Strategy“[18] angestrebt werden. Nach der Dokumentation der Parameterwahl und Benchmarkfunktionen wird auf Basis der Fitnessverlaufsdiagramme ein Fazit gegeben.

5.1 Convariance-Matrix-Adaptation Evolution Strategy

Wie erwähnt simulieren Evolutionstrategien den Prozess der Mutation und Rekombination über eine Normalverteilung, wobei die mutierten Individuen in dieser Distribution erzeugt werden. Betrachtet man die Konturlinien der untenstehenden Funktionen, so wird deutlich, dass sich diese lediglich durch Streckung in x-Richtung unterscheiden. Es wäre also von Vorteil, eine Anpassung vornehmen zu können, die diese Transformation kompensiert und somit eine Rückführung des transformierten Problems auf ein symmetrisches Problem bewirkt. Der geniale Gedanke der CMA-ES ist der, dass aus der Position und der Fitness der Individuen die Kovarianzmatrix der Normalverteilung so adaptiert wird, dass der oben beschriebe Effekt erreicht wird.

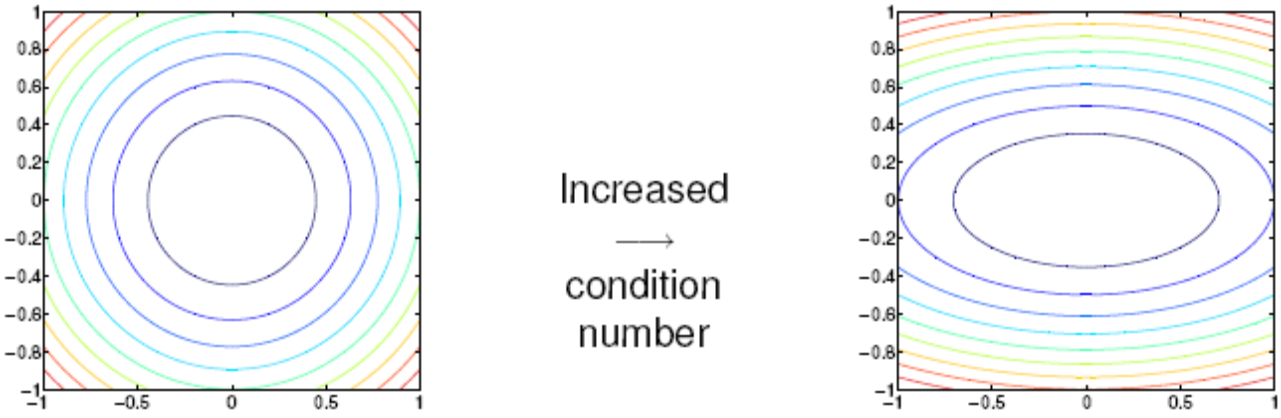


Abbildung: 5.1a Transformierte Funktion

Hieraus resultiert ein hoher Grad an Invarianz bezüglich diverser Transformationen wie Verschiebung, Rotation, etc. Aus Umfangsgründen wird hier nicht genauer auf das Verfahren eingegangen, jedoch sollte der Leser ohne Vorwissen sich zunächst über das „Quasi-Newton-Verfahren“ informieren und im Anschluss Quelle [18] für das detaillierte Verständnis des Verfahrens studieren.

Für die Funktionsauswertungen wurde in dieser Arbeit die „Shark machine learning libary“ [23] verwendet, wobei die Parameter nach Quelle [18] wie folgt bestimmt wurden:

$$\lambda = 4 + ceil(3\ln n) \qquad \mu = floor(\mu^1) \qquad \mu^1 = \frac{\lambda - 1}{2} \qquad \sigma_{init} = 0.3(xMax - xMin)$$

Formel 26*:* CMA-ES Parameter

Der verwendete Sourcecode der CMA-ES, sowie die Ergebnis-Daten können der Beiliegenden CD entnommen werden.

5.2 Vergleichsbasis

Um die Leistungsfähigkeit eines Optimierungsverfahrens einschätzen und vergleichen zu können, sollte dieses Verfahren in der Lage sein, diverse Problemklassen repräsentiert durch Benchmark-Funktionen zu optimieren.

Es wird davon ausgegangen, dass ein „Black-Box“-Optimierungsproblem vorliegt. Das heißt, dass keinerlei Information über die Funktion vorliegt, von denen aus auf eine optimale Parameterwahl geschlossen werden kann. Dies beinhaltet auch die Separabilität. Was bedeutet, dass sich die D-dimensionale Funktion in D eindimensionale Optimierungen aufteilen lässt, was die Optimierung stark vereinfachen würde.

Grundsätzlich sollten nicht-lineare Optimierungsverfahren in der Lage sein, unimodale und multimodale Funktionen zu optimieren. Die im nächsten Kapitel vorgestellten Funktionen haben ihr Optimum im Vektor 0^D . Die Verfahren sollten darüber hinaus in der Lage sein, auch eine verschobene Funktion optimieren zu können. Dies spiegelt reale Probleme besser wider und zeigt, ob das Verfahren vom Koordinatensystem oder von der Symmetrie der Funktion abhängt. Aufgrund eventueller Symetrie- oder Separabilitätsbefangenheit sollten die Verfahren die gleichen Funktionen auch rotiert und skaliert optimieren können. Um reale Probleme noch näher zu kommen, sollte zusätzlich untersucht werden, inwiefern kleine Störungen(„noise“) die Optimierung beeinflussen.

Im nächsten Kapitel werden neun entweder unimodale oder multimodale Funktionen vorgestellt, die verschoben werden können. Da zunächst eine allgemeine Aussage über das Potenzial der Verfahren gegeben werden soll, wird in dieser Arbeit nicht auf Rotation und „noise“ eingegangen. Eine detaillierte Untersuchung würde den Umfang dieser Arbeit übersteigen, allerdings kann dies unter Nutzung der Quelle [24] nachgeholt werden.

Als Vergleichskriterium dienen die Funktionsauswertungen, da diese bei realen Optimierungsproblemen in der Regel die meiste Rechenzeit in Anspruch nehmen und so eine Minimierung der Funktionsaufrufe anzustreben ist.

Die Fitnessverläufe ergeben sich als arithmetisches Mittel von jeweils 100 Durchläufen der jeweils 30-dimensionalen Benchmarkfunktionen.

In Anschluss werden die Funktionen auch verschoben untersucht, wobei der Verschiebungsvektor jeweils ein zufälliger Vektor aus dem jeweiligen Definitionsbereich ist.

5.3 Benchmark Funktionen

5.3.1 Unimodale Funktionen

Unimodale Funktionen besitzen lediglich das globale Optimum, so dass Optimierungsverfahren nicht in lokale Optima konvergieren können. Dementsprechend sollte die Optimierung auf solch „einfache“ Funktionen entsprechend schnell verlaufen.

Die einfachste der neun Funktionen stellt die „Linear Funktion“ dar. Sie ist unimodal, linear und separierbar und gibt Auskunft über die generelle Konvergenzgeschwindigkeit des Verfahrens.

$$f(\vec{x})_{Linear} = -\sum_{d=0}^{D-1} x_d$$
$$D = 30 \qquad \vec{x} \in \mathbb{R}^D \wedge -\infty \le x_d \le \infty$$
$$Initialisierung: -100 \le x_d \le 100$$

$$Globales\ Optimum\ bei\ f(\vec{\infty})_{linear} = -\infty$$

Formel 27: Linear – Funktion

Die „Sphere Funktion“ und ihre Varianten „Paraboloid“, „Cigar“ und „Tablet“ sind konvex, unimodal, separierbar und werden auch verschoben untersucht. Die Varianten stellen Streckungen bzw. Stauchungen der „Sphere“ Funktion dar.

$$f(\vec{x})_{Sphere} = \sum_{d=0}^{D-1} (x_d - o_d)^2$$
$$D = 30 \qquad \vec{x}, \vec{o} \in \mathbb{R}^D \wedge -100 \le x_d, o_d \le 100$$

$$Globales\ Optimum\ bei\ f(\vec{o})_{sphere} = 0.0$$

Formel 28: Sphere – Funktion

Die „Paraboloid Funktion“ ist in jeder Dimension unterschiedlich gestreckt.

$$f(\vec{x})_{Paraboloid} = \sum_{d=0}^{D-1} (c^{(\frac{d}{D})} (x_d - o_d)^2)$$

$$c = 1000 \qquad D = 30 \qquad \vec{x}, \vec{o} \in \mathbb{R}^D \wedge -100 \leq x_d, o_d \leq 100$$

$$Globales\ Optimum\ bei\ f(\vec{o})_{Paraboloid} = 0.0$$

Formel 29: Paraboloid – Funktion

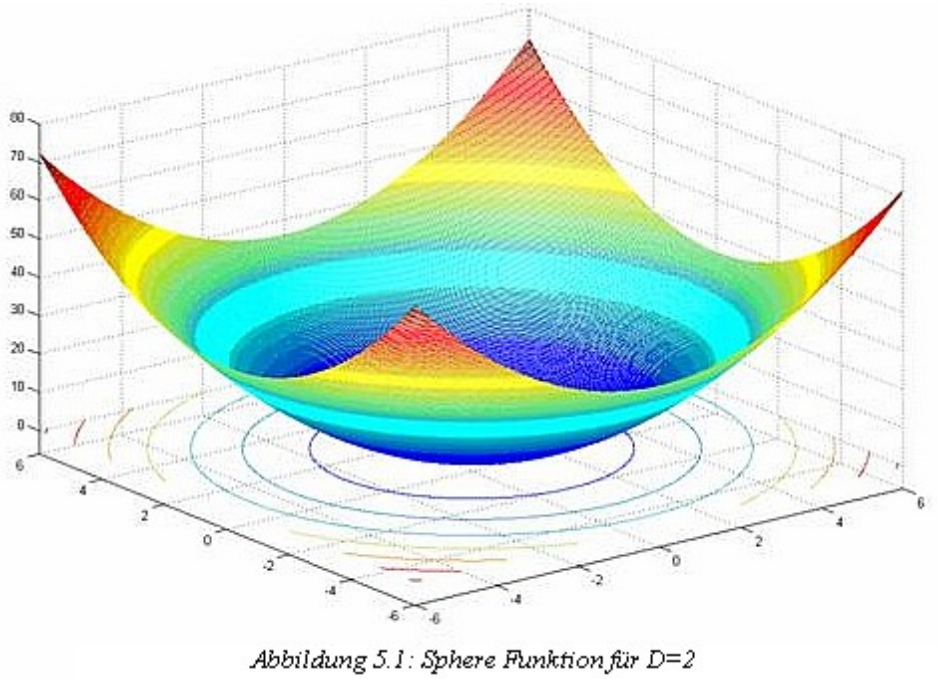


Abbildung 5.1: Sphere Funktion für D=2

Die beiden folgenden Funktionen unterscheiden sich von der „Sphere Funktion“ dadurch, dass der Streckungfaktor c nur in der ersten Dimension vorkommt, wie bei f_{Tablet} oder gerade nicht in dieser, sondern in allen anderen, wie bei f_{Cigar}.

$$f(\vec{x})_{Cigar} = (x_0 - o_0)^2 + \sum_{d=1}^{D-1} (c(x_d - o_d))^2$$

$$c = 1000 \qquad D = 30 \qquad \vec{x}, \vec{o} \in \mathbb{R}^D \wedge -5 \leq x_d, o_d \leq 5$$

$$Globales\ Optimum\ bei\ f(\vec{o})_{Cigar} = 0.0$$

Formel 30: Cigar – Funktion

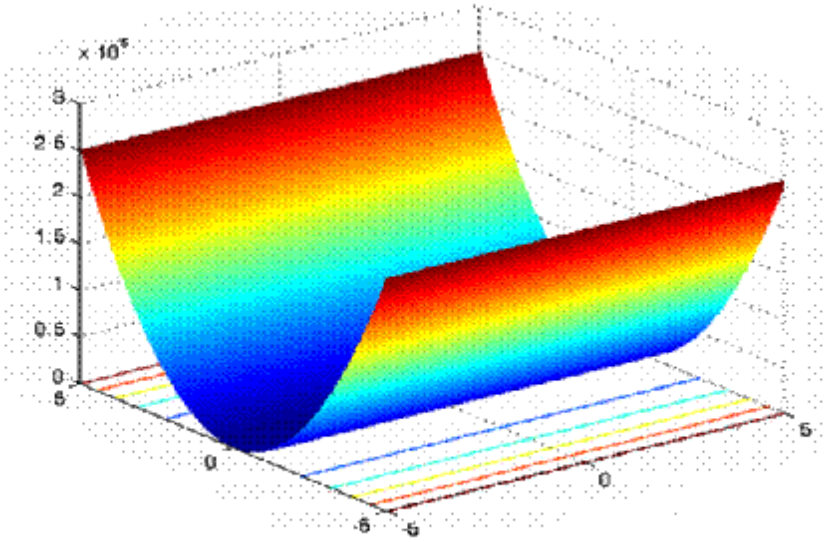


Abbildung 5.2: Cigar Funktion D = 2

$$f(\vec{x})_{Tablet} = (c(x_0 - o_0))^2 + \sum_{d=1}^{D-1} (x_d - o_d)^2$$
$$c = 1000 \qquad D = 30 \qquad \vec{x}, \vec{o} \in \mathbb{R}^D \wedge -5 \leq x_d, o_d \leq 5$$

$$Globales\ Optimum\ bei\ f(\vec{o})_{Tablet} = 0.0$$

Formel 31: Tablet – Funktion

5.3.2 Multimodale Funktionen

Die multimodalen Funktionen weisen teilweise extrem viele lokale Optima auf, wie die „Ackley Funktion“, die multimodal, verschiebbar, aber nicht separierbar ist. Auch alle multimodalen Funktionen werden verschoben untersucht.

$$f(\vec{x})_{Ackley} = -20\exp\left(-\frac{1}{5}\sqrt{\frac{1}{D}\sum_{d=0}^{D-1}(x_d - o_d)^2}\right) - \exp\left(\frac{1}{D}\sum_{d=0}^{D-1}\cos(2\pi(x_d - o_d))\right) + 20 + e$$
$$D = 30 \qquad \vec{x}, \vec{o} \in \mathbb{R}^D \wedge -32 \leq x_d, o_d \leq 32$$

$$Globales\ Optimum\ bei\ f(\vec{o})_{Ackley} = 0.0$$

Formel 32: Ackley – Funktion

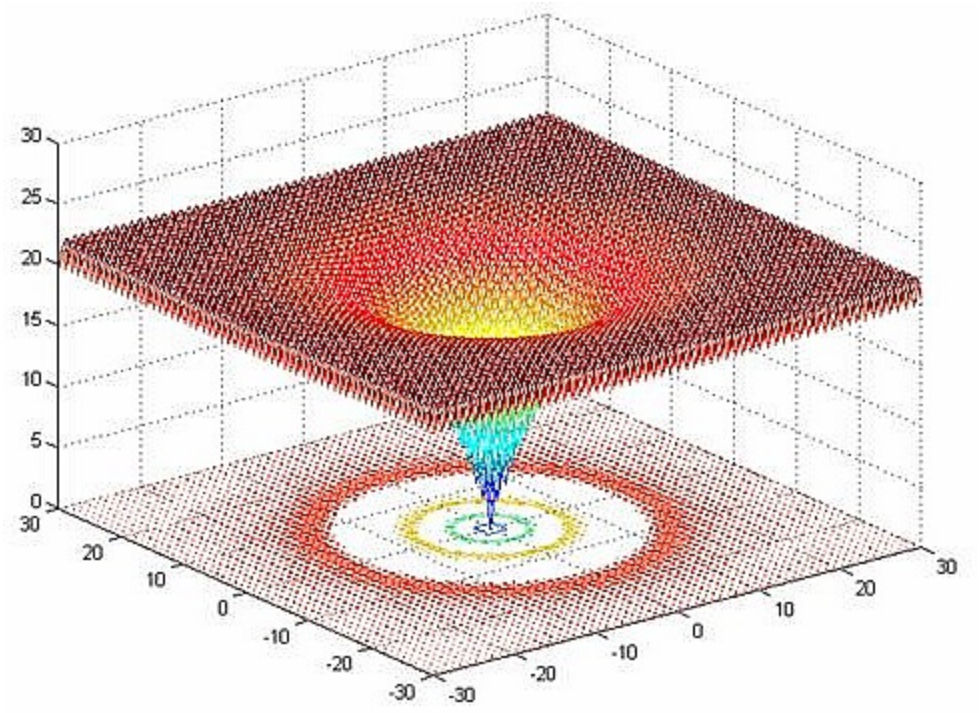


Abbildung 5.3: Ackley Funktion D = 2

Die Griewank Funktion ist multimodal, verschiebbar und nicht separierbar. Es existieren extrem viele lokale Optima.

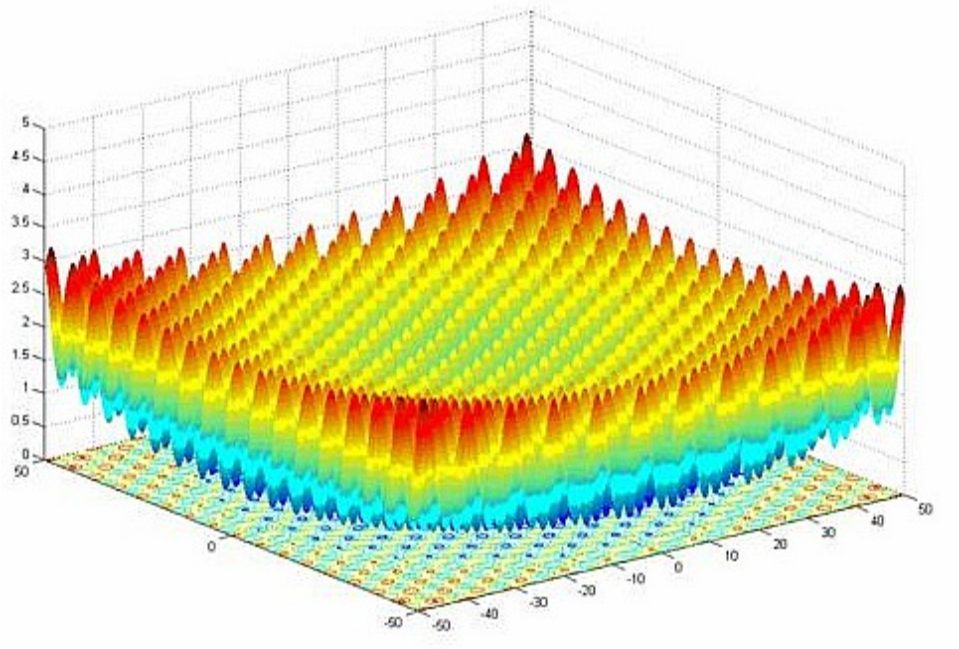


Abbildung 5.4: Griewank Funktion D=2

$$f(\vec{x})_{Griewank} = \sum_{d=0}^{D-1} \frac{(x_d - o_d)^2}{4000} - \prod_{d=0}^{D-1} \cos\left(\frac{(x_d - o_d)}{\sqrt{i+1}}\right) + 1$$
$$D = 30 \qquad \vec{x}, \vec{o} \in \mathbb{R}^D \wedge -600 \le x_d, o_d \le 600$$

$$Globales\ Optimum\ bei\ f(\vec{o})_{Griewank} = 0.0$$

Formel 33: Griewank – Funktion

Die „Rastrigin Funktion“ ist multimodal, verschiebbar und separierbar. Auch hier existieren extrem viele lokale Optima.

$$f(\vec{x})_{Rastrigin} = \sum_{d=1}^{D-1} ((x_d - o_d)^2 - 10\cos(2\pi(x_d - o_d)) + 10)$$
$$D = 30 \qquad \vec{x}, \vec{o} \in \mathbb{R}^D \wedge -5 \le x_d, o_d \le 5$$

$$Globales\ Optimum\ bei\ f(\vec{o})_{Rastrigin} = 0.0$$

Formel 34: Rastrigin – Funktion

Die „Rosenbrock Funktion“ ist multimodal, verschiebbar und nicht separierbar. Während sie an den Seiten sehr steil ansteigt, wird das Gefälle zum Optimum hin sehr klein.

$$f(\vec{x})_{Rosenbrock} = \sum_{d=0}^{D-2} (100((x_d - o_d)^2 - (x_{d+1} - o_{d+1}))^2 + ((x_d - o_d) - 1)^2)$$
$$D = 30 \qquad \vec{x}, \vec{o} \in \mathbb{R}^D \wedge -2.048 \le x_d, o_d \le 2.048$$

$$Globales\ Optimum\ bei\ f(\vec{o})_{Rosenbrock} = 0.0$$

Formel 35: Rosenbrock – Funktion

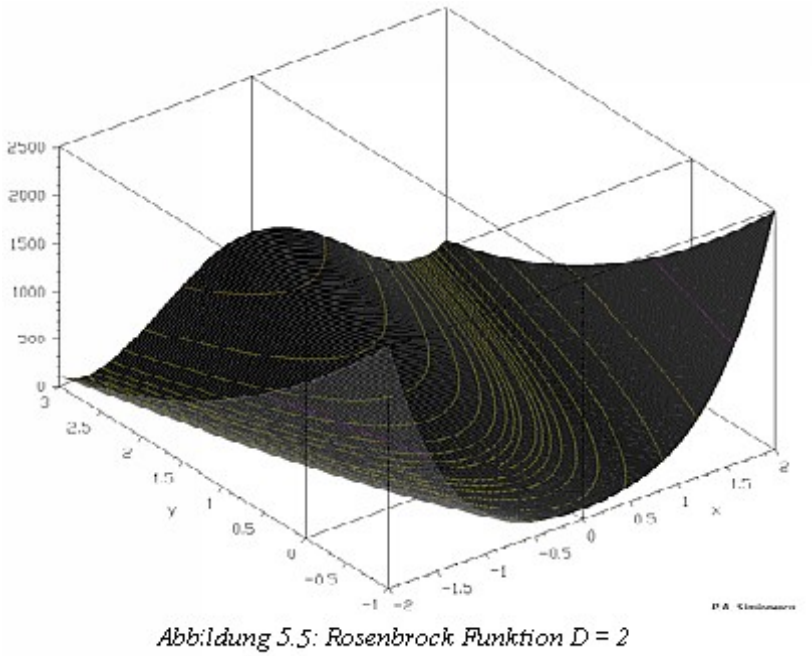


Abbildung 5.5: Rosenbrock Funktion D = 2

5.4 Optimierungsvergleiche

5.4.1 PSO-Parameterwahl

Im folgenden soll differenziert werden, welche Parameter sich unter Berücksichtigung der Allgemein-Performance als gut erwiesen haben.

Die Fitnessfunktionen sind 30-dimensional und sollen unter Verwendung von 20.000 Fitnessauswertungen optimiert werden. Die Schwarmgröße beträgt jeweils 20 Partikel, wodurch der Algorithmus eintausend Iterationen durchläuft.

Die Verwendung der Kugel-Verteilung erwies sich bei allen Testfunktionen als etwas besser als die Rechteck-Verteilung, wie auch in Quelle [2] beschrieben wird. Abbildung 5.7 zeigt den unterschiedlichen Fitnessverlauf der beiden Verteilungen, der im gleichen Verhältnis auch bei den anderen Funktion zu beobachten war.

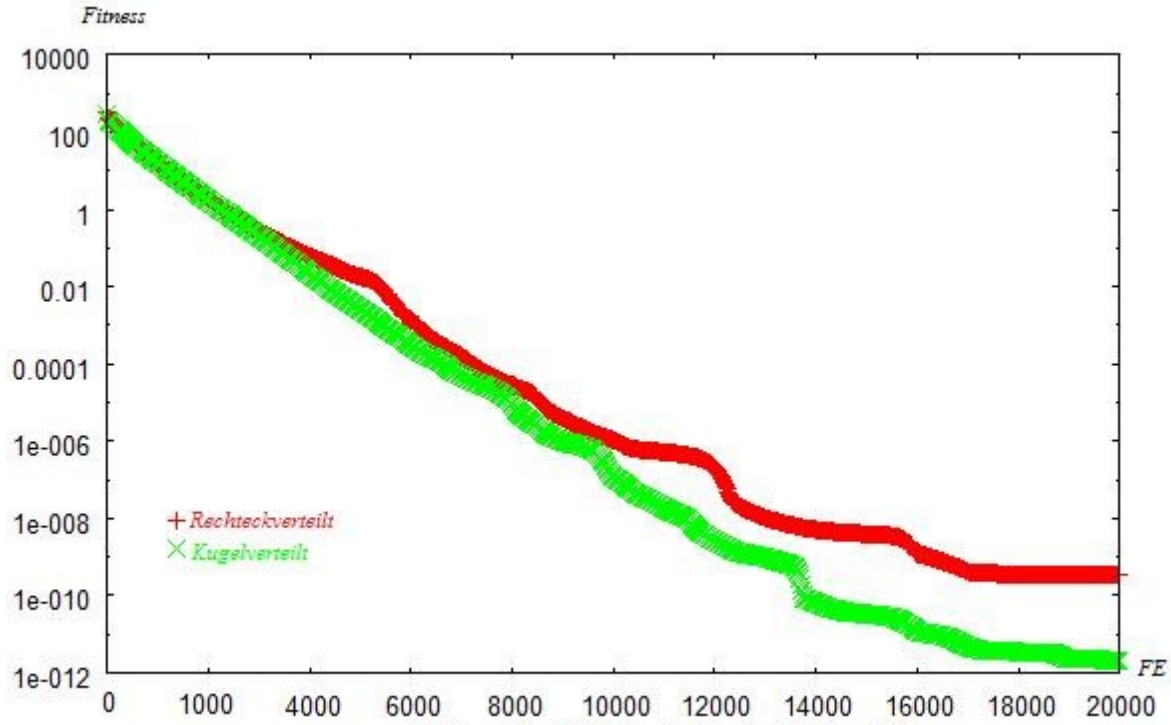


Abbildung 5.7: Zufallsvariablen Vergleich Sphere 30D

Die in Kapitel 2 vorgestellten Nachbarschaftsbeziehungen werden in Abbildung 5.8 und 5.9 verglichen, wobei die Funktionsoptima zufällig verschoben sind. Die globalen Nachbarschaften gBest und Wheel liefern in Kombination mit der Kugelverteilung schlechte Werte. Auch unter Verwendung der Rechteckverteilung können diese Topologien nicht überzeugen, da sie nur zu Beginn schnelle Konvergenz aufweisen.

Die Distanzfunktion zeigt in beiden Verläufen sehr gute Ergebnisse. Allerdings sind die Ergebnisse bei der „Ackley Funktion“ relativ schlecht, was durch die frühe Konvergenz in ein lokales Optimum zu erklären ist. Die Neumann-Topologie liefert durchschnittliche Ergebnisse. Es kann beobachtet werden, dass die dynamischen

Nachbarschaften die konstantesten Ergebnisse auf unimodalen und multimodalen Funktionen liefern.

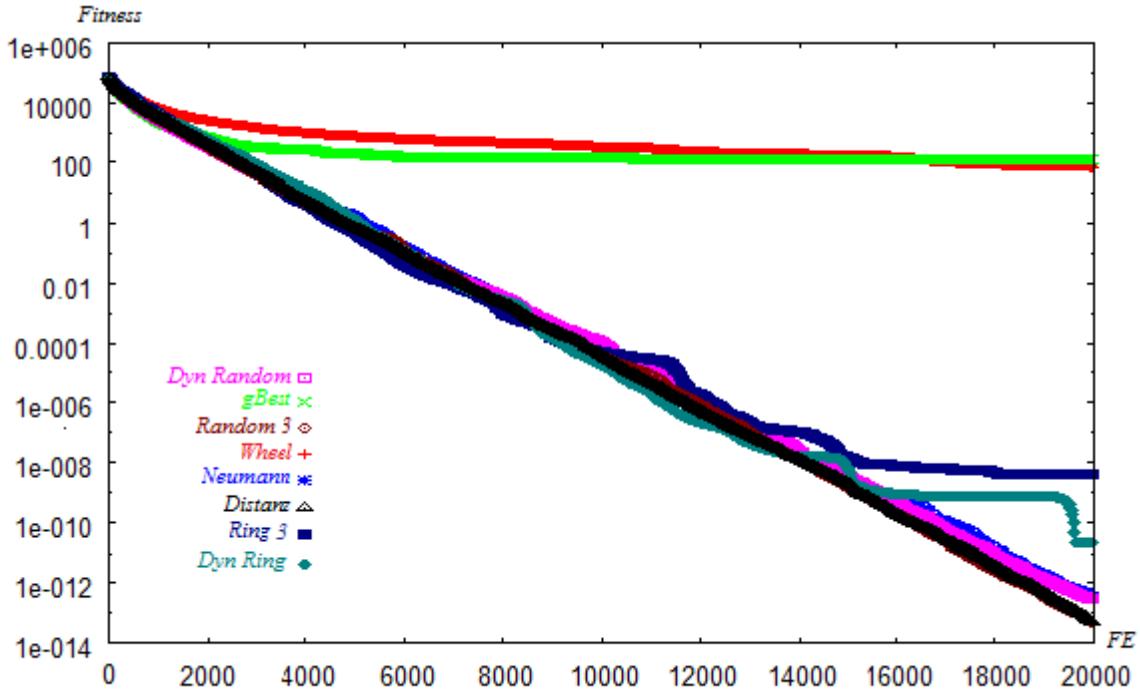


Abbildung 5.8: Topologievergleich Sphere (Unimodal)

Die zufällige Nachbarschaft - ob dynamisch oder festgelegt - liefert etwas bessere Ergebnisse als die Ring Topologie.

Auf allen Funktionen liefert die dynamische zufällige Nachbarschaft die besten Ergebnisse, so dass ich bei der Auswertung diese Nachbarschaft verwenden werde.

Hierfür wird die Nachbarschaftsgröße kontinuierlich von 2 auf 15 erhöht.

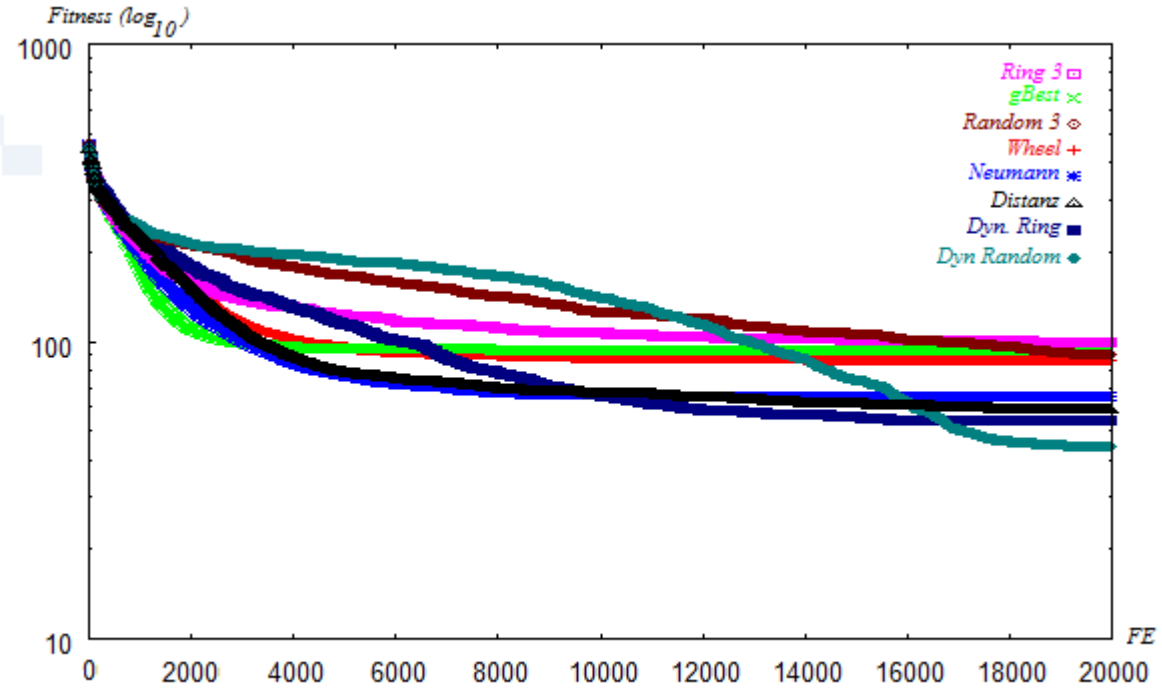


Abbildung 5.9: Topologievergleich Rastrigin (Multimodal)

Die Bestimmung der „inertia weight“ der GL-PSO funktioniert nicht in Kombination mit globalen Nachbarschaftsbeziehungen, daher wird eine zufällige Nachbarschaft der Größe drei verwendet. Bevor der eigentliche Vergleich mit CMA-ES durchgeführt wird, soll zunächst geklärt werden, welche der unten aufgeführten PSO-Varianten die besten Ergebnisse liefert. Die jeweiligen Parameter wurden den einzelnen Kapiteln entsprechend gesetzt. Die Lernfaktoren c_i wurden für die Versionen die den „constriction factor“ verwenden auf 2.1 und für alle anderen auf 2.0 gesetzt.

CF-PSO: Particle-Schwarm-Optimierung mit „Constriction Factor“

IW-PSO: Particle-Schwarm-Optimierung mit „Inertia Weight“

GL-PSO: Particle-Schwarm-Optimierung mit „Global/Local Inertia Weight“

LA-PSO: „Landscape Adaptive“- Particle-Schwarm-Optimierung

MB-PSO: „Mean Based“- Particle-Schwarm-Optimierung

(ESC – PSO wird später detailliert untersucht)

Zunächst der Fitnessverlauf der „Sphere Funktion“ aller Varianten:

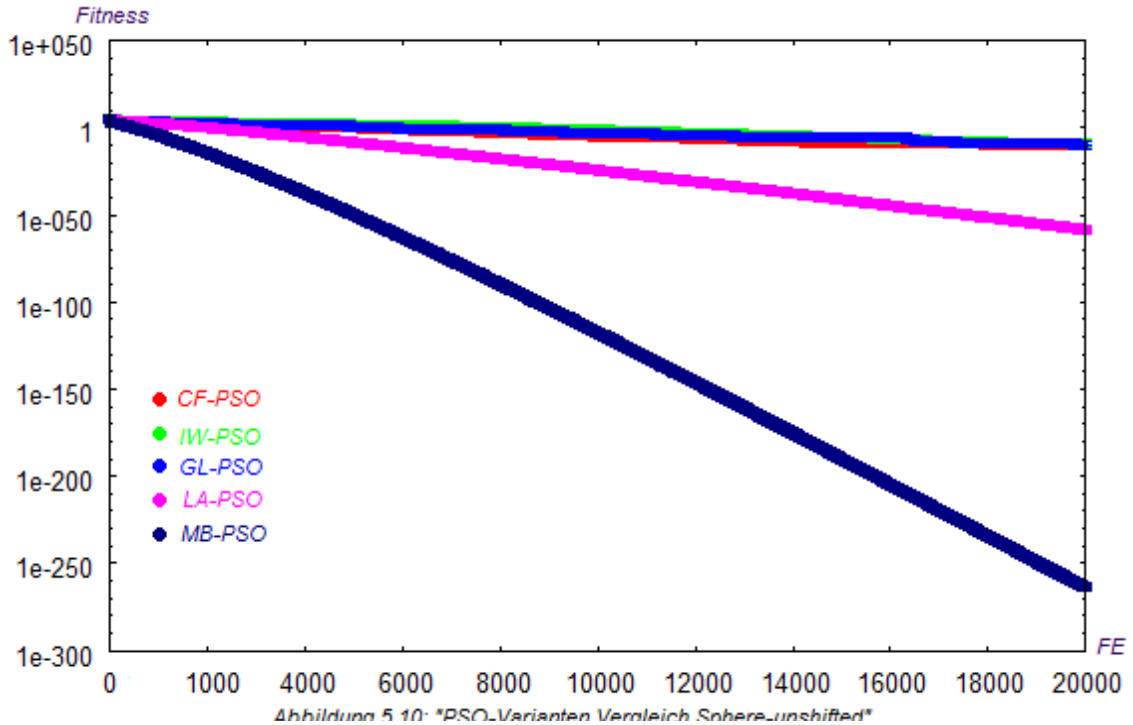


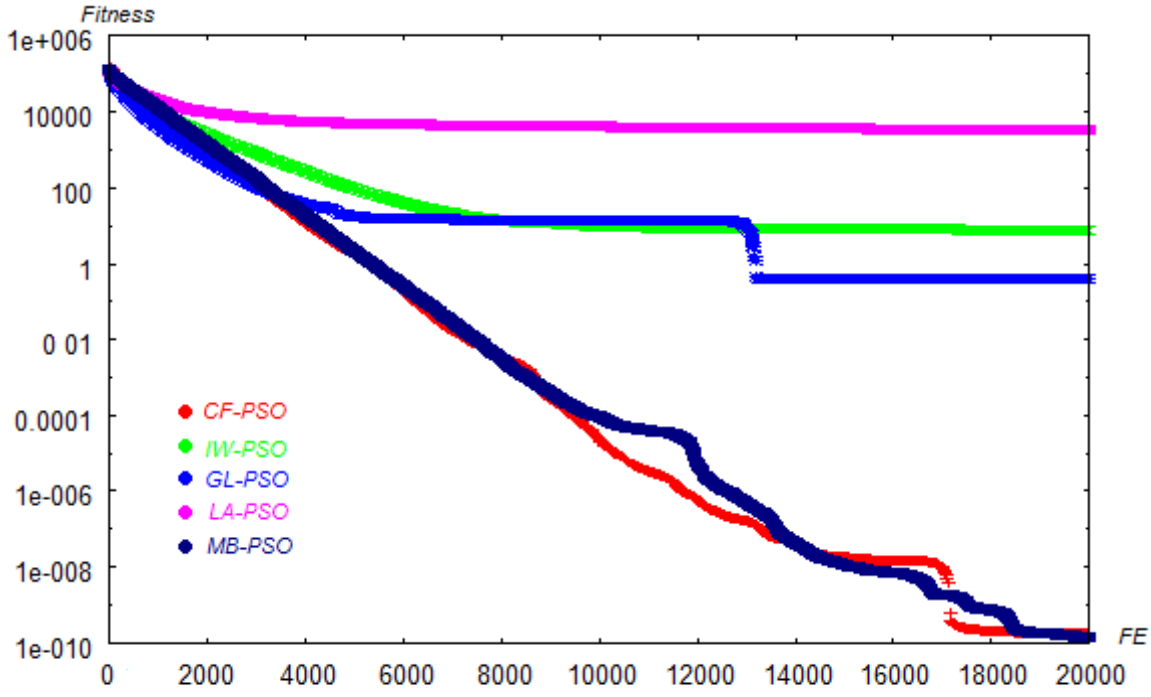
Abbildung 5.10: "PSO-Varianten Vergleich Sphere-unshifted"

LA-PSO und MB-PSO zeigen eine außergewöhnliche Performance, wobei in dem jeweiligen Kapitel bereits eine gewisse Skepsis gegenüber diesen Varianten geäußert wurde, da sie vom Koordinatensystem befangen sind.

Im nachfolgenden Verlauf wurde das Optimum der „Sphere-Funktion“ auf eine zufällige Position im Definitionsbereich $[-100,100]^D$ verschoben. Der Graph bestätigt die Vermutung der Koordinatenbefangenheit von LA-PSO. Dies zeigt sich ebenfalls bei

allen anderen acht Funktionen. Akzeptable Ergebnisse liefern lediglich CF-PSO und MB-PSO. Allerdings geht der angenommene Vorteil der MB-PSO verloren, wenn die Funktion verschoben wird. Das arithmetische Mittel weist bei Optimumsverschiebung sehr selten gute Fitnesswerte auf und scheint somit von der Symmetrie befangen zu sein.

Aus der Tabelle kann entnommen werden, dass IW-PSO und GL-PSO durch „Ausreißer“ eine schlechte durchschnittliche Fitness aufweisen. Dieser Effekt wird nicht durch die Verwendung der Kugel-Verteilung hervorgerufen, sondern hängt unmittelbar mit der Stärke der Verschiebung zusammen.



Method	Median	AVG	best	worst
CF	4,22356634042212E-13	1,65513687150465E-10	4,46736533109579E-17	1,32445117887911E-08
IW	1,45995481239757E-10	7,89993750842231	1,25229313049902E-13	121,978272278963
GL	2,32178831203837E-12	0,384459891051424	8,17869564692698E-15	19,2229551972929
LA	2805,23445077113	3279,07493144485	442,839151795325	11331,2034083481
MB	3,31069371162882E-12	1,44879331639945E-10	6,39111255347748E-16	6,05031927254383E-09

Abbildung 5.11: "PSO-Varianten Vergleich Sphere-shifted"

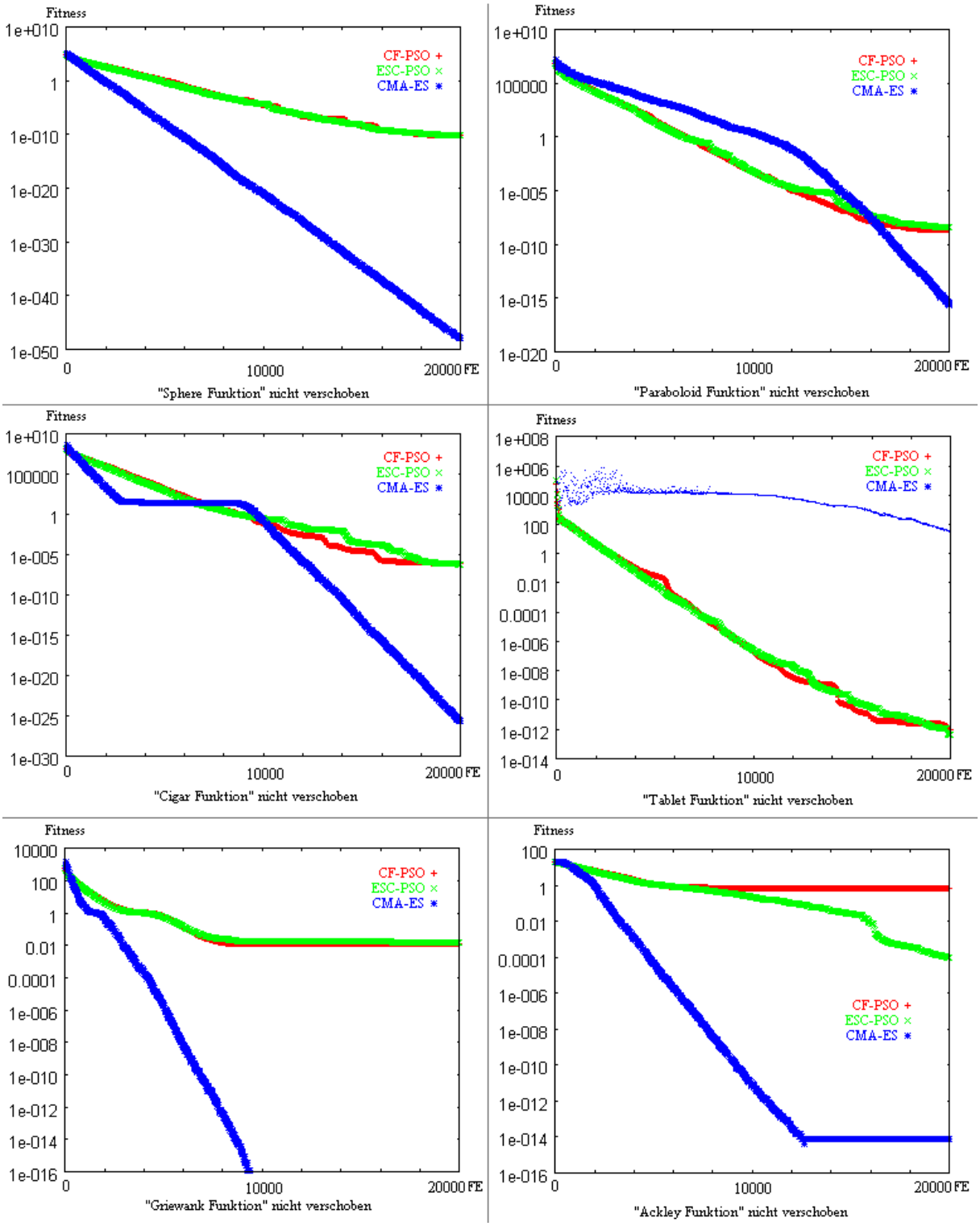
Da nur CF-PSO auch auf stark verschobenen Funktionen akzeptable Ergebnisse erzielte, wird im Nachhinein nur noch dieses Verfahren und das hier Außen vor gelassene ESC-PSO für die Auswertung verwendet.

MB-PSO liefert ähnliche Ergebnisse, da sie durch die fehlende Verwendung des gewichteten Mittels in etwa der CF-PSO entspricht.

Neben der CF-PSO wird auch die ESC-PSO detailliert untersucht, so dass die folgenden Diagramme stets den Fitnessverlauf der CF-PSO, ESC-PSO und der CMA-ES enthalten.

5.4.2 Vergleich mit der Evolutions Strategie „CMA-ES“

Zunächst die Verlaufsgraphen für alle nicht verschoben Funktionen außer „Linear“. Aufgrund der „Derandomisierung“ [27] des CMA-ES und den daraus resultierenden konstanten Ergebnissen, macht die Angabe des Mittelwertes, Medians, des schlechtesten und besten Wertes nur für PSO Verfahren Sinn.



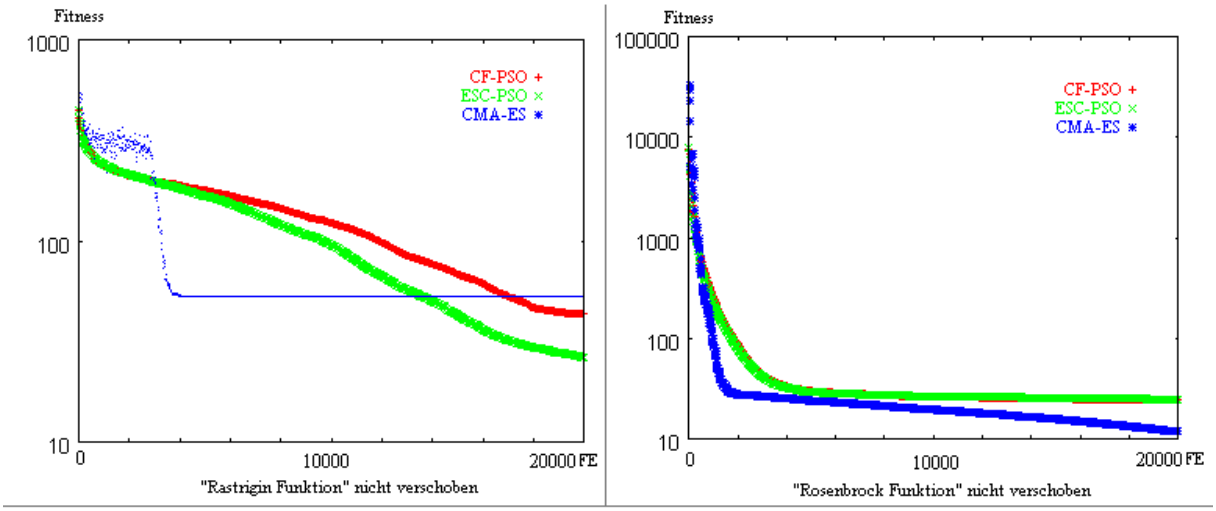


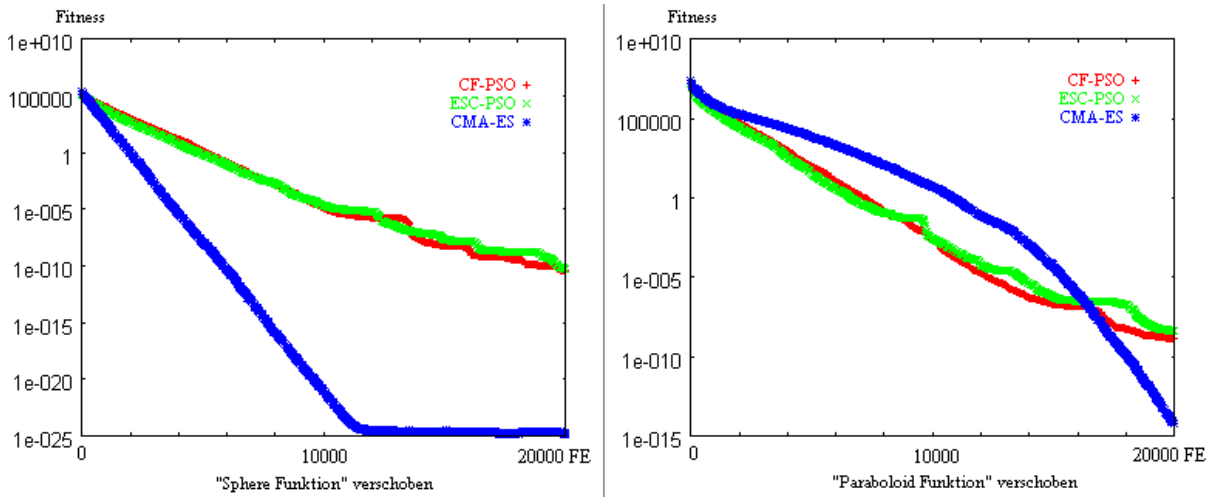
Abbildung 5.12: Fitnessverlauf nicht verschobener Funktionen

	Median	AVG	Best	Worst
	Linear			
CF	-1,55129031264008E+70	-1,25270070733807E+74	-7,5744930080534E+75	-2,8729714012786E+65
ESC	-3,39770321738935E+70	-8,97512949198584E+74	-4,39439293348111E+76	-4,2747456056068E+64
CMA	-1.21633e+095	-1.21633e+095	-1.21633e+095	-1.21633e+095
	Sphere nicht verschoben			
CF	3,13020692924346E-13	6,68320777366103E-11	1,59371957966798E-16	2,39437123762322E-09
ESC	6,72276025335433E-13	5,95578909083363E-11	1,36947566659503E-17	3,91146328871476E-09
	Paraboloid nicht verschoben			
CF	1.95017e-015	5.66008e-015	2.18318e-016	1.50082e-014
ESC	3,1508602059782E-11	4,17589173773965E-09	6,68812116910138E-14	3,36104468899968E-07
	Cigar nicht verschoben			
CF	4,91399272014757E-10	7,67018957508593E-07	9,8048965285746E-15	3,90468281187391E-05
ESC	2,88173339387904E-09	6,27578880116897E-07	2,93250877837765E-14	4,58565025765687E-05
	Tablet nicht verschoben			
CF	5,80335989996474E-15	8,54398062189648E-13	4,18424574380943E-19	5,13248430654289E-11
ESC	9,39312025548278E-15	4,19495096316997E-13	2,97488466334357E-18	1,12760348658429E-11
	Griewank nicht verschoben			
CF	0,00739604033612051	0,0123094083370203	1,11022302462516E-15	0,0852341242605816
ESC	0,00739618202320746	0,0157057118018992	6,99440505513849E-15	0,282125181534596
	Ackley nicht verschoben			
CF	0,931304601816838	0,747039196911237	3,49893669593371E-09	2,11895780300742
ESC	5,24835534365664E-06	0,00010074508347758	1,97022917980405E-08	0,00262536681183834
	Rastrigin nicht verschoben			
CF	41,7882252566808	43,2892039851248	15,9194369373246	102,619214993488
ESC	23,1730972999357	26,3670855253759	5,00401698839516	72,4237692989369
	Rosenbrock nicht verschoben			
CF	24,0639697511078	24,2732276028085	18,3418951189861	28,7546765288512
ESC	24,4485641939972	24,620328911023	9,9856687895497	28,9273123531379

Es kann beobachtet werden, dass der Fitnessverlauf der beiden PSO Varianten auf unimodalen Funktionen mit Außnahme der „Tablet Funktion“, sowie der „Griewank Funktion“ und „Rosenbrock Funktion“ in etwa übereinstimmt. Auf keinem der Verlaufsgraphen hat ESC-PSO einen Nachteil gegenüber CF-PSO hervorgerufen. In Bezug auf die „Ackley Funktion“ weisst ESC-PSO eine höhere Konvergenzsicherheit auf, da der schlechteste Wert der 100 Durchläufe nur geringfügig über dem Mittelwert liegt. Auch für die „Rastrigin-Funktion“ liefert ESC-PSO einen deutlich besseren mittleren Fitnessswert als CF-PSO und konvergiert weiterhin gegen das globale Optimum, während die anderen Verfahren bereits lokal konvergierten. CMA-ES konvergiert auf der „Rastrigin Funktion“ schnell , erreicht jedoch nach ca. 2000 Fitnessauswertungen(FE) ein lokales Optimum, aus dem sie nicht mehr entkommt.

Für die „Tablet Funktion“ liefern beide PSO-Verfahren deutliche bessere Werte. Würde man jedoch den Fitnessverlauf auf 40.000 FE ausweiten, so würde CMA-ES etwa mit den PSO Varianten gleich aufliegen. Interessant ist auch die Beobachtung, dass die Fitness des CMA-ES bei der „Tablet Funktion“ und „Rastrigin Funktion“ zu Beginn stark schwankt. Es scheint, als reagiere CMA-ES sehr empfindlich auf starke Transformation bezüglich einer oder weniger Dimensionen. Der zu Beginn stark schwankende Fitnessverlauf lässt auf Probleme bezüglich einer gerichteten Adaption schließen. Trotzdem soll hier nochmals betont werden, dass CMA-ES in der Lage ist, Funktionen wie „Tablet“ zu optimieren, dies aber verhältnismäßig langsam.

Im Folgenden werden die gleichen Funktionen mit Ausnahme der „Linear Funktion“ verschoben untersucht. Auf Grund der langen Berechnungszeiten der CMA-ES dienen hier 40 Wiederholungen als Referenz, wobei vor jeder Optimierung das Funktionsoptimum zufällig innerhalb des unveränderten Definitionsbereiches der Funktion verschoben wurde. Um jedoch die Ausgangsbedingungen für alle drei Verfahren identisch zu halten, waren für jedes Verfahren alle 40 Zufallsvektoren identisch, um die das Optimum verschoben wurde.



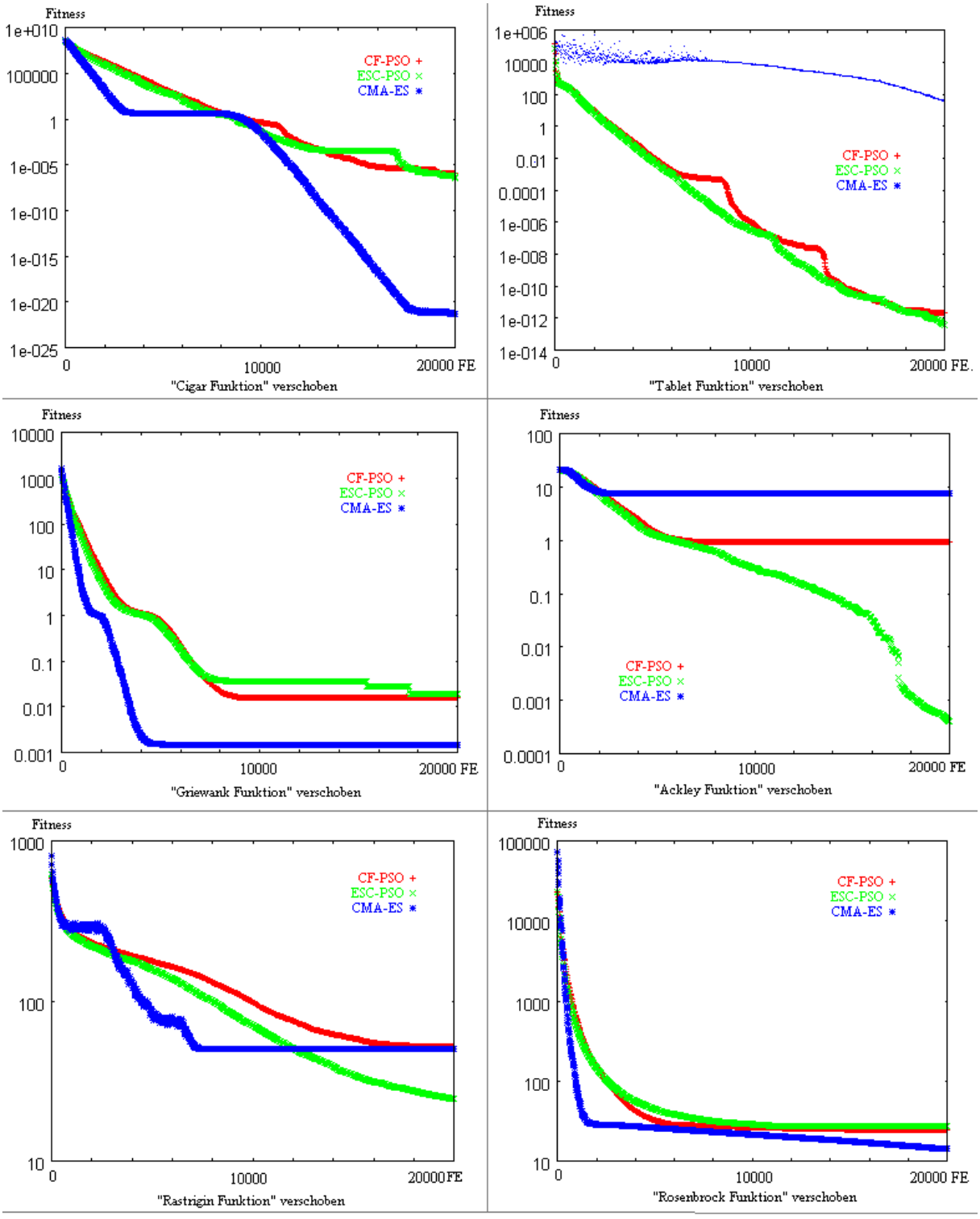


Abbildung 5.13: Fitnessverlauf verschobener Funktionen

CF-PSO und ESC-PSO weisen nahezu den identischen Fitnessverlauf mit unverschobenen Fitnessfunktionen auf, was auf eine allgemeine Invarianz bezüglich Verschiebung schließen lässt. CMA-ES weißt teilweise etwas schlechtere Werte auf verschobenen Funktionen auf. Auf den Funktionen auf denen es unverschoben überzeugte, überzeugte es jedoch auch verschoben, mit Ausnahme der „Ackley Funktion“. Die

frühe Konvergenz der CMA-ES bezüglich der verschobenen „Ackley-Funktion“ war überraschend. Ich vermute, dass der sehr flache Verlauf der Ackley Funktion nicht über genügend Information für eine erfolgreiche Adaption der CMA verfügt. So fand CMA-ES in mehr als 50% der Durchläufe nicht in die Nähe des globalen Optimums.

	Median	**AVG**	**Best**	**Worst**
	Sphere nicht verschoben			
CF	3,41731287632523E-13	3,41616795632516E-11	1,4569954160423E-16	7,46620249597258E-10
ESC	7,82211301035593E-13	4,62209770072406E-11	1,05968041778196E-15	3,73484359554788E-09
CMA	1.49852e-025	1.4433e-025	9.43432e-026	2.01548e-025
	Paraboloid nicht verschoben			
CF	2,9315477142684E-11	1,38075793854516E-09	5,16409031345241E-15	2,73204618371521E-08
ESC	7,43617603234527E-11	3,95963184619561E-09	1,7636584549002E-14	1,02559111997118E-07
CMA	1.95017e-015	5.66008e-015	2.18318e-016	1.50082e-014
	Cigar nicht verschoben			
CF	8,23902880973257E-10	1,20823232001329E-06	5,39888206649453E-13	0,000101053537769345
ESC	3,24464158286138E-09	3,81742372875962E-07	3,78797979045823E-12	1,90410672387403E-05
CMA	5.30963e-022	5.0933e-022	4.14201e-022	5.75328e-022
	Tablet nicht verschoben			
CF	6,41050354343928E-15	2,12137961555799E-12	1,24144218666717E-18	1,41794448529958E-10
ESC	1,70209245826372E-14	3,72053578005524E-13	5,34170561045075E-18	7,44062090936226E-12
CMA	32.8647	36.4694	6.14404	133.622
	Griewank nicht verschoben			
CF	0,00739604036047781	0,0155131245206988	3,33066907387547E-16	0,15666012948061
ESC	2,95194124699805E-09	0,018854147350532	2,22044604925031E-16	0,311711726806872
CMA	0	0.00147921	0	0.00739604
	Ackley nicht verschoben			
CF	0,931304601824526	0,926563658285846	1,18225291778629E-08	2,81259637952329
ESC	3,71365386842015E-06	0,000407453592584783	1,66037037452327E-08	0,0107001491514933
CMA	7.14984e-014	7.50119	4.30767e-014	21.5836
	Rastrigin nicht verschoben			
CF	49,9412379310022	51,7835103623384	22,8841079329019	95,4169105693024
ESC	22,9154360552648	24,3247479973064	8,96370163459653	53,2406648285157
CMA	51.7378	49.7479	28.8538	68.6521
	Rosenbrock nicht verschoben			
CF	23,9384738827681	24,081940441447	16,0370824124055	28,8570322030465
ESC	26,7725089422495	26,8506468031975	26,7725089411867	29,3787515587332
CMA	13.8764	13.9836	10.4831	16.0856

5.5 Fazit

Bezüglich ESC-PSO konnten keine signifikanten Nachteile gegenüber CF-PSO festgestellt werden. ESC-PSO wies darüber hinaus deutliche Verbesserungen bezüglich stark multimodaler Funktionen wie „Rastrigin“ und „Ackley“ auf.

Die PSO-Versionen reichten bezüglich der unimodalen Funktionen mit Ausnahme der „Tablet Funktion“ nicht an die Geschwindigkeit von CMA-ES heran. Obwohl die Graphen der verschobenen Funktionen eine Reduzierung der Konvergenzgeschwindigkeit des CMA-ES zeigten, war das Verfahren dennoch deutlich schneller als PSO.

Lediglich bezüglich der „Rastrigin Funktion“ und der „Ackley Funktion“ konnte ESC-PSO überzeugen. An dieser Stelle soll jedoch angemerkt werden, dass es auch Versionen der CMA-ES gibt - wie zum Beispiel die „Restart CMA-ES“ [27], die die Sicherheit hinsichtlich stark multimodaler Funktionen erhöhen sollen.

Es wurde bereits erwähnt, dass die Funktionsauswertung meist sehr viel Zeit in Anspruch nimmt, so dass die Optimierung mittels PSO eine ungenügende Performance im Vergleich mit CMA-ES aufweist.

Es stellt sich nun die Frage, wofür und wann PSO Verwendung finden kann!

Je geringer der Zeitaufwand der Funktionsauswertungen wird, desto attraktiver wird eine Optimierung mit PSO. Dies resultiert aus der grundsätzlichen Performance des Algorithmus, wenn man die Funktionsauswertungen außen vor lässt. Der Berechnungsaufwand der zuvor beschriebenen Benchmarkfunktionen ist verschwindend gering. So benötigte CMA-ES für einen Optimierungdurchlauf mit 20000 Fitnessauwertungen der „Sphere-Funktion“ so viel Zeit wie PSO für ca. 500 Durchläufe. Die PSO wurde beispielsweise verwendet, um den Spannungsverlauf in elektrischen Systemen zu optimieren [25] . Die Tatsache das die Funktionsauwertungen verhältnismäßig gering ausfallen und eine ständige Optimierung des Spannungssignals über kurze Zeitdistanzen gefordert ist, macht die PSO für diesen Bereich interessant.

Allgemein betrachtet konnten die in dieser Arbeit vorgestellten PSO-Varianten bezüglich der Konvergenzgeschwindigkeit nicht gegen CMA-ES bestehen. Dennoch weist das Verfahren Potential auf, wie ESC-PSO zeigen konnte. So zeigt auch das „Tribes“ Verfahren [20] eine starke Verbesserung auf, so dass ein Studium der PSO über diese Arbeit hinaus sich mit diesem Algorithmus befassen sollte.

Literaturverzeichnis

[1] Kennedy, James und Eberhart, Russell C. : *Swarm Intelligence.* San Francisco USA: Morgan Kaufmann, März 2001.

[2] Clerc, Maurice: *Particle Swarm Optimization*. London UK: ISTE Publishing Company, Februar 2006.

[3] Ernst, Hartmut: *Grundkurs Informatik.* 3. überarbeitete Auflage, Braunschweig/ Wiesbaden: Friedr. Vieweg & Sohn Verlagsgesellschaft mbH, März 2003.

[4] Prof. Dr. Claus, Volker und Prof. Dr. Schwill, Andreas: *Duden Informatik*. 3. Auflage, Mannheim. Bibliographisches Institut & F.A. Brockhaus AG, 2003.

[5] Niederhauser, Jürg: *Duden – Die schriftliche Arbeit*. 3. Auflage, Mannheim. Bi bliographisches Institut & F.A. Brockhaus AG, 2000.

[6] Dawkin, Richard : *„The Selfish Gene"*. 2. Auflage, Oxford. Oxford University Press Paperback, 1999.

[7] Shi, Yuhui (Februar 2004): *„Particle Swarm Optimization IEEE Neural Network Society"*. Kokomo USA. Electronic Data Systems Inc.:
URL http://www.computelligence.org/download/pso.pdf

[8] Montes de Oca, Marco A. (Mai 2007): *„Particle Swarm Optimization Introducti on"*. IRIDIA-CoDE, Universit´e Libre de Bruxelles (U.L.B.):
URL http://iridia.ulb.ac.be/~mmontes/slidesCIL/slides.pdf

[9] Eberhart, Russell C. und Shi, Yuhui (Juni 2004): *„Guest Editorial Special Issue on Particle Swarm Optimization"*, IEEE Transactions on Evolutionary Computa tion, Vol. 8, No. 3. Page 201:
URL http://ieeexplore.ieee.org/iel5/4235/28981/01304842.pdf

[10] Arumugam, M. Senthil und Rao, M. V. C. (Januar 2006): *„On the performance of the Particle Swarm Optimaziation algorithmn with various Inertia Weight variants for computing optimal control of a class of hybrid systems"*. Hindawi Publishing Corporation Discrete Dynamics in Nature and Society, Article ID 79295, Pages 1–17: URL
http://downloads.hindawi.com/GetPDF.aspx?doi=10.1155/DDNS/2006/79295

[11] Hamdan, S. A. (Januar 2008): *„Hybrid Particle Swarm Optimiser using multi-neighborhood topologies"*. Department of Computing and Information Technolo gy Arab Open University Kuwait:
URL http://www.dcc.ufla.br/infocomp/artigos/v7.1/art05.pdf

[12] Mendes, Rui und Kennedy, James und Neves, José (Januar 2005): *„The Fully Informed Particle Swarm : Simpler Maybe Better"*. IEEE TRANSACTIONS OF EVOLUTIONARY COMPUTATION, VOL. 1, NO. 1:
URL http://www.di.uminho.pt/~rcm/publications/FIPS_TEC.pdf

[13] Riccardo Poli und James Kennedy und Tim Blackwell (August 2007): *„Particle swarm optimization - An overview"*. Swarm Intell (2007) 1: 33–57:
URL http://sci2s.ugr.es/docencia/sf1/Particle-swarm-optimization-SI-Vol-1.pdf

[14] Eberhart, Russell C. und Hu, Xiaohui : *„Multiobjective Optimization using Dy namic Neighborhood Particle Swarm Optimization“*. Department of Biomedical Engineering Purdue University, West Lafayette, Indiana, USA und Departe ment of Electrical and Computer Engineering Purdue School of Engineering and Technology, Indianapolis, Indiana, USA: URL http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=8FA4F2A2191E8A5D70C02D3E71C60F61?doi=10.1.1.20.199&rep=rep1&type=pdf

[15] Yisu, Jin und Knowles, Joshua und Hongmei, Lu und Yizeng, Liang und Kell, Douglas B. (Januar 2007): *„The landscape adaptive particle swarm optimizer“*. Col lege of Chemistry and Chemical Engineering, Central South University, Changsha 410083, PR China und School of Chemistry, University of Manches ter, Manchester M60 1QD, UK:
URL http://dbkgroup.org/Papers/jin_adaptivePSO_published.pdf

[16] Müller, Alexander (August 2008): *„Evolutionsalgorithmen (Evolutionäre Algorith men)“*.
URL http://www.evocomp.de/themen/evolutionsalgorithmen/evoalg.html

[17] Weicker, Karsten : *„Evolutionäre Algorithmen“*. Institut für Informatik, Abtei lung Formale Konzepte Universität Stuttgart, Breitwiesenstr. 20-22, 70565 Stuttgart:
URL http://www.imn.htwk-leipzig.de/~weicker/publications/sctreff_ea.pdf

[18] Hansen, Nikolaus (April 2008): *„The CMA Evolution Strategy: A Tutorial“*. Technische Universität Berlin Fachbereich Bionik
URL http://www.bionik.tu-berlin.de/user/niko/cmatutorial.pdf

[19] Clerc, Maurice (2001): *„PSO: A few words about constriction“*.
URL http://clerc.maurice.free.fr/pso/A_few_words_about_constri.zip

[20] Y. COOREN, M. CLERC, P. SIARRY (November 2006): *„Tribes – a parameter free particle swarm optimization algorithmn“*.
URL http://www.particleswarm.info/Tribes_2006_Cooren.pdf

[21] Clerc, Maurice : *„PSO, Tribes“*.
URL http://clerc.maurice.free.fr/pso/

[22] Wikipedia : *„Repulsive Particle Swarm Optimization“*
URL http://en.wikipedia.org/wiki/Repulsive_particle_swarm_optimization

[23] Institut für Neuroinformatik, Ruhr-Universität Bochum: *„The official Shark-Project site“*
URL http://shark-project.sourceforge.net/

[24] Hansen, Nikolaus (April 2005): *„Problem Definitions and Evaluation Criteria for the CEC 2005* Special Session on Real-Parameter Optimization“.
Technische Universität Berlin Fachbereich Bionik
URL http://www.bionik.tu-berlin.de/user/niko/Tech-Report-May-30-05.pdf

[25] Hirotaka, Yoshida ; Kawata, Kenichi ; Fukuyama, Yoshikazu ; Takayama,

Shinichi und Nakanishi Yosuke (November 2001): „A PARTICLE SWARM OPTIMIZATION FOR REACTIVE POWER AND VOLTAGE CONTROL CONSIDERING VOLTAGE SECURITY ASSESMENT“.

URL http://www.particleswarm.info/PSOVQC_fukuyama.PDF

[26] Wikipedia : URL: http://www.wikipedia.de

[27] Hansen, Nikolaus (2005): “A Restart CMA Evolution Strategy With In creasing Population Size”

URL http://www.bionik.tu-berlin.de/user/niko/cec2005ipopcmaes.pdf

Abbildungsverzeichnis

1.1 *„Visualisierung konvexer Funktionsoptimierung durch Verwendung des Gradienten“:* Erstellt mit Gnuplot

1.2 *„Beispielabbildung einer multimodalen Funktion“*
URL: http://www-optima.amp.i.kyoto-u.ac.jp/member/student/hedar/Hedar_files/TestGO_files/Page2607.htm

1.3 *„Optimierungsschemata evolutionärer Algorithmen“*
URL: http://www.imn.htwk-leipzig.de/~weicker/publications/sctreff_ea.pdf

2.1 *„Aufbau eines Partikels der PSO“:* Erstellt mit Open Office Draw

2.2 *„Grafische Visualisierung der Partikelbewegung“:* Erstellt mit Open Office Draw

2.3 *„Der Partikel-Schwarm-Grundalgorithmus als Pseudo-Code“:*
Erstellt mit Open Office Math

2.4 *„Der PSO-Grundalgorithmus als Pseudo-Code für binäre Probleme“*:
Erstellt mit Open Office Math

2.5 *„Grafische darstellung der Partikelbewegungung unter Nutzung einer Zufallsvariable einer D-Kugel“:* Erstellt mit Open Office Draw

2.6 *„Mean und Varianz bei Verwendung der Normalverteilung“:*
Erstellt mit Open Office Math

2.8 *„Grafische darstellung der Partikelbewegungung unter Nutzung einer Zufallsvariable der Normalverteilung“:* Erstellt mit Open Office Draw

2.9 *„Die Ring-Nachbarschaftsbeziehung“:* Erstellt mit Open Office Draw

2.10 *„Die Zufall-Nachbarschaftsbeziehung“:* Erstellt mit Open Office Draw

2.11 *„Die Wheel-Nachbarschaftsbeziehung“:* Erstellt mit Open Office Draw

2.12 *„Die Von-Neumann-Nachbarschaftsbeziehung“:*
Erstellt mit Open Office Draw

2.13 *„Nachbarschaftsgrößen-Verlauf dynamischer Nachbarschaften“:*
Erstellt mit MS Paint

2.14 *„Memory-Swarm-Nachbarschaftsbeziehung“:*
Erstellt mit Open Office Draw

2.15 *„Darstellung des DistributionVektors“:*
Erstellt mit Open Office Draw

2.16 *„Landscape Adaptive Pseudo-Code“:*
Erstellt mit Open Office Draw

2.17 *„Tribes – Schwärme“*
URL http://www.particleswarm.info/Tribes_2006_Cooren.pdf

2.18 *„Tribes – Aktualisierungsregeln“*

URL http://www.particleswarm.info/Tribes_2006_Cooren.pdf

3.1 *„Partikel folgen dem gewichteten Mittel“*: Erstellt mit OpenOffice Draw

3.2 *„Partikel folgen nicht dem gewichteten Mittel“:* Erstellt mit OpenOffice Draw

3.3 *„Mean Based PSO Pseudo-Codel“:* Erstellt mit OpenOffice Math

4.1 *„Die Klasse Particle“* : Erstellt mit Visual Studio 2005

4.2 *„Die Klasse Swarm“* : Erstellt mit Visual Studio 2005

4.3 *„Die Schnittstelle IDistribution“* : Erstellt mit Visual Studio 2005

4.4 *„Die Schnittstelle ITopology“* : Erstellt mit Visual Studio 2005

4.5 *„Die Schnittstelle IFitness“* : Erstellt mit Visual Studio 2005

4.6 *„Die abstrakte Klasse Function“* : Erstellt mit Visual Studio 2005

4.7 *„Die abstrakte Klasse Pso“* : Erstellt mit Visual Studio 2005

5.1a *„Transformierte Funktion“* :
„http://www.bionik.tu-berlin.de/user/niko/handout.pdf“

5.1 *„Die Funktion Sphere für D=2“* : http://www-optima.amp.i.kyoto-u.ac.jp/member/student/hedar/Hedar_files/TestGO_files/Page364.htm

5.2 *„Die Funktion Ackley für D=2“* : http://www-optima.amp.i.kyoto- u.ac.jp/member/student/hedar/Hedar_files/TestGO_files/Page364.htm

5.3 *„Die Funktion Griewank für D=2“* : http://www-optima.amp.i.kyoto-u.ac.jp/mem ber/student/hedar/Hedar_files/TestGO_files/Page364.htm

5.4 *„Die Funktion Rastrigin für D=2“* : http://www-optima.amp.i.kyoto-u.ac.jp/mem ber/student/hedar/Hedar_files/TestGO_files/Page364.htm

5.5 *„Die Funktion Rosenbrock für D=2“* : http://www-optima.amp.i.kyoto-u.ac.jp/mem ber/student/hedar/Hedar_files/TestGO_files/Page364.htm

5.6 *-- Entfernt --*

5.7 *„ZufallsvariablenVergleich Sphere 30D“* : Gnu-Plot

5.8 *„Topologievergleich Sphere (unimodal)“* : Gnu-Plot

5.9 *„Topologievergleich Rastrigin (multimodal)“* : Gnu-Plot

5.10 *„PSO Varianten Vergleich „Sphere unshifted““* : Gnu-Plot

5.11 *„PSO Varianten Vergleich „Sphere shifted““* : Gnu-Plot

5.12 *„Fitnessverlauf nicht verschobener Funktionen“* : Gnu-Plot

5.13 *„Fitnessverlauf verschobener Funktionen“* : Gnu-Plot

Formelverzeichnis

1. *„Statische Geschwindigkeitsadaption“:* Erstellt mit OpenOffice Math
2. *„Variable Lernfaktoren“:* Erstellt mit OpenOffice Math
3. *„Geschwindigkeitsadaption der klassischen PSO“:* Erstellt mit OpenOffice Math
4. *„Die Sigmoidfunktion“:* Erstellt mit OpenOffice Math
5. *„Volumen einer D-dimensionalen Kugel“:* Erstellt mit OpenOffice Math
6. *„Radiusberechnung einer D-dimensionalen Kugel mit dem Volumen 1“*: Erstellt mit OpenOffice Math
7. *„Dichtefunktion der Normalverteilung“:* Erstellt mit OpenOffice Math
8. *„Berechnung der neuen Partikelposition durch die Pivot-Methode“:* Erstellt mit OpenOffice Math
9. *„Beschränkung der Partikel auf den Definitionsbereich“:* Erstellt mit OpenOffice Math
10. *„Zufälliges Setzen der Partikel-Dimension beim Verlassen des Def. Bereichs“:* Erstellt mit OpenOffice Math
11. *„Geschwindigkeitsbeschränkung durch Vmax“:* Erstellt mit OpenOffice Math
12. *„Sinnvolle Wahl von Vmax“:* Erstellt mit OpenOffice Math
13. *„Die „inertia weight“ Konvergenz-Methode“:* Erstellt mit OpenOffice Math
14. *„Die "global-local-inertia wieght" Konvergenz-Methode“:* Erstellt mit OpenOffice Math
15. *„Euklidische-Distanz“:* Erstellt mit OpenOffice Math
16. *„Berechnung des Distribution Vectors in LAPSO“:* Erstellt mit OpenOffice Math
17. *„Geschwindigkeitsberechung in LAPSO“:* Erstellt mit OpenOffice Math
18. *„Beschränkung des Distribution Vectors“:* Erstellt mit OpenOffice Math
19. *„FIPS-Geschwindigkeitsberechnung“*: Erstellt mit OpenOffice Math
20. *„Die Berechnung des gewichteten arthimetischen Mittels“:* Erstellt mit OpenOffice Math
21. *"Berechung des Pseudo-Gradienten"*: Erstellt mit OpenOffice Math
22. *"Berechung der Spannweite"*: Erstellt mit OpenOffice Math
23. *"Pseudo-Gradienten-Schranke"*: Erstellt mit OpenOffice Math
24. *"Reinitialisierungsauswahl"*: Erstellt mit OpenOffice Math
25. *"Berechung der reinitialisierten Partikelposition"*: Erstellt mit OpenOffice Math
26. *„CMA-ES Parameter“*: Erstellt mit OpenOffice Math
27. *„Linear Funktion“*: Erstellt mit OpenOffice Math
28. *„Sphere Funktion“*: Erstellt mit OpenOffice Math

29.*„Paraboloid Funktion“*: Erstellt mit OpenOffice Math
30.*„Cigar Funktion“*: Erstellt mit OpenOffice Math
31.*„Tablet Funktion“*: Erstellt mit OpenOffice Math
32.*„Ackley Funktion“*: Erstellt mit OpenOffice Math
33.*„Griewank Funktion“*: Erstellt mit OpenOffice Math
34.*„Rastrigin Funktion“*: Erstellt mit OpenOffice Math
35.*„Rosenbrock Funktion“*: Erstellt mit OpenOffice Math

Anhang (CD)

Die beiliegende CD enthält den PSO-Quellcode, dessen UML-Klassendiagramm, den verwendeten Quellcode der CMA-ES sowie die Ergebnis-Plots.