Otto-von-Guericke-Universität Magdeburg

Fakultät für Informatik
Institut für Intelligente Kooperierende Systeme

Masterarbeit

Dynamische Distanzminimierungprobleme mit variablem Schwierigkeitsgrad für multikriterielle Optimierung

Verfasser:
André Kottenhahn

14. Dezember 2016

Betreuer:
Prof. Dr.-Ing. habil. Sanaz Mostaghim
Heiner Zille, M.Sc.

Erstprüfer:
Prof. Dr.-Ing. habil. Sanaz Mostaghim

Zweitprüfer:
Prof. Dr. rer. nat. habil. Rudolf Kruse

Universität Magdeburg
Fakultät für Informatik
Postfach 4120, D–39016 Magdeburg

Kottenhahn, André:
Dynamische Distanzminimierungprobleme mit variablem Schwierigkeitsgrad für multikriterielle Optimierung
Masterarbeit, Otto-von-Guericke-Universität Magdeburg, 2016.

Zusammenfassung

In dieser Arbeit wird ein dynamisches Distanzminimierungsproblem basierend auf dem statischen Distanzminimierungsproblem definiert und implementiert. Diese beiden Probleme werden durch drei Algorithmen bearbeitet und die Ergebnisse ausgewertet. Als Algorithmen werden NSGA-II (evolutionärer Algorithmus), SMPSO (schwarmbasierter Algorithmus) und GRA (Algorithmus nach dem Prinzip „Teile und Herrsche“) verwendet. GRA stellt dabei eine neue Herangehensweise an multikriterielle Probleme dar und wird in dieser Arbeit erstmals beschrieben. Weiterhin wird untersucht, welche Eigenschaften einer Dynamik die Lösung eines Distanzminimierungsproblems erschweren. Hierbei wird deutlich, dass die Schwierigkeit eines dynamischen Problems im Kern von der Häufigkeit und den Auswirkungen der Änderungen abhängt. Es wird weiterhin gezeigt, dass mit GRA, im Falle der Verwendung der Summennorm als Abstandsmaß, deutlich bessere Ergebnisse erzielt werden können als bei der Verwendung der anderen Algorithmen.

Inhaltsverzeichnis

Inhaltsverzeichnis **i**

Abbildungsverzeichnis **iv**

Tabellenverzeichnis **viii**

Listings **ix**

Verzeichnis der Abkürzungen **x**

1 Einleitung **1**

1.1 Hintergrund 1

1.2 Ziele der Arbeit 2

1.3 Methodik der Arbeit 2

1.4 Aufbau der Arbeit 2

2 Grundlagen **3**

2.1 Multikriterielle Optimierungsprobleme 3

2.2 Distanzminimierungsproblem 6

2.2.1 p-Normen 7

2.2.2 Pareto-optimale Lösungsmenge des Distanzminimierungsproblems 8

2.2.3 Statisches und dynamisches DMP 10

2.3 Evolutionäre Algorithmen 10

2.4 Partikelschwarmoptimierung 12

2.5 Qualitätsindikatoren 13

2.5.1 Anzahl der Lösungen in pareto-optimaler Menge 14

2.5.2 Streuung 14

2.5.3 Hypervolumen . . . 15
2.6 Verwandte Literatur . . . 19

3 Ausgewählte Algorithmen 21
3.1 NSGA-II . . . 21
3.2 DNSGA-II-A . . . 24
3.3 SMPSO . . . 24
3.4 dSMPSO . . . 25
3.5 GRA . . . 26
3.5.1 GRA und dynamische Probleme . . . 29

4 Dynamisches Distanzminimierungsproblem 32
4.1 Motivation . . . 32
4.2 Problemstellung . . . 33
4.2.1 Änderung der Anzahl an Variablen . . . 34
4.2.2 Änderung der Anzahl an Zielpunkten . . . 34
4.2.3 Änderung der Lage der Zielpunkte . . . 35
4.2.4 Änderung der Zielfunktionen . . . 36
4.3 Umsetzung . . . 36
4.3.1 Änderung der Anzahl an Variablen . . . 38
4.3.2 Änderung der Anzahl an Zielpunkten . . . 39
4.3.3 Änderung der Lage der Zielpunkte . . . 40
4.3.4 Änderung der Zielfunktionen . . . 43
4.4 Implementierte Änderungen . . . 43

5 Evaluation 45
5.1 Parametereinstellungen für die Algorithmen . . . 45
5.2 Test mit dynamischem DMP . . . 46
5.2.1 Dynamisches DMP mit Summennorm . . . 47
5.2.2 Dynamisches DMP mit euklidischer Norm . . . 50
5.3 Test mit statischem DMP . . . 54
5.3.1 Statisches DMP mit Summennorm und Maximalprinzip . . . 55
5.3.2 Statisches DMP mit Summennorm und Minimalprinzip . . . 56
5.3.3 Statisches DMP mit euklidischer Norm und Maximalprinzip . . . 59

5.3.4 Statisches DMP mit euklidischer Norm und Minimalprinzip . . . 62
5.4 Analyse der Ergebnisse 66

6 Zusammenfassung und Ausblick **70**
6.1 Schlussfolgerung 70
6.2 Zukünftige Forschung 71

Literaturverzeichnis **72**

Abbildungsverzeichnis

2.1 Grundriss des Raums 4
2.2 Pareto-optimale Lösungsmenge P 5
2.3 Pareto-optimale Front POF 5
2.4 Einzig möglicher Abstand nach euklidischer Norm 7
2.5 Drei mögliche, gleichgroße Abstände nach Summennorm 7
2.6 Pareto-optimale Lösungsmenge für $p = 2$ 8
2.7 Pareto-optimale Menge für zwei Zielpunkte und Summennorm 9
2.8 Pareto-optimale Menge für drei Zielpunkte und Summennorm 9
2.9 Dominanz zweier Punkte bei Summennorm 9
2.10 Schematischer Ablauf eines evolutionären Algorithmus 11
2.11 Anziehung 12
2.12 Abstossung 12
2.13 Ausrichtung 12
2.14 Berechnung des Bewegungsvektors $\vec{v}(t+1)$ 13
2.15 Streuungen von 0 nach [1] 15
2.16 Streuung > 0 nach [2] 16
2.17 Streuung von 0 nach [2] 16
2.18 Berechnung des Hypervolumens aus gefunden Lösungen 16
2.19 Einfaches DMP 17

3.1 Binäre Kreuzung zweier 8-Bit-Blöcken ab Block 6 22
3.2 Crowding Distance nach [2] 24
3.3 GRA - Initiale Lösung auf Gitter 26
3.4 PAP für GRA 27

3.5 PAP zur Entscheidung über die Aufnahme einer neuen Lösung in die Population 28
3.6 GRA - Erzeugte Lösungen für $\gamma = 1.0$ 28
3.7 GRA - Mögliche Lösungen für $\gamma = 0.5$ 28
3.8 GRA - Keine neuen Punkte möglich 29
3.9 GRA - Verfeinerung des Gitters 29

4.1 Probleminstanz mit zwei Variablen 34
4.2 Probleminstanz mit drei Variablen 34
4.3 Probleminstanz mit drei Zielpunkten 35
4.4 Probleminstanz mit vier Zielpunkten 35
4.5 Ausgangslage vor Verschiebung entlang der x_0-Achse 35
4.6 Ergebnis nach Verschiebung entlang der x_0-Achse 35
4.7 Pareto-optimale Menge dreier Zielpunkte mit $p = 2$ 36
4.8 Pareto-optimale Menge der Zielpunkte mit $p = 1$ 36
4.9 Schematischer Ablauf von Evaluation und dynamischer Änderung 37
4.10 Pareto-optimale Menge der Zielpunkte bei Nicht-Beachtung der Zielfunktion für Z_4: Diskrepanz zwischen Darstellung im Entscheidungsraum und der pareto-optimalen Menge 39
4.11 Verschieben von Punkt Z_4 auf Position von Punkt Z_3 40
4.12 Übereinstimmung von Darstellung im Entscheidungsraum und pareto-optimaler Menge 40
4.13 Translation von Punkt $\vec{Z}_1$ entlang der x_0-Achse 41
4.14 Ergebnis der Translation 41
4.15 Rotation von Punkt Z_3 42
4.16 Ergebnis der Rotation 42
4.17 Vor der Skalierung ($r_1 = r_3 = a$) 43
4.18 Nach der Skalierung (Verdoppelung von a) 43

5.1 Dynamischer Testfall für Summennorm 48
5.2 Median-HVR-Werte vor und nach einer Änderung (Summennorm) . . . 49
5.3 Medianwerte für die Streuung vor und nach einer Änderung (Summennorm) 49
5.4 Dynamischer Testfall für euklidische Norm bei zwei Variablen 51

5.5 Median-HVR-Werte vor und nach einer Änderung bei zwei Variablen (euklidische Norm) . . . 52
5.6 Median-HVR-Werte vor und nach einer Änderung bei zehn Variablen (euklidische Norm) . . . 52
5.7 Median-HVR-Werte vor und nach einer Änderung bei 50 Variablen (euklidische Norm) . . . 52
5.8 Median-HVR-Werte vor und nach einer Änderung bei 100 Variablen (euklidische Norm) . . . 52
5.9 Streuung vor und nach einer Änderung bei zwei Variablen (euklidische Norm) . . . 53
5.10 Streuung vor und nach einer Änderung bei zehn Variablen (euklidische Norm) . . . 53
5.11 Streuung vor und nach einer Änderung bei 50 Variablen (euklidische Norm) 53
5.12 Streuung vor und nach einer Änderung bei 100 Variablen (euklidische Norm) 53
5.13 Testfall 1 für Summennorm . . . 54
5.14 Testfall 2 für Summennorm . . . 54
5.15 Statischer Testfall für euklidische Norm mit zwei Variablen . . . 55
5.16 Verlauf der HVR des Median-Durchlaufs für Testfall 1 (Summennorm) . 56
5.17 Verlauf der HVR des Median-Durchlaufs für Testfall 2 (Summennorm) . 56
5.18 Boxplot der HVR der 51 Durchläufe für Testfall 1 (Summennorm) 56
5.19 Boxplot der HVR der 51 Durchläufe für Testfall 2 (Summennorm) 56
5.20 Erreichung der HVR-Werte von 0.8, 0.85 und 0.9 für Testfall 1 für Median-Durchlauf (log. Darstellung) . . . 58
5.21 Erreichung der HVR-Werte von 0.8, 0.85 und 0.9 für Testfall 2 für Median-Durchlauf (log. Darstellung) . . . 58
5.22 Verlauf der HVR des Median-Durchlaufs bei Testfall 1 . . . 59
5.23 Verlauf der HVR des Median-Durchlaufs bei Testfall 2 . . . 59
5.24 Boxplot für die Erreichung einer HVR von 0.9 der 51 Durchläufe bei Testfall 1 . . . 59
5.25 Boxplot für die Erreichung einer HVR von 0.85 der 51 Durchläufe bei Testfall 2 . . . 59
5.26 Entwicklung der Median-HVR bei zunehmender Anzahl an Variablen . . 60
5.27 Entwicklung der Median-HVR_{opt} bei zunehmender Anzahl an Variablen . 60

5.28 Entwicklung der Median-n_{opt} bei zunehmender Anzahl an Variablen . . . 60
5.29 Entwicklung der Median-Streuung bei zunehmender Anzahl an Variablen 60
5.30 Verlauf der HVR des Median-Durchlaufs bei zwei Variablen 61
5.31 Verlauf der HVR des Median-Durchlaufs bei zehn Variablen 61
5.32 Verlauf der HVR des Median-Durchlaufs bei 50 Variablen 61
5.33 Verlauf der HVR des Median-Durchlaufs bei 100 Variablen 61
5.34 Boxplot der HVR der 51 Durchläufe bei zwei Variablen 61
5.35 Boxplot der HVR der 51 Durchläufe bei zehn Variablen 61
5.36 Boxplot der HVR der 51 Durchläufe bei 50 Variablen 62
5.37 Boxplot der HVR der 51 Durchläufe bei 100 Variablen 62
5.38 Erreichung der HVR-Werte von 0.8, 0.85 und 0.9 für Median-Durchlauf bei zwei Variablen 64
5.39 Erreichung der HVR-Werte von 0.8, 0.85 und 0.9 für Median-Durchlauf bei zehn Variablen (log. Darstellung) 64
5.40 Erreichung der HVR-Werte von 0.8, 0.85 und 0.9 für Median-Durchlauf bei 50 Variablen (log. Darstellung) 64
5.41 Erreichung der HVR-Werte von 0.8, 0.85 und 0.9 für Median-Durchlauf bei 100 Variablen (log. Darstellung) 64
5.42 Verlauf der HVR des Median-Durchlaufs bei zwei Variablen 65
5.43 Verlauf der HVR des Median-Durchlaufs bei zehn Variablen 65
5.44 Verlauf der HVR des Median-Durchlaufs bei 50 Variablen 65
5.45 Verlauf der HVR des Median-Durchlaufs bei 100 Variablen 65
5.46 Boxplot für die Erreichung einer HVR von 0.9 der 51 Durchläufe bei zwei Variablen 65
5.47 Boxplot für die Erreichung einer HVR von 0.9 der 51 Durchläufe bei zehn Variablen 65
5.48 Boxplot für die Erreichung einer HVR von 0.9 der 51 Durchläufe bei 50 Variablen 66
5.49 Boxplot für die Erreichung einer HVR von 0.9 der 51 Durchläufe bei 100 Variablen 66

Tabellenverzeichnis

5.1 Dynamischer Testfall - Änderung der HVR bei Summennorm 48
5.2 Dynamischer Testfall - Änderung der Streuung bei Summennorm 49
5.3 Dynamischer Testfall - Änderung der HVR bei euklidischer Norm 50
5.4 Dynamischer Testfall - Änderung der Streuung bei euklidischer Norm . . 52
5.5 Maximalprinzip für Summennorm 55
5.6 Minimalprinzip für Summennorm - Evaluationen zur Erreichnung einer HVR 57
5.7 Minimalprinzip für Summennorm - Evaluationen zur Erreichnung einer HVR_{opt} 57
5.8 Minimalprinzip für Summennorm - Erreichte Werte am Ende der maximalen Evaluationen 58
5.9 Maximalprinzip für euklidische Norm 59
5.10 Minimalprinzip für euklidische Norm - Evaluationen zur Erreichnung einer HVR 62
5.11 Minimalprinzip für euklidische Norm - Evaluationen zur Erreichnung einer HVR_{opt} 63
5.12 Minimalprinzip für euklidische Norm - erreichte Werte am Ende der maximalen Evaluationen 63
5.13 Vergleich von GRA mit NSGA-II / DNSGA-II-A 68
5.14 Vergleich von GRA mit SMPSO / dSMPSO 69

Listings

3.1 Pseudo-Code für NSGA-II nach [2] 21
3.2 Pseudo-Code für Binary Tournament nach [2] 22
3.3 Crowding Distance nach [2] 23
3.4 Pseudo-Code für SMPSO nach [3] 25
3.5 Pseudo-Code für GRA 31
4.1 Beispiel Schrittart und Schrittweite 37
4.2 Beispiel Änderung der Parameters p 43
4.3 Pseudo-Code Parameterprüfung 44

Verzeichnis der Abkürzungen

DMP Distanzminimierungsproblem

DNSGA-II-A Dynamic non-dominated Sorting Genetic Algorithm II A

dSMPSO Dynamic speed-constrained Particle Swarm Optimization

EA Evolutionärer Algorithmus

FDA Farina-Deb-Amato

GRA Grid Refinement Algorithm

HV Hypervolume

HVR Hypervolume Ratio

NSGA-II Non-dominated Sorting Genetic Algorithm II

P Pareto-optimale Lösungsmenge

PAP Programmablaufplan

POF Pareto-optimale Front

PSO Partikelschwarmoptimierung

SMPSO Speed-constrained Particle Swarm Optimization

ZDT Zitzler–Deb–Thiele

Kapitel 1

Einleitung

In diesem Kapitel werden die Hintergründe der in dieser Arbeit besprochenen multikriteriellen Optimierungsprobleme erläutert. Es wird ausgehend von einem praktischen Beispiel in die Thematik eingeführt. Weiterhin werden Ziele, Methodik und Aufbau der Arbeit kurz beschrieben.

1.1 Hintergrund

Bei der Lösung von Problemen werden oft mehrere Ziele verfolgt, welche in der Regel konkurrierend und daher nicht gleichzeitig erreichbar sind. Diese Art von Problemen werden als multikriterielle Probleme bezeichnet. Es gibt bei ihnen nicht eine Lösung, sondern eine Menge von Lösungen, die jeweils ein Optimum darstellen. Ein Beispiel dafür wäre die Herstellung eines Autos. Hierbei versucht der Hersteller verschiedene, konkurrierende Aspekte zu vereinigen. So soll das Auto beispielsweise besonders sicher, leistungsstark, kraftstoffsparend und günstig sein. Um die Sicherheit zu erhöhen könnten stabilere Materialien verbaut werden oder zusätzliche Sicherheitstechnik. Die zusätzliche Technik wird den Preis des Fahrzeuges erhöhen, die stabileren Materialien das Gewicht. Mit einem höheren Gewicht ist das Fahrzeug wiederum träger, was beim Kunden als Mangel an Motorleistung angesehen werden könnte. Ein leistungsstärkerer Motor wiederum wird mit einem höheren Verbrauch einhergehen und dem Ziel des geringen Kraftstoffverbrauchs entgegenwirken. Für die Lösung dieses Beispielproblems wird ein Kompromiss aus den anfangs gesetzten Zielen notwendig sein. Je nach Schwerpunkt bei der Kompromissfindung sind hierbei verschiedene Lösungen denkbar. Damit wird deutlich, dass multikriterielle Optimierungsprobleme eine Menge an möglichen Lösungen als Ergebnis haben. Diese optimalen Lösungen können sich zusätzlich mit der Zeit verändern, indem sich beispielsweise die Beschaffungspreise für die jeweiligen Materialien ändern. Um solche Probleme formal lösen zu können ist es notwendig die Aspekte und Zusammenhänge der verschiedenen Zielstellungen mathematisch zu modellieren. Zur Lösungsfindung selbst gibt es bereits verschiedene Algorithmen mit unterschiedlichen Ansätzen, welche solche Probleme lösen bzw. gleichwertige Lösungsalternativen aufzeigen können. Für den Test der Leistungsfähigkeit solcher Algorithmen werden oft genormte Tests und Benchmarks verwendet. Jedoch sind diese oftmals nicht in ihrer Komplexität skalierbar und/oder nicht gut visualisierbar und analysierbar. Mit dieser Arbeit soll der Versuch unternommen werden diese Fähigkeitslücke für dynamische Probleme zu schließen.

1.2 Ziele der Arbeit

Diese Arbeit verfolgt drei aufeinander aufbauende Ziele. Das erste Ziel ist die Ausarbeitung und Vorstellung einer dynamischen Variante des Distanzminimierungsproblems. Als zweites Ziel sollen die Leistungen dynamischer Algorithmen für dieses Problem bestimmt und untereinander verglichen werden. Das dritte Ziel ist der Vergleich der Leistungsfähigkeit der entsprechenden statischen Algorithmen für das statische Distanzminimierungsproblem und die Untersuchung, ob ein Zusammenhang zwischen der Schwierigkeit des statischen und des dynamischen Distanzminimierungsproblems existiert.

1.3 Methodik der Arbeit

Zur Erreichung der soeben vorgestellten Ziele wird folgende methodische Vorgehensweise gewählt. Zuerst werden die Anforderungen an das dynamische Distanzminimierungsproblem definiert und selbiges darauf aufbauend erstellt. Danach werden die Algorithmen ausgewählt, welche sowohl für die Lösung des dynamischen als auch des statischen Distanzminimierungsproblems angewendet werden. Als nächstes wird eine Testumgebung zur Generierung beliebig komplexer dynamischer Distanzminimierungsprobleme erstellt. Danach erfolgt die Durchführung der Tests. Zum Abschluss werden die Ergebnisse ausgewertet und Schlussfolgerungen gezogen.

1.4 Aufbau der Arbeit

Im ersten Kapitel „Einleitung“ werden Inhalt, Ziele und Methodik der Arbeit beschrieben. Im zweiten Kapitel „Grundlagen“ erfolgt eine Darstellung der wichtigsten Themenbereiche, auf welche im Rahmen dieser Arbeit Bezug genommen wird. Das dritte Kapitel „Ausgewählte Algorithmen“ stellt die zur Evaluation des statischen und dynamischen Distanzminimierungsproblems verwendeten Algorithmen vor. Im vierten Kapitel „Dynamisches Distanzminimierungsproblem“ wird die Motivation, welche zur Erstellung des neuen Problems geführt hat beschrieben. Weiterhin werden die Anforderungen und deren Umsetzung vorgestellt. Im fünften Kapitel „Evaluation“ werden die durchgeführten Tests beschrieben und deren Ergebnisse dargestellt und analysiert. Im letzten Kapitel „Abschluss“ werden die im Rahmen dieser Arbeit gewonnenen Erkenntnisse zusammengefasst. Auch werden mögliche zukünftige Forschungsthemen erwähnt, die auf den Erkenntnissen dieser Arbeit aufbauen.

Kapitel 2

Grundlagen

In diesem Kapitel werden die in dieser Arbeit verwendeten Begriffe und Verfahren vorgestellt. Diese werden innerhalb der einzelnen Abschnitte aufeinander aufbauend beschrieben. Im letzten Abschnitt findet sich eine Aufzählung verwandter Literatur, die diesem Forschungsgebiet zuzuordnen ist.

2.1 Multikriterielle Optimierungsprobleme

Multikriterielle Probleme sind Probleme, bei welchen mehr als ein Ziel verfolgt wird. Oft sind diese Ziele miteinander konkurrierend und können nicht alle gleichzeitig optimal erreicht werden, da die Verbesserung bezüglich eines Ziels eine Verschlechterung bezüglich eines anderen Ziels zur Folge hat. Mathematisch kann ein Minimierungsproblem wie folgt formuliert werden:

$$\text{Minimiere } \vec{f}(\vec{x}) = (f_1(\vec{x}), ..., f_k(\vec{x})) \text{ mit } \vec{x} \in \Omega$$

Hierbei ist Ω der Entscheidungsraum (*decision space*), in welchem sich eine Lösung $\vec{x}$ befinden kann. Das Optimierungsproblem besteht weiterhin aus k Zielfunktionen. Diese bilden jede Lösung aus Ω in $\mathbb{R}^k$ ab, wobei $\mathbb{R}^k$ den k-dimensionalen Zielraum (*objective space*) darstellt: $f : \Omega \to \mathbb{R}^k$. Der Funktionswert $\vec{f}(\vec{x})$ einer Lösung $\vec{x}$ aus Ω stellt somit einen Vektor der Größe k dar, welcher aus $(f_1(\vec{x}), ..., f_k(\vec{x}))^T$ besteht:

$$\vec{f}(\vec{x}) = (f_1(\vec{x}), f_2(\vec{x}), f_3(\vec{x}), ..., f_k(\vec{x}))^T$$

Aufgrund der Tatsache, dass die Verbesserung bezüglich eines Ziels in der Regel die Verschlechterung bezüglich eines anderen Ziels zur Folge hat ergibt sich, dass es nicht nur eine einzige optimale Lösung für das Problem gibt, sondern dass es eine Menge von optimalen Lösungen gibt. Jede Lösung aus dieser Menge erreicht, im Vergleich mit einer anderen Lösung dieser Menge, ein bestimmtes Ziel besser und ein anderes Ziel schlechter. Formal wird diese Lösungsmenge als pareto-optimal bezeichnet [4]. Pareto-optimale Lösungsmengen (P) zeichnen sich dadurch aus, dass ihre Lösungen nicht-dominiert sind. Eine Lösung dominiert eine andere Lösung, wenn sie alle Ziele mindestens gleich gut erreicht und mindestens ein Ziel besser. Mathematisch formuliert ergibt sich für ein Minimierungsproblem mit k Zielen und den entsprechenden Zielfunktionen f_1 bis f_k, dass

die Lösung $\vec{x_1}$ eine andere Lösung $\vec{x_2}$ dominiert (dargestellt als $\vec{x_1} \prec \vec{x_2}$), genau dann, wenn gilt:

$$f_n(\vec{x_1}) \leq f_n(\vec{x_2}), \forall n \in \{1, ..., k\} \wedge \exists i : f_i(\vec{x_1}) < f_i(\vec{x_2}) \tag{2.1}$$

Die pareto-optimale Lösungsmenge befindet sich dabei im Entscheidungsraum Ω. Die zugehörigen Funktionswerte der einzelnen Lösungen befinden sich im Zielraum $\mathbb{R}^k$.
Die Menge der Funktionswerte aller pareto-optimalen Lösungen bildet die pareto-optimale Front (POF). Bei der Lösung multikriterieller Probleme ist das Ziel eine möglichst große Annäherung an die pareto-optimale Front, um möglichst optimale Lösungen zu erhalten. Man spricht hierbei von Konvergenz. Ein weiteres Ziel ist das Erreichen einer größtmöglichen Diversität, also der Unterschiedlichkeit der einzelnen Lösungen. Dadurch stehen verschiedenartige Lösungen zur Verfügung, aus denen nach weiteren Kriterien ausgewählt werden kann.
Nachfolgend sollen die Begriffe der pareto-optimalen Menge P und der pareto-optimalen Front POF an einem einfachen Beispiel veranschaulicht werden. Gegeben ist der Grundriss eines Raumes welcher über zwei Eingänge E_1 und E_2 verfügt (s. Abbildung 2.1).

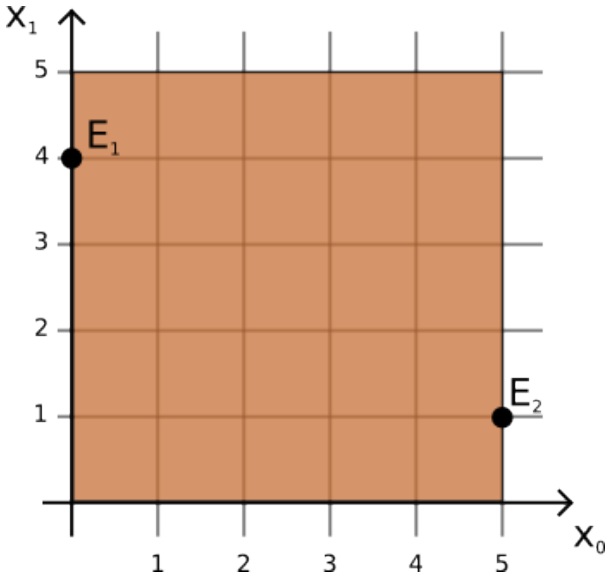

Abbildung 2.1: Grundriss des Raums

Das Ziel ist nun eine Position innerhalb dieses Raumes zu finden, bei welcher sowohl der Abstand zu $\vec{E_1}$ als auch der Abstand zu $\vec{E_2}$ minimal ist. Der Abstand eines Punktes $\vec{A}$ berechnet sich über die Abstandsfunktionen f_1 für $\vec{E_1}$ und f_2 für $\vec{E_2}$.

$$f_1(\vec{A}) = \sqrt{(E_{1_{x_0}} - A_{x_0})^2 + (E_{1_{x_1}} - A_{x_1})^2}$$

$$f_2(\vec{A}) = \sqrt{(E_{2_{x_0}} - A_{x_0})^2 + (E_{2_{x_1}} - A_{x_1})^2}$$

Mit den in Abbildung 2.1 angegebenen Koordinaten $\overrightarrow{E_1} = \begin{pmatrix} 0 \\ 4 \end{pmatrix}$ und $\overrightarrow{E_2} = \begin{pmatrix} 5 \\ 1 \end{pmatrix}$ ergibt sich somit für die Abstandsfunktionen

$$f_1(\overrightarrow{A}) = \sqrt{(-A_{x_0})^2 + (4 - A_{x_1})^2}$$

$$f_2(\overrightarrow{A}) = \sqrt{(5 - A_{x_0})^2 + (1 - A_{x_1})^2}$$

In diesem Fall ist es leicht zu erkennen, dass als optimale Lösungen nur die Punkte auf der Strecke zwischen $\overrightarrow{E_1}$ und $\overrightarrow{E_2}$ in Frage kommen. Da die beiden Eingänge nicht auf dem gleichen Punkt im Raum liegen, kann es aber keine Lösung geben, welche beide Funktionen gleichzeitig auf 0 minimiert. Eine Annäherung zu einem der beiden Eingänge hat immer eine Entfernung vom anderen Eingang zur Folge. Somit gibt es eine Menge an Lösungen, welche pareto-optimal sind. Diese pareto-optimale Lösungsmenge P ist im vorliegenden Fall genau die Menge der Punkte auf der Strecke zwischen den beiden Eingängen. Da sich diese Strecke durch $y = -\frac{3}{5} \cdot x + 4$ mit $0 \leq x \leq 5$ beschreiben lässt ergibt sich aus ihren Funktionswerten die entsprechende pareto-optimale Front POF. Sowohl P als auch POF sind nachfolgend dargestellt (Abbildungen 2.2 und 2.3).

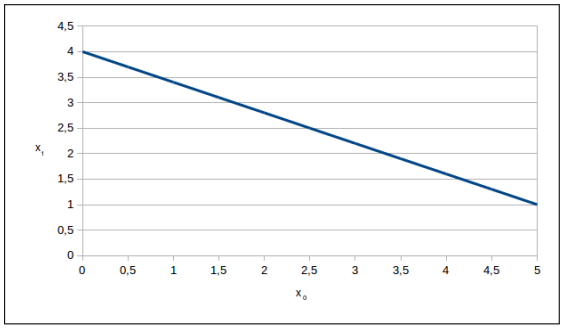

Abbildung 2.2: Pareto-optimale Lösungsmenge P

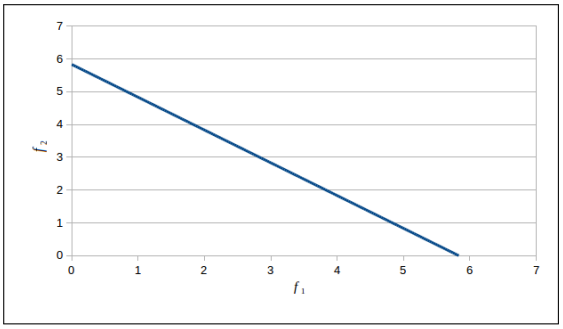

Abbildung 2.3: Pareto-optimale Front POF

Im Rahmen dieser Arbeit werden für die pareto-optimalen Lösungsmengen und Fronten P bzw. POF verwendet, für die von den Algorithmen ermittelten entsprechend P^* und POF^*.

Bei der Lösung von multikriteriellen Optimierungsproblemen unterscheidet man verschiedene Herangehensweisen [4]. Bei der Vorgehensweise *a priori* sind dem Algorithmus zu Beginn die Präferenzen des Entscheiders bekannt. Er hat dann das Ziel eine Lösungsmenge oder auch nur eine Lösung zu finden, welche diesen Wünschen am stärksten entspricht. Bei der Vorgehensweise *a posteriori* werden zu Beginn der Problemlösung keine Vorgaben gemacht und der Algorithmus hat das Ziel eine möglichst breit gefächerte Menge an optimalen Lösungen zu finden. Diese Menge wird am Ende vom Entscheider genutzt, um eine Auswahl zu treffen. Eine dritte Möglichkeit ist die *interaktive* Vorgehensweise, in welcher der Entscheider während der Problemlösung Einfluss auf die Schwerpunkte der Problemlösung nimmt.

Diese multikriteriellen Probleme können sowohl als statische sowie auch als dynamische multikriterielle Probleme auftreten.

Bei statischen multikriteriellen Problemen bleiben die Eigenschaften des Problems über die gesamte Zeit hinweg bestehen. Das heißt, dass sowohl die Position als auch die Funktionswerte der optimalen Lösungen sich nicht ändern. Ein Funktionswert $\overrightarrow{f}(\overrightarrow{x})$ besitzt

somit zu jeder Zeit den gleichen Wert.
Im Unterschied dazu können sich in dynamischen multikriteriellen Problemen entscheidende Parameter des Problems im Laufe der Zeit ändern. Unterschieden werden drei grundsätzliche Arten von Änderungen:

- Änderung der Position der optimalen Lösungen
- Änderung der Funktionswerte der optimalen Lösungen
- Änderung der Position und der Funktionswerte der optimalen Lösungen

Eine Änderung des Problems kann beispielsweise einem bestimmten zeitlichen Zyklus folgen, eine Reaktion auf eine sich verändernde Umwelt sein oder zufällig und azyklisch auftreten. Eine zeitliche Abhängigkeit sorgt somit dafür, dass der ehemals statische Funktionswert $\vec{f}(\vec{x})$ nicht mehr nur von der Lösung $\vec{x}$ abhängig ist, sondern auch vom aktuellen Zeitschritt t und sich damit für den Funktionswert $\vec{f}(\vec{x}, t)$ ergibt.
Diese Art der Probleme sind in der Forschung besonders interessant, da reale Probleme in der Regel Änderungen unterliegen. Dynamische multikriterielle Probleme stellen besondere Herausforderungen an die Algorithmen, die zur Lösung verwendet werden. Diese müssen mindestens in der Lage sein eine Änderung zu erkennen und entsprechend zu reagieren.

2.2 Distanzminimierungsproblem

Das Distanzminimierungsproblem (DMP) ist ein spezielles multikriterielles Problem. Es geht hierbei darum Punkte im Entscheidungsraum zu finden, die einen minimalen Abstand zu den vorgegebenen Zielpunkten haben. Der Abstand eines Punktes zu einem Zielpunkt ist durch die Zielfunktion dieses Zielpunktes definiert. Das Problem besteht somit aus einer Menge von Zielpunkten $\{\vec{Z_1}, \vec{Z_2}, ..., \vec{Z_k}\}$, die jeweils n Komponenten im n-dimensionalen Entscheidungsraum besitzen: $\vec{Z_i} = (x_{i1}, x_{i2}, ..., x_{in})^T$. Für jeden Zielpunkt $\vec{Z_i}$ existiert eine Zielfunktion $f_i(\vec{x})$, die den Abstand einer Lösung $\vec{x}$ vom Zielpunkt $\vec{Z_i}$ bestimmt. Mathematisch formuliert ergibt sich somit für ein Distanzminimierungsproblem:

$$\text{Minimiere } \vec{f}(\vec{x}) = (f_1(\vec{x}), f_2(\vec{x}), ..., f_k(\vec{x}))^T$$

$$\text{mit } f_i(\vec{x}) = dist(\vec{x}, \vec{Z_i}) \ \forall i = 1, ..., k$$

Für die Zielfunktion zur Abstandsbestimmung ($dist(\vec{x}, \vec{Z_i})$) können verschiedene Abstandsmaße verwendet werden. Im Falle des euklidischen Abstandes ist der Wert der Zielfunktion die Länge der Strecke zwischen dem Punkt und dem entsprechenden Zielpunkt (s. Abbildung 2.4). Im Allgemeinen wird eine weitere Annäherung an einen der Zielpunkte gleichzeitig dazu führen, dass die Entfernung zu einem anderen Zielpunkt zunimmt. Somit ist es für einen Punkt nicht möglich alle Zielfunktionen gleichzeitig zu minimieren.
Für die Festlegung der Zielfunktion, welche den Abstand eines Punktes zu einem Zielpunkt bestimmt, können p-Normen verwendet werden.

2.2.1 p-Normen

Die p-Normen sind eine Klasse von Vektornormen. Sie sind alle durch die gleiche Grundfunktion mit einem veränderlichen Parameter p definiert.

$$||x||_p := (\sum_{i=1}^{n} |x_i|^p)^{1/p}$$

Im Bezug auf das Abstandsmaß werden die Abstände der einzelnen Komponenten der beiden Vektoren mit p exponentiert, die Ergebnisse der Potenzen summiert und diese Summe mit p radiziert. Die bekannteste p-Norm ist jene für $p = 2$, welche den euklidischen Abstand zweier Punkte berechnet. Im Rahmen dieser Arbeit werden für p nur die Werte $p = 1$ (Summennorm, s. Gleichung 2.2) und $p = 2$ (euklidische Norm, s. Gleichung 2.3) verwendet. Die p-Normen sind jedoch für alle Werte $p \geq 1$ definiert. Bei der Summennorm ($p = 1$) ergibt sich als Abstandsmaß die sogenannte Manhattan-Metrik. Diese zeichnet sich dadurch aus, dass der Abstand zweier Punkte über die kürzeste Strecke entlang der Koordinatenachsen definiert ist. Namensgebend ist hierbei der New Yorker Stadtteil Manhattan. In diesem ergeben sich, aufgrund seines rechtwinkligen Straßenverlaufs, Entfernungen für zurückzulegende Strecken in gleicher Weise. Bei der euklidischen Norm ($p = 2$) ist der Abstand definiert als Länge der Strecke zwischen dem Punkt und dem entsprechenden Zielpunkt.

$$||x||_1 := \sum_{i=1}^{n} |x_i| \tag{2.2}$$

$$||x||_2 := \sqrt{\sum_{i=1}^{n} |x_i|^2} \tag{2.3}$$

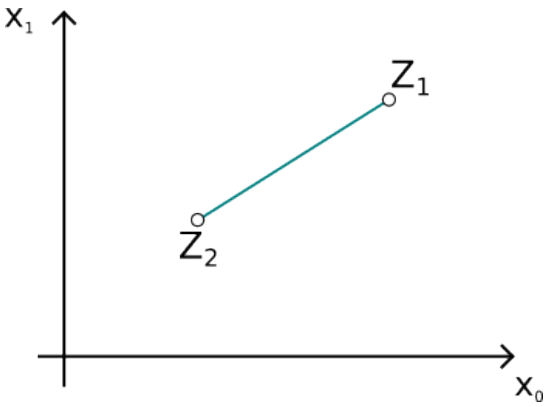

Abbildung 2.4: Einzig möglicher Abstand nach euklidischer Norm

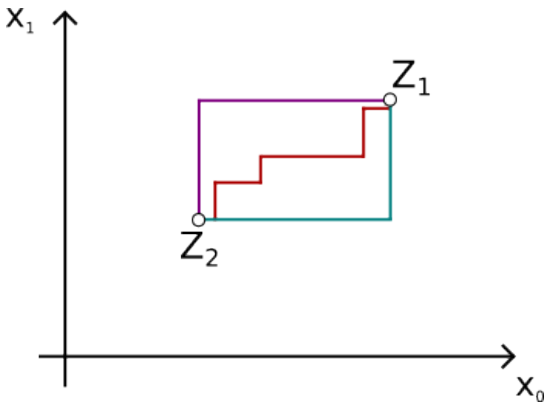

Abbildung 2.5: Drei mögliche, gleichgroße Abstände nach Summennorm

2.2.2 Pareto-optimale Lösungsmenge des Distanzminimierungsproblems

Wie in der Abbildung 2.4 zu erkennen ist, gibt es bei der euklidischen Norm nur eine Lösung für die Strecke mit minimalem Abstand zwischen zwei Punkten. Bei der Summennorm ergeben sich im allgemeinen Fall jedoch unendlich viele (s. Abbildung 2.5). Dies hat Folgen für die pareto-optimale Lösungsmenge der jeweiligen DMP.
Zille und Mostaghim [5] untersuchten die Eigenschaften der pareto-optimalen Lösungsmengen des Distanzminimierungsproblems für $p = 1$ und $p = 2$.
Im Falle der euklidischen Norm ($p = 2$) ergibt sich die pareto-optimale Lösungsmenge als konvexe Hülle des Raumes, welcher durch die Zielpunkte aufgespannt wird. Alle Punkte, welche sich innerhalb dieser Hülle befinden sind nicht-dominiert.
Beispielhaft ist dies in der folgenden Abbildung 2.6 für einen zweidimensionalen Entscheidungsraum und drei Zielpunkte dargestellt. Alle Abstände von den einzelnen Zielpunkten können als Kreise um den entsprechenden Zielpunkt dargestellt werden. Jeder Schnittpunkt zwischen verschiedenen minimalen Kreisen definiert einen nicht-dominierten Punkt. Die Menge dieser Punkte bildet ein konvexes Polygon, welches die pareto-optimale Menge P darstellt [5].

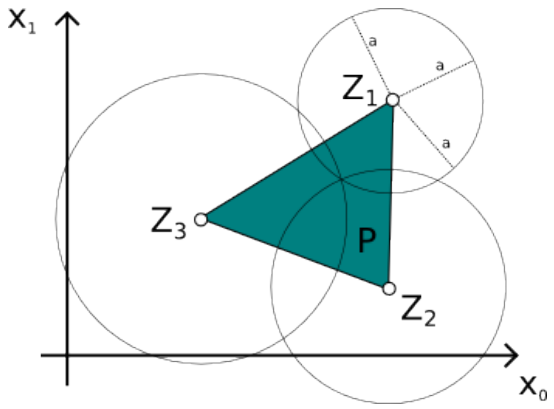

Abbildung 2.6: Pareto-optimale Lösungsmenge für $p = 2$

Im Fall der Summennorm ($p = 1$) ist die pareto-optimale Lösungsmenge aufwendiger zu ermitteln. Wie soeben gezeigt gibt es für den minimalen Abstand zwischen zwei Punkten unendlich viele Möglichkeiten. Die pareto-optimale Lösungsmenge für zwei Zielpunkte ergibt sich wiederum als Schnittmenge der minimalen Distanzen der Zielpunkte. Anstelle von Kreisen wie bei der euklidischen Norm ergeben sich jedoch durch die Summennorm hier für alle Abstände zu einem einzelnen Zielpunkt gleichseitige Rauten um diesen Zielpunkt. Die Menge aller Schnittpunkte zweier solcher Rauten ergibt die pareto-optimale Menge für zwei Zielpunkte (s. Abbildung 2.7). Für eine beliebige Anzahl an Zielpunkten ergibt sich die pareto-optimale Lösungsmenge aus der Vereinigung der Schnittmengen aller paarweisen pareto-optimalen Lösungsmengen [5] (s. Abbildung 2.8).
Die Verwendung der Summennorm hat neben der aufwendigeren Ermittlung der pareto-optimalen Lösungsmenge eine weitere entscheidende Eigenschaft bezüglich der Dominanz zwischen zwei Punkten. So dominiert eine Lösung eine andere nur, wenn diese auf der gleichen Schnittlinie (also im Winkel von 45° zueinander) liegen, wie exemplarisch in der Abbildung 2.9 dargestellt.

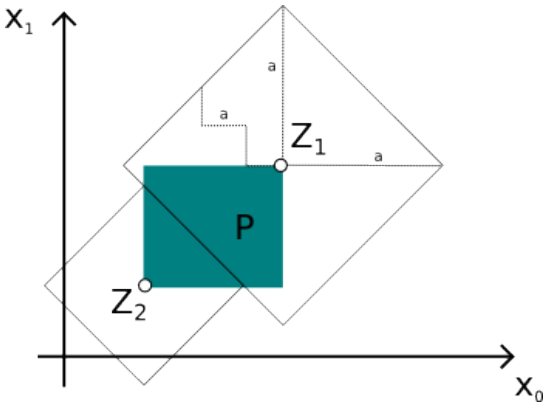

Abbildung 2.7: Pareto-optimale Menge für zwei Zielpunkte und Summennorm

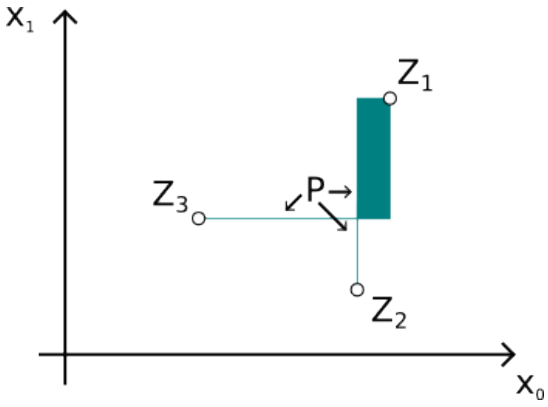

Abbildung 2.8: Pareto-optimale Menge für drei Zielpunkte und Summennorm

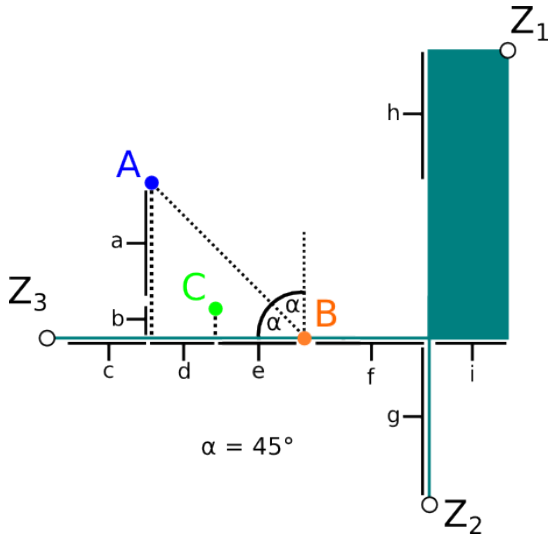

Abbildung 2.9: Dominanz zweier Punkte bei Summennorm

Es ergeben sich folgende Funktionswerte und Dominanzen für die Lösungen A, B und C:

$$\vec{f}(A) = (f_1(A), f_2(A), f_3(A)) = ((d+e+f+h+i), (a+b+d+e+f+g), (a+b+c))$$

$$\vec{f}(B) = (f_1(B), f_2(B), f_3(B)) = ((f+b+a+h+i), (f+g), (e+d+c))$$

$$\vec{f}(C) = (f_1(C), f_2(C), f_3(C)) = ((a+e+f+h+i), (b+e+f+g), (b+d+c))$$

mit $a+b=d+e$ und $a+b+d+e>0$ ergibt sich somit

$$f_1(A) = f_1(B) \text{ und } f_2(A) > f_2(B) \text{ und } f_3(A) = f_3(B)$$

Damit gilt nach Gleichung 2.1:

$$B \prec A$$

Hierbei ist auch zu sehen, dass der Punkt C den Punkt A nicht dominiert, obwohl C näher an der pareto-optimalen Menge liegt als A. Aufgrund dieser Eigenschaft bezüglich der Dominanz stellt das Distanzminimierungsproblem mit Summennorm auch bei einer geringen Anzahl an Variablen und Zielpunkten ein schwieriges Problem dar [5].

2.2.3 Statisches und dynamisches DMP

Ein Distanzminimierungsproblem ist gekennzeichnet durch die Lage der Zielpunkte und die verwendeten Abstandsfunktionen. Diese beiden Eigenschaften sind beim statischen Distanzminimierungsproblem unveränderlich. Somit sind auch die pareto-optimale Lösungsmenge im Entscheidungsraum und die pareto-optimale Front im Zielraum unveränderlich.
Beim dynamischen Distanzminimierungsproblem hingegen ist sowohl die Lage der Zielpunkte als auch die Gestaltung der Zielfunktionen veränderlich. Ein Zielpunkt $\vec{Z_i}$ und eine Zielfunktion $f_i(\vec{x})$ sind dann, wie in Abschnitt 2.1 beschrieben, von einem Parameter, wie z.B. einem Zeitschritt t, abhängig. Es ergeben sich dann für den Zielpunkt $\vec{Z_i}(t)$ und für die Zielfunktion $f_i(\vec{x}, t)$ als neue mathematische Modellierungen. Die Erstellung eines solchen Problems ist eines der Ziele dieser Arbeit. Deshalb werden die Möglichkeiten der Veränderungen ausführlich im Kapitel 4 behandelt. Aus diesen Veränderungen heraus können sich neue und teilweise stark veränderte pareto-optimale Lösungsmengen und Fronten ergeben.

2.3 Evolutionäre Algorithmen

Bei evolutionären Algorithmen (EA) handelt es sich um ein metaheuristisches Verfahren zur näherungsweisen Lösung von Optimierungsproblemen. Metaheuristisch bedeutet, dass diese Art von Algorithmen auf beliebige Probleme (*meta*) anwendbar ist und eine näherungsweise Lösung (*heuristisch*) liefert. Hierzu werden Mechanismen, die primär aus der Evolutionsbiologie bekannt sind, verwendet.
Die Abbildung 2.10 zeigt den schematischen Ablauf eines evolutionären Algorithmus. Grundlegendes Element eines evolutionären Algorithmus ist die Population, welche die Gesamtmenge an Lösungen darstellt [6]. Zu Beginn wird eine solche Gesamtmenge erstellt, wobei die Lösungen zufällig im Entscheidungsraum verteilt sein können oder bestimmten Regeln folgen. Danach werden die Mitglieder dieser Population evaluiert, d.h. die Eignung der Lösungen bezüglich des gegebenen Problems wird durch eine sogenannte Fitness-Funktion bestimmt. Diese Funktion ist problemabhängig und stellt eine charakteristische Eigenschaft des jeweiligen Problems dar. Im nächsten Schritt, der *Selektion*, wird eine bestimmte Anzahl an Mitgliedern anhand ihrer (guten) Eignung ausgewählt. Von diesen wiederum werden (meist zwei) ausgewählt, welche im Prozess der *Kreuzung* (engl. *cross-over*) neue Lösungen erzeugen [7]. Diese neu erzeugten „Kinder“-Lösungen stellen eine Kombination der Eigenschaften ihrer „Eltern“-Lösungen dar. Es wird hierbei die Annahme getroffen, dass gute „Eltern“-Lösungen (durch *Selektion* erhalten) auch gute „Kinder“-Lösungen (durch *Kreuzung* erzeugt) hervorbringen. Nachdem die „Kinder“-Lösungen erstellt sind, kann noch ein weiterer evolutionärer Schritt erfolgen, die *Mutation*. Hierbei werden in den neu geschaffenen Lösungen einzelne Bestandteile (z.B. einzelne Bits) verändert. Dies dient dazu möglicherweise neue und nützliche Eigenschaften, die keiner der Elternteile besitzt, mit in die nächste Generation zu vererben [8]. Anschließend werden die „Kinder“-Lösungen ebenfalls evaluiert und aus der Menge der „Eltern“- und „Kinder“-Lösungen durch Ersetzungsmechanismen die neue Population gebildet. Diese bildet den Ausgangspunkt für die nächste Generation, in welcher sich das soeben Beschriebene wiederholt.

Der Prozess endet in der Regel wenn eine bestimmte Anzahl von Generationen oder Evaluationen erreicht ist. Aber auch wenn die Qualität der erzeugten Lösungen einen bestimmten Grenzwert überschreitet oder sich im Vergleich zur vorherigen Generation nicht um ein Mindestmaß verbessert, kann dies ein Grund dafür sein, den evolutionären Prozess zu stoppen.

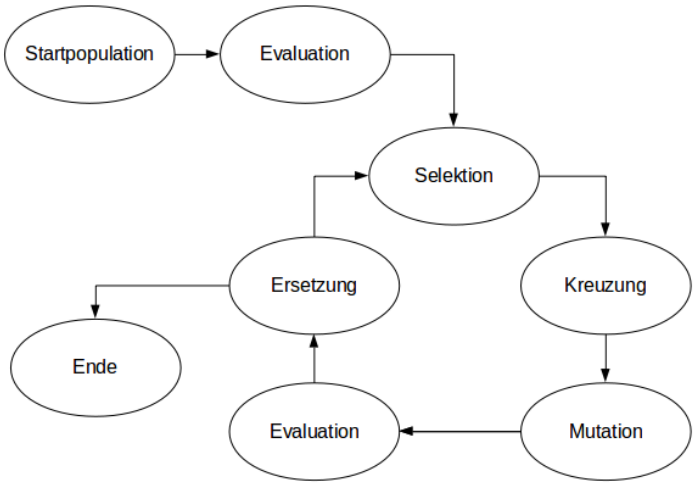

Abbildung 2.10: Schematischer Ablauf eines evolutionären Algorithmus

2.4 Partikelschwarmoptimierung

Auch bei der Partikelschwarmoptimierung (PSO) handelt es sich um ein metaheuristisches Verfahren zur Lösung von Optimierungsproblemen, welches seine Grundidee aus einem in der Natur vorkommenden Verhalten bezieht. Es entstand aus der Beobachtung des Verhaltens von Tierschwärmen (z.B. Vögeln oder Fischen) heraus. Diese bestehen aus einer bestimmten Menge an Individuen, welche als *Partikel* bezeichnet werden und in ihrer Gesamtheit den *Schwarm* bilden. Dieser Schwarm ist in der Lage komplexe Verhaltensweisen zu zeigen. Diese basieren jedoch nicht auf einer komplexen Menge von Regeln, sondern auf wenigen einfachen Regeln, welche von jedem Individuum im Schwarm befolgt werden [9]. Grundlegend besitzt ein Individuum drei verschiedene Regeln [10], eine für die *Anziehung* (s. Abbildung 2.11), eine für die *Abstoßung* (s. Abbildung 2.12) und eine dritte für die *Ausrichtung* (s. Abbildung 2.13). Die *Anziehung* dient dazu, dass sich das Individuum einem bestimmten Ziel nähert, beispielsweise dem Zentrum des Schwarms. Die *Abstoßung* verhindert, dass alle Individuuen exakt die gleiche Position einnehmen und in einem realen Schwarm kollidieren würden. Die *Ausrichtung* sorgt dafür, dass alle Individuen den gleichen Richtungssinn annehmen.

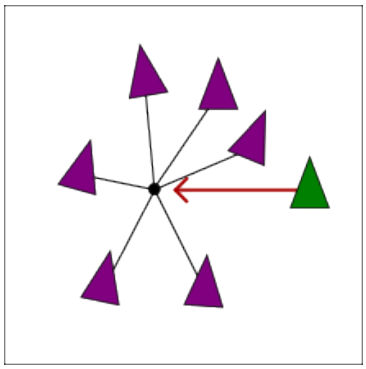

Abbildung 2.11: Anziehung

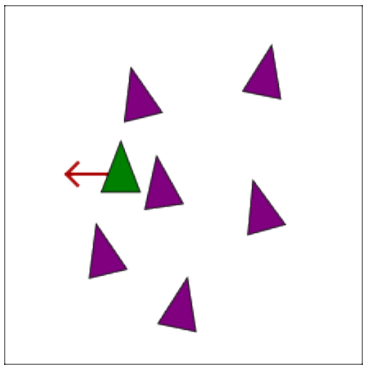

Abbildung 2.12: Abstossung

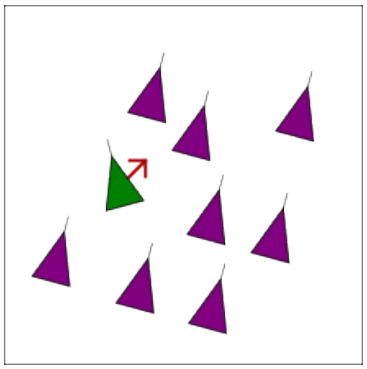

Abbildung 2.13: Ausrichtung

Die drei Regeln des Schwarms in Anlehnung an [10]

Durch eine entsprechende Gestaltung der jeweiligen Regeln in mathematischer Form lässt sich das Verhalten des Individuums und damit des gesamten Schwarms mannigfaltig steuern. Ein Ergebnis solcher Regeln ist in der Natur als die typische V-Formation von Zugvögeln zu beobachten.
Im Rahmen von multikriteriellen Problemen gibt es gewissen Gemeinsamkeiten zwischen der Partikelschwarmoptimierung und den im vorherigen Kapitel besprochenen evolutionären Algorithmen. In beiden Fällen besteht die Population (der *Schwarm*) aus möglichen Lösungen des entsprechenden Problems. Ausgehend von der Startpopulation wird diese verändert um neue und bessere Lösungen zu erhalten. In der Partikelschwarmoptimierung geschieht dies über die Bewegung der einzelnen Individuen. Dazu besitzt jedes Individuum drei Komponenten [11], eine *kognitive*, eine *soziale* und die *Trägheitskomponente*. Die *kognitive* Komponente stellt sicher, dass das Individuum sich seine beste, bisher erreichte Position merkt. Die *soziale* Komponente dient dazu, dass alle Individuen die beste Position innerhalb des Schwarms kennen, also wissen welches Individuum gera-

de die beste aller Lösungen besitzt. Hierzu muss ein Informationsaustausch zwischen den Individuen sichergestellt sein. Über die Gewichtung der *Trägheit* kann bestimmt werden, wie groß die Auswirkung der Bewegung des aktuellen Zeitschrittes auf die Bewegung des nächsten Zeitschrittes sein soll. Damit ergeben sich folgende Gleichungen für das Verhalten eines Partikels $\vec{x_i}$.
Zur Ermittlung des Positionsvektors für den nächsten Zeitschritts wird zum aktuellen Positionsvektor der Bewegungsvektor des nächsten Zeitschritts addiert.

$$\vec{x_i}(t+1) = \vec{x_i}(t) + \vec{v_i}(t+1) \tag{2.4}$$

Die Ermittlung dieses Bewegungsvektors $\vec{v_i}(t+1)$ ergibt sich aus:

$$\vec{v_i}(t+1) = \omega \cdot \vec{v_i}(t) + C_1 \cdot r_1 \cdot (\vec{x_{p_i}}(t) - \vec{x_i}(t)) + C_2 \cdot r_2 \cdot (\vec{x_g}(t) - \vec{x_i}(t))$$

Hierbei ist $\vec{x_{p_i}}(t)$ die beste Lösung, die der i-te Partikel bisher gefunden hat (kognitive Komponente) und $\vec{x_g}(t)$ die aktuelle beste Lösung des gesamten Schwarms (soziale Komponente). Die Gewichtung der Trägheitskomponente ist durch ω beschrieben, C_1 und C_2 sind Konstanten, welche die Anziehungskräfte des persönlichen und schwarmweiten besten bisherigen Ergebnisses definieren. r_1 und r_2 sind Vektoren in $[0,1]^n$, welche das Lernverhalten der Partikel beeinflussen.
In Abbildung 2.14 wird die Berechnung des Bewegungsvektors $\vec{v}(t+1)$ als Vektorkombination dargestellt. Dabei werden die Trägheit ω, die bisherige Geschwindigkeit $\vec{v}(t)$ und die beste Lösung des gesamten Schwarms $\vec{x_g}(t)$ beachtet. Die Parameter sind mit $\omega = 1$ und $r_2 = 0.5$ festgelegt.

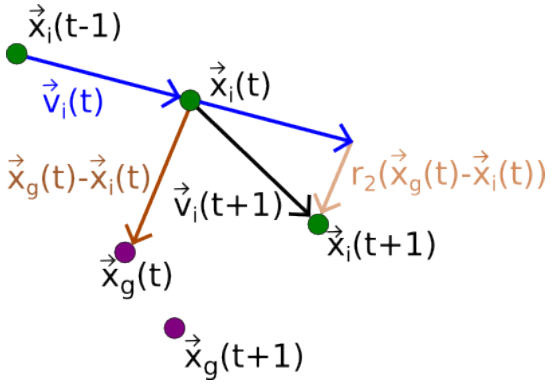

Abbildung 2.14: Berechnung des Bewegungsvektors $\vec{v}(t+1)$

2.5 Qualitätsindikatoren

In diesem Abschnitt werden die verwendeten Qualitätsindikatoren vorgestellt. Sie dienen dazu das Ergebnis eines Algorithmus zu quantifizieren. Dies ist notwendig, da es bei multikriteriellen Problemen keine eindeutig „beste“ Lösung gibt. Um verschiedene Lösungsmengen vergleichen zu können werden daher Eigenschaften dieser Lösungsmengen ermittelt und diese Eigenschaften miteinander verglichen. Damit ist eine Aussage über die Qualität der Leistung eines Algorithmus bezüglich eines bestimmten Problems sowie ein Vergleich der Leistungsfähigkeit der getesteten Algorithmen untereinander möglich.

2.5.1 Anzahl der Lösungen in pareto-optimaler Menge

Bei einer gegebenen Probleminstanz eines DMP ist durch die Lage der Zielpunkte und die entsprechenden Zielfunktionen die pareto-optimale Menge P definiert. Da die Lösungen dieser Menge nicht von anderen Lösungen dominiert werden können, ist es ein Hinweis darauf, dass die vom Algorithmus gefundenen Lösungen qualitativ gut sind, wenn ein hoher Anteil der Lösungen in P liegt. Die Ermittlung der entsprechenden Anzahl an Lösungen innerhalb dieser Menge kann über die absolute Anzahl n_{opt} erfolgen.

$$n_{opt} = |\{\vec{x} \in P \cap \vec{x} \in P^*\}|$$

Hierbei ist P die pareto-optimale Lösungsmenge und P^* die vom Algorithmus gefundene Lösungsmenge. Der Vergleich der Ergebnisse verschiedener Algorithmen wird erschwert, wenn die Algorithmen unterschiedliche maximale Populationsgrößen haben. So ist es mit einer großen Populationsgröße leichter mehr Lösungen in der pareto-optimalen Menge zu platzieren als mit einer kleinen Populationsgröße. Dies kann zu einer verzerrten Wahrnehmung der Leistungsfähigkeit der unterschiedlichen Algorithmen führen. Zur Schaffung der Vergleichbarkeit bietet sich als eine einfache Möglichkeit eine Verhältniszahl bezüglich der Populationsgröße des Algorithmus n_{ges} an.

$$r_{opt} = \frac{n_{opt}}{n_{ges}}$$

Da im Rahmen dieser Arbeit für alle Algorithmen die gleiche Populationsgröße festgelegt wird, ist hier eine Vergleichbarkeit auch ohne Berechnung der relativen Anzahl gegeben. Für die Testfälle, in welchen das euklidische Abstandsmaß verwendet wird, ergibt sich der pareto-optimale Bereich als konvexe Hülle, welche durch die Zielpunkte gebildet wird. Im Falle des Manhattan-Abstandsmaßes kann sich, wie in Abbildung 2.8 gezeigt, ein pareto-optimaler Bereich mit einer Fläche von 0 ergeben. Diesen exakt zu treffen stellt eine deutliche Schwierigkeit dar. Aus diesem Grund wird hier ein Punkt zur pareto-optimalen Menge gezählt, wenn sein Abstand von dieser höchstens 1 % der Ausdehnung des Entscheidungsraum für die jeweilige Variable beträgt.

2.5.2 Streuung

Die Streuung (engl. *spread*) gibt an, wie gleichmäßig die nicht-dominierten Lösungen im Zielraum verteilt sind. Zur Berechung des Abstandes wird das euklidische Abstandsmaß verwendet. Grundsätzlich zeigt ein kleinerer Wert für die Streuung eine gleichmäßigere Verteilung der Funktionswerte an. In der ursprünglichen Variante von Schott [1] wird die Streuung als korrigierte Stichprobenvarianz berechnet.

$$S = \sqrt{\frac{1}{n_{POF^*} - 1} \sum_{m=1}^{n_{POF^*}} (d_{avg} - d_m)^2}$$

$$d_m = min_{j=1,\ldots,n_{POF^*}} \{\sum_{i=1}^{k} |f_i(\overrightarrow{POF^*}_m) - f_i(\overrightarrow{POF^*}_j)|\} \text{ mit } m \neq j$$

Hierbei ist d_m der der kleinste Wert für die absoluten Differenzen aller Funktionswerte zwischen der Lösung m und allen anderen Lösungen. Die Anzahl der Lösungen in

der Lösungsmenge wird mit n_{POF^*} bezeichnet, d_{avg} ist das arithmetische Mittel aller Abstände d_m. ${POF^*}_m$ und ${POF^*}_j$ stellen das m-te bzw. j-te Element der Lösungsmenge dar und k ist die Anzahl der Zielfunktionen. Bei dieser Variante der Streuungsberechnung besteht das Problem, dass eine gleichmäßige Verteilung der Funktionswerte bereits eine Streuung von 0 erzeugt, auch wenn die Funktionswerte die pareto-optimale Front nicht gänzlich abdecken. Dies wird in den beiden in Abbildung 2.15 dargestellten Fällen deutlich. Es handelt sich offensichtlich um unterschiedliche Lösungsmengen, die jeweils eine unterschiedliche Abdeckung der pareto-optimalen Front besitzen. Da die Lösungen im Zielraum jedoch alle den gleichen Abstand voneinander haben ist die Streuung in beiden Fällen 0. Damit ist diese Variante der Streuung als Qualitätsindikator nur begrenzt aussagefähig.

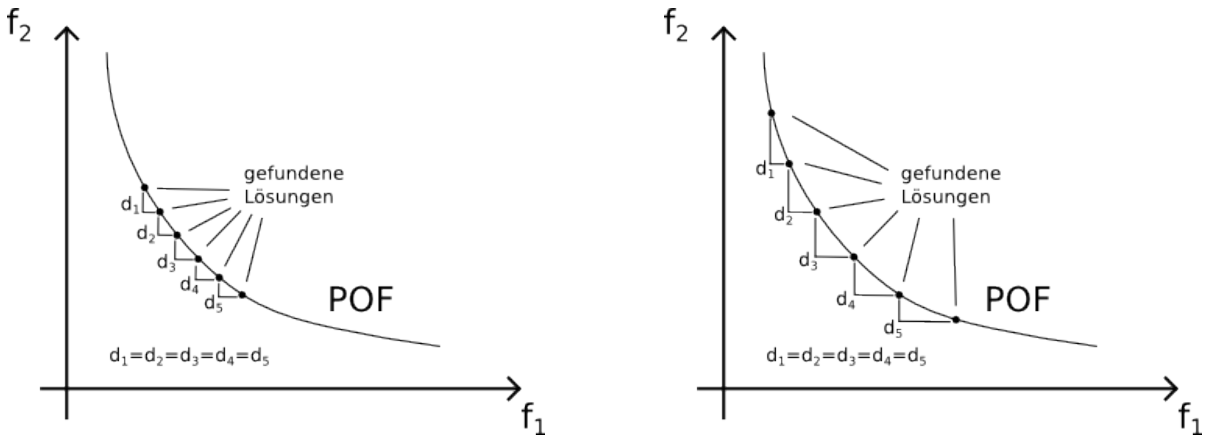

Abbildung 2.15: Streuungen von 0 nach [1]

In dieser Arbeit wird daher die Variante von Deb [12] [2] verwendet, welche neben der gleichmäßigen Verteilung auch den Grad der Abdeckung der pareto-optimalen Front beachtet. Es handelt sich hierbei um ein normalisiertes Maß im Intervall $[0, 1]$, wobei auch hier ein Wert von 0 eine ideal gleichmäßige Verteilung anzeigt. In dieser Variante ist ein Streuungswert von 0 jedoch nur möglich, wenn die Extremwerte der pareto-optimalen Front gefunden wurden und die Abstände aller Funktionswerte gleich sind (s. Abbildung 2.17).

$$\Delta = \frac{d_f + d_l + \sum_{i=1}^{N-1} |d_i - d_{avg}|}{d_f + d_l + (N-1)d_{avg}} \tag{2.5}$$

Hierbei sind d_f und d_l die euklidischen Abstände der extremsten gefundenen Lösung zu den Extremlösungen der pareto-optimalen Front (s. Abbildung 2.16), d_{avg} der Mittelwert aller Abstände und N die Anzahl der Lösungen der Population.

2.5.3 Hypervolumen

Beim Hypervolumen (engl. *hypervolume*) handelt es sich um einen Qualitätsindikator bezüglich der Abdeckung des Zielraums [13]. Allgemein bestimmt sich die Dimensionalität des Zielraums nach der Anzahl an Zielfunktionen des Optimierungsproblems. Bei zwei Zielfunktionen ist der Zielraum zweidimensional, bei drei Zielfunktionen dreidimensional etc..

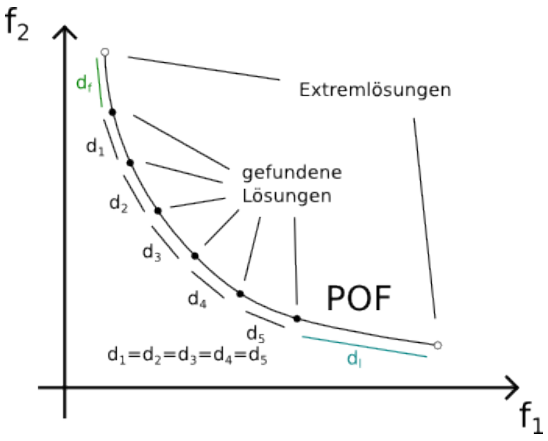

Abbildung 2.16: Streuung > 0 nach [2]

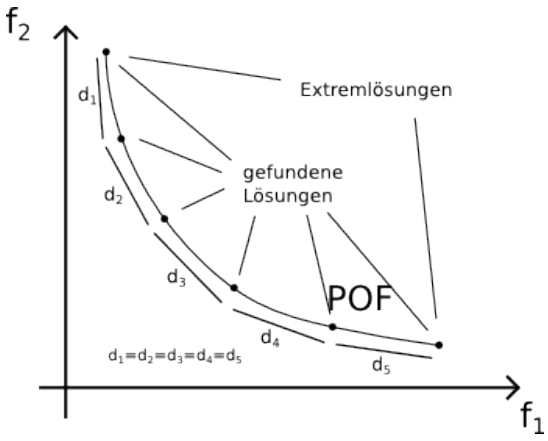

Abbildung 2.17: Streuung von 0 nach [2]

Innerhalb des Zielraumes werden Hypervolumen (Ausdehnungen entsprechend der Dimensionalität des Raumes) berechnet (s. Abbildung 2.18). Dies sind im zweidimensionalen Raum Flächen, im dreidimensionalen Raum Volumen usw..
Diese Hypervolumen werden für jede Lösung aus P^* im Zielraum ermittelt. Dabei wird ein für alle Lösungen gleicher Referenzpunkt P_{ref} verwendet. Dieser besteht üblicherweise aus den jeweils schlechtesten Werten aller möglichen Zielfunktionen und Lösungen. Für jede Lösung aus P^* wird, ausgehend vom Funktionswert der jeweiligen Lösung, das abgedeckte Volumen bezüglich dieses Referenzpunktes berechnet. Das Ergebnis der Berechnung des Hypervolumens ist somit das durch alle Lösungen abgedeckte Volumen bezüglich des Referenzpunktes. Hierbei ist zu beachten, dass Bereiche, welche durch mehrere Punkte abgedeckt werden nicht mehrfach gezählt werden.
Allgemein gilt, dass sich eine Lösungsmenge den optimalen Lösungen besser annähert, je höher das durch sie abgedeckte Hypervolumen ist. Der optimale Wert für das Hypervolumen eines Problems lässt sich durch die pareto-optimale Front berechnen.

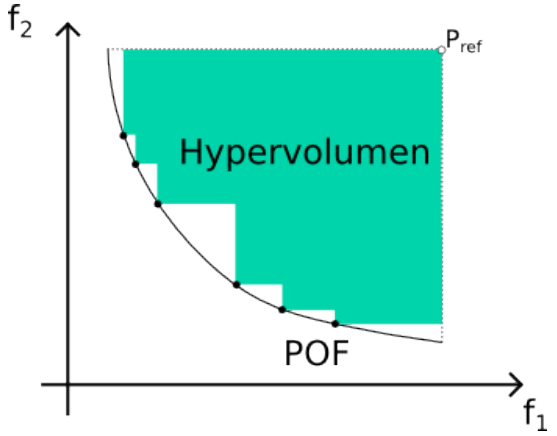

Abbildung 2.18: Berechnung des Hypervolumens aus gefunden Lösungen

Hypervolumenverhältnis

Beim Hypervolumenverhältnis (engl. *hypervolume ratio* (HVR)), welches von van Veldhuizen [14] beschrieben wurde, handelt es sich um einen Verhältniswert.

$$HVR = \frac{HV(POF^*)}{HV(POF)} \tag{2.6}$$

Das von einer nicht-dominierten Front POF^* der Lösungsmenge erzeugte Hypervolumen wird durch das Hypervolumen der pareto-optimalen Front POF geteilt. Es ergibt sich ein Wert zwischen 0 und 1, wobei größere Werte eine bessere Abdeckung und damit ein besseres Ergebnis darstellen. Mittels dieses Verhältniswerts ist es möglich die Ergebnisse verschiedener Probleminstanzen zu vergleichen, da eine Normalisierung der absoluten Werte erfolgt.

Hypervolumenverhältnis optimaler Punkte

Da die Berechnung des Hypervolumens einer Lösungsmenge nur im Zielraum erfolgt, kann aus ihr nicht erkannt werden, wie gut die Abdeckung der pareto-optimalen Lösungsmenge im Entscheidungsraum ist. Auch Punkte außerhalb der pareto-optimalen Lösungsmenge können Hypervolumen erzeugen. Dazu folgt ein kurzes Beispiel.
Es handelt sich um ein Distanzminimierungsproblem mit drei Zielpunkten, zwei Variablen und euklidischem Abstandsmaß (s. Abbildung 2.19). Die Ergebnisse werden zum Zwecke der Übersichtlichkeit auf zwei Dezimalstellen gerundet. Die Zielpunkte sind wie folgt verteilt:

$$\overrightarrow{Z_1} = (4, 4) \qquad \overrightarrow{Z_2} = (-4, -4) \qquad \overrightarrow{Z_3} = (2, -3)$$

Die beiden zu untersuchenden Punkte $\vec{A}$ und $\vec{B}$ haben die Koordinaten:

$$\vec{A} = (-1, 0) \qquad \vec{B} = (0, -1)$$

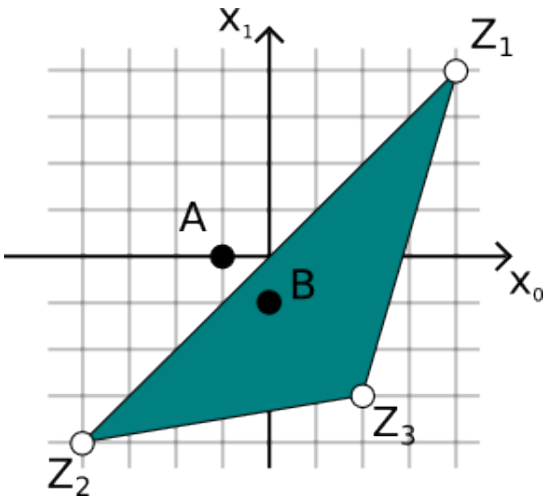

Abbildung 2.19: Einfaches DMP

Zu erkennen ist, dass $\vec{A} \notin P$ und $\vec{B} \in P$, also nur $\vec{B}$ im pareto-optimalen Bereich liegt. Die Funktionswerte der Zielpunkte sind die paarweisen euklidischen Abstand zueinander, so dass sich folgende Werte ergeben:

$$\vec{f}(\vec{Z_1}) = \begin{pmatrix} 0 \\ 11.31 \\ 7.28 \end{pmatrix} \qquad \vec{f}(\vec{Z_2}) = \begin{pmatrix} 11.31 \\ 0 \\ 6.08 \end{pmatrix} \qquad \vec{f}(\vec{Z_3}) = \begin{pmatrix} 7.28 \\ 6.08 \\ 0 \end{pmatrix}$$

Für die beiden Punkte ergeben sich analog:

$$\vec{f}(\vec{A}) = \begin{pmatrix} 6.4 \\ 5 \\ 4.24 \end{pmatrix} \qquad \vec{f}(\vec{B}) = \begin{pmatrix} 6.4 \\ 5 \\ 2.83 \end{pmatrix}$$

Damit ergibt sich der Referenzpunkt aus dem jeweils schlechtesten Wert für jede Funktion:

$$\overrightarrow{P_{Ref}} = \begin{pmatrix} 11.31 \\ 11.31 \\ 7.28 \end{pmatrix}$$

Für die Punkte $\vec{A}$ und $\vec{B}$ berechnet sich das von ihrem Funktionswert ausgehende Volumen bis zum Referenzpunkt mit:

$$\vec{f}(\vec{A}) = \begin{pmatrix} 6.4 \\ 5 \\ 4.24 \end{pmatrix} \rightarrow HV(\vec{A}) = (11.31 - 6.4) \cdot (11.31 - 5) \cdot (7.28 - 4.24) = 94.19$$

$$\vec{f}(\vec{B}) = \begin{pmatrix} 6.4 \\ 5 \\ 4.24 \end{pmatrix} \rightarrow HV(\vec{B}) = (11.31 - 6.4) \cdot (11.31 - 5) \cdot (7.28 - 2.83) = 137.87$$

Zu sehen ist hier deutlich, dass eine Lösung im pareto-optimalen Bereich ein größeres Hypervolumen erzeugt als eine Lösung außerhalb. Jedoch kann auch letztere einen deutlichen Beitrag zum Hypervolumen leisten.
Daher wird in dieser Arbeit auch untersucht, wie groß der Anteil des Hypervolumens ist, der durch Lösungen im pareto-optimalen Bereich erzeugt wird. Dazu wird der Indikator HVR_{opt} eingeführt, der nur die Lösungen für das Hypervolumen betrachtet, die sich im Entscheidungsraum im pareto-optimalen Bereich befinden.

$$HVR_{opt} = \frac{HV(POF_{opt})}{HV(POF)} \tag{2.7}$$

mit

$$\vec{f}(\vec{x}) \in POF_{opt} \Leftrightarrow \vec{x} \in P \tag{2.8}$$

2.6 Verwandte Literatur

Die Thematik der multikriteriellen Probleme ist in vielen Bereichen des täglichen und Wirtschaftslebens anzutreffen, da oft komplexe Probleme gelöst werden müssen, deren Teilprobleme verschiedene Ziele verfolgen. Neben mathematischen Anwendungsgebieten ist dies vor allem im ökonomischen und Finanzbereich der Fall. Hierbei liegt der Schwerpunkt meist bei der Abwägung von Nutzen und Kosten. So wurde in [15] ein Kühlsystem in Form auf Basis eines multikriteriellen Problems entworfen und simuliert, wobei sehr gute Werte für Leistung und Effizienz erreicht werden konnten. Gleichartiges wurde in [16] für den Entwurf einer Gasturbine durchgeführt. Aufgrund der Häufigkeit und Realitätsnähe dieses Problemtyps gibt es vielfältige Literatur zu diesem Thema. Grundlegende Untersuchungen und Beschreibungen finden sich beispielsweise in den Fachbüchern [17] und [4].
Auch über die verschiedenen Arten, an diese Probleme heranzugehen sind Veröffentlichungen zu finden. So behandeln [9, 18, 11] die grundlegenden Eigenschaften und Herangehensweisen im Rahmen der Partikelschwarmoptimierung. In [9] wird ausgehend von der Bewegung des einzelnen Partikels im Schwarm ein allgemeines Modell beschrieben, welches Parameter enthält, mit denen es möglich ist das Konvergenzverhalten des Schwarms zu steuern. [18] zeigt, wie die Partikelschwarmoptimierung nicht nur für die Lösung von Optimierungsproblemen mit nur einem Ziel verwendet werden können, sondern erweitert den Anwendungsbereich auf multikriterielle Probleme. In [11] werden die drei Komponenten des Verhaltens eines Schwarms, basierend auf natürlichen Vorgängen, erstmals beschrieben.
Als Pendants dazu dienen [12, 19, 6, 8, 7] im Bereich der evolutionären Algorithmen. In [12] wird zuerst grundlegend in die Thematik der multikriteriellen Problem eingeführt und darauf aufbauend werden verschiedene Algorithmentypen zur Lösung dieser Probleme vorgestellt. Einen ähnlichen Aufbau haben auch die Fachbücher [19, 6, 8]. [7] fokussiert sich im Gegensatz dazu stark auf den Themenbereich der evolutionären Algorithmen.
Neben dieser Literatur zur grundsätzlichen Bearbeitung entsprechender Probleme finden sich auch eine Vielzahl an Veröffentlichungen zu einzelnen Algorithmen. Neben dem im nächsten Kapitel vorgestellten NSGA-II [2] findet sich beispielsweise noch SPEA2 [20] als weiterer evolutionärer Algorithmus. Dieser erweitert die Vorgehensweise seines Vorgängers SPEA beispielsweise um eine neue Strategie zur Bestimmung der Eignung von Individuen und erzielt damit bessere Ergebnisse. AbYSS [21] verfolgt den Ansatz einer Streusuche und erreicht damit eine bessere Diversität als NSGA-II und SPEA2. OMOPSO [22] stellt die Grundlage für den auch im nächsten Kapitel vorgestellten SMPSO [3] dar. Es handelt sich bei OMOPSO um einen schwarmbasierten Algorithmus, der Mutation als neues Element enthält und damit verhindert, dass die Lösungsmenge nicht mehr aus einem lokalen Optimum herauskommt.
Zur Erstellung von Testreihen steht eine große Auswahl an Frameworks zu Verfügung. So ist neben dem für diese Arbeit genutzten JMetal [23] auch das MOEA Framework [24], das Java Evolutionary Computation Toolkit [25] und Paradiseo [26] zu nennen.
An Testfunktionen zur Prüfung und Bewertung von Algorithmen sind als statische ZTD [27, 28], WFG [29] und DTLZ [30] zu nennen. Diese Testfunktionen dienen als standardisierte Benchmarks für multikriterielle Probleme mit unterschiedlichen Schwierigkeitsgraden. Dynamische Benchmarks dieser Art stellen FDA [31], DTF [32], die dMOP-

Funktionen [33] oder die T1- bis T4-Funktion [34] dar, welche wiederum verschiedene Varianten und damit Schwierigkeitsstufen beinhalten. Eine ausführliche Übersicht und Beschreibung dynamischer Testfunktionen findet sich in [35].
Um die Qualität der durch die Algorithmen erzeugten Ergebnisse bewerten zu können sind entsprechende Indikatoren notwendig. Häufig werden in der Literatur das Hypervolumen [13] zur Bestimmung der Konvergenz und Abdeckung des Volumens im Zielraum, und die Streuung [1] zur Messung der Diversität der erreichten Lösungen im Zielraum verwendet. Weitere interessante Indikatoren sind die Hypervolume Ratio [14] als Verhältniswert und die Generational Distance [36] zur Bestimmung der Entfernung der nichtdominierten Lösungsmenge zur pareto-optimalen Front. Eine Übersicht findet sich in [37].
In [38] wurde der Einfluss der Anzahl an Zielpunkten auf die pareto-optimale Menge eines Problems und die Lösungsqualität der Algorithmen untersucht. Hierbei wurde festgestellt, dass die Wahrscheinlichkeit in einem lokalen Optimum zu verbleiben und nicht das globale Optimum zu erreichen mit der Anzahl an Zielpunkten steigt. Das Distanzminimierungsproblem wurde in verschiedenen Varianten untersucht [39, 40, 41, 42], wobei in [40] ein Realweltbezug thematisiert wird. Diese Veröffentlichung beschäftigt sich mit der Frage nach einer optimalen Wohnposition in einer Stadt und stellt fest, dass keinem Algorithmus gelungen ist sowohl für das Hypervolumen als auch für die Diversität gleichzeitig gute Werte zu erreichen. In [39] wird ein Ansatz zur Visualisierung von Problemen mit vielen Zielpunkten in einem zwei- oder dreidimensionalen Entscheidungsraum vorgestellt, wodurch eine einfache Visualisierung des Verhaltens der Algorithmen möglich wird, was wiederum dazu führt, dass das Verhalten leichter analysiert werden kann. [41] untersucht die Auswirkungen der Anzahl an Variablen auf die Schwierigkeit eines Problems und stellt fest, dass diese Anzahl einen besonders starken Einfluss auf die Diversität der ermittelten Lösungen hat. In [42] wurde festgestellt, dass sich die Schwierigkeit des Distanzminimierungsproblems nicht signifikant erhöht, wenn die Anzahl an Zielpunkten erhöht wird. Diese Untersuchung bezieht sich jedoch nur auf das euklidische Abstandsmaß. Dass das Distanzminimierungsproblem unter Verwendung der Summennorm nicht zufriedenstellend gelöst werden konnte wurde in [5] gezeigt. Aus dieser Veröffentlichung wird auch deutlich, dass die verwendente Abstandsfunktion einen deutlichen Einfluss auf die Schwierigkeit eines Problems hat.

Kapitel 3

Ausgewählte Algorithmen

Zur Überprüfung, wie gut bestehende Algorithmen die im Rahmen dieser Arbeit erstellten Testszenarien lösen werden Algorithmen mit verschiedenen Strategien verwendet, um ein ausgeglichenes Bild bestehender Verfahren darzustellen. Die verwendeten Algorithmen werden nachfolgend vorgestellt.

3.1 NSGA-II

Bei NSGA-II [2] (*Non-dominated Sorting Genetic Algorithm II*) handelt es sich um einen evolutionären Algorithmus. Mittels einer sich ständig verbessernden Population wird hierbei versucht eine größtmögliche Annäherung an die pareto-optimale Front zu erreichen. Der nachfolgende Pseudo-Code verdeutlicht dieses bereits im Anschnitt 2.3 vorgestellte Verfahren.

Listing 3.1: Pseudo-Code für NSGA-II nach [2]

```
1  erstellePopulation(P_0);          #Startpopulation P_0 mit n Individuen erzeugen
2  evaluierePopulation(P_0);         #Jedes Individuum der Population wird evaluiert
3  i = 0;                            #Generationsnummer
4  wiederhole bis Abbruchkriterium
5    K_i = erzeugeKinder(P_i);       #P_i erzeugt neue Population mittels
6                                    #Selektion, Kreuzung und Mutation
7    evaluierePopulation(K_i);       #Die neu erstellte Population wird evaluiert
8    V_i = vereinige(P_i, K_i);      #Eltern- und Kindpopulation werden vereinigt
9    P_i+1 = auswerten(V_i);         #Mittels einer Auswertung werden die n besten
10                                   #Individuen als Folgegeneration ermittelt
11   i++;                            #Generationsnummer um 1 inkrementieren
12 ausgabe(P_i);                     #Ergebnis ist Population der letzten Generation
```

Der Algorithmus wird mit vier Parametern gestartet. Der erste Parameter bestimmt die Größe der Population, also die Anzahl an Individuen. Der zweite Parameter legt fest, nach welcher Verfahrensweise Individuen als Elternteile selektiert werden. Als dritter Parameter wird die Art der Kreuzung und als vierter Parameter die Art der Mutation festgelegt.
Die Selektion kann beispielsweise mittels *Binary Tournament* nach [2] erfolgen. Hierbei werden zwei Individuen verglichen und von diesen dasjenige mit der besseren Eignung ausgewählt. Danach werden die nächsten zwei Individuen betrachet, bis die gesamte Population geprüft wurde. Dies führt dazu, dass sich als Ergebnis der Selektion genau halb so viele Individuen ergeben wie in der Ausgangspopulation vorhanden waren. Hier-

bei muss jedoch eine gerade Anzahl an Individuen in der Population vorhanden sein. Nachfolgender Pseudo-Code stellt die Vorgehensweise für ein Minimierungsproblem dar.

Listing 3.2: Pseudo-Code für Binary Tournament nach [2]

```
1 n = 0;                                    #Setze Index n = 0
2 wiederhole bis n >= Anzahl Individuen
3   wenn f(P[n]) < f(P[n+1])                #Vergleiche Individuum n mit Individuum n+1
4     ergebnis = P[n];                      #Individuum mit besserer Eignung
5   sonst                                   #als Ergebnis
6     ergebnis = P[n+1];
7   selektion += ergebnis                   #Ergebnis in Selektionsmenge legen
8   n = n + 2;
```

Eine mögliche Variante für die Kreuzung ist *Simulated Binary Crossover (SBX)* [43]. Hierbei handelt es sich um ein Verfahren, welches die Idee der binären Kreuzung (s. Abbildung 3.1) auf Fließkommazahlen erweitert und dabei dessen relevante mathematische Eigenschaften beibehält. Es werden zwei Eltern für eine Kreuzung benötigt, welche dann wiederum zwei Kinder erzeugen. Dieses Verfahren führt zu einer Verdopplung der Population.

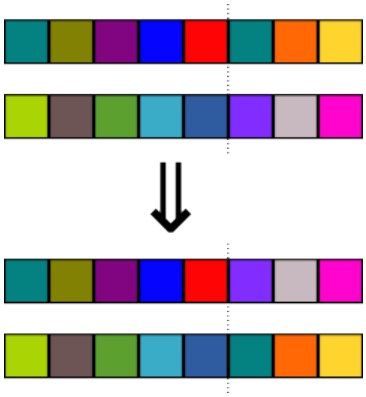

Abbildung 3.1: Binäre Kreuzung zweier 8-Bit-Blöcken ab Block 6

Die Mutation kann beispielsweise über das Verfahren der *polynomialen Mutation* [44] erfolgen. Dieses Verfahren ermöglicht die Mutation von Fließkommazahlen. Für jede Variable eines Individuum (z.B. dessen x-Koordinate, y-Koordinate etc.) wird abhängig von einer Wahrscheinlichkeit der entsprechende Wert verändert. Der Umfang der Veränderung folgt einer vorgegebenen Wahrscheinlichkeitsverteilung, wobei das Ergebnis einer Mutation prinzipiell jeden Wert im Entscheidungsraum annehmen kann.

Die Auswertung der kombinierten Eltern- und Kindpopulation erfolgt bei NSGA-II über die Bildung nicht-dominierter Fronten verbunden mit dem Verfahren *Crowding Distance* zur Wahrung der Diversität der Lösungen [45]. Diese beiden Bestandteile werden nachfolgend vorgestellt. Ausgangspunkt ist hierbei die Population an Lösungen V, welche aus der Vereinigung von Eltern- und Kindpopulation hervorgegangen ist (s. Listing 3.1 Zeile 8). Diese Population ist doppelt so groß wie die jeweils einzelnen Eltern- bzw. Kindpopulationen und muss daher halbiert werden, um die vorgegebene Größe zu erreichen. Zuerst werden dazu nicht-dominierte Fronten gebildet. Dabei wird eine leere Menge F_1 erstellt, welche die erste nicht-dominierte Front darstellt. Die erste Lösung in V wird nun in F_1 aufgenommen. Dann wird die zweite Lösung in V daraufhin geprüft, ob eine Dominanz bezüglich der Lösungen in F_1 (aktuell enthält

F_1 nur die erste Lösung) vorliegt. Dominiert die zweite Lösung die erste, so wird die erste aus F_1 entfernt und die zweite Lösung in F_1 hinzugefügt. Wird jedoch die zweite Lösung durch die Lösungen in F_1 dominiert, so wird die zweite Lösung nicht weiter betrachtet. Liegt keine Dominanz vor, so wird die zweite Lösung F_1 hinzugefügt und F_1 besteht dann aus der ersten und der zweiten Lösung. Dieses Verfahren wird mit allen weiteren Lösungen in V fortgesetzt, wobei immer eine Prüfung auf Dominanz gegen alle bestehenden Lösungen in F_1 durchgeführt wird. Am Ende enthält F_1 eine bestimmte Anzahl an nicht-dominierten Lösungen. Diese Lösungen werden nun aus V entfernt: $V = V \backslash F_1$. Sollte V weiterhin Lösungen enthalten, so werden weitere Fronten $(F_2, F_3, ...)$ erzeugt. Dies geschieht analog zum soeben beschriebenen Verfahren, jedoch ohne die bisherigen nicht-dominierten Lösungen, welche aus V entfernt wurden. Das Ergebnis dieser Frontenbildung ist eine Menge an Fronten $F = (F_1, F_2, ...)$ welche jeweils für sich genommen nur nicht-dominierte Lösungen enthalten. Hierbei sind die Lösungen der Fronten mit kleinerem Index als besser anzusehen, als solche mit höherem Index, da die Lösungen aus Fronten mit kleinerem Index jene aus Fronten mit höherem Index dominieren.
Nachdem nun eine Sortierung der Lösungen über die Zuordnung zu bestimmten Fronten erfolgt ist, kann anhand dieser Ordnung die neue, halbierte Population aufgebaut werden. Hierbei werden zuerst die Lösungen der ersten Front in die Population übertragen, dann die der zweiten usw.. Jedoch kann es vorkommen, dass es innerhalb einer Front mehr Lösungen gibt, als noch in der Population Platz zur Verfügung steht. In diesem Fall müssen die Lösungen einer Front weiter unterschieden werden. Hier kommt *Crowding Distance* als weiteres Verfahren hinzu. Dieses sortiert die Lösungen innerhalb einer Front jeweils bezogen auf eine Zielfunktion. Der gesamte Ablauf ist in Listing 3.3 als Pseudo-Code für eine Front F dargestellt. Es wird für jede Lösung i eine Distanz bestimmt, indem die Differenz der Funktionswerte, bezogen auf die aktuelle Zielfunktion, der beiden angrenzenden Lösungen $i-1$ und $i+1$ bestimmt wird (s. Abbildung 3.2). Die beiden extremsten Lösungen, welche den Anfang bzw. das Ende der innerhalb der Sortierung darstellen erhalten einen Abstandswert von unendlich. Dies stellt sicher, dass diese Punkte immer bevorzugt werden.

Listing 3.3: Crowding Distance nach [2]

```
1 j = |F|                          #j = Anzahl an Lösungen in der Front
2 für i=0 bis j-1                  #alle Distanzwerte auf 0 setzen
3   F[i]_distance = 0
4 für jede Zielfunktion k
5   F = sortiere(F,k)              #Front bzgl. der Zielfunktion sortieren
6   F[0]_distance = ∞              #Distanzwert der Extremwertlösungen = ∞
7   F[j-1]_distance = ∞
8   für i=1 bis j-2
9     F[i]_distance = F[i]_distance + (F[i+1].m - F[i-1].m) #Distanz angrenzender Funktionswerte
```

Sollte es also dazu kommen, dass nicht alle Lösungen einer Front in die Population aufgenommen werden können, so werden diejenigen Lösungen mit einem höheren Distanzwert bevorzugt. Durch dieses Kriterium werden Lösungen aus Bereichen mit einer hohen Lösungsdichte potentiell nicht in die Population aufgenommen. Dies sorgt dafür, dass die endgültige Lösungsmenge ein höheres Maß an Diversität aufzeigt.

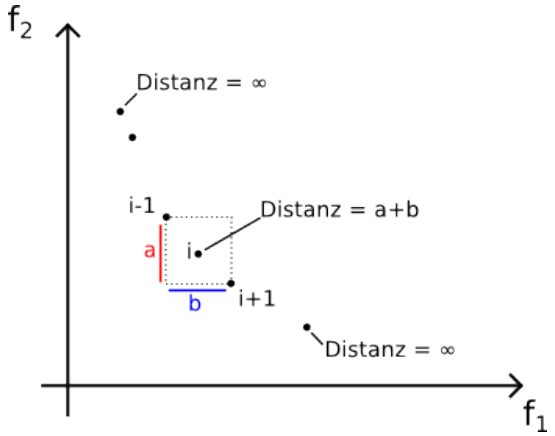

Abbildung 3.2: Crowding Distance nach [2]

3.2 DNSGA-II-A

Bei DNSGA-II-A [45] (Dynamic non-dominated Sorting Genetic Algorithm II A) handelt es sich um die dynamische Variante des soeben vorgestellten NSGA-II-Algorithmus. Diese ist in der Lage auf Veränderungen der Probleminstanz zu reagieren und ihr Verhalten entsprechend anzupassen. Neben DNSGA-II-A gibt es auch eine Variante DNSGA-II-B, welche ebenfalls in [45] beschrieben wird und eine andere Strategie verfolgt, sobald eine Änderung erkannt wurde. Im Rahmen dieser Arbeit wird ausschließlich die Variante DNSGA-II-A verwendet, da diese auch in der Literatur häufiger Anwendung findet. Nachfolgend wird die Funktionsweise dieser Variante beschrieben.
Das grundsätzliche Vorgehen entspricht jenem von NSGA-II. Jedoch wird nach der Erstellung der Kindergeneration und vor der Vereinigung dieser mit der Elterpopulation (also vor Zeile 8 in Listing 3.1) ein gewisser Anteil der Elternpopulation neu evaluiert. Sollte sich hierbei zeigen, dass sich einer der Funktionswerte oder eine der Einschränkungen des Problems verändert hat, so wird davon ausgegangen, dass sich die Probleminstanz verändert haben muss. Dann wird ein festgelegter Anteil an Individuen der Elterngeneration zufällig im Entscheidungsraum verteilt, um vollkommen neue Lösungen in die Population einzuführen. Darauf folgend wird die gesamte Elternpopulation neu evaluiert und erst danach mit der Kindergeneration vereinigt. Das weitere Vorgehen nach diesen Aktualisierungen entspricht jenem des statischen NSGA-II.

3.3 SMPSO

SMPSO (*Speed-constrained Particle Swarm Optimization*) [3] ist ein metaheuristischer Lösungsansatz für multikriterielle Probleme, der auf Partikelschwarmoptimierung basiert. Wie in Abschnitt 2.4 beschrieben bewegen sich dabei die einzelnen Partikel eines Schwarms hin zu einem vermuteten Optimum. Dabei ist die Änderung der Position eines Partikels nach Gleichung 2.4 nur von der Geschwindigkeit $\vec{v_i}(t)$ abhängig. Um extreme Werte für diese Geschwindigkeit und damit extreme Bewegungen der Partikel zu vermeiden (*speed-constrained*) verwendetet SMPSO einen Beschränkungsfaktor χ. Dieser

ergibt sich aus

$$\chi = \frac{2}{2 - \varphi - \sqrt{\varphi^2 - 4\varphi}} \tag{3.1}$$

mit

$$\varphi = \begin{cases} C_1 + C_2 & \text{falls } C_1 + C_2 > 4 \\ 0 & \text{sonst} \end{cases} \tag{3.2}$$

Wie in [3] gezeigt ist diese Vorgehensweise im Hinblick auf Geschwindigkeit und Genauigkeit der ermittelten Lösungen anderen Verfahren, wie beispielsweise SPEA2 oder OMOPSO, überlegen. Aus diesem Grund wird SMPSO als ein leistungsfähiger Algorithmus für die Tests herangezogen. Listing 3.3 zeigt den Ablauf von SMPSO in Pseudo-Code.

Listing 3.4: Pseudo-Code für SMPSO nach [3]

```
1  erstelleSchwarm();                #Schwarm mit initialen Positionen wird erstellt
2  evaluation();                     #Funktionswerte aller Partikel berechnen
3  ermittleBestePartikel();          #Ermittlung der besten Partikel
4  wiederhole bis Abbruchkriterium
5    berechneGeschwindigkeit();      #Geschwindigkeit aller Partikel berechnen
6    aktualisierePosition();         #Position aller Partikel aktualisieren
7    mutation();                     #Mutation
8    evaluation();                   #Funktionswerte aller Partikel aktualisieren
9    ermittleBestePartikel();        #Aktualisierung der besten Partikel
10   ermittleBesteEigenePosition(); #Aktualisierung der jeweiligen besten Position
11 ausgabe(BestePartikel);           #Ergebnis ist Population der letzten Generation
```

3.4 dSMPSO

Aufgrund der weiten Verbreitung und großen Leistungsfähigkeit wurde SMPSO für die Tests im Rahmen dieser Arbeit ausgewählt. Eine entsprechende dynamische Variante von SMPSO, wie DNSGA-II-A für NSGA-II, ist nicht bekannt. Um diesen Umstand zu beheben wird für die dynamischen Tests im kommenden Teil dieser Arbeit eine dynamische Variante von SMPSO erzeugt. Diese wird mit *dSMPSO* (d für *dynamic*) bezeichnet. Das prinzipielle Vorgehen ist identisch mit jenem von DNSGA-II-A. Dies hat zwei Gründe. Zum Einen ist der Ansatz von DNSGA-II-A generisch und daher gut auf andere Verfahren übertragbar. Zum Anderen ergibt sich eine gute Vergleichbarkeit der beiden Algorithmentypen. Dies ist der Fall, da die Erkennung und Reaktion einer Veränderung losgelöst vom eigentlichen Prozess der Problemlösung ist. Damit kann mit größerer Sicherheit eine Aussage über die Qualität der jeweiligen Problemlösung getroffen werden. Der Algorithmus wird demnach so erweitert, dass vor der Aktualisierung der Positionen (Zeile 6 in Listing 3.4) ein Teil der Partikel neu evaluiert werden. Dabei wird überprüft, ob sich ein Funktionswert oder eine Einschränkung des Problems geändert hat. Ist dies der Fall wird der gesamte Schwarm neu evaluiert. Zuvor wird jedoch, wie bei DNSGA-II-A, ein bestimmter Anteil der Partikel zufällig im Entscheidungsraum verteilt. Danach werden sowohl die persönlichen als auch die schwarmweiten besten Positionen aktualisiert. Mit diesen neuen Werten für $\overrightarrow{x_{p_i}}$ und $\overrightarrow{x_g}$ werden die Geschwindigkeiten neu berechnet. Anschließend werden die Partikel auf ihre neuen Positionen bewegt. Diese Bewegungen entsprechen dann Zeile 6 in Listing 3.4. Von da an wird der Algorithmus wie im statischen Fall fortgesetzt.

3.5 GRA

Der GRA (*Grid Refinement Algorithm*) wurde im Rahmen dieser Arbeit vom Autor erstellt mit dem Ziel das (dynamische) Distanzminimierungsproblem besser verstehen und lösen zu können. Er basiert auf einem Gitter im Entscheidungsraum, welches im Laufe der Berechnung immer feiner wird. Zu Beginn wird dem Algorithmus ein Startabstand d zwischen zwei Lösungen im Gitter als Parameter übergeben. Aus diesem Abstand ergibt sich das initiale Gitter, wobei ein kleiner Abstand ein feineres Gitter und ein größerer Abstand ein gröberes Gitter erzeugt. Das Gitter selbst liegt hierbei immer parallel bzw. orthogonal zu den Koordinatenachsen. Der Ablauf des Algorithmus ist in einem Programmablaufplan in Abbildung 3.4 dargestellt, wobei die Prüfung über die Aufnahme einer neuen Lösung in die Population detailliert in Abbildung 3.5 dargestellt ist. Weiterhin enthält Listing 3.5 den entsprechenden Pseudo-Code für GRA.
Im ersten Schritt wird eine Lösung zufällig im Entscheidungsraum auf dieses Gitter gesetzt und evaluiert (s. Abbildung 3.3).

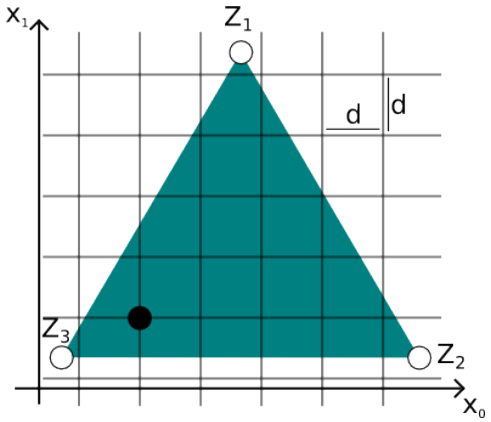

Abbildung 3.3: GRA - Initiale Lösung auf Gitter

Nun werden, ausgehend von dieser initialen Lösung, die Variablen des Entscheidungsraums nacheinander abgearbeitet. Hierfür wird dem GRA ein weiterer Parameter γ übergeben, welcher festlegt mit welcher Wahrscheinlichkeit eine Variable behandelt wird. Dies hat zur Folge, dass vor allem für Probleme mit vielen Variablen die Möglichkeit besteht nicht für jede Lösung alle Variablen behandeln zu müssen und damit die Konvergenzgeschwindigkeit drastisch zu erhöhen. Die Abbildung 3.6 zeigt hier das Ergebnis für die erzeugten Lösungen bei einem Wert von $\gamma = 1.0$. Im Gegensatz dazu zeigt Abbildung 3.7 eines der vier möglichen Ergebnisse für einen Wert von $\gamma = 0.5$.
Soll mit einer Variable eine Berechnung stattfinden, so wird ausgehend vom der initialen Lösung bezüglich der gewählten Variable einmal der Abstand der Gitterpunkte sowohl addiert als auch subtrahiert. Damit ergeben sich aus der initialen Lösung zwei neue Lösungen pro Variable. Die Koordinaten der initialen Lösung werden einer Liste hinzugefügt, durch welche gesichert wird, dass dieser Lösung nicht noch einmal neue Lösungen erzeugt. Für die Festlegung des Wertes für γ spielt neben der Anzahl an Variablen auch die Anzahl an zur Verfügung stehenden Evaluationen eine Rolle. Bei wenigen zur Verfügung stehenden Evaluationen sollte der Wert für γ kleiner gewählt werden als bei vielen.

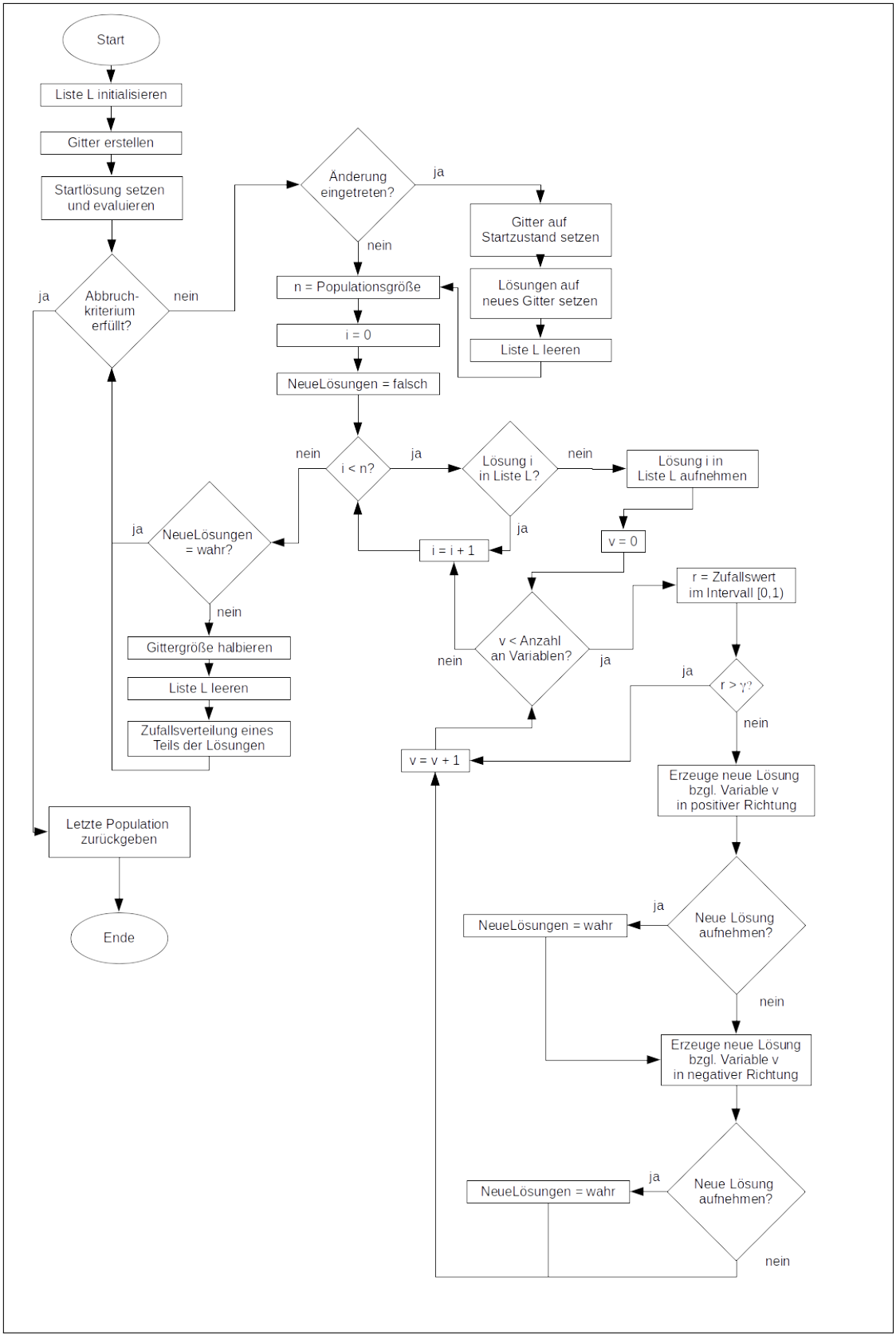

Abbildung 3.4: PAP für GRA

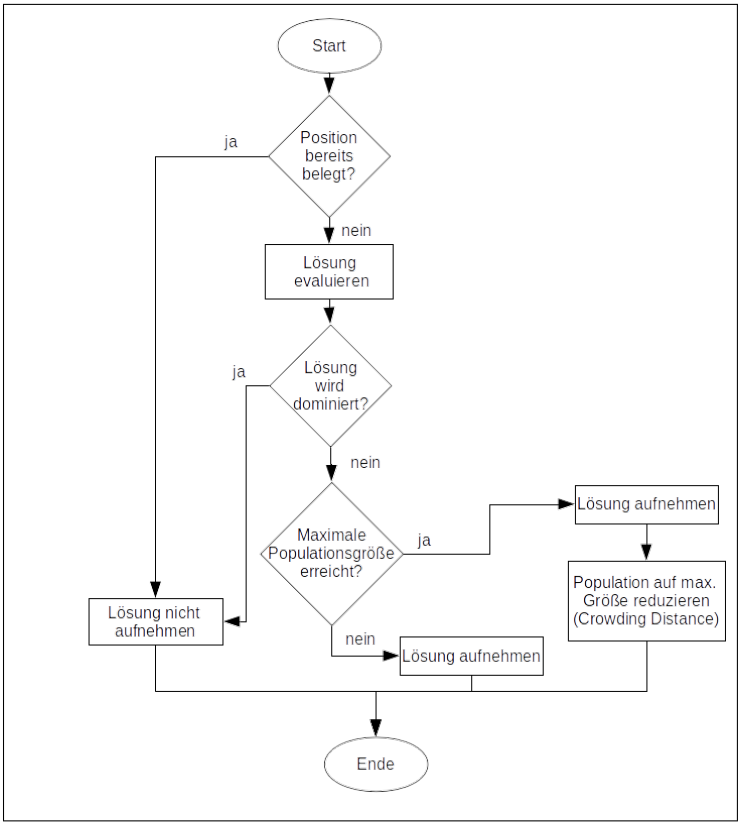

Abbildung 3.5: PAP zur Entscheidung über die Aufnahme einer neuen Lösung in die Population

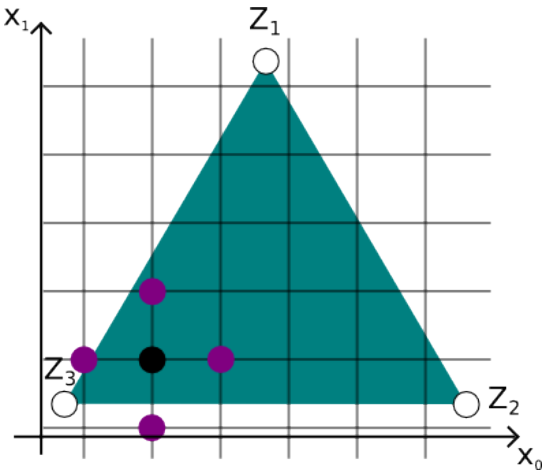

Abbildung 3.6: GRA - Erzeugte Lösungen für $\gamma = 1.0$

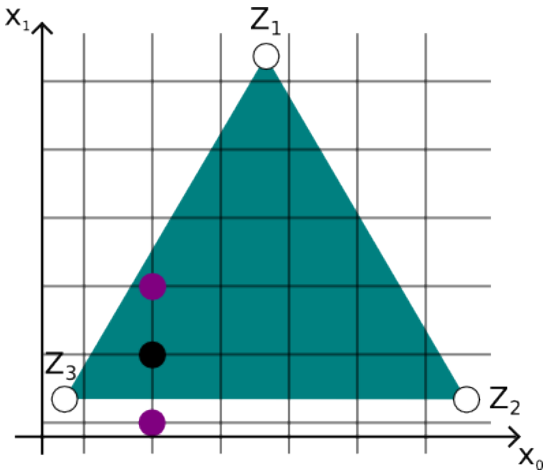

Abbildung 3.7: GRA - Mögliche Lösungen für $\gamma = 0.5$

Um über die Aufnahme neuer Lösungen in die Population entscheiden zu können wird ein dritter Parameter, die maximale Populationsgröße, benötigt.
Nach der initialen Generation, bei welcher die Startlösung neue Lösungen erzeugt hat, folgen nun weitere Generationen. Bei diesen dienen alle Lösungen der Population als Ausgangspunkte für neu zu erzeugende Lösungen. Diejenigen, welche jedoch bereits Lösungen erzeugt haben, werden dabei übersprungen.

Sollte es sich ergeben, dass in einer Generation keine neuen Lösungen in die Population aufgenommen werden können, so wird der Abstand zwischen zwei Lösungen auf dem Gitter halbiert (s. Abbildungen 3.8 und 3.9), wodurch das Gitter feiner wird. Weiterhin wird die Liste mit Lösungen, welche bereits neue Lösungen erzeugt haben, geleert, so dass nun alle Lösungen wieder neue Punkte erzeugen können. Zusätzlich werden 10 % der bestehenden Lösungen zufällig auf dem Gitter verteilt, um gänzlich neue Lösungen erhalten zu können.

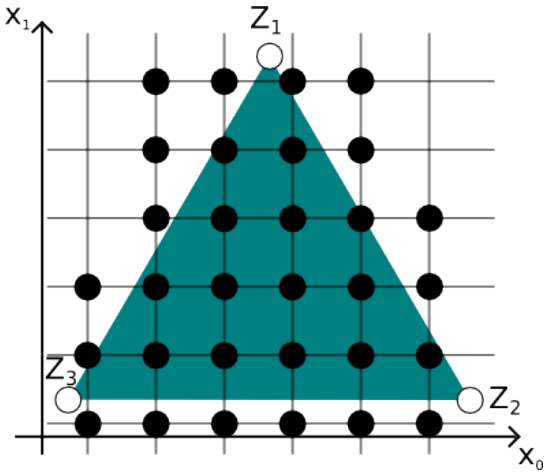

Abbildung 3.8: GRA - Keine neuen Punkte möglich

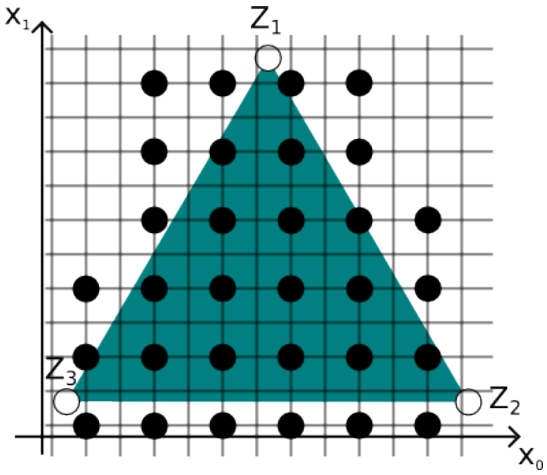

Abbildung 3.9: GRA - Verfeinerung des Gitters

In der nächsten Generation werden die neuen Lösungen, passend zum neuen Gitter, mit der Hälfte des bisherigen Abstands gesetzt. Dadurch ergibt sich eine weiterere Verfeinerung des von den Lösungen aufgespannten Raumes innerhalb des Entscheidungsraum. Dies wird solange wiederholt, bis ein entsprechendes Abbruchkriterium (z.B. eine bestimmte Anzahl an Evaluationen) erreicht wird.
Dieser Ansatz des Teile-und-Herrsche-Prinzips basierend auf einer Gitterstruktur wurde im Rahmen der Lösung von multikriteriellen Problemen bisher nicht verfolgt. Er stellt damit eine andere Lösungsstrategie als NSGA-II oder SMPSO dar. Hierdurch stehen im Evaluationsteil dieser Arbeit drei verschiedene Lösungsansätze für multikriterielle Optimierungsprobleme zu Verfügung, die auf ihre spezielle Eignung zur Lösung des (dynamischen) Distanzminimierungsproblems getestet werden.

3.5.1 GRA und dynamische Probleme

Die bisher beschriebene Funktionsweise des GRA erkennt keine Änderungen innerhalb der Probleminstanz und ist daher nur für statische Probleme anwendbar. Um den GRA auch für dynamische Probleme nutzbar zu machen, ist das Erkennen von und die Reaktion auf Veränderungen notwendig. Hierzu wird wiederum das von DNSGA-II-A bekannte generische Verfahren benutzt. Dies bedeutet, dass vor der Erstellung der neuen Lösungen entlang des Gitters ein Teil der bestehenden Lösungen daraufhin geprüft werden, ob sich die Funktionswerte oder die Einschränkungen des Problems geändert haben. Ist dies der Fall wird das Gitter auf seinen anfänglichen Abstand zurückgesetzt. Die bestehenden Lösungen werden auf dem neuen Gitter verteilt. Dabei werden sie auf die Koordinaten

gesetzt, die ihren ursprünglichen am nächsten sind. Weiterhin dürfen nun alle Lösungen wieder neue Lösungen erzeugen. Damit endet die Reaktion auf Veränderung und die Erzeugung neuer Lösungen beginnt, wie im vorherigen Abschnitt beschrieben.
Das soeben beschriebene Verfahren mit Prüfung auf Dynamik findet bei GRA immer statt, d.h. es gibt keine explizite Variante des GRA für statische Probleme. Die Bezeichnung GRA schließt damit im Rahmen dieser Arbeit immer die Prüfung und Reaktion auf Veränderungen durch den Algorithmus ein.
Ein Grund hierfür ist die Universalität dieses Ansatzes. Ein dynamisches Problem besteht aus einer Probleminstanz mit einer Änderungsfunktion, welche abhängig von bestimmten Parametern die Probleminstanz ändert. Diese Art von Problemen kann von dynamischen Algorithmen gelöst werden. Statische Algorithmen hingegen werden Schwierigkeiten haben hier gute Ergebnisse zu erzielen. Ein statisches Problem kann als Probleminstanz mit einer leeren Änderungsfunktion angesehen werden, da die Probleminstanz nie verändert wird. Diese Art von Problemen kann sowohl von statischen als auch von dynamischen Algorithmen gelöst werden, wobei statische Algorithmen hier in der Regel bessere Ergebnisse erzielen werden, da sie keine Ressourcen auf die Prüfung einer (in diesen Problemen nicht vorhandenen) Änderung verbrauchen. Ein dynamischer Algorithmus ist jedoch für beide Arten von Problemen geeignet und somit universell. Liegt ein unbekanntes Problem vor, so muss davon ausgegangen werden, dass es sich um ein dynamisches Problem handelt (*worst case*).
Der Nachteil des dynamischen Algorithmus liegt offensichtlich in den für die Prüfung auf Änderung verbrauchten Evaluationen. Die Anzahl der verbrauchten Evaluationen wird oft als Maßeinheit für die Lösungsgeschwindigkeit herangezogen bzw. die maximale Anzahl an Evaluationen ist begrenzt. Aufgrund der ersten Erfahrungen mit GRA ist erkennbar, dass zum Ermitteln guter Lösungsmengen sehr wenige Evaluationen erforderlich sind. Dies ist trotz der nicht notwendigen Prüfung auf Veränderungen von statischen Problemen der Fall. Gründe hierfür liegen zum einen in der kleinen Startpopulation von nur einer Lösung und zum anderen daran, dass nur nicht-dominierte Lösungen in der Lösungsmenge vorkommen.
Zusammenfassend lässt sich sagen, dass durch die Universalität und die bisher beobachtete gute Konvergenzgeschwindigkeit eine rein statische Variante von GRA zumindest in den vorliegenden Testfällen keine nennenswerten Vorteile mit sich bringt. Sollte im Rahmen zukünftiger Arbeiten die Notwendigkeit für eine statische Variante von GRA bestehen, so ist diese leicht aus der dynamischen Variante durch ein Überspringen der Prüfung auf Veränderung erstellbar.

Listing 3.5: Pseudo-Code für GRA

```
1  L = ∅;                                #Liste mit Lösungen, die bereits Lösungen erzeugt
2                                        #haben initialisieren
3  erstelleGitter(d_start);              #Startgitter mit Abstand d_start erstellen
4  P_0 = setzeStartpunkt();              #Erste Lösung P_0 zufällig auf Gitter setzen
5  evaluierePunkt(P_0);                  #Startlösung evaluieren
6  wiederhole bis Abbruchkriterium
7    Änderung = prüfeAufÄnderung();      #Prüfung ob sich Probleminstanz geändert hat
8    wenn Änderung
9      d = d_start;
10     erstelleGitter(d);                #Gitter auf Startgitter zurücksetzen
11     setzeLösungen();                  #Alle Lösungen auf neues Gitter setzen
12     L = ∅;                            #Liste leeren
13   n = |P|;                            #n = Anzahl der Lösungen auf dem Gitter
14   ∀P_i|(i < n ∧ P_i ∉ L)              #Alle Lösungen, die bisher auf dem Gitter und nicht
15                                       #in der Liste sind
16     L += P_i;                         #werden in die Liste aufgenommen und
17     erzeugeLösungen(γ);               #erzeugen mit Wahrscheinlichkeit γ pro Variable
18                                       #zwei neue Lösungen
19                                       #Die Aufnahme der neuen Lösungen in die Population
20                                       #erfolgt, wenn die Kriterien erfüllt sind
21   wenn keineNeuenLösungen
22     d = d/2;
23     erstelleGitter(d);                #Gitter verfeinern
24     L = ∅;                            #Liste leeren
25     zufallsverteilung();              #10 % der Lösungen zufällig auf Gitter verteilen
26 ausgabe(P);                           #Ergebnis ist Population der letzten Generation
```

Kapitel 4

Dynamisches Distanzminimierungsproblem

In diesem Kapitel wird die Erstellung des dynamischen Distanzminimierungsproblems beschrieben. Zuerst wird auf die Motivation zur Erstellung dieses Problems eingegangen. Es geht hierbei darum, welche Vorteile dieses Problem gegenüber anderen Problemen hat und warum es sich deshalb für bestimmte Zwecke besser eignet.
Danach werden die Problemstellung und die gestellten Anforderungen beschrieben. Es werden Eigenschaften und Möglichkeiten vorgestellt, die das dynamische Distanzminimierungsproblem haben sollte.
Darauf aufbauend erfolgt die Beschreibung, wie diese gewünschten Eigenschaften umgesetzt wurden und welche Möglichkeiten mit dem erstellten Problemtyp gegeben sind.

4.1 Motivation

Der grundsätzliche Wunsch, der zur Erstellung des dynamischen Distanzminimierungsproblems geführt hat, war es, eine dynamische Problemklasse zu haben, bei welcher die Komplexität und Schwierigkeit beliebig skalierbar ist. Weiterhin sollte das Problem selbst leicht nachvollziehbar und visualisierbar sein.
Es gibt bereits eine Reihe von statischen (z.B. ZTD [27, 28]) und dynamischen (z.B. FDA [31]) Problemklassen (s. Gleichungen 4.1 und 4.2). Diese haben jedoch die Nachteile, dass sich die Komplexität des Problems durch die Dynamik nicht ändert oder sie nicht gut visualisierbar und analysierbar sind. Das statische Distanzminimierungsproblem hingegen ist gut visualisierbar und analysierbar und durch Veränderungen, welche in diesem Kapitel beschrieben werden, lässt sich auch die Komplexität dynamisch verändern.

$$\text{ZDT1: Minimiere} = \begin{cases} f_1(x) = x_1 \\ f_2(x) = g(x)h(f_1(x), g(x)) \\ g(x) = 1 + \frac{9}{29}\sum_{i=2}^{30} x_i \\ h(f_1(x), g(x)) = 1 - \sqrt{\frac{f1_(x)}{g(x)}} \end{cases} \tag{4.1}$$

$$\text{FDA1: Minimiere} = \begin{cases} f_1(x_|) = x_1 \\ g(x_{||}, t) = 1 + \sum_{x_i \in x_{||}} (x_i - G(t))^2 \\ h(f_1, g) = 1 - \sqrt{\frac{f_1}{g}} \\ mit \\ G(t) = \sin(0.5\pi t), t = \frac{1}{n_t} \lfloor \frac{\tau}{\tau_t} \rfloor \\ x_| \in [0, 1] \\ x_{||} = (x2, \ldots, x_n) \in [-1, 1]^{n-1} \end{cases} \tag{4.2}$$

Weiterhin ist die Anzahl an Variablen in diesen Problemen fest definiert und liegt beispielsweise bei der ZDT-Klasse bei 30. Diese Anzahl ist prinzipiell änderbar, jedoch wird in der Literatur beim Vergleich von Algorithmen unter Verwendung dieser Probleme oft die standardisierte Anzahl an Variablen referenziert [2, 3, 20, 21, 46].
Auch die Anzahl an Zielfunktionen ist in diesen Problemklassen fest vorgegeben. Dies ist darin begründet, dass diese Probleme das Ziel haben eine bestimmte parteo-optimale Front zu erzeugen, welche durch den Algorithmus gefunden werden soll. Das Hinzufügen, Ändern oder Entfernen einer Zielfunktion würde die Form dieser Front verändern und damit auch die gewünschten mathematischen Eigenschaften des Problems. Dies gilt analog für die sich vom aktuellen Zeitschritt abhängig ändernde Front bei dynamischen Problemen.
Auch gibt es bei den genannten Probleme keine Möglichkeit festzustellen, ob die Schwierigkeit eines Problems durch das Problem selbst oder durch die verwendete Dynamik entsteht. Dies liegt daran, dass es zu den statischen Problemen keine äquivalenten dynamischen Probleme gibt und umgekehrt.
Diese Einschränkungen sollen mit dem dynamischen Distanzminimierungsproblem nicht auftreten.

4.2 Problemstellung

In diesem Abschnitt werden die Anforderungen an das zu erstellende Problem einzeln vorgestellt. Diese ergeben sich zum größten Teil aus den bestehenden Einschränkungen bekannter Probleme, wie im vorherigen Abschnitt dargestellt.
Die Anforderung der Vergleichbarkeit von statischen und dynamischen Problemen wird hierbei nicht explizit behandelt, da sich diese automatisch daraus ergibt, dass zu jedem statischen Problem unter Verwendung der gleichen Parametern (Anzahl an Variablen und Zielpunkten, gleiche Zielfunktionen) ein entsprechendes dynamisches Vergleichsproblem erstellt werden kann und umgekehrt.

4.2.1 Änderung der Anzahl an Variablen

Es soll als erste Anforderung eine beliebig skalierbare Anzahl an Variablen möglich sein. Diese wird einmalig beim Definieren des Problems festgelegt, soll aber während der Bearbeitung des Problems veränderbar sein. Die Anzahl der Variablen legt in einem Distanzminimierungsproblem die Dimensionalität des Entscheidungsraumes fest. Eine größere Anzahl an Variablen erzeugt dabei in der Regel ein schwierigeres Problem, da die Menge an Lösungsmöglichkeiten mit der Erhöhung der Anzahl an Variablen stark zunimmt. Somit bestünde über die Veränderung der Anzahl an Variablen eine einfache Möglichkeit die Schwierigkeit des Problems dynamisch zu beeinflussen. Über diese Änderungsmöglichkeit lassen sich die pareto-optimale Menge und die pareto-optimale Front während der Laufzeit des Problems verändern. Die Abbildungen 4.1 und 4.2 zeigen diese Änderung an einem Beispiel. Der Entscheidungsraum besteht vor der Änderung aus den beiden Variablen x_0 und x_1. Bei der Einführung einer zusätzlichen Variablen bleiben die Werte die Zielpunkte $\overrightarrow{Z_1}$ bis $\overrightarrow{Z_3}$ für diese bisherigen Variablen gleich. Der pareto-optimale Bereich ändert sich jedoch dadurch, dass der Zielpunkt $\overrightarrow{Z_2}$ für die neue Variable x_2 einen Wert $\neq 0$ aufweist, wie in Abbildung 4.2 zu erkennen ist.

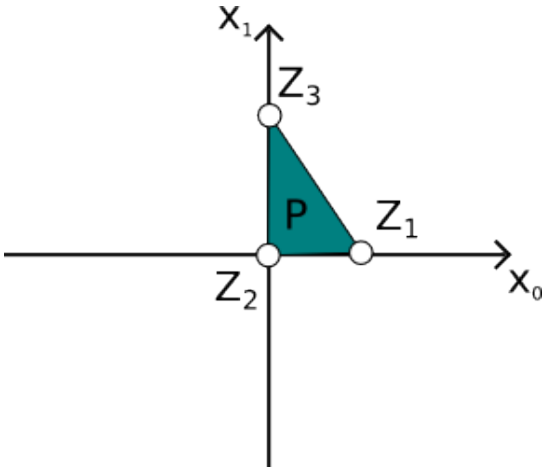

Abbildung 4.1: Probleminstanz mit zwei Variablen

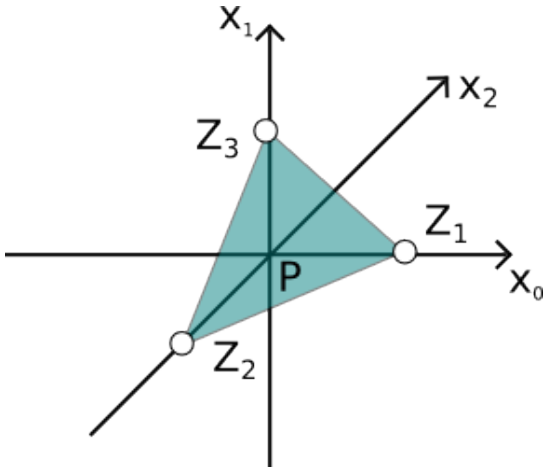

Abbildung 4.2: Probleminstanz mit drei Variablen

4.2.2 Änderung der Anzahl an Zielpunkten

Die zweite Anforderung stellt die beliebige Skalierbarkeit der Anzahl an Zielpunkten dar. Auch diese werden mit einem Initialwert bei der Erstellung des Problems definiert und sollen sich während der Laufzeit des Problems dynamisch verändern können. Durch das Ändern der Anzahl an Zielpunkten verändert sich potentiell die pareto-optimale Menge im Distanzminimierungsproblem, da die Zielpunkte die Extremwerte dieser Menge darstellen. Dies ist beispielhaft in den Abbildungen 4.3 und 4.4 dargestellt. Hierbei ist zu beachten, dass Punkte, welche vor der Veränderung nicht im pareto-optimalen Bereich lagen nach der Änderung durchaus in diesem liegen können. Dies betrifft in den Abbildungen die Punkte, welche sich im Dreieck ($\overrightarrow{Z_1}$,$\overrightarrow{Z_3}$,$\overrightarrow{Z_4}$) befinden. Gleichzeitig ändert sich die Anzahl an Zielfunktionen, da es im Distanzminimierungsproblem eine Zielfunktion pro Zielpunkt gibt.

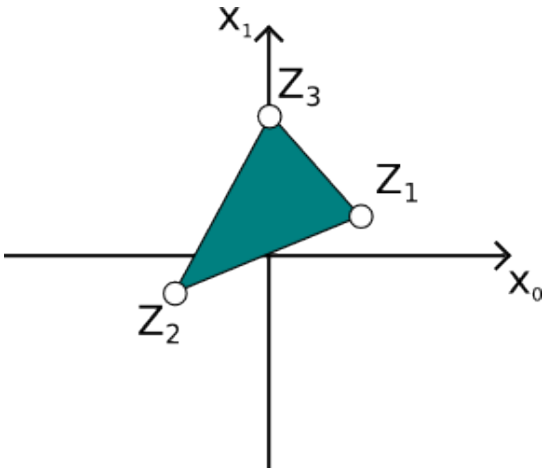

Abbildung 4.3: Probleminstanz mit drei Zielpunkten

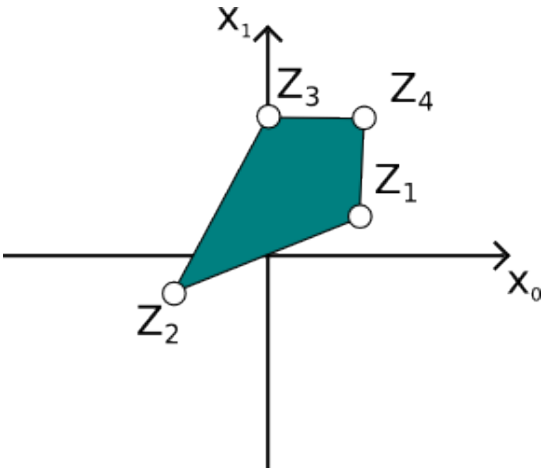

Abbildung 4.4: Probleminstanz mit vier Zielpunkten

4.2.3 Änderung der Lage der Zielpunkte

Als dritte Anforderung ist die Möglichkeit der Änderung der Lage der Zielpunkte im Entscheidungsraum zu nennen. Hierfür sollen einfach zu nutzende Funktionen zur Verfügung stehen, welche die Position der Zielpunkte im Entscheidungsraum (mit beliebiger Dimensionalität; s. Abschnitt 4.2.1) verändern können. Diese Möglichkeit der Veränderung verdeutlicht auf einfache Weise die Änderung der Lage und des Funktionswertes der optimalen Lösungen, wie in Abschnitt 2.1 beschrieben. Hierdurch können wiederum veränderte pareto-optimale Bereiche und Fronten entstehen. Die Abbildungen 4.5 und 4.6 stellen die Zielpunkte vor bzw. nach einer Verschiebung entlang der x_0-Achse dar. Durch die Verschiebung der Zielpunkte verschiebt sich auch der pareto-optimale Bereich entsprechend.

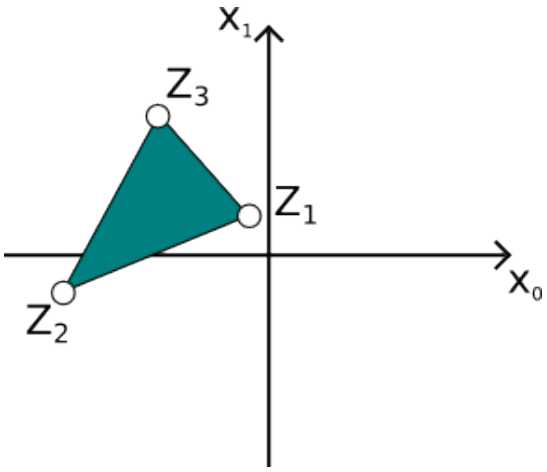

Abbildung 4.5: Ausgangslage vor Verschiebung entlang der x_0-Achse

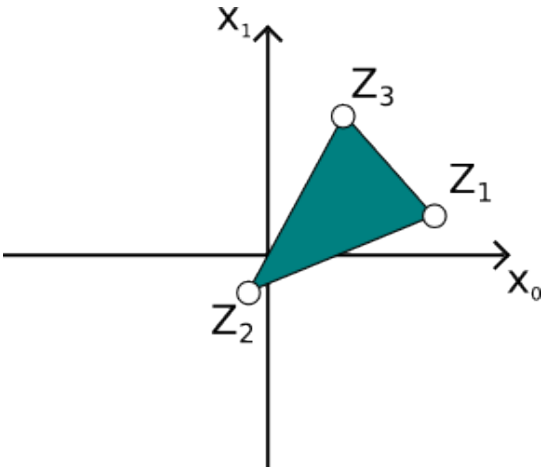

Abbildung 4.6: Ergebnis nach Verschiebung entlang der x_0-Achse

4.2.4 Änderung der Zielfunktionen

Die innerhalb eines Distanzminimierungsproblems verwendete Zielfunktion stellt einen erheblichen Indikator für die Schwierigkeit des Problems dar, wie in [5] gezeigt. Daher ist die Möglichkeit der dynamischen Änderung dieser Funktion ein guter Ansatzpunkt zur Beeinflussung der Schwierigkeit eines Problems. Diese Änderung stellt die vierte Anforderung dar. Hierbei muss darauf hingewiesen werden, dass im Rahmen dieser Arbeit immer dieselbe Zielfunktion für alle Zielpunkte verwendet wird. Die Zuordnung verschiedener Zielfunktionen für einzelne Zielpunkte stellt einen möglichen Aspekt zukünftiger Forschung, wie in Abschnitt 6.2 beschrieben, dar. Die Änderung der Zielfunktion kann einen erheblichen Einfluss auf die pareto-optimale Menge und Front haben. Die Abbildungen 4.7 und 4.8 machen dies deutlich. Die gleichen Zielpunkte erzeugen bei unterschiedlichen p-Normen ganz unterschiedliche pareto-optimale Bereiche.

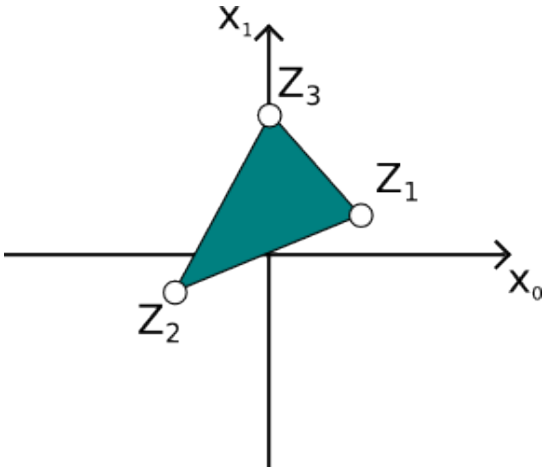

Abbildung 4.7: Pareto-optimale Menge dreier Zielpunkte mit $p = 2$

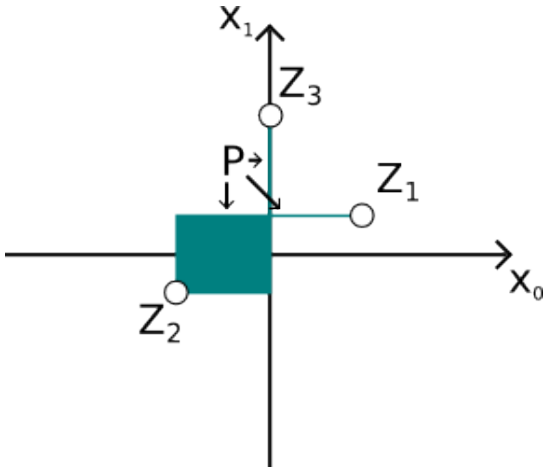

Abbildung 4.8: Pareto-optimale Menge der Zielpunkte mit $p = 1$

4.3 Umsetzung

Dieser Abschnitt beinhaltet die Beschreibungen, wie die einzelnen Anforderungen aus dem letzten Abschnitt umgesetzt wurden. Hierbei wird jeweils das Vorgehen beschrieben und gegebenenfalls ausschnittsweise mit dem entsprechenden Quelltext vorgestellt. Grundsätzlich ist die gesamte Dynamik des Problems in einer Funktion gekapselt, welche abhängig von der Anzahl an bereits durchgeführten Evaluationen Anpassungen durchführt. Diese Funktion wird nach jeder Evaluation aufgerufen (s. Abbildung 4.9). Wann eine Änderung durch die entsprechende Funktion vorgenommen wird, ist auf zwei Arten einstellbar, wobei die zu nutzende Variante dem Problem über einen Parameter mitgeteilt wird.
Die erste Möglichkeit ist es eine Änderung nach einer bestimmten Anzahl an Evaluationen vorzunehmen. Diese Möglichkeit wird hier als „Riesenschritte“ (*giant steps*) bezeichnet und eignet sich für die Umsetzung weniger aber großer Änderungen, welche selten auftreten. Hierfür kann als Parameter die Anzahl an Evaluationen angegeben werden,

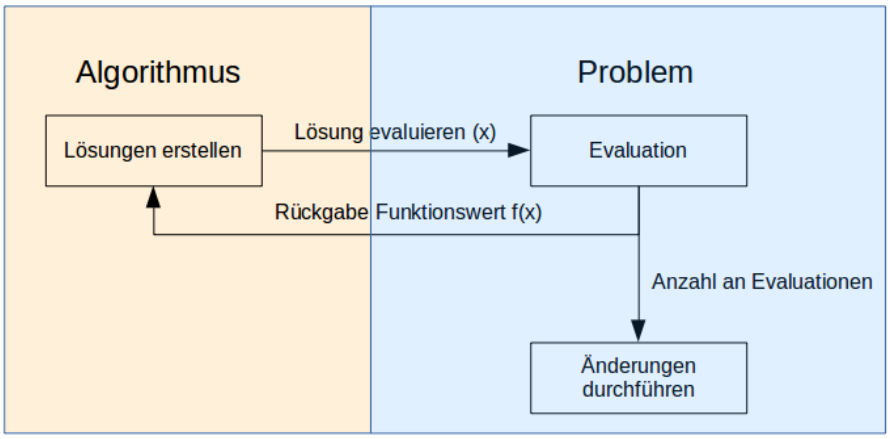

Abbildung 4.9: Schematischer Ablauf von Evaluation und dynamischer Änderung

nach denen eine solche Änderung erfolgen soll. Mathematisch ist dies mit einem Zeitschritt t definiert, welcher sich als Ergebnis einer ganzzahligen Division ergibt (s. Zeile 14 in Listing 4.1). Dieses t erhöht sich immer dann, wenn die vorgegebene Menge an Evaluationen erreicht wurde. Im Rahmen der Änderung dient t als direkter Parameter für die in den folgenden Abschnitten beschriebenen Änderungsfunktionen. Die Anzahl der Evaluationen dient dabei als Ausgangsinformation für die Berechnung von t.
Das Gegenstück hierzu stellen die „Babyschritte“ (*baby steps*) dar. Hierbei wird nach jeder einzelnen Evaluation eine Veränderung durchgeführt. Diese Variante eignet sich für Probleme, welche sich stetig ändern, jedoch nur in geringem Umfang. In diesem Fall ist nicht zwingend ein spezieller Zeitschritt nötig, da die Anzahl der bereits erfolgten Evaluationen bestimmt, wie sich das Problem verändern soll.
Soll ein Vergleich eines dynamischen Problems zwischen den Varianten mit Riesen- und Babyschritten durchgeführt werden, so ist darauf zu achten, die Parameter passend einzustellen. Bei Durchführung einer Änderung mit Riesenschritten muss zu dieser Anzahl an Evaluationen auch die Variante mit Babyschritten das gleiche Problem darstellen, um eine Vergleichbarkeit sicherzustellen.

Listing 4.1: Beispiel Schrittart und Schrittweite

```
1  #Parameter bei der Definition des Problems übergeben
2  parameters.put("steps",0);       #0 = giant steps; 1 = baby steps
3  parameters.put("stepsize",1500); #Schrittweite = 1500 Evaluationen
4
5  #Auswertung in der Änderungsmethode des Problems
6  int steps_ = (int) (parameters.get("steps"));
7  int stepsize_ = (int) (parameters.get("stepsize"));
8
9  private void dynamicChange(int evaluations)
10 {
11   ...
12   if (steps_ == GIANTSTEPS)
13   {
14     t = (evaluations / stepsize_);
15   }
16   else
17   {
18     ...
19   }
20   ...
21 }
```

4.3.1 Änderung der Anzahl an Variablen

Die Anforderung der dynamischen Skalierbarkeit der Anzahl an Variablen war die erste Anforderung an das dynamische Distanzminimierungsproblem. Das Hauptproblem, welches hier bei der Umsetzung auftrat war, dass die bestehenden Frameworks und Algorithmen nicht darauf ausgelegt sind auf eine solche dynamische Veränderung einzugehen. Die Erstellung eines Problems mit einer solchen Dynamik hätte damit mit den bestehenden Möglichkeiten nicht zu einem auswertbaren Ergebnis geführt. Eine eigene Erstellung eines solchen Frameworks und die Anpassung bestehender Algorithmen an diese Dynamik hätte den Rahmen der Arbeit bei weitem überschritten. Daher wurde für diese Anforderung eine Zwischenlösung geschaffen, welche die geforderte Dynamik in die bestehenden Frameworks und Algorithmen einbettet. Diese besteht darin, dass zu Beginn der Bearbeitung das Problem mit der maximalen Anzahl an Variablen definiert wird. Damit können die Algorithmen entsprechende Datenstrukturen anlegen, die sie für die Bearbeitung des Problems benötigen. Auch das Problem initialisiert die Zielpunkte entsprechend mit Werten für alle Variablen. Um nun den Effekt der Änderung der Anzahl an Variablen zu erreichen werden die Evaluationsfunktionen angepasst. Diese werten für die Funktionswerte der einzelnen Lösungen nur die aktuell aktiven Variablen aus. Hierzu wird die Funktion zur Berechnung des Abstandes zwischen der Lösung $\vec{y_i}$ und dem Zielpunkt $\overrightarrow{Z_m}$ folgendermaßen definiert:

$$f(\vec{y_i}, \overrightarrow{Z_m}) = \sqrt[p]{\sum_{n=0}^{max(D_{min}, D_{act})-1} (y_{i,n} - Z_{m,n})^p}$$

Die Zielfunktion ist über eine p-Norm, wie in Abschnitt 2.2.1 beschrieben, definiert. In dieser Variante werden jedoch nur die ersten D_{min} bzw. D_{act} Variablen ausgewertet. Das D steht hierbei für Dimensionalität, weil im Distanzminimierungsproblem jede Variable eine Dimension im Entscheidungsraum darstellt. Prinzipiell ist dieses Verfahren aber auch auf andere Probleme anwendbar. Der Wert für D_{act} (für *aktive Dimensionen*) legt hierbei fest, dass die ersten D_{act} Variablen in der Funktion ausgewertet werden sollen. Dieser Wert wird typischerweise durch eine Gleichung bestimmt, wobei das Ergebnis dynamisch von der aktuellen Anzahl an Evaluationen bzw. vom Zeitschritt t abhängig ist. Der Parameter D_{min} legt die Mindestanzahl an auszuwertenden Variablen fest. Hierbei ist anzumerken, dass die Zählung der beiden Variablen bei 0 beginnt und ein $D_{min} = 2$ somit bedeutet, dass die ersten zwei Variablen (0 und 1) ausgewertet werden. Da bestimmte Variablen nicht in das Ergebnis der Funktionswerte einfließen, haben diese auch keinen Einfluss auf die Ordnung der Lösungen untereinander. Somit ist der Effekt gleich dem, welcher auftreten würde, wenn es diese Variablen nicht gäbe.
Zusammenfassend lässt sich sagen, dass alle Variablen sowohl von der Probleminstanz als auch von den Algorithmen so behandelt werden, als wären sie vollständig definiert. Die Erfüllung der Anforderung an eine dynamisch skalierbare Anzahl an Variablen wird durch die Anpassung der Zielfunktionen erreicht.

4.3.2 Änderung der Anzahl an Zielpunkten

Auch bei der zweiten Anforderung, der dynamisch skalierbaren Anzahl an Zielpunkten, ergaben sich Einschränkungen durch die bestehenden Frameworks und Algorithmen. So sind diese nicht darauf ausgelegt auf eine sich ändernde Anzahl an Zielpunkten zu reagieren. Zur Umgehung dieser Einschränkung wurde eine ähnliche Lösung gefunden wie bei der Änderung der Anzahl an Variablen. Auch hier wird die maximale Anzahl an Zielpunkten bei der Erstellung des Problems definiert und alle Zielpunkte bestehen während der gesamten Lebenszeit des Problems. Jedoch genügt es nicht die Zielfunktion dahingehend anzupassen, dass nur die aktiven Zielpunkte beachtet werden. Dies würde dazu führen, dass die entsprechenden Zielpunkte nicht in die Evaluation der Lösungen einbezogen werden, jedoch weiterhin im Entscheidungsraum definiert sind. Da sich die pareto-optimale Menge und damit die pareto-optimale Front jedoch aus der Position aller Zielpunkte zueinander definiert, ergäbe sich hier eine Diskrepanz zwischen der Darstellung im Entscheidungsraum und der optimalen Lösungsmenge. In Abbildung 4.10 wird dies beispielhaft dargestellt. Das Problem selbst besitzt vier Zielpunkte und die Bestimmung der Distanzen erfolgt über das euklidische Abstandsmaß. Würde die Zielfunktion dahingehend geändert, dass der Zielpunkt $\overrightarrow{Z_4}$ nicht betrachtet wird ergäbe sich der pareto-optimale Bereich in der dargestellten Weise. Der Algorithmus würde daher nur diesen Bereich als optimal für seine Lösungen ansehen, weil er nur die Ergebnisse der Evaluationen durch die Zielfunktion zur Verfügung hat. Als Resultat würden die gefundenen Lösungen im besten Fall nur im dargestellten Bereich liegen. Auf einen Betrachter, der keine Kenntnis über diesen Bereich und die Einschränkungen durch die Zielfunktion hat, wirkt dieses Ergebnis jedoch als nicht-optimal, da der Bereich des Dreiecks $(\overrightarrow{Z_1}, \overrightarrow{Z_3}, \overrightarrow{Z_4})$ nicht durch den Algorithmus gefunden wurde.

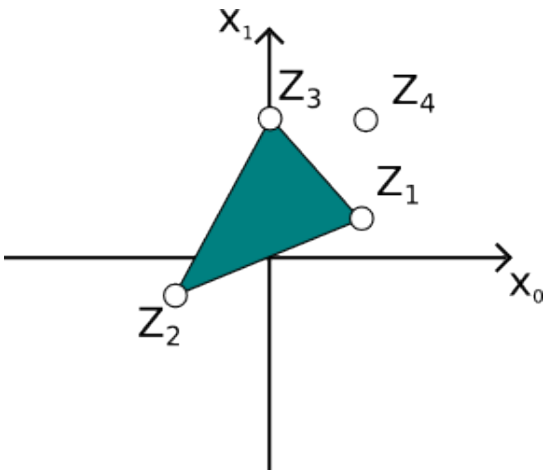

Abbildung 4.10: Pareto-optimale Menge der Zielpunkte bei Nicht-Beachtung der Zielfunktion für Z_4: Diskrepanz zwischen Darstellung im Entscheidungsraum und der pareto-optimalen Menge

Aus diesem Grund wurde eine andere Lösung zur Erreichung dieser Anforderung gewählt. Die aktuell nicht relevanten Zielpunkte werden auf die gleiche Position wie ein aktuell relevanter Zielpunkt gesetzt. Damit ergibt sich der für diesen Fall korrekte pareto-optimale Bereich und damit auch die korrekte pareto-optimale Front (s. Abbildung 4.11 und 4.12).

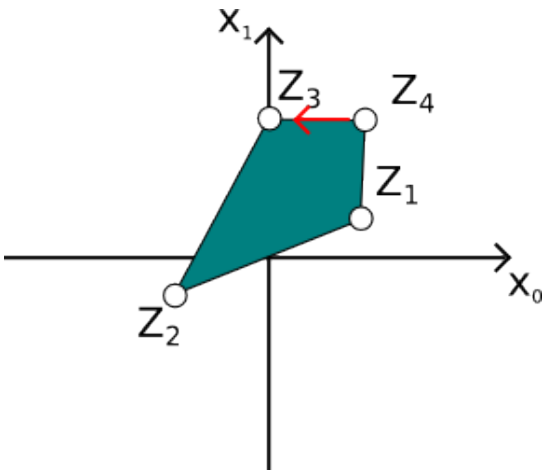

Abbildung 4.11: Verschieben von Punkt Z_4 auf Position von Punkt Z_3

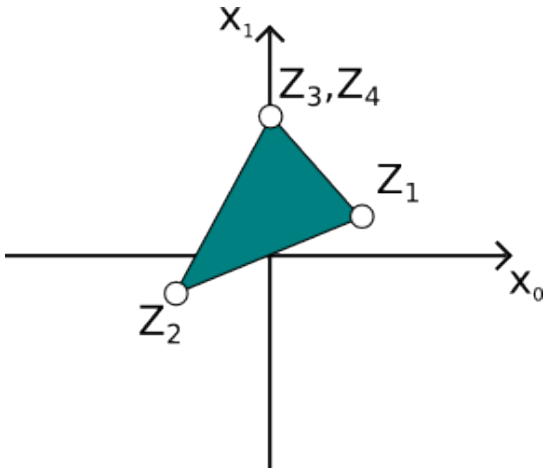

Abbildung 4.12: Übereinstimmung von Darstellung im Entscheidungsraum und pareto-optimaler Menge

Für das Distanzminimierungsproblem stellt diese Variante eine adäquate Lösung dar. Bei anderen Problemen, welche eine Gewichtung der einzelnen Zielpunkte enthalten, ist zu beachten, dass diese für die verdeckten Punkte auf 0 gesetzt bzw. die Zielfunktion entsprechend angepasst werden muss.

4.3.3 Änderung der Lage der Zielpunkte

Die Möglichkeit zur Änderung der Lage einzelner Zielpunkte im Entscheidungsraum stellt die anschaulichste Anforderung an das dynamische Distanzminimierungsproblem dar. Ein Zielpunkt $\vec{Z}$ besteht dabei, wie bereits in 2.2 beschrieben, aus n Komponenten oder Entscheidungsvariablen und ist wie folgt beschrieben:

$$\overrightarrow{Z_m} = \begin{pmatrix} x_{m,0} \\ x_{m,1} \\ \vdots \\ x_{m,n} \end{pmatrix}$$

Für die Anwendung der Änderungsmöglichkeiten ist es zweckmäßig die Zielpunkte und deren einzelne Komponenten in Form von Polarkoordinaten zu definieren:

$$\begin{aligned}
x_{m,0} &= r_m \cdot \cos \alpha_{m,0} \cdot \sin \alpha_{m,1} \cdot \sin \alpha_{m,2} \cdot \ldots \cdot \sin \alpha_{k,n-2} \cdot \sin \alpha_{m,n-1} \\
x_{m,1} &= r_m \cdot \sin \alpha_{m,0} \cdot \sin \alpha_{m,1} \cdot \sin \alpha_{m,2} \cdot \ldots \cdot \sin \alpha_{m,n-2} \cdot \sin \alpha_{m,n-1} \\
x_{m,2} &= r_m \cdot \cos \alpha_{m,1} \cdot \sin \alpha_{m,2} \cdot \ldots \cdot \sin \alpha_{m,n-2} \cdot \sin \alpha_{m,n-1} \\
&\vdots \\
x_{m,n-1} &= r_m \cdot \cos \alpha_{m,n-2} \cdot \sin \alpha_{k,n-1} \\
x_{m,n} &= r_m \cdot \cos \alpha_{m,n-1}
\end{aligned}$$

An den einzelnen Bestandteilen dieser Komponenten können nun auf bestimmte Weise Veränderungen vorgenommen werden. Diese werden nachfolgend einzeln erläutert.

Translation

Die Translation dient zur Verschiebung eines Zielpunktes entlang einer oder mehrerer Koordinatenachsen. Hierfür wird ein Translationsvektor $\vec{z}(t)$ verwendet, welcher die gleiche Dimensionalität wie der Zielpunkt hat. Dieser Vektor kann mittels einer beliebig komplexen Funktion gebildet werden. Ein Beispiel dafür ist:

$$\overrightarrow{z_{m,n}}(t) = t \cdot (n \pmod 2) \Rightarrow \overrightarrow{z_{m,n}}(t) = t \cdot \begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \\ \vdots \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ t \\ 0 \\ t \\ \vdots \\ t \end{pmatrix}$$

Diese Funktion verschiebt den Zielpunkt in jeder zweiten Dimension um t Einheiten in positive Richtung.
Der entstehende Translationvektor kann auf vielfältige Weise (Addition, Subtraktion, Multiplikation etc.) auf den bestehenden Zielpunktvektor angewendet werden. Hier wird als Beispiel die Addition des Translationsvektors durchgeführt:

$$\overrightarrow{Z_m(t)} = \begin{pmatrix} x_{m,0} + z_{m,0}(t) \\ x_{m,1} + z_{m,1}(t) \\ \vdots \\ x_{m,n} + z_{m,n}(t) \end{pmatrix}$$

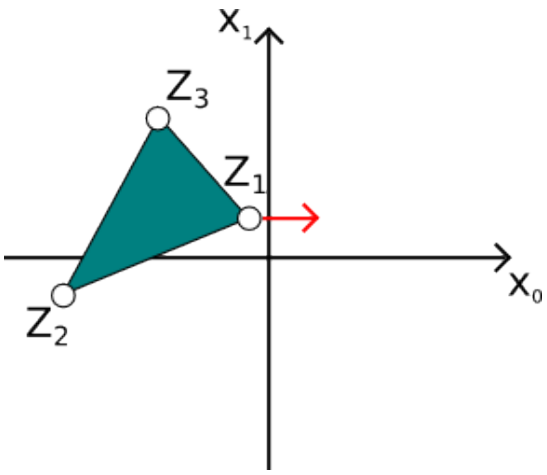

Abbildung 4.13: Translation von Punkt $\vec{Z}_1$ entlang der x_0-Achse

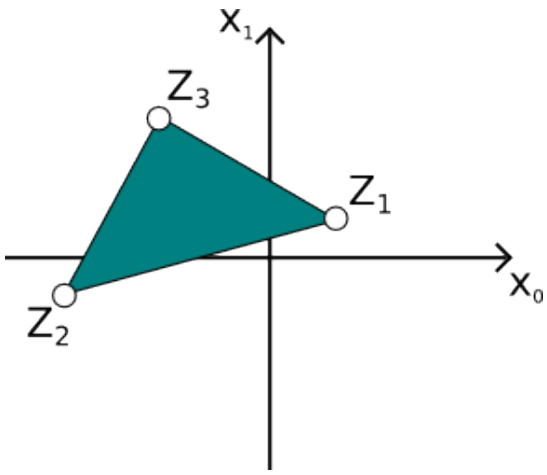

Abbildung 4.14: Ergebnis der Translation

Rotation

Durch die Beschreibung eines Zielpunktes in Polarkoordinaten besitzt jeder Zielpunkt eine Rotationskomponente $\overrightarrow{\alpha_{m,n}}$. Hierbei ist m der Index des Zielpunktes und mit n der Bezug auf die jeweilige Achse gegeben. Die Dimensionalität dieses Vektor ist um eine Komponente geringer als die des Zielpunktes, was sich aus der Definiton über die Polarkoordinaten ergibt. Um eine dynamische Rotation zu erzeugen muss dieser Vektor vom aktuellen Zeitschritt t abhängig sein. Hier kann zur Beschreibung des Vektors wieder eine beliebig komplexe Funktion verwendet werden. Nachfolgende Funktion erhöht den Roationswinkel bezüglich der Achsen jeweils um 30° pro Zeitschritt t:

$$\overrightarrow{\alpha_{m,n}}(t) = (t \cdot (n+1) \cdot \frac{\pi}{6}) \pmod{2\pi} \Rightarrow \overrightarrow{\alpha_{m,n}}(t) = t \cdot \begin{pmatrix} \frac{\pi}{6} \\ \frac{\pi}{3} \\ \vdots \end{pmatrix} = \begin{pmatrix} t \cdot \frac{\pi}{6} \\ t \cdot \frac{\pi}{3} \\ \vdots \end{pmatrix}$$

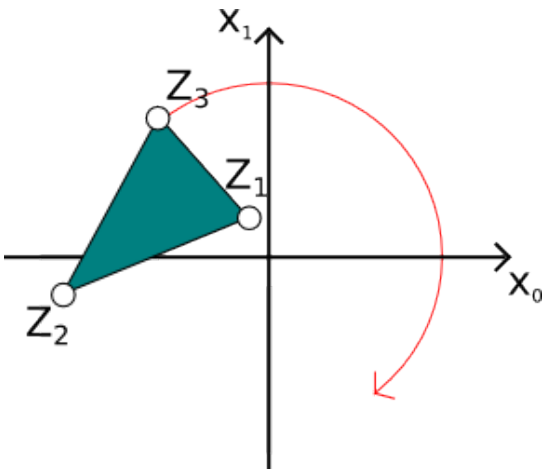

Abbildung 4.15: Rotation von Punkt Z_3

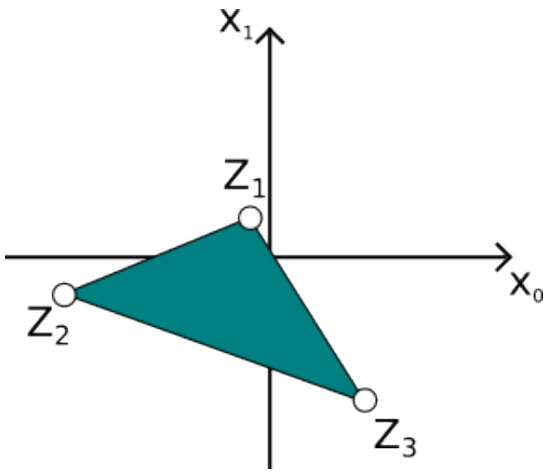

Abbildung 4.16: Ergebnis der Rotation

Skalierung

Neben der Rotationskomponente besitzt ein Zielpunkt in der Polarkoordinatenform auch eine Abstandskomponente r. Hierbei handelt es sich um ein Skalar, welcher den Abstand des Punktes vom Koordinatenursprung definiert. Über eine Änderung von r wird somit der Punkt näher am oder weiter entfernt vom Koordinatenursprung gesetzt. Dies kann beispielsweise zur Vergrößerung und Verkleinerung des pareto-optimalen Bereiches genutzt werden, um die Reaktion von Algorithmen auf diese Art der Veränderung zu untersucht. Hierzu wird r wiederum vom aktuellen Zeitschritt t abhängig definiert. Eine sehr einfache Funktion wäre:

$$r_m(t) = t \cdot m$$

Diese definiert den Abstand abhängig vom Index des Zielpunktes und dem aktuellen Zeitschritt. Der Zielpunkt mit Index 0 liegt dann immer genau auf dem Koordinatenursprung, jener mit dem Index 1 hat einen Abstand von t, jener mit Index 2 einen Abstand von $2t$ etc..

In den Abbildungen 4.17 und 4.18 wird dargestellt, wie sich die Änderung der Abstandskomponente auf den pareto-optimalen Bereich im Entscheidungsraum auswirkt. Die beiden Zielpunkte $\overrightarrow{Z_1}$ und $\overrightarrow{Z_3}$ haben zu Beginn einen Abstand von a, welcher durch eine Skalierung verdoppelt wird.

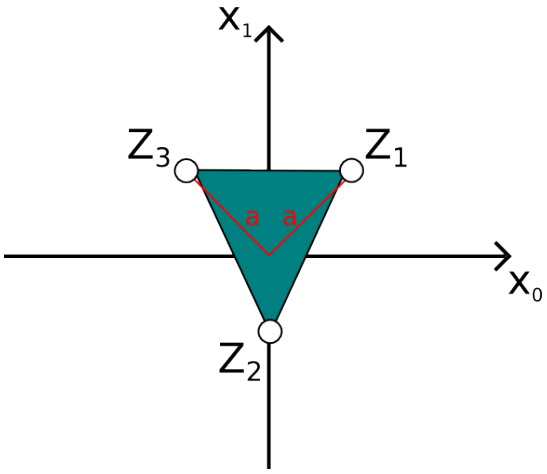

Abbildung 4.17: Vor der Skalierung ($r_1 = r_3 = a$)

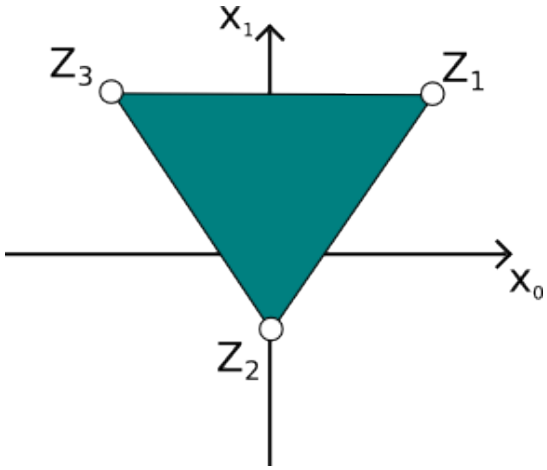

Abbildung 4.18: Nach der Skalierung (Verdoppelung von a)

4.3.4 Änderung der Zielfunktionen

Wie in [5] gezeigt, hat die verwendete Zielfunktion einen großen Einfluss auf die Schwierigkeit eines Problems. Die Definition der Zielfunktion erfolgt für das Distanzminimierungsproblems üblicherweise über die in 2.2.1 beschriebenen p-Normen. Diese bieten die Möglichkeit der Einstellung des Parameters p, durch welchen sich die Zielfunktion und damit auch der pareto-optimale Bereich und die pareto-optimale Front eines Problems erheblich ändern können. Aufgrund dieser Mächtigkeit ist im dynamischen Distanzminimierungsproblem die Möglichkeit gegeben p abhängig vom aktuellen Zeitschritt t zu ändern. Listing 4.2 zeigt hierzu eine Möglichkeit.

Listing 4.2: Beispiel Änderung der Parameters p

```
1 #p wechselt alle 2000 Evaluationen zwischen 1 und 2
2 p = 1 + ((evaluations / 2000) % 2);
```

Dadurch sind Änderungen wie in den Abbildungen 4.7 und 4.8 dargestellt möglich.

4.4 Implementierte Änderungen

Bei der Erstellung des dynamischen Distanzminimierungsproblems wurde die Möglichkeit geschaffen die hier vorgestellten Änderungsmöglichkeiten auf einfache Weise an- und abzuschalten. Hierzu wertet das Problem eine ihm übergebene Parametermenge aus und führt entsprechend Änderungen durch. Listing 4.3 zeigt dieses Vorgehen im Pseudo-Code.

Listing 4.3: Pseudo-Code Parameterprüfung

```
wenn parameterEnthält("AnzahlVariablen")
  ändereAnzahlVariablen();
wenn parameterEnthält("AnzahlZielpunkte")
  ändereAnzahlZielpunkte();
wenn parameterEnthält("Translation")
  führeTranslationAus();
...
```

Die entsprechenden Funktionen müssen im Problem definiert werden. Über die Aktivierung kann jedoch mit dem Setzen oder Nicht-Setzen des entsprechenden Parameters entschieden werden. Neben diesen mitgelieferten Möglichkeiten sind weitere beliebige Änderungen und Erweiterungen durch Anpassung der Änderungsfunktion des dynamischen Distanzminimierungsproblem problemlos möglich.

Kapitel 5

Evaluation

In diesem Kapitel werden die in Kapitel 3 ausgewählten Algorithmen auf ihre Leistungsfähigkeit bezüglich der Lösung von dynamischen und statischen Distanzminimierungsproblemen untersucht. Die Tests in diesem Kapitel ist dabei wie folgt gegliedert:

- Dynamisches DMP mit Summennorm (Abschnitt 5.2.1)
- Dynamisches DMP mit euklidischer Norm (Abschnitt 5.2.2)
- Statisches DMP mit Summennorm und Maximalprinzip (Abschnitt 5.3.1)
- Statisches DMP mit Summennorm und Minimalprinzip (Abschnitt 5.3.2)
- Statisches DMP mit euklidischer Norm und Maximalprinzip (Abschnitt 5.3.3)
- Statisches DMP mit euklidischer Norm und Minimalprinzip (Abschnitt 5.3.4)

Bei den Tests mit dem dynamischen Distanzminimierungsproblem ist anzumerken, dass es im begrenzten Rahmen dieser Arbeit leider nicht möglich war alle Änderungsmöglichkeiten des dynamischen DMP einzeln im Detail zu untersuchen. Zu allen Änderungsmöglichkeiten wurden einzelne Untersuchungen angestellt und aus diesen Erkenntnissen eine begrenzte Menge ausgewählt. Diese soll einen repräsentativen Eindruck über die Leistungsfähigkeit der einzelnen Algorithmen bei der Lösung dynamischer Distanzminimierungsprobleme wiedergeben.
Im Anschluss an die Tests werden die Ergebnisse verglichen und Schlussfolgerungen sowohl über die Leistungsfähigkeit der Algorithmen als auch über den Zusammenhang zwischen Statik, Dynamik und Schwierigkeit eines Problems getroffen.

5.1 Parametereinstellungen für die Algorithmen

Um die in Kapitel 3 vorgestellten Algorithmen zur Problemlösung einsetzen zu können müssen für diese Parameter festgelegt werden. Um eine vergleichbare Anzahl an Lösungen zu erhalten wird für alle Algorithmen die maximale Populationsgröße auf 100 Individuen bzw. Partikeln festgelegt. Für NSGA-II erfolgt die Selektion mittels Binary Tournament und die Kreuzung mittels SBX. Die Kreuzungswahrscheinlichkeit wird mit 90 %

definiert. NSGA-II und SMPSO verwenden polynomiale Mutation mit einer Mutationswahrscheinlichkeit von 1/3. Für SMPSO werden die Werte für C_1 und C_2 zufallsverteilt aus dem Intervall $[1.5, 2.5)$ gewählt. Gleiches gilt für r_1 und r_2, hier jedoch im Intervall $[0, 1)$. Für die dynamischen Varianten DNSGA-II-A und dSMPSO werden die in [45] vorgeschlagenen Werte verwendet. So wird der Anteil an Individuen, deren Funktionswerte auf Veränderungen geprüft werden, mit 10 % festgelegt und der Anteil an zufällig zu verteilenden Lösungen mit 20 %. Für GRA wird der initiale Gitterabstand d auf 1.0 und der Parameter γ abhängig von der Anzahl an Variablen gesetzt. Für 2, 10, 50 und 100 Variablen beträgt der Wert für γ jeweils 1.0, 0.5, 0.2 und 0.05. Die Prüfung auf eine Veränderung der Funktionswerte erfolgt auch hier bei 10 % der Population.

5.2 Test mit dynamischem DMP

Bei den dynamischen Tests erfolgt eine grundsätzliche Unterscheidung nach der Art der verwendeten Abstandsnorm. So gibt es einen Testfall für die Summennorm und einen für die euklidische Norm. Beide Testfälle enthalten Rotation, Skalierung und Translation der Zielpunkte. Beim Testfall der euklidischen Norm wird weiterhin von der Möglichkeit Gebrauch gemacht, die Schwierigkeit über die Anzahl an Variablen zu steigern. So werden hier aus dem grundlegenden Testfall vier Testfälle (mit zwei, zehn, 50 und 100 Variablen) abgeleitet. Bei der Summennorm wird von dieser Möglichkeit Abstand genommen und es werden nur zwei Variablen verwendet. Dies liegt darin begründet, dass in [5] bereits gezeigt wurde, dass es sich auch bei dieser geringen Anzahl an Variablen bereits um ein schweres Problem handelt. Für jeden Testfall werden pro Algorithmus 51 Durchläufe ausgeführt.
Nachfolgend werden die Testfälle mathematisch definiert, grafisch dargestellt und die Ergebnisse vorgestellt.
Als Qualitätsindikatoren für die Leistungsfähigkeit der Algorithmen werden die HVR und die Streuung genutzt. Es wird hierbei bestimmt, welche Werte für diese Indikatoren direkt vor dem Eintreten einer Änderung erreicht wurden. Weiterhin werden die Werte direkt nach dem Eintreten der Änderung ermittelt, um festzustellen wie groß die Auswirkung der Änderung auf die Indikatoren ist. In den Durchläufen hat sich gezeigt, dass die 4.000 Evaluationen zwischen den Änderungen ausreichend groß bemessen sind, sodass es keine Einschwingphase bei der Bearbeitung durch die Algorithmen auftritt, welche die Werte zu Beginn der Bearbeitung im Vergleich zum weiteren Verlauf verfälschen würde. Nach der Änderung arbeiten die Algorithmen daran die neue Probleminstanz zu optimieren. Dafür steht eine bestimmte Anzahl an Evaluationen, genau die Schrittgröße, zur Verfügung. Danach erfolgt die nächste Änderung. Vor dem Durchführen der nächsten Änderung werden wiederum die erreichten Werte für die Qualitätsindikatoren bestimmt. Durch den Vergleich zweier aufeinanderfolgender Werte vor einer Änderung kann erkannt werden, wie gut der durch eine Änderung verursachte Unterschied wieder behoben wurde.

5.2.1 Dynamisches DMP mit Summennorm

Dieser Testfall verwendet drei Zielpunkte $\overrightarrow{Z_1}$, $\overrightarrow{Z_2}$ und $\overrightarrow{Z_3}$. Der Zeitschritt t wird alle 4.000 Evaluationen inkrementiert (Riesenschritte). Insgesamt gibt es acht Änderungen (s. Abbildung 5.1), welche sich zyklisch wiederholen. Damit ergeben sich für die Steuerung der Änderungen eine Schrittgröße von 4000 und ein $t \equiv$ (Evaluationen/Schrittgröße) (mod 8).
Bezüglich Rotation, Skalierung und Translation gelten folgende Festlegungen.

$$\overrightarrow{\alpha_{1,n}}(t) = \begin{cases} \frac{\pi}{4} + t \cdot \frac{\pi}{8} = 45° + t \cdot 22.5° & \text{falls } t < 5 \\ \frac{\pi}{4} + (8-t) \cdot \frac{\pi}{8} = 45° + (8-t) \cdot 22.5° & \text{sonst} \end{cases}$$

$$\overrightarrow{\alpha_{2,n}}(t) = \overrightarrow{a_{1,n}}(t) + \frac{3}{4}\pi = \overrightarrow{a_{1,n}}(t) + 135°$$

$$\overrightarrow{\alpha_{3,n}}(t) = \begin{cases} \frac{3}{2}\pi = 270° & \text{falls } t = 0 \\ \frac{13}{8}\pi = 292.5° & \text{falls } t = 1 \vee t = 7 \\ \frac{7}{4}\pi = 315° & \text{sonst} \end{cases}$$

$$r_1(t) = \begin{cases} \sqrt{10} & \text{falls } t \equiv 1 \text{ (mod 2)} \\ \sqrt{8} & \text{falls } t = 0 \vee t = 4 \\ 4 & \text{sonst} \end{cases}$$

$$r_2(t) = \begin{cases} \sqrt{10} & \text{falls } t \equiv 1 \text{ (mod 2)} \\ 4 & \text{falls } t = 0 \vee t = 4 \\ \sqrt{8} & \text{sonst} \end{cases}$$

$$r_3(t) = \begin{cases} 0 & \text{falls } t = 4 \\ \sqrt{2} & \text{falls } t = 3 \vee t = 5 \\ \sqrt{8} & \text{falls } t = 2 \vee t = 6 \\ \sqrt{10} & \text{falls } t = 1 \vee t = 7 \\ 4 & \text{sonst} \end{cases}$$

$$\overrightarrow{z_{m,n}}(t) = (8-t) \cdot \frac{1}{4}$$

In den nachfolgenden Tabellen 5.1 bis 5.4 werden die Mediane der Werte für HVR und Streuung für die 51 Durchläufe aufgeführt. Dafür wurden zuerst pro Durchlauf alle Werte vor bzw. nach einer Änderung bestimmt. Von diesen Werten wurde dann ein Medianwert für die Werte vor der Änderung ein ein Medianwert für die Werte nach der Änderung gebildet. Die Medianwerte dieser Werte aller Durchläufe sind in den Tabellen 5.1 bis 5.4 dargestellt. Die Werte in Klammern geben den Standardfehler wieder. Die Spalte Δ gibt die Differenz der Mediane vor und nach der Änderung an. Im Falle der HVR steht ein positives Δ für einen Verlust an HVR und damit für eine Verschlechterung des Ergebnisses. Für die Streuung gibt ein positiver Wert eine Verbesserung und ein negativer eine Verschlechterung des Ergebnisses wieder.

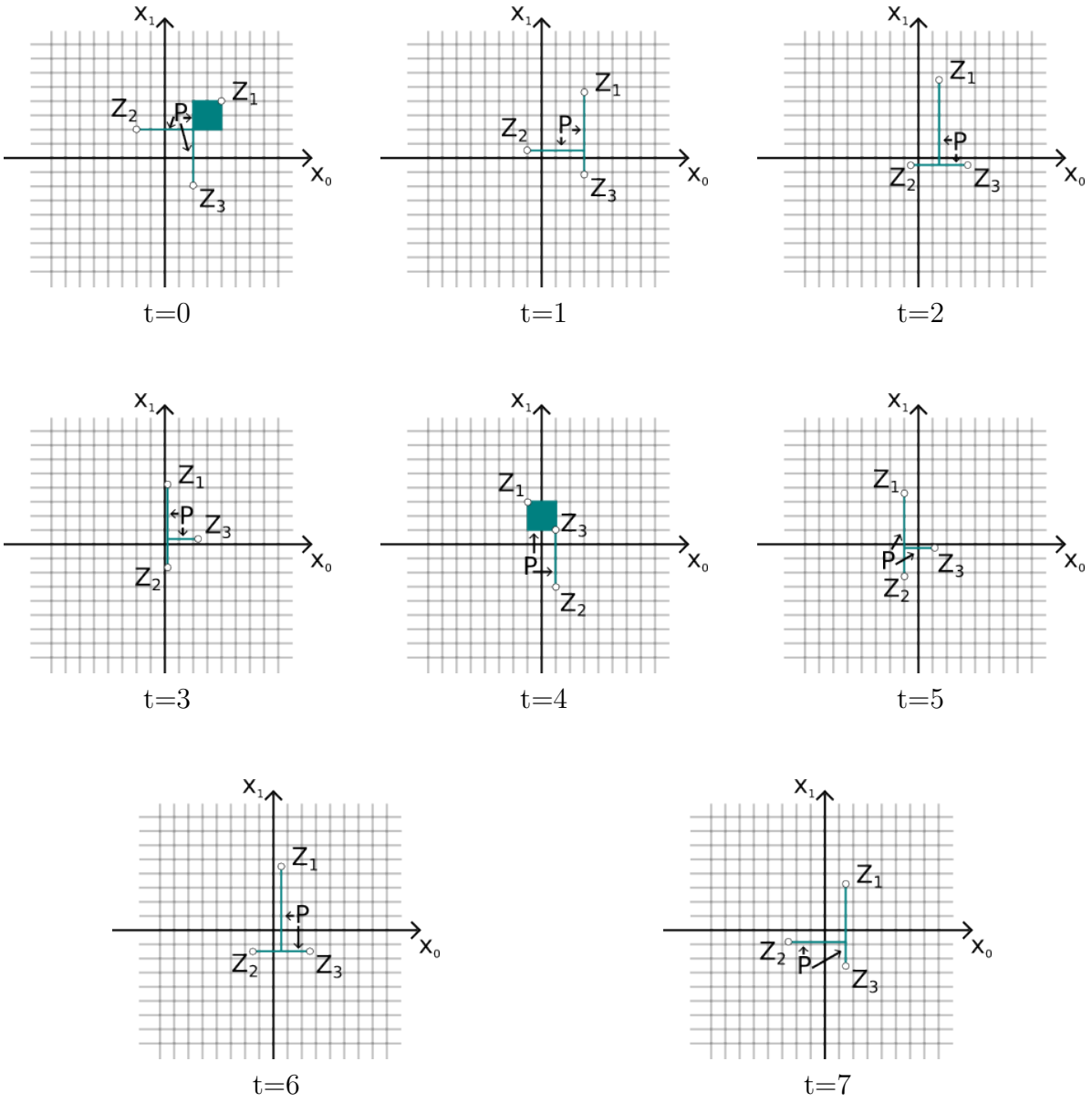

Abbildung 5.1: Dynamischer Testfall für Summennorm

In allen Tabellen, welche Ergebnisse aus Tests beinhalten, wird der beste Wert einer Spalte grün und der schlechteste Wert einer Spalte rot hervorgehoben. Hierbei ist zu beachten, dass nur Ergebnisse mit gleichen Parametern verglichen werden. So findet beispielsweise in Tabelle 5.3 für jede Anzahl an Variablen eine separate Darstellung des besten und des schlechtesten Ergebnisses pro Spalte statt.

Algorithmus	vor Änderung	nach Änderung	Δ
DNSGA-II-A	0.898 (0.013)	0.852 (0.011)	0.046
dSMPSO	0.813 (0.011)	0.802 (0.010)	0.011
GRA	0.982 (0.002)	0.719 (0.011)	0.263

Tabelle 5.1: Dynamischer Testfall - Änderung der HVR bei Summennorm

Wie in Tabelle 5.1 zu erkennen ist, erreicht GRA die besten Werte für HVR vor dem Eintreten einer Änderung. Gleichzeitig verliert GRA durch eine Änderung aber auch am stärksten an HVR. Dies liegt darin begründet, dass dieser Algorithmus die Lösungen im Entscheidungsraum im pareto-optimalen Bereich konzentriert platziert nur in geringem Maß um diesen Bereich herum. Dadurch kommt es bei einer Änderung dazu, dass ein großer Teil der Lösungen nicht mehr im pareto-optimalen Bereich liegt. Da vorher kaum Lösungen um diesen Bereich herum lagen, befinden sich nach der Änderung auch deutlich weniger Lösungen im neuen pareto-optimalen Bereich. Dadurch ergibt sich ein starker Einbruch an HVR. Jedoch ist es GRA möglich den sehr guten Wert vor dem Eintreten der nächsten Änderung wieder zu erreichen, was durch den kleinen Standardfehler deutlich wird.

Algorithmus	vor Änderung	nach Änderung	Δ
DNSGA-II-A	0.808 (0.034)	0.695 (0.017)	0.113
dSMPSO	0.761 (0.011)	0.739 (0.010)	0.022
GRA	1.180 (0.026)	1.083 (0.032)	0.097

Tabelle 5.2: Dynamischer Testfall - Änderung der Streuung bei Summennorm

Anhand der Werte in Tabelle 5.2 wird deutlich, dass ein guter Wert für die Streuung nicht mit einer guten Abdeckung der idealen Front einhergehen muss. So liefert dSMPSO den besten Wert für die Streuung, jedoch den schlechsten für die HVR. Genau umgekehrt verhält es sich mit GRA. Bei diesem Algorithmus wird der beste Wert für die HVR und der schlechteste für die Streuung erreicht.
Die Abbildungen 5.2 und 5.3 zeigen deutlich die Schwankungen für HVR und Streuung, die durch die Dynamik des Problems hervorgerufen werden.

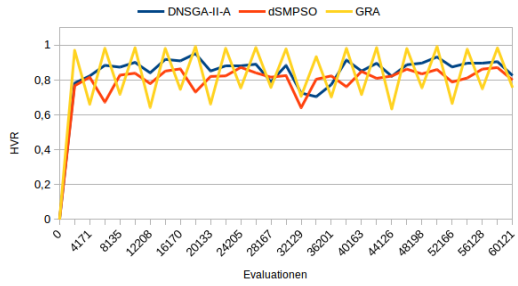

Abbildung 5.2: Median-HVR-Werte vor und nach einer Änderung (Summennorm)

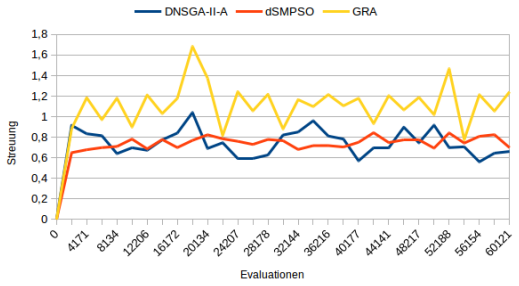

Abbildung 5.3: Medianwerte für die Streuung vor und nach einer Änderung (Summennorm)

5.2.2 Dynamisches DMP mit euklidischer Norm

Auch dieser Testfall verwendet drei Zielpunkte $\overrightarrow{Z_1}$, $\overrightarrow{Z_2}$ und $\overrightarrow{Z_3}$. Der Zeitschritt t wird alle 4.000 Evaluationen inkrementiert (Riesenschritte). Insgesamt gibt es zwölf Änderungen (s. Abbildung 5.4), welche sich zyklisch wiederholen. Damit ergeben sich für die Steuerung der Änderungen eine Schrittgröße von 4000 und ein $t \equiv$ (Evaluationen/Schrittgröße) (mod 12).
Bezüglich Rotation, Skalierung und Translation gelten folgende Festlegungen.

$$\overrightarrow{\alpha_{m,n}}(t) = (m-1) \cdot \frac{2\pi}{3} + t \cdot \frac{\pi}{6} = (m-1) \cdot 120° + t \cdot 30°$$

$$r_m(t) = (5-m) - |(6-t)| \cdot \frac{1}{6}$$

$$\overrightarrow{z_{m,n}}(t) = |6-t| \cdot \frac{1}{2}$$

Variablen	Algorithmus	vor Änderung	nach Änderung	Δ
2	DNSGA-II-A	0.933 (0.001)	0.897 (0.002)	0.036
	dSMPSO	0.729 (0.009)	0.716 (0.009)	0.013
	GRA	0.906 (0.002)	0.805 (0.004)	0.101
10	DNSGA-II-A	0.806 (0.009)	0.377 (0.039)	0.429
	dSMPSO	0.368 (0.024)	0.342 (0.024)	0.026
	GRA	0.760 (0.034)	0.392 (0.034)	0.368
50	DNSGA-II-A	0.028 (0.032)	0.0 (0.007)	0.028
	dSMPSO	0.121 (0.026)	0.084 (0.025)	0.037
	GRA	0.0 (0.029)	0.0 (0.0001)	0.0
100	DNSGA-II-A	0.0 (0.003)	0.0 (0.0)	0.0
	dSMPSO	0.0 (0.028)	0.0 (0.018)	0.0
	GRA	0.0 (0.014)	0.0 (0.0)	0.0

Tabelle 5.3: Dynamischer Testfall - Änderung der HVR bei euklidischer Norm

In allen Messwerten in Tabelle 5.3 ist ein kleiner Standardfehler erkennbar. Dies zeigt an, dass die Werte, welche vor einer Änderung erreicht wurden bis zum Eintritt der nächsten Änderung auch wieder erreicht wurden. Jedoch ist sehr deutlich zu erkennen, dass die Schwierigkeit des Problems mit zwei oder zehn Variablen deutlich unter der des Problems mit 50 oder 100 Variablen liegt. Dies wird auch durch die Darstellung der Entwicklung in den Abbildungen 5.5 bis 5.8 deutlich. Bei wenigen Variablen werden gute Werte für HVR erreicht, welche durch eine Änderung nur leicht absinken. Mit steigender Anzahl an Variablen wird es schwerer gute Werte für HVR zu erreichen und die Änderungen haben ein stärkeres Absinken des Wertes zur Folge. Die ist besonders im Fall von 100 Variablen ist zu erkennen, dass es in den meisten Fällen nicht möglich war einen Wert $HVR > 0$ zu erreichen. Dies kann in der schnellen Änderung des Problems (alle 4.000 Evaluationen) begründet sein, welche nicht ausreichend sein mögen, um eine gute Lösungsmenge zu finden. Um diese Vermutung zu überprüfen erfolgen in den Abschnitten 5.3.1 und 5.3.2 entsprechende statische Tests unter gleichartigen Bedingungen.

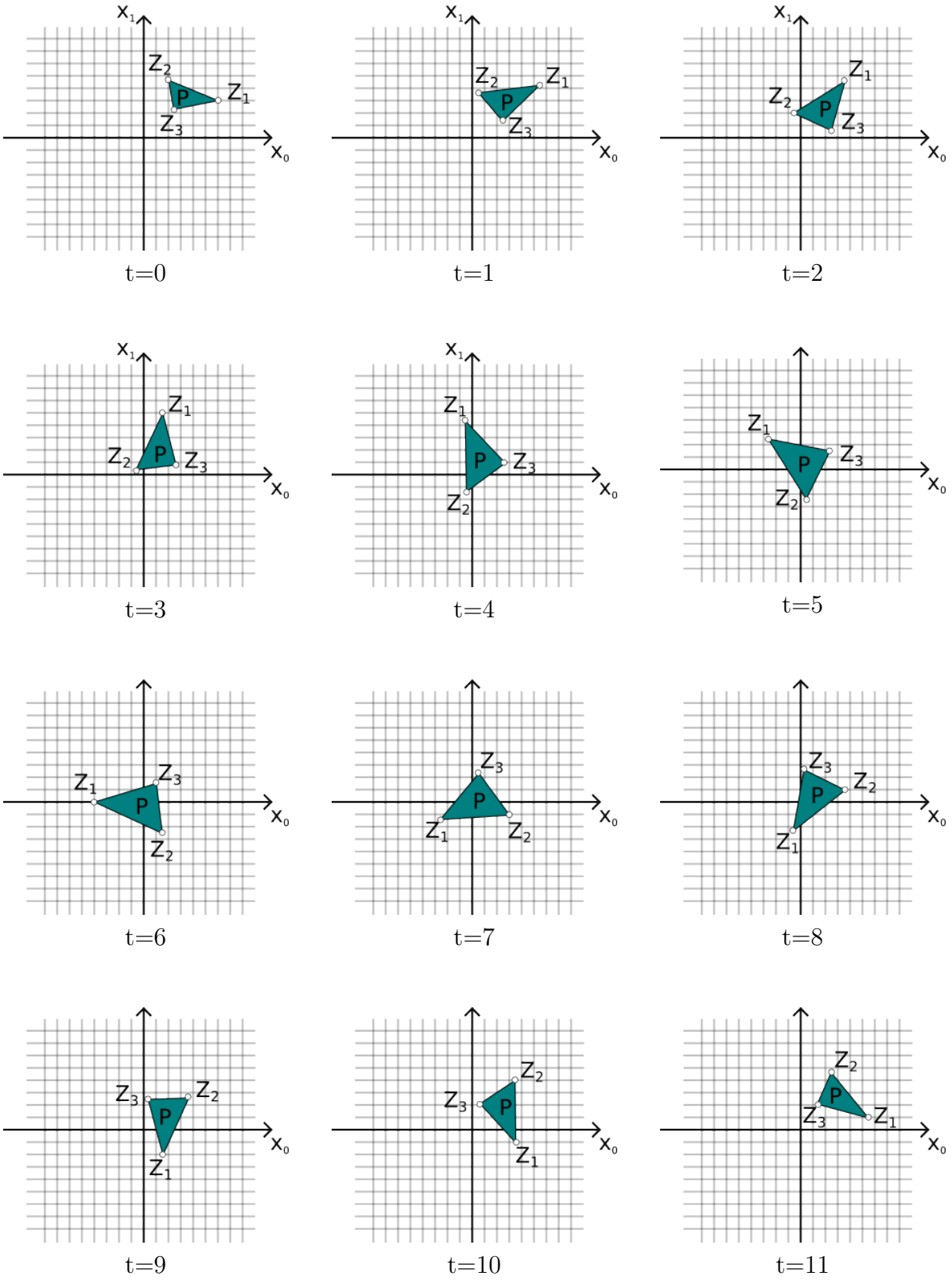

Abbildung 5.4: Dynamischer Testfall für euklidische Norm bei zwei Variablen

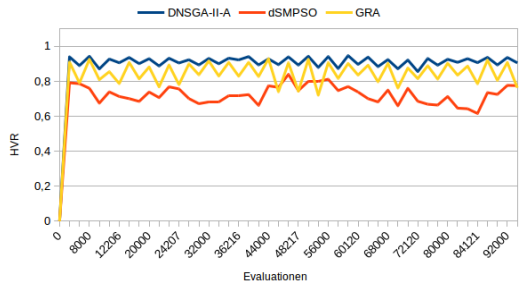

Abbildung 5.5: Median-HVR-Werte vor und nach einer Änderung bei zwei Variablen (euklidische Norm)

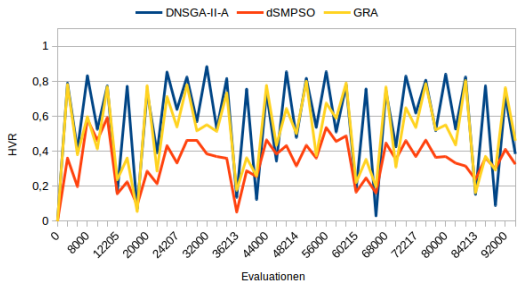

Abbildung 5.6: Median-HVR-Werte vor und nach einer Änderung bei zehn Variablen (euklidische Norm)

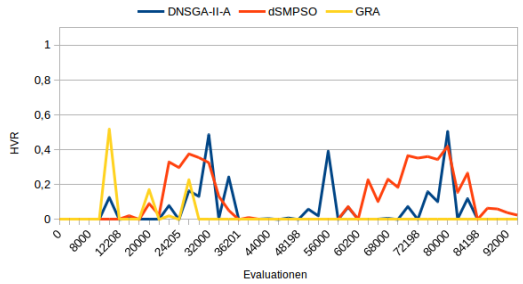

Abbildung 5.7: Median-HVR-Werte vor und nach einer Änderung bei 50 Variablen (euklidische Norm)

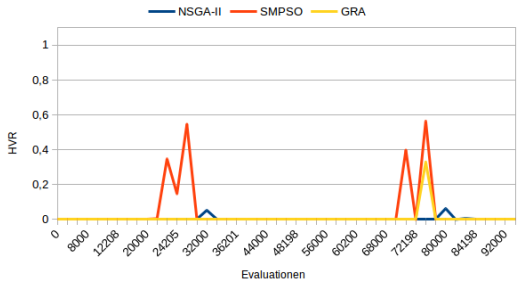

Abbildung 5.8: Median-HVR-Werte vor und nach einer Änderung bei 100 Variablen (euklidische Norm)

Variablen	Algorithmus	vor Änderung	nach Änderung	Δ
2	DNSGA-II-A	0.785 (0.002)	0.679 (0.004)	0.106
	dSMPSO	0.704 (0.011)	0.699 (0.011)	0.005
	GRA	1.029 (0.005)	0.896 (0.007)	0.133
10	DNSGA-II-A	0.735 (0.005)	0.660 (0.014)	0.075
	dSMPSO	0.709 (0.010)	0.708 (0.009)	0.001
	GRA	0.759 (0.036)	0.625 (0.018)	0.134
50	DNSGA-II-A	0.697 (0.009)	0.713 (0.010)	-0.016
	dSMPSO	0.773 (0.011)	0.790 (0.010)	-0.017
	GRA	1.000 (0.032)	0.949 (0.031)	0.051
100	DNSGA-II-A	0.753 (0.011)	0.791 (0.014)	-0.038
	dSMPSO	0.946 (0.012)	0.943 (0.011)	0.003
	GRA	1.000 (0.0)	1.000 (0.0)	0.0

Tabelle 5.4: Dynamischer Testfall - Änderung der Streuung bei euklidischer Norm

Auch bei der Streuung ist, wie aus Tabelle 5.4 zu erkennen, der Standardfehler gering, was auf ein hohes Maß an Gleichartigkeit der Ergebnisse jeweils vor und nach der Änderung hindeutet. Auffällig ist, dass sich die Streuung nach einer Änderung in fast allen Testfällen verbessert hat. Dies liegt möglicherweise darin begründet, dass durch eine Änderung des Problems eine geringere Anzahl an Lösungen nicht-dominiert ist. Bei wenigen Lösungen ist es wiederum wahrscheinlicher eine gleichmäßige Verteilung dieser zu

erreichen als bei vielen Lösungen.
Die Abbildungen 5.9 bis 5.12 zeigen auf, dass die Streuung über die wechselnde Anzahl an Variablen relativ konstant bleibt und für DNSGA-II-A und dSMPSO nur kleine Wertänderungen durch eine Änderung des Problems erfährt. Bei GRA hingegen sind die Schwankungen teilweise sehr deutlich, vor allem bei einer Probleminstanz mit 50 Variablen (s. Abbildung 5.11).

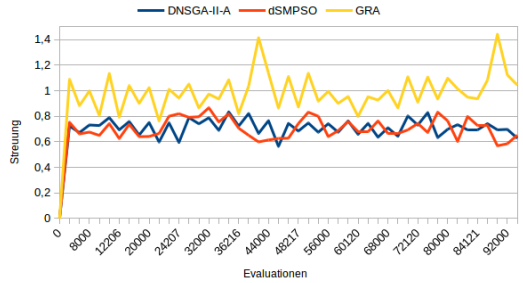

Abbildung 5.9: Streuung vor und nach einer Änderung bei zwei Variablen (euklidische Norm)

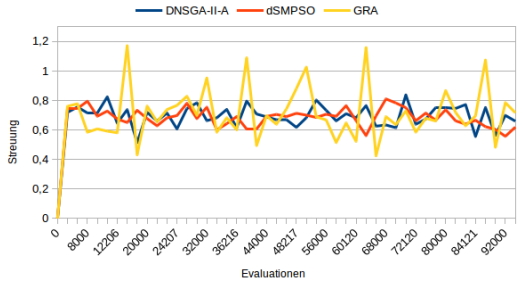

Abbildung 5.10: Streuung vor und nach einer Änderung bei zehn Variablen (euklidische Norm)

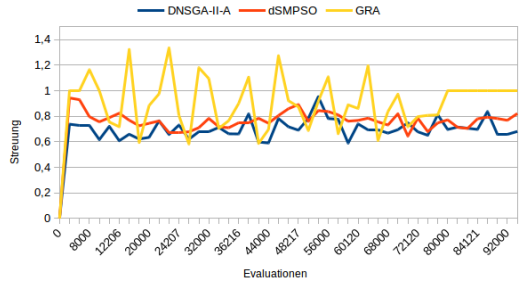

Abbildung 5.11: Streuung vor und nach einer Änderung bei 50 Variablen (euklidische Norm)

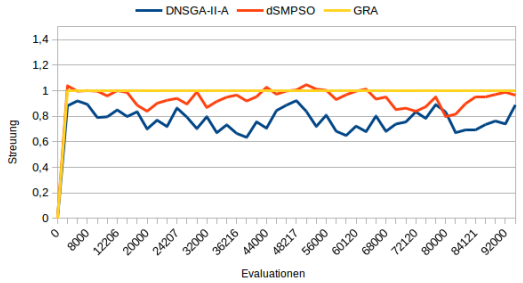

Abbildung 5.12: Streuung vor und nach einer Änderung bei 100 Variablen (euklidische Norm)

5.3 Test mit statischem DMP

In diesem Abschnitt werden NSGA-II, SMPSO und GRA zur Lösung statischer Distanzminimierungprobleme herangezogen. Wiederum werden Summennorm und euklidische Norm in den Testfällen verwendet. Auch die Anzahl an Durchläufen ist mit 51 identisch mit der im dynamischen Fall. Im Unterschied zu den Tests mit dem dynamischen Distanzminimierungsproblem werden hier zwei Zielstellungen unterschieden.
Bei der ersten handelt es sich um die Verwendung des Maximalprinzips. Hier besteht das Ziel darin innerhalb einer fest vorgegebenen Anzahl an Evaluationen bestmögliche Werte für HVR, HVR_{opt}, n_{opt} und Streuung zu erreichen.
Die zweite Zielstellung verwendet das Minimalprinzip. Das Ziel liegt hierbei darin einen bestimmten Schwellwert für HVR bzw. HVR_{opt} in möglichst wenigen Evaluationen zu erreichen.
Für die Summennorm ($p = 1$) werden zwei Testfälle unterschieden (s. Abbildungen 5.13 und 5.14). In beiden Fällen besitzt das Problem zwei Variablen und drei Zielpunkte. Aufgrund der Anordnung der Zielpunkte ergeben sich die entsprechenden pareto-optimalen Lösungsmengen. Im Testfall 1 besitzen die Zielpunkte die Koordinaten $\overrightarrow{Z_1} = (-4, 0)$, $\overrightarrow{Z_2} = (0, -4)$ und $\overrightarrow{Z_3} = (2, 2)$, im Testfall 2 $\overrightarrow{Z_1} = (-4, 0)$, $\overrightarrow{Z_2} = (0, -4)$ und $\overrightarrow{Z_3} = (0, 4)$.

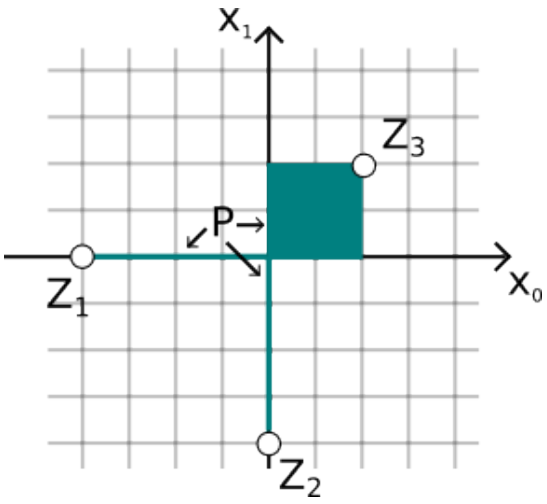

Abbildung 5.13: Testfall 1 für Summennorm

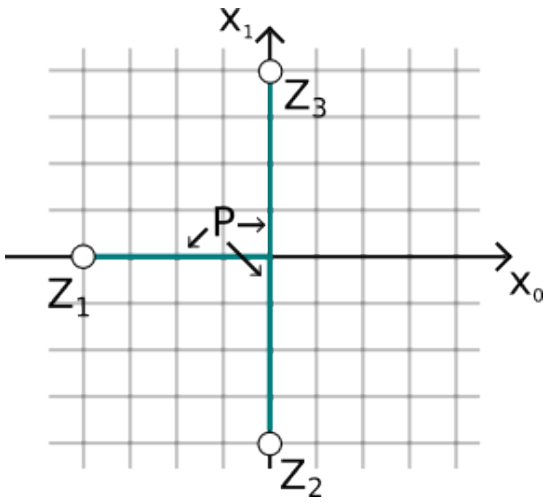

Abbildung 5.14: Testfall 2 für Summennorm

Zu erkennen ist, dass im Testfall 2 die pareto-optimale Menge eine Fläche von 0 besitzt. Dieser Fall sollte daher nach den Erfahrungen aus [5] schwer zu lösen sein.
Für die euklidische Norm ($p = 2$) gibt es vier Testfälle. Diese besitzen einen Aufbau, welcher sich in allen Fällen ähnelt (s. Abbildung 5.15). Die Anzahl der Variablen unterscheidet sich jedoch erheblich zwischen den einzelnen Testfällen. Hierdurch soll eine unterschiedliche Schwierigkeit des Problems erzeugt werden. In allen Fällen besitzt das Problem drei Zielpunkte, die abhängig von der Anzahl der Variablen definiert sind.

$$\vec{Z_1} = \begin{pmatrix} -4 \\ -4 \\ -4 \\ \vdots \\ -4 \end{pmatrix} \qquad \vec{Z_2} = \begin{pmatrix} 4 \\ 4 \\ 4 \\ \vdots \\ 4 \end{pmatrix} \qquad \vec{Z_3} = \begin{pmatrix} 2 \\ -3 \\ 2 \\ \vdots \\ -3 \end{pmatrix}$$

In den einzelnen Testfällen werden zwei, zehn, 50 bzw. 100 Variablen verwendet.

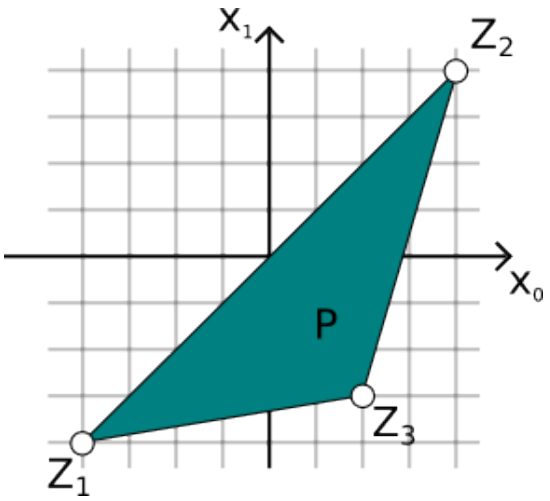

Abbildung 5.15: Statischer Testfall für euklidische Norm mit zwei Variablen

5.3.1 Statisches DMP mit Summennorm und Maximalprinzip

Für diesen Test liegt die Anzahl der maximalen Evaluationen bei 100.000.
In Tabelle 5.5 sind die erreichten Werte der Algorithmen nach dieser Anzahl an Evaluationen aufgeführt. Die Werte vor der Klammer stellen den Median der 51 Durchläufe dar und die Werte in Klammern geben den Standardfehler an.

Testfall	Algorithmus	HVR	HVR_{opt}	n_{opt}	Streuung
1	NSGA-II	0.768 (0.005)	0.500 (0.014)	20 (0.367)	0.840 (0.009)
	SMPSO	0.817 (0.004)	0.672 (0.010)	10 (0.368)	0.689 (0.008)
	GRA	0.970 (0.001)	0.942 (0.003)	67 (0.689)	1.063 (0.009)
2	NSGA-II	0.756 (0.008)	0.271 (0.027)	8 (0.241)	0.724 (0.008)
	SMPSO	0.759 (0.005)	0.586 (0.008)	4 (0.330)	0.653 (0.007)
	GRA	0.966 (0.003)	0.942 (0.003)	68 (0.722)	1.046 (0.010)

Tabelle 5.5: Maximalprinzip für Summennorm

Aus Tabelle 5.5 ist zu erkennen, dass trotz der geringen Anzahl von nur zwei Variablen NSGA-II und SMPSO keine sehr guten Werte für HVR erreichen. Dies bestätigt die in [5] beschriebenen Ergebnisse. Nur GRA ist hier in der Lage gute Ergebnisse zu liefern. Auch für n_{opt} und HVR_{opt} ist dieser Zusammenhang erkennbar. Bei den ermittelten Werten für die Streuung fällt auf, dass diese für alle Algorithmen im (schwierigeren)

Testfall 2 besser sind, was, wie bereits angesprochen, durch die geringere Anzahl an nicht-dominierten Punkten begründet sein könnte.
Wie in den Verläufen von HVR für die beiden Testfälle in den Abbildungen 5.16 und 5.17 zu erkennen ist, schwankt der durch NSGA-II und SMPSO erreichte Wert während der Bearbeitung deutlich, während GRA sein gutes Ergebnis gleichmäßig beibehält. Weiterhin ist zu erkennen, dass GRA sehr schnell einen hohen Wert für HVR erreicht. Daher wird ein Großteil der zur Verfügung stehenden 100.000 Evaluationen nicht zur Verbesserung des Ergebnisses benötigt. Die Prüfung auf die benötigten Evaluationen der Algorithmen ist Gegenstand des nächsten Abschnitts 5.3.2, in welchem das Minimalprinzip zur Anwendung kommt. Auch die Boxplots in den Abbildungen 5.18 und 5.19 zeigen, dass GRA im Falle der Summennorm den beiden anderen Algorithmen überlegen ist. Es ist zu erkennen, dass das schlechteste Ergebnis von GRA besser ist als das beste Ergebnis einer der anderen Algorithmen.

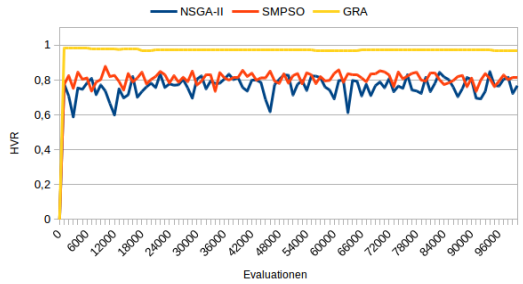

Abbildung 5.16: Verlauf der HVR des Median-Durchlaufs für Testfall 1 (Summennorm)

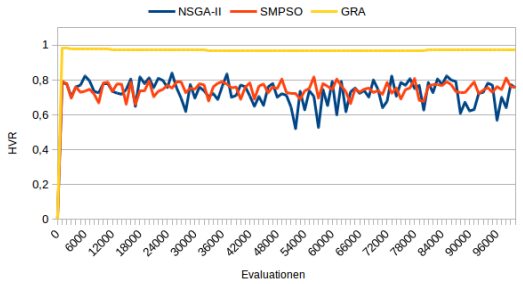

Abbildung 5.17: Verlauf der HVR des Median-Durchlaufs für Testfall 2 (Summennorm)

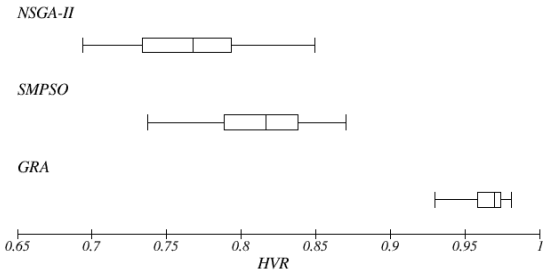

Abbildung 5.18: Boxplot der HVR der 51 Durchläufe für Testfall 1 (Summennorm)

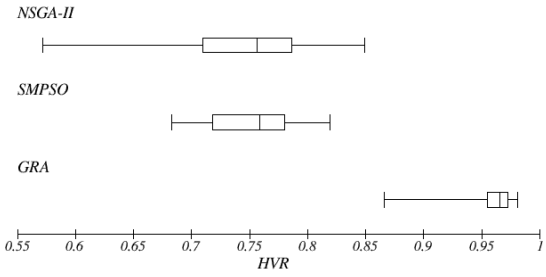

Abbildung 5.19: Boxplot der HVR der 51 Durchläufe für Testfall 2 (Summennorm)

5.3.2 Statisches DMP mit Summennorm und Minimalprinzip

In diesem Abschnitt werden die Ergebnisse bezüglich der Erreichung vorgegebener Werte für HVR und HVR_{opt} dargestellt. Als maximale Anzahl an Evaluationen wird ein Wert von 5.000.000 festgelegt.
In Tabelle 5.6 sind die erreichten Werte der Algorithmen aufgeführt. Die Werte vor der Klammer stellen den Median der 51 Durchläufe dar und die Werte in Klammern geben

den Standardfehler (gerundet auf ganze Evaluationen) an. Falls die vorgegebenen Werte von jeweils 0.8, 0.85 und 0.9 nicht innerhalb von 5.000.000 Evaluationen erreicht werden könnten, so wird der entsprechende Durchlauf als nicht erfolgreich gewertet.

Testfall	Algorithmus	HVR		
		0.8	0.85	0.9
1	NSGA-II	400 (41)	500 (1 790)	- (199 177)
	SMPSO	200 (13)	500 (78)	2 607 300 (208 795)
	GRA	57 (3)	120 (2)	143 (3)
2	NSGA-II	500 (90)	25 700 (4598)	- (-)
	SMPSO	400 (62)	75 200 (18 550)	- (-)
	GRA	62 (3)	123 (3)	144 (3)

Tabelle 5.6: Minimalprinzip für Summennorm - Evaluationen zur Erreichnung einer HVR

Bei den Werten in Tabelle 5.6 ist anzumerken, dass NSGA-II im Testfall 1 nur bei drei der 51 Durchläufe eine $HVR \geq 0.9$ erreicht. Daher wird der Mediandurchlauf als nicht erfolgreich gewertet. Der Standardfehler wurde aus den Werten der drei erfolgreichen Durchläufe gebildet. Bei SMPSO haben im Testfall 1 nur 41 der 51 Durchläufe das Ziel einer $HVR \geq 0.9$ erreicht. Auch hier wurde der Standardfehler nur aus diesen Durchläufen ermittelt. Wie aus den Werten eindeutig hervorgeht erreicht GRA alle vorgegebenen Werte für die HVR und dies auch deutlich schneller als die anderen Algorithmen.

Testfall	Algorithmus	HVR_{opt}		
		0.8	0.85	0.9
1	NSGA-II	1 459 300 (211 220)	- (-)	- (-)
	SMPSO	2400 (696)	259 600 (38 721)	- (-)
	GRA	57 (3)	120 (2)	143 (3)
2	NSGA-II	2 822 300 (460 356)	- (-)	- (-)
	SMPSO	410 100 (67 534)	- (-)	- (-)
	GRA	62 (3)	123 (3)	144 (3)

Tabelle 5.7: Minimalprinzip für Summennorm - Evaluationen zur Erreichnung einer HVR_{opt}

Wie in Tabelle 5.7 zu erkennen ist, stellt die Erreichung eines guten Wertes für HVR_{opt} für NSGA-II und SMPSO ein schwieriges Problem dar. GRA hingegen findet innerhalb weniger Evaluationen gute Lösungen. Im Testfall 2 erreicht NSGA-II nur in 11 von 51 Fällen eine $HVR_{opt} \geq 0.8$. Der Median und die Standardabweichung für diese 11 Durchläufe sind in Tabelle 5.7 angegeben. Dass die ermittelten Werte bei GRA für HVR und HVR_{opt} identisch sind zeigt, dass GRA seine Lösungen im Entscheidungsraum gezielt im pareto-optimalen Bereich platziert, sobald dieser gefunden wurde.

Die folgende Tabelle 5.8 beinhaltet die für die beiden Testfälle erreichten Werte am Ende der 5.000.000 Evaluationen. Diese Anzahl an Evaluationen wurde von NSGA-II und SMPSO durchlaufen, da das Abbruchkriterium $HVR \geq 0.9 \wedge HVR_{opt} \geq 0.9$ nicht erfüllt wurde.

Testfall	Algorithmus	HVR	HVR_{opt}	n_{opt}
1	NSGA-II	0.751 (0.007)	0.508 (0.015)	20 (0.405)
	SMPSO	0.820 (0.004)	0.698 (0.008)	11 (0.378)
2	NSGA-II	0.751 (0.008)	0.357 (0.025)	8 (0.300)
	SMPSO	0.750 (0.006)	0.595 (0.020)	5 (0.305)

Tabelle 5.8: Minimalprinzip für Summennorm - Erreichte Werte am Ende der maximalen Evaluationen

Anhand der Werte in Tabelle 5.8 ist gut erkennbar, dass die durch NSGA-II und SMPSO erreichten Werte sich im Laufe der Bearbeitung wieder verschlechtern können. So liegt beispielsweise SMPSO beim Testfall 1 am Ende deutlich unter dem Wert von 0.9 für die HVR, während im Laufe der Bearbeitung jeder Durchlauf dieses Algorithmus den Wert von 0.9 mindestens einmal erreicht hat. Wie in Tabelle 5.6 gezeigt benötigte der Mediandurchlauf von SMPSO hierfür 2.607.300 Evaluationen.
In den Abbildungen 5.20 und 5.21 wird grafisch dargestellt, nach welcher Anzahl die einzelnen Algorithmen die entsprechenden Schwellwerte für HVR in den beiden Testfällen erreicht haben, wobei NSGA-II und SMPSO deutlich mehr Evaluationen benötigen als GRA. Die Abbildungen 5.22 und 5.23 stellen den Verlauf der HVR des Median-Durchlaufs dar. Hier werden die Schwankungen in den Werten von NSGA-II und SMPSO deutlich. Die Kurve für GRA endet in diesen Abbildungen vorzeitig, da das gesetzte Ziel von $HVR \geq 0.9 \wedge HVR_{opt} \geq 0.9$ erreicht wurde und damit das Abbruchkriterium für den Algorithmus erfüllt war.
Die Boxplots in den Abbildungen 5.24 und 5.25 verdeutlichen wie wenig Evaluationen durch GRA im Vergleich mit NSGA-II und SMPSO benötigt werden. In all diesen Abbildungen zeigt sich deutlich, dass GRA im Falle der Summennorm eine hohe Leistungsfähigkeit besitzt.

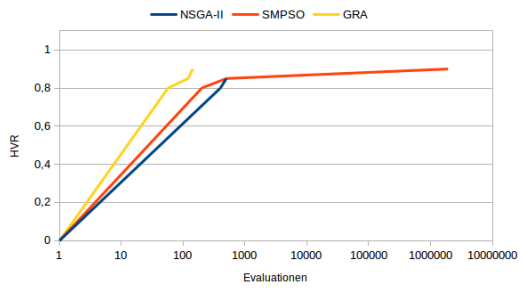

Abbildung 5.20: Erreichung der HVR-Werte von 0.8, 0.85 und 0.9 für Testfall 1 für Median-Durchlauf (log. Darstellung)

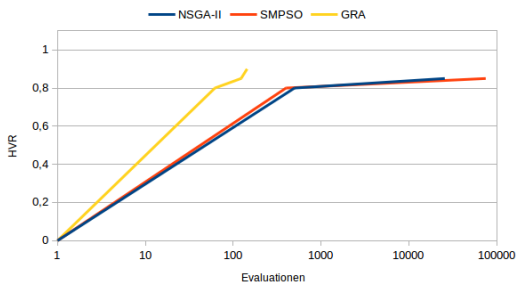

Abbildung 5.21: Erreichung der HVR-Werte von 0.8, 0.85 und 0.9 für Testfall 2 für Median-Durchlauf (log. Darstellung)

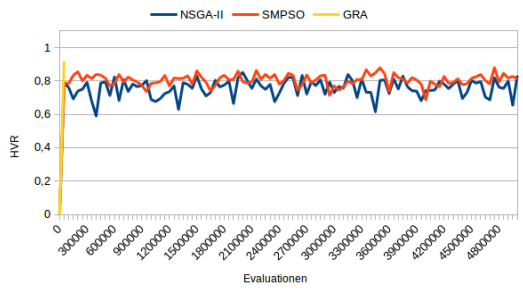

Abbildung 5.22: Verlauf der HVR des Median-Durchlaufs bei Testfall 1

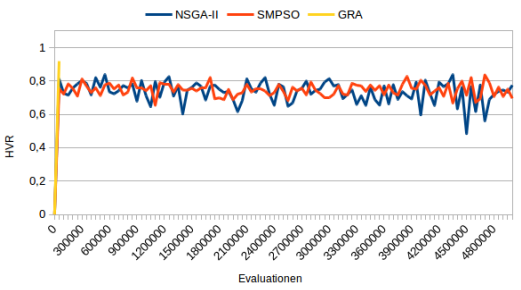

Abbildung 5.23: Verlauf der HVR des Median-Durchlaufs bei Testfall 2

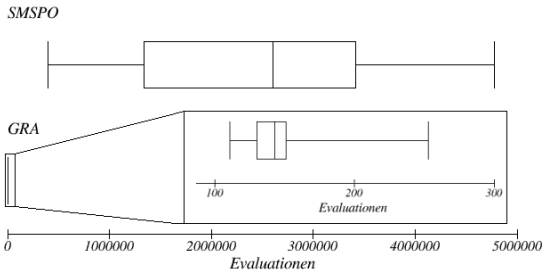

Abbildung 5.24: Boxplot für die Erreichung einer HVR von 0.9 der 51 Durchläufe bei Testfall 1

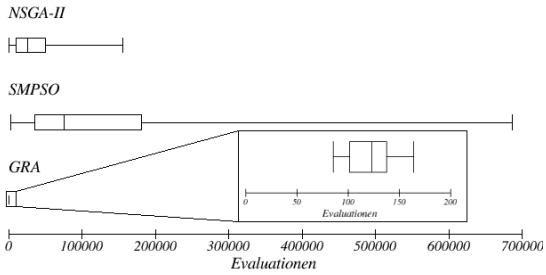

Abbildung 5.25: Boxplot für die Erreichung einer HVR von 0.85 der 51 Durchläufe bei Testfall 2

5.3.3 Statisches DMP mit euklidischer Norm und Maximalprinzip

Für diesen Test liegt die Anzahl der maximalen Evaluationen wiederum bei 100.000. In folgender Tabelle 5.9 sind die erreichten Werte der Algorithmen nach dieser Anzahl an Evaluationen aufgeführt. Die Werte vor der Klammer stellen den Median der 51 Durchläufe dar und die Werte in Klammern geben den Standardfehler an.

Variablen	Algorithmus	HVR	HVR_{opt}	n_{opt}	Streuung
2	NSGA-II	0.957 (0.0005)	0.933 (0.002)	52 (0.840)	0.763 (0.007)
	SMPSO	0.885 (0.003)	0.839 (0.004)	27 (0.495)	0.855 (0.007)
	GRA	0.937 (0.001)	0.908 (0.004)	62 (1.420)	0.820 (0.017)
10	NSGA-II	0.876 (0.001)	0 (0)	0 (0)	0.745 (0.009)
	SMPSO	0.858 (0.004)	0.334 (0.015)	2 (0.213)	0.833 (0.007)
	GRA	0.933 (0.001)	0.436 (0.031)	1 (0.115)	0.703 (0.016)
50	NSGA-II	0.764 (0.001)	0 (0)	0 (0)	0.721 (0.009)
	SMPSO	0.774 (0.005)	0.334 (0.017)	2 (0.157)	0.918 (0.009)
	GRA	0.887 (0.001)	0 (0.009)	0 (0.020)	0.699 (0.018)
100	NSGA-II	0.690 (0.002)	0 (0)	0 (0)	0.789 (0.006)
	SMPSO	0.684 (0.003)	0.334 (0.014)	2 (0.208)	1.027 (0.009)
	GRA	0.788 (0.004)	0 (0)	0 (0)	0.629 (0.018)

Tabelle 5.9: Maximalprinzip für euklidische Norm

Aus den Werten der Tabelle 5.9 und den Abbildungen 5.26 bis 5.29 ist zu erkennen, dass mit steigender Anzahl an Variablen die erreichten Werte erwartungsgemäß schlechter werden. Sehr stark ist dieser Effekt bei n_{opt} und HVR_{opt} zu verzeichnen. Bei der Streuung ist auffällig, dass diese für SMPSO bei steigender Anzahl an Variablen schlechter wird, sich bei GRA jedoch verbessert. Für NSGA-II ist bei der Streuung kein eindeutiger Trend erkennbar. Grundsätzlich sind die Werte für die Streuung sehr gleichmäßig, was durch den kleinen Standardfehler deutlich wird.

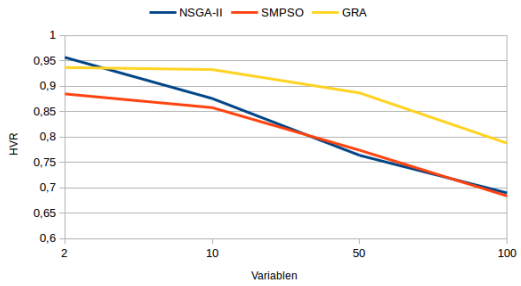

Abbildung 5.26: Entwicklung der Median-HVR bei zunehmender Anzahl an Variablen

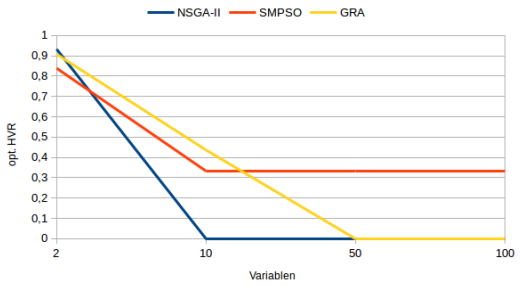

Abbildung 5.27: Entwicklung der Median-HVR_{opt} bei zunehmender Anzahl an Variablen

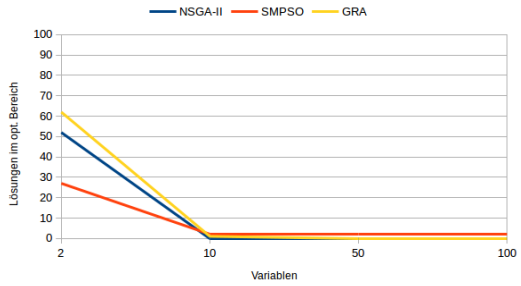

Abbildung 5.28: Entwicklung der Median-n_{opt} bei zunehmender Anzahl an Variablen

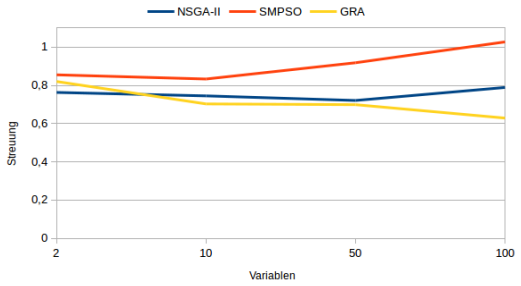

Abbildung 5.29: Entwicklung der Median-Streuung bei zunehmender Anzahl an Variablen

Die Abbildungen 5.30 bis 5.33 zeigen die Entwicklung der HVR für den Median-Durchlauf des jeweiligen Algorithmus. Vor allem in den Abbildungen 5.32 und 5.33 ist zu erkennen, dass der Lösungsansatz von GRA sich von denen der beiden anderen Algorithmen unterscheidet, denn zu Beginn der Bearbeitung steigt für GRA der Wert für HVR deutlich langsamer an. Der Grund hierfür liegt darin, dass GRA nur einen Teil der Variablen untersucht (gesteuert durch den Parameter γ) und sich damit der pareto-optimalen Lösungsmenge langsamer annähert. Die Herangehensweise von SMPSO (Anpassung des nächsten Bewegungsvektors) und NSGA-II (Kreuzung guter Lösungen) bearbeitet im Gegensatz dazu in einem Schritt immer alle Variablen und nähert sich daher deutlich schneller der Lösungsmenge an. Jedoch sorgen diese Ansätze auch dafür, dass ab einem bestimmten Punkt der Bearbeitung keine große Verbesserung der Annäherung an die Lösungsmenge mehr geschieht, was in den Abbildungen 5.32 und 5.33 durch das Abflachen der Kurve deutlich wird. Sobald GRA hingegen die Grenzen der pareto-optimalen

Lösungsmenge ermittelt hat, beginnt eine Verdichtung der Lösungen (durch die Verfeinerung des Gitters) innerhalb dieser Lösungsmenge. Dieses Vorgehen hat zur Folge, dass GRA am Ende der zur Verfügung stehenden Evaluationen in den Testfällen mit mehr als zwei Variablen bessere Ergebnisse erzielt als NSGA-II und SMPSO.

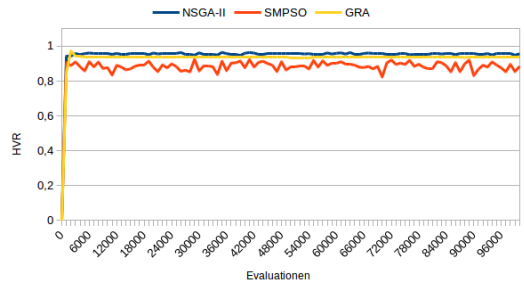

Abbildung 5.30: Verlauf der HVR des Median-Durchlaufs bei zwei Variablen

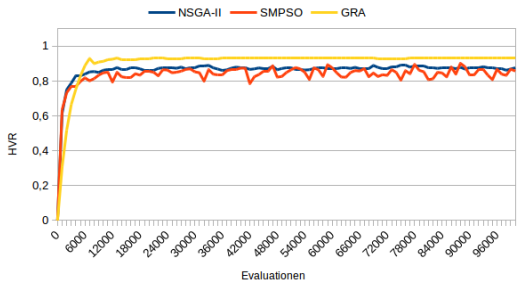

Abbildung 5.31: Verlauf der HVR des Median-Durchlaufs bei zehn Variablen

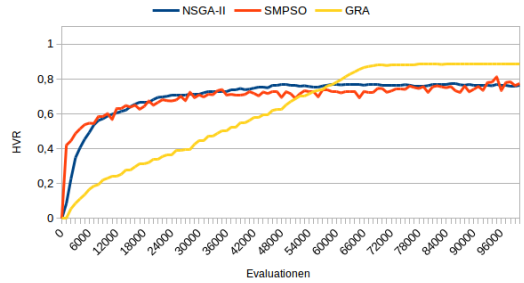

Abbildung 5.32: Verlauf der HVR des Median-Durchlaufs bei 50 Variablen

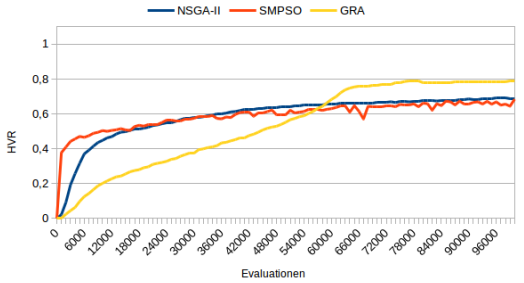

Abbildung 5.33: Verlauf der HVR des Median-Durchlaufs bei 100 Variablen

Die Boxplots der Ergebnisse für alle Durchläufe der Algorithmen in den Abbildungen 5.34 bis 5.37 lassen erkennen, dass sich die Ergebnisse für alle Algorithmen mit zunehmender Anzahl an Variablen verschlechtern. Bei GRA fällt diese Verschlechterung am wenigsten deutlich aus, da durch die Einstellung des Parameters γ direkt die Vorgehensweise in Abhängigkeit von der Anzahl an Variablen beeinflusst werden kann. Dieser Parameter stellt damit einen wichtigen Faktor für die Leistungsfähigkeit des Algorithmus dar.

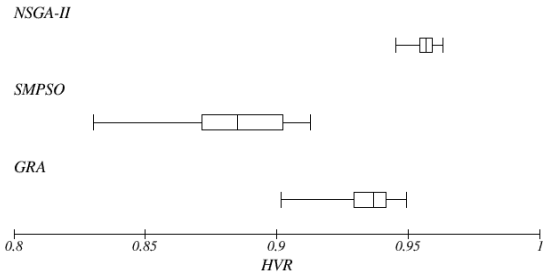

Abbildung 5.34: Boxplot der HVR der 51 Durchläufe bei zwei Variablen

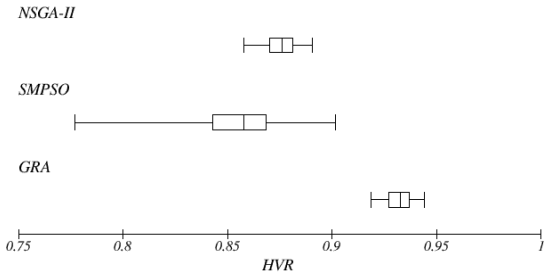

Abbildung 5.35: Boxplot der HVR der 51 Durchläufe bei zehn Variablen

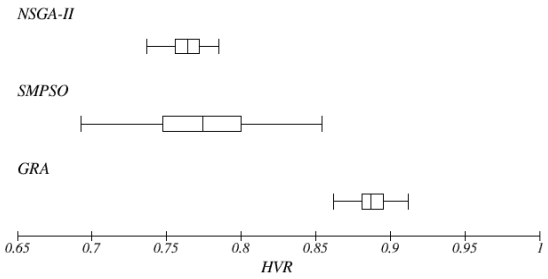

Abbildung 5.36: Boxplot der HVR der 51 Durchläufe bei 50 Variablen

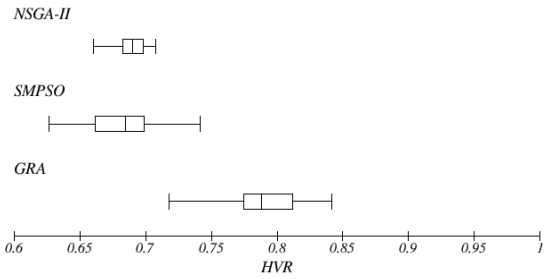

Abbildung 5.37: Boxplot der HVR der 51 Durchläufe bei 100 Variablen

5.3.4 Statisches DMP mit euklidischer Norm und Minimalprinzip

In diesem Abschnitt werden die Ergebnisse bezüglich der Erreichung vorgegebener Werte für HVR und HVR_{opt} dargestellt. Als maximale Anzahl an Evaluationen wird ein Wert von 5.000.000 festgelegt. Aufgrund dieser großen Anzahl an zur Verfügung stehenden Evaluationen wurde der Wert für γ bei GRA im Falle von 100 Variablen auf 0.2 erhöht (s. a. Beschreibung in Abschnitt 3.5).
In Tabelle 5.10 sind die erreichten Werte der Algorithmen aufgeführt. Die Werte vor der Klammer stellen den Median der 51 Durchläufe dar und die Werte in Klammern geben den Standardfehler (gerundet auf ganze Evaluationen) an. Falls die vorgegebenen Werte von jeweils 0.8, 0.85 und 0.9 nicht innerhalb von 5.000.000 Evaluationen erreicht werden konnten, so wurde der entsprechende Durchlauf als nicht erfolgreich gewertet. Zu erkennen ist, dass die benötigte Anzahl an Evaluationen sich mit steigender Anzahl an Variablen erhöht. Dies bestätigt, dass die Schwierigkeit eines Problems durch die Erhöhung der Anzahl an Variablen gesteigert werden kann.

Variablen	Algorithmus	HVR		
		0.8	0.85	0.9
2	NSGA-II	200 (9)	400 (9)	500 (10)
	SMPSO	100 (9)	300 (15)	900 (93)
	GRA	116 (5)	135 (6)	165 (6)
10	NSGA-II	3 400 (91)	6 600 (297)	577 300 (72 169)
	SMPSO	4 700 (190)	10 100 (372)	69 800 (4 506)
	GRA	4 722 (94)	5 263 (100)	6 049 (179)
50	NSGA-II	1 090 500 (61 853)	- (-)	- (-)
	SMPSO	91 900 (5 042)	144 500 (8 121)	274 300 (13 691)
	GRA	63 509 (697)	69 139 (928)	168 768 (52 688)
100	NSGA-II	- (-)	- (-)	- (-)
	SMPSO	404 300 (15 328)	599 900 (18 775)	897 000 (28 173)
	GRA	248 632 (2 827)	268 013 (2 976)	569 018 (104 849)

Tabelle 5.10: Minimalprinzip für euklidische Norm - Evaluationen zur Erreichnung einer HVR

Variablen	Algorithmus	HVR_{opt} 0.8	HVR_{opt} 0.85	HVR_{opt} 0.9
2	NSGA-II	300 (11)	500 (12)	600 (24)
	SMPSO	300 (18)	600 (47)	11 100 (1 956)
	GRA	122 (6)	148 (6)	379 (5)
10	NSGA-II	- (-)	- (-)	- (-)
	SMPSO	- (-)	- (-)	- (-)
	GRA	- (-)	- (-)	- (-)
50	NSGA-II	- (-)	- (-)	- (-)
	SMPSO	- (-)	- (-)	- (-)
	GRA	- (-)	- (-)	- (-)
100	NSGA-II	- (-)	- (-)	- (-)
	SMPSO	- (-)	- (-)	- (-)
	GRA	- (-)	- (-)	- (-)

Tabelle 5.11: Minimalprinzip für euklidische Norm - Evaluationen zur Erreichnung einer HVR_{opt}

Wie aus Tabelle 5.11 zu entnehmen ist, stellt die Erreichung eines guten Wertes für HVR_{opt} ein schwieriges Problem dar. Keiner der Algorithmen konnte einen Wert von 0.8 für mehr als zwei Variablen erreichen. Die Erreichung einer bestimmten HVR gestaltet sich weniger schwierig, wobei auch hier deutlich zu erkennen ist, dass das Problem mit steigender Anzahl an Variablen schwieriger wird.
Die folgende Tabelle beinhaltet die für alle Testfälle mit mehr als zwei Variablen erreichten Werte am Ende der 5.000.000 Evaluationen. Diese Anzahl an Evaluationen wurde von allen Algorithmen durchlaufen, da das Abbruchkriterium $HVR \geq 0.9 \ \wedge \ HVR_{opt} \geq 0.9$ nicht erfüllt wurde.

Variablen	Algorithmus	HVR	HVR_{opt}	n_{opt}
10	NSGA-II	0.880 (0.001)	0 (0)	0 (0)
	SMPSO	0.863 (0.003)	0.333 (0.016)	2 (0.215)
	GRA	0.923 (0.001)	0.436 (0.029)	2 (0.357)
50	NSGA-II	0.784 (0.001)	0 (0)	0 (0)
	SMPSO	0.864 (0.003)	0.333 (0.019)	2 (0.208)
	GRA	0.919 (0.001)	0 (0.009)	0 (0.020)
100	NSGA-II	0.736 (0.001)	0 (0)	0 (0)
	SMPSO	0.859 (0.003)	0.333 (0.020)	2 (0.188)
	GRA	0.914 (0.001)	0 (0)	0 (0)

Tabelle 5.12: Minimalprinzip für euklidische Norm - erreichte Werte am Ende der maximalen Evaluationen

Aus Tabelle 5.12 ist gut erkennbar, dass die durch NSGA-II und SMPSO erreichten Werte sich im Laufe der Bearbeitung wieder verschlechtern können. So liegt beispielsweise SMPSO beim Testfall mit 100 Variablen am Ende deutlich unter dem Wert von 0.9 für die HVR, während im Laufe der Bearbeitung jeder Durchlauf den Wert von 0.9 mindestens einmal erreicht hat. Wie in Tabelle 5.10 gezeigt benötigte der Mediandurchlauf von

SMPSO hierfür 879.000 Evaluationen.
In den Abbildungen 5.38 bis 5.41 sind die im Median benötigten Evaluationen zur Erreichung der HVR-Werte von 0.8, 0.85 und 0.9 aufgezeigt. Hierbei ist zu erkennen, dass NSGA-II mit steigender Anzahl an Variablen deutlich mehr Evaluationen benötigt, um gute Ergebnisse zu erzielen. Bei 50 Variablen kann NSGA-II nur noch einen HVR-Wert von 0.8 erreichen, bei 100 Variablen ist auch dies nicht mehr möglich. SMPSO erreicht für jede Anzahl an Variablen einen Wert von 0.9, benötigt dafür aber immer mehr Evaluationen als GRA.

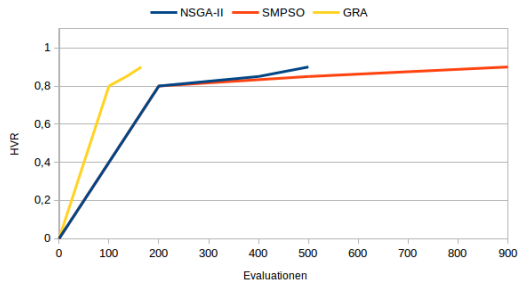

Abbildung 5.38: Erreichung der HVR-Werte von 0.8, 0.85 und 0.9 für Median-Durchlauf bei zwei Variablen

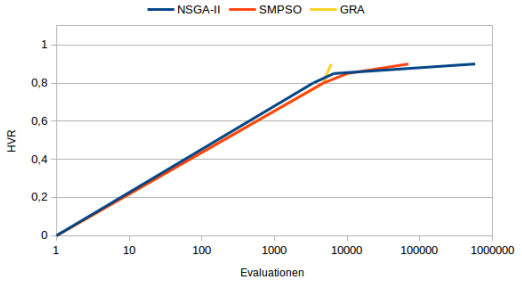

Abbildung 5.39: Erreichung der HVR-Werte von 0.8, 0.85 und 0.9 für Median-Durchlauf bei zehn Variablen (log. Darstellung)

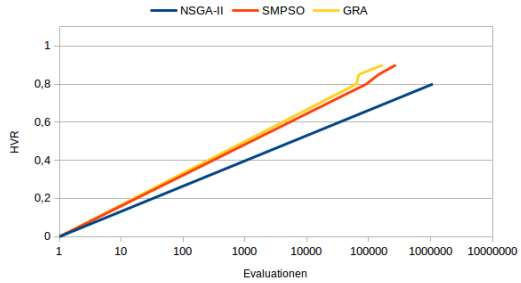

Abbildung 5.40: Erreichung der HVR-Werte von 0.8, 0.85 und 0.9 für Median-Durchlauf bei 50 Variablen (log. Darstellung)

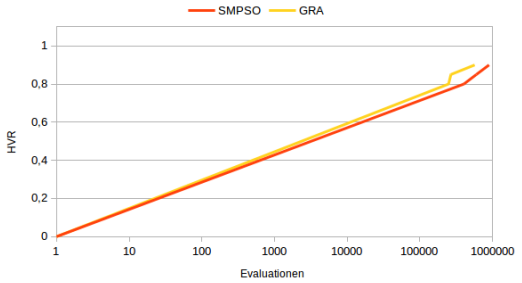

Abbildung 5.41: Erreichung der HVR-Werte von 0.8, 0.85 und 0.9 für Median-Durchlauf bei 100 Variablen (log. Darstellung)

Bei den in den Abbildungen 5.42 bis 5.45 dargestellten Verläufen des Wertes für HVR für die Median-Durchläufe werden wieder die Schwankungen der Werte bei NSGA-II und SMPSO deutlich. Aufgrund der sehr großen Anzahl an Evaluationen ist hier der anfänglich flachere Anstieg bei GRA (im Vergleich zu den Abbildungen 5.32 und 5.33) kaum zu erkennen. Auffällig ist auch wieder die Stabilität des Wertes für GRA im Gegensatz zu den Schwankungen bei NSGA-II und SMPSO.

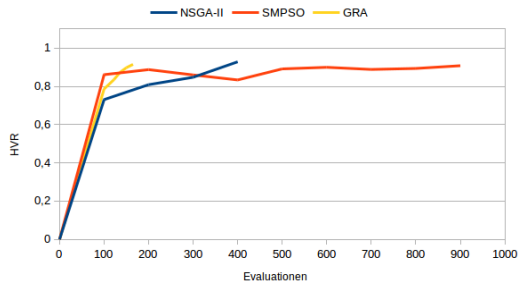

Abbildung 5.42: Verlauf der HVR des Median-Durchlaufs bei zwei Variablen

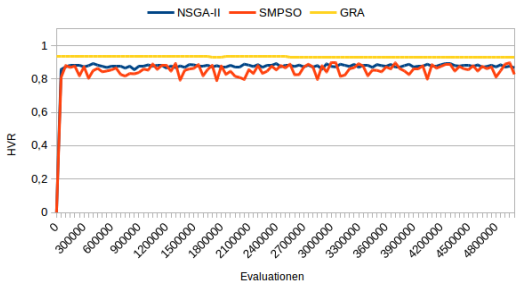

Abbildung 5.43: Verlauf der HVR des Median-Durchlaufs bei zehn Variablen

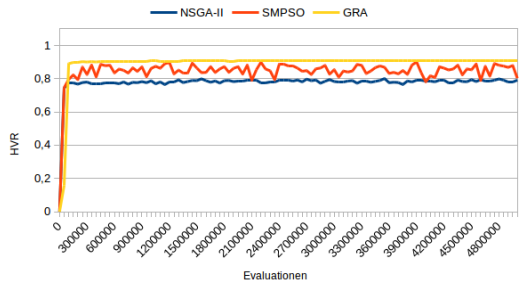

Abbildung 5.44: Verlauf der HVR des Median-Durchlaufs bei 50 Variablen

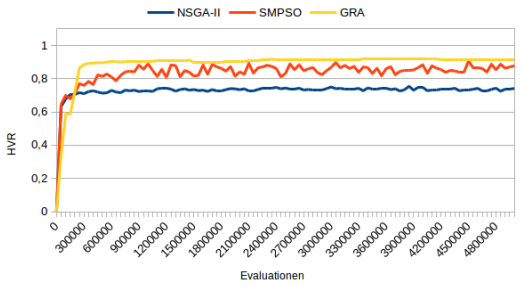

Abbildung 5.45: Verlauf der HVR des Median-Durchlaufs bei 100 Variablen

Die Abbildungen 5.46 bis 5.49 zeigen die Boxplots für alle Durchläufe der Algorithmen mit der jeweiligen Anzahl an Variablen. Hier ist besonders im den Fällen von zwei und zehn Variablen zu erkennen, dass GRA deutlich schneller einen HVR-Wert von 0.9 erreicht als die anderen beiden Algorithmen. In den Abbildungen 5.48 und 5.49 ist zu erkennen, dass GRA im Median schneller die geforderten Werte erreicht, jedoch durch die zufällige Auswahl an zu bearbeitenden Variablen (durch Parameter γ) auch durchaus Durchläufe erzeugen kann, die als Ausreißer deutlich mehr Evaluationen benötigen.

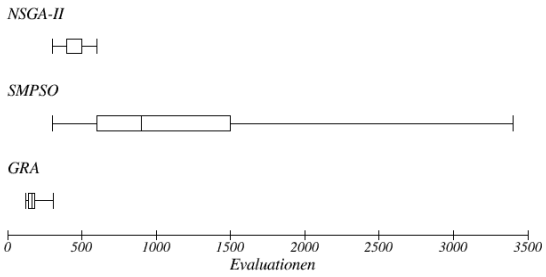

Abbildung 5.46: Boxplot für die Erreichung einer HVR von 0.9 der 51 Durchläufe bei zwei Variablen

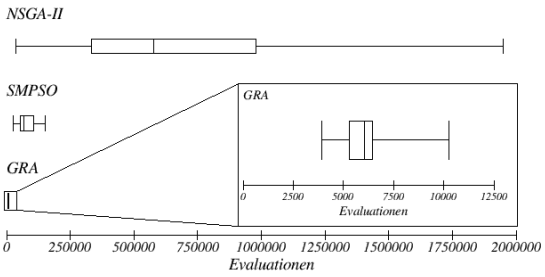

Abbildung 5.47: Boxplot für die Erreichung einer HVR von 0.9 der 51 Durchläufe bei zehn Variablen

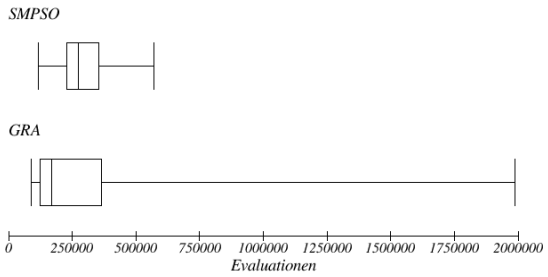

Abbildung 5.48: Boxplot für die Erreichung einer HVR von 0.9 der 51 Durchläufe bei 50 Variablen

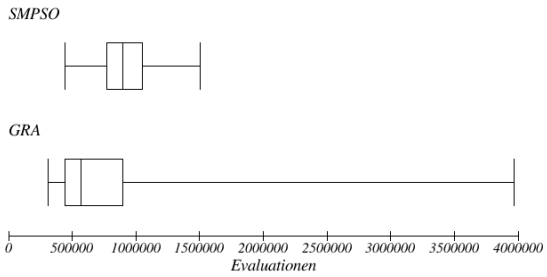

Abbildung 5.49: Boxplot für die Erreichung einer HVR von 0.9 der 51 Durchläufe bei 100 Variablen

5.4 Analyse der Ergebnisse

In diesem Abschnitt werden aus den erfolgten Tests und ihren Ergebnissen Schlussfolgerungen gezogen. Zuerst kann aufgrund der unterschiedlichen Qualität der Lösungen festgestellt werden, dass die Testfälle unterschiedliche Schwierigkeit besitzen. Der statische Testfall mit nur zwei Variablen und euklidischer Metrik ist der einfachste der im Rahmen dieser Arbeit betrachteten Testfälle. Durch eine Erhöhung der Anzahl an Variablen lässt sich erkennen, dass sich dadurch die Schwierigkeit des Problems steigern lässt, wie im statischen Fall mit euklidischer Metrik gut zu sehen ist. Auch die Einführung einer Dynamik in die Testfälle erhöht in gewisser Weise die Schwierigkeit. Durch die Änderung, welche durch eine Dynamik hervorgerufen wird, wird ein statisches Problem durch ein anderes statisches Problem abgelöst. Das Gesamtproblem ist hierbei dynamisch, da es sich verändert, jedoch sind die einzelnen Teilprobleme für die Dauer ihrer Anwesenheit (Schrittgröße) statisch. Die Ergebnisse lassen erkennen, dass dynamische Probleme dann einfach lösbar sind, wenn die entsprechenden statischen (Teil-)Probleme innerhalb der Spanne bis zu nächsten Änderung einfach lösbar sind. Ist dies nicht der Fall, so sind diese Probleme kaum lösbar.
Gut zu erkennen ist dies am Beispiel der Testfälle mit euklidischer Metrik und 50 Variablen. Im statischen Fall nimmt der Wert für HVR für alle Algorithmen am Anfang der Bearbeitung stark zu, erreicht jedoch erst gegen 6.000 Evaluationen für NSGA-II und SMPSO einen Wert von > 0.5. Im dynamischen Fall steht diese Anzahl an Evaluationen jedoch nicht zur Verfügung, weshalb die Werte für HVR deutlich unter 0.2 liegen. Es lässt sich also sagen, dass die Schwierigkeit in der Dynamik eines Problems in der Stärke und der Häufigkeit der Änderung liegt, weshalb die dynamischen Testfälle mit vielen Variablen und der schnellen Änderung die schwierigsten im Rahmen dieser Arbeit getesteten darstellen.
Dadurch lassen sich auch kaum Unterschiede in der Lösungsqualität eines statischen mit dem entsprechenden dynamischen Algorithmus (also z.B. NSGA-II mit DNSGA-II-A) unterscheiden, da der dynamische Algorithmus nur die Erkennung und initiale Reaktion auf die Änderung vornimmt. Das weitere Vorgehen entspricht dann dem statischen Algorithmus mit dem (kurzzeitig) statischen Problem.
Auffällig ist, dass NSGA-II und SMPSO bereits im statischen Fall und bei nur zwei Variablen Schwierigkeiten haben Problem mit Summennorm zu lösen. Dies fällt in der reinen Betrachtung der HVR weniger auf, sondern erst bei Untersuchung von n_{opt} und HVR_{opt}. Im Testfall 1 besitzt der von den drei Zielpunkten umschlossene Bereich eine

Fläche von 6 x 6 Einheiten. In diesem Bereich liegt ein pareto-optimaler Bereich von 2 x 2 Einheiten, d.h. dass $\frac{1}{9}$ (ca. 11 %) der umschlossenen Fläche pareto-optimal sind. Nach 100.000 Evaluationen kann NSGA-II jedoch im Median nur 20 % der Individuen diesem Bereich zuordnen. Bei SMPSO sind es im Median sogar nur 10 %, was ungefähr einer reinen Zufallsverteilung entspräche. Trotz dieser relativ schlechten Werte erhalten die Lösungen einen relative guten Wert für die HVR, was darin begründet liegt, dass auch Lösungen außerhalb des pareto-optimalen Bereichs Hypervolumen erzeugen können (s. Abschnitt 2.5.3). Dies bestätigt die in [5] beschriebenen Erfahrungen bezüglich der Lösungsqualität bei der Verwendung der Summennorm und verdeutlicht den Einfluss der Metrik auf die Schwierigkeit eines Problems.
Weiterhin konnte festgestellt werden, dass eine große Schwierigkeit für alle Algorithmen besteht den pareto-optimalen Bereich im Entscheidungsraum abzudecken bzw. überhaupt zu erreichen. Dies wurde in den hier durchgeführten Tests durch die Qualitätsindikatoren n_{opt} und HVR_{opt} gezeigt. Der Grund hierfür liegt darin, dass der in einem Distanzminimierungsproblem mit euklidischer Metrik durch k Zielpunkte aufgespannte Raum maximal eine Dimensionalität von $k-1$ hat. Damit erzeugen drei Zielpunkte ein Dreieck, welches im zweidimensionalen Raum leicht zu treffen ist. Im dreidimensionalen und höheren Raum ist es jedoch aufgrund seines Volumens von 0 sehr schwer eine Lösung exakt darin zu positionieren. Somit wird das Hypervolumen ausschließlich von Lösungen außerhalb des pareto-optimalen Bereiches gebildet und damit nicht vollkommen ausgeschöpft.

Im Rahmen dieser Arbeit wurde GRA als neuer Algorithmus vorgestellt und es soll nun die Qualität der Ergebnisse dieses Algorithmus mit denen der anderen (NSGA-II und SMPSO bzw. DNSGA-II-A und dSMPSO) verglichen werden. Hierzu werden für jeden Testfall paarweise Vergleiche der Ergebnisse durchgeführt, d.h. das Ergebnis des ersten Durchlaufs von GRA wird mit den Ergebnissen aller Durchläufe von NSGA-II verglichen, danach wird das Ergebnis des zweiten Durchlaufs von GRA mit den Ergebnissen aller Durchläufe von NSGA-II verglichen etc.. Dadurch wird erreicht, dass jeder Durchlauf von GRA mit jedem Durchlauf der anderen Algorithmen verglichen wird. Als Indikator wird aus Gründen der Übersichtlichkeit ausschließlich die HVR betrachtet. Es wird dabei die Frage gestellt, in wie vielen Fällen GRA eine bessere Leistung erbracht hat als der Vergleichsalgorithmus. Die Bewertung, wann ein Ergebnis „besser“ ist als ein anderes, hängt von der Fragestellung des Problems ab. Im Falle des Maximalprinzips ist ein Ergebnis besser, wenn es am Ende der zur Verfügung stehenden Evaluationen einen größeren Wert für HVR erreichen konnte. Im Falle des Minimalprinzips ist ein Ergebnis besser, wenn es zur Erreichung eines bestimmten Schwellwerts weniger Evaluationen benötigte. Im dynamischen Testfall ist ein Ergebnis besser, wenn vor dem Eintreten einer Änderung ein größerer Wert für HVR erreicht wurde. Für jeden Vergleich wird eine Punktzahl von 0, 0.5 oder 1 vergeben. Wird beim Vergleich zweier Ergebnisse ein besserer Wert durch GRA erzielt, so wird dieser Fall mit 1 bewertet, ist das Ergebnis gleich mit 0.5, ist es schlechter mit 0.

Norm	Variablen	Test	Punkte
Summennorm	2	Dynamisch	2 601/2 601 (100 %)
		Statisch (Maximalp.)	5 202/5 202 (100 %)
		Statisch (Minimalp. - $HVR \geq 0.8$)	5 202/5 202 (100 %)
		Statisch (Minimalp. - $HVR \geq 0.85$)	5 202/5 202 (100 %)
		Statisch (Minimalp. - $HVR \geq 0.9$)	5 202/5 202 (100 %)
Euklidische Norm	2	Dynamisch	0/2 601 (0 %)
		Statisch (Maximalp.)	6/2 601 (0.2 %)
		Statisch (Minimalp. - $HVR \geq 0.8$)	2 411/2 601 (92.7 %)
		Statisch (Minimalp. - $HVR \geq 0.85$)	2 581/2 601 (99.2 %)
		Statisch (Minimalp. - $HVR \geq 0.9$)	2 598/2 601 (99.9 %)
Euklidische Norm	10	Dynamisch	0/2 601 (0 %)
		Statisch (Maximalp.)	2 601/2 601 (100 %)
		Statisch (Minimalp. - $HVR \geq 0.8$)	239/2 601 (9.2 %)
		Statisch (Minimalp. - $HVR \geq 0.85$)	2 317/2 601 (89.1 %)
		Statisch (Minimalp. - $HVR \geq 0.9$)	2 601/2 601 (100 %)
Euklidische Norm	50	Dynamisch	563/2 601 (21.6 %)
		Statisch (Maximalp.)	2 601/2 601 (100 %)
		Statisch (Minimalp. - $HVR \geq 0.8$)	2 601/2 601 (100 %)
		Statisch (Minimalp. - $HVR \geq 0.85$)	2 601/2 601 (100 %)
		Statisch (Minimalp. - $HVR \geq 0.9$)	2 601/2 601 (100 %)
Euklidische Norm	100	Dynamisch	1 300.5/2 601 (50 %)
		Statisch (Maximalp.)	2 601/2 601 (100 %)
		Statisch (Minimalp. - $HVR \geq 0.8$)	2 601/2 601 (100 %)
		Statisch (Minimalp. - $HVR \geq 0.85$)	2 601/2 601 (100 %)
		Statisch (Minimalp. - $HVR \geq 0.9$)	2 601/2 601 (100 %)

Tabelle 5.13: Vergleich von GRA mit NSGA-II / DNSGA-II-A

Bei diesem Vergleich zwischen GRA und NSGA-II bzw. DNSGA-II-A ist zu erkennen, dass GRA im Falle der Summennorm eindeutig bessere Ergebnisse erzielt. Dies deckt sich mit den Ergebnissen aus den vorherigen Abschnitten. Auch bei der Erreichung hoher Werte für HVR mittels möglichst weniger Evaluationen (Minimalprinzip) scheint eine Eignung von GRA durchaus gegeben sein. Dies gilt beim Vergleich mit NSGA-II vor allem in Fällen mit ≥ 50 Variablen, da NSGA-II hier deutliche Schwierigkeiten hat gute Werte zu erreichen. In den dynamischen Fällen mit euklidischer Abstandfunktion ist GRA weniger erfolgreich als NSGA-II.

Norm	Variablen	Test	Punkte
Summennorm	2	Dynamisch	2 601/2 601 (100 %)
		Statisch (Maximalp.)	5 202/5 202 (100 %)
		Statisch (Minimalp. - $HVR \geq 0.8$)	5 201/5 202 (99.9 %)
		Statisch (Minimalp. - $HVR \geq 0.85$)	5 202/5 202 (100 %)
		Statisch (Minimalp. - $HVR \geq 0.9$)	5 202/5 202 (100 %)
Euklidische Norm	2	Dynamisch	2 601/2 601 (100 %)
		Statisch (Maximalp.)	2 577/2 601 (99.1 %)
		Statisch (Minimalp. - $HVR \geq 0.8$)	1 419/2 601 (54.6 %)
		Statisch (Minimalp. - $HVR \geq 0.85$)	2 521/2 601 (96.7 %)
		Statisch (Minimalp. - $HVR \geq 0.9$)	2 600/2 601 (99.9 %)
Euklidische Norm	10	Dynamisch	2 601/2 601 (100 %)
		Statisch (Maximalp.)	2 601/2 601 (100 %)
		Statisch (Minimalp. - $HVR \geq 0.8$)	1 493/2 601 (57.4 %)
		Statisch (Minimalp. - $HVR \geq 0.85$)	2 570/2 601 (98.8 %)
		Statisch (Minimalp. - $HVR \geq 0.9$)	2 601/2 601 (100 %)
Euklidische Norm	50	Dynamisch	65.5/2 601 (2.5 %)
		Statisch (Maximalp.)	2 601/2 601 (100 %)
		Statisch (Minimalp. - $HVR \geq 0.8$)	2 119/2 601 (84.5 %)
		Statisch (Minimalp. - $HVR \geq 0.85$)	2 466/2 601 (94.8 %)
		Statisch (Minimalp. - $HVR \geq 0.9$)	1 719/2 601 (66.1 %)
Euklidische Norm	100	Dynamisch	1 300.5/2 601 (50 %)
		Statisch (Maximalp.)	2 599/2 601 (99.9 %)
		Statisch (Minimalp. - $HVR \geq 0.8$)	2 403/2 601 (92.4 %)
		Statisch (Minimalp. - $HVR \geq 0.85$)	2 600/2 601 (99.9 %)
		Statisch (Minimalp. - $HVR \geq 0.9$)	1 851/2 601 (71.2 %)

Tabelle 5.14: Vergleich von GRA mit SMPSO / dSMPSO

Beim Vergleich zwischen GRA und SMPSO bzw. dSMPSO fällt wiederum auf, dass GRA im Falle der Summennorm eindeutig bessere Ergebnisse erzielt. Auch dies passt mit den Ergebnissen aus den vorherigen Abschnitten zusammen. Bei der Zielsetzung der Erreichung hoher Werte für HVR mit möglichst weniger Evaluationen (Minimalprinzip) wird eine Schwäche von GRA deutlich. Während der Algorithmus im Mittel gute Werte erzielt, gibt es auch Ausreißer, welche deutlich mehr Evaluationen benötigen als der Durchschnitt (s. auch Abbildung 5.49) und damit die Wertung verschlechtern. Bei der Anwendung des Maximalprinzips hingegen erreicht GRA im Vergleich zu SMPSO sehr gute Ergebnisse.

Kapitel 6

Zusammenfassung und Ausblick

Diese Arbeit verfolgte drei Ziele. Das erste Ziel war die Ausarbeitung und Vorstellung einer dynamischen Variante des Distanzminimierungsproblems. Für dieses neue Problem wurden im Kapitel 4 die Anforderungen sowie deren Umsetzung beschrieben. Hierbei konnte allen Anforderungen entsprochen werden und das dynamische Distanzminimierungsproblem wurde erfolgreich implementiert. Als zweites Ziel dieser Arbeit sollten die Leistungen dynamischer Algorithmen für die Lösung dieses neuen Problems untersucht werden. Die Beschreibung der Tests und Ergebnisse ist im Abschnitt 5.2 erfolgt, wobei sich zeigte, dass die Algorithmen grundsätzlich in der Lage sind gute Lösungen zu erreichen. Das dritte Ziel war der Vergleich der Ergebnisse der Tests mit dem dynamischen Distanzminimierungsproblem mit den Ergebnissen von Tests mit einem statischen Distanzminimierungsproblem. Dies erfolgte im Abschnitt 5.3, wobei die Ergebnisse zeigten, dass ein Zusammenhang der Schwierigkeit von statischen und dynamischen Problemen über die Häufigkeit und Auswirkung der Dynamik gegeben ist.
Zusätzlich zu diesen erreichten Zielen wurde in dieser Arbeit in Abschnitt 3.5 ein neuer Algorithmus (GRA) vorgestellt, welcher für das Distanzminimierungsproblem gute Lösungen, vor allem bei der Verwendung der Summennorm, erzielt.

6.1 Schlussfolgerung

Abschließend sollen hier die Erkenntnisse, welche im Rahmen dieser Arbeit gewonnen wurden, kurz zusammengefasst werden.
Eine der Fragen, die vor dem Beginn dieser Arbeit bestand, war, welcher Zusammenhang zwischen der Schwierigkeit eines statischen Problems und einem entsprechenden dynamischen Problem besteht. Hierzu kann nun gesagt werden, dass ein dynamisches Problem, welches sich nicht kontinuierlich, sondern schrittweise, ändert, eine Art kurzlebiges statisches Problem darstellt. Die Schwere des Problems ist damit davon abhängig, wie schwierig dieses statische Problem ist (Anzahl an Variablen, Metrik etc.), wie schnell es sich ändert und wie stark es sich ändert. Ist ein Algorithmus in der Lage innerhalb der zur Verfügung stehenden Zeit das statische Problem zu lösen, so sollte er auch in der Lage sein das dynamische Problem zu lösen, da dieses eine Folge von statischen Problemen darstellt.
Die Abdeckung der pareto-optimalen Menge im Entscheidungsraum für das Distanzminimierungsproblem stellt für alle Algorithmen ein schwieriges Problem dar. Dies liegt

daran, dass die Anzahl an Variablen in der Regel größer ist als die Anzahl an Zielpunkten und damit die pareto-optimale Menge ein Volumen von 0 besitzt.
Der in dieser Arbeit vorgestellte Algorithmus GRA stellt eine neue Herangehensweise an multikriterielle Optimierungsprobleme dar und hat gezeigt, dass es durchaus Anwendungsfelder für diese Art des Teile-und-Herrsche-Prinzips in diesem Bereich der Informatik gibt. Hier wurde dies ausschließlich am Distanzminimierungsproblem dargestellt, bei welchem unter Verwendung der Summennorm als Abstandsmaß bessere Lösungen als in [5] erreicht wurden.

6.2 Zukünftige Forschung

Bei der Untersuchung des typischen Verlaufs der Abdeckung an Hypervolumen fällt auf, dass SMPSO schnell gute Werte erreicht, während GRA eine weniger steile Verlaufskurve aufweist (vgl. Abbildung 5.32 und 5.33). Andererseits ist es GRA möglich im späteren Verlauf der Bearbeitung bessere Werte für HVR zu erreichen als SMPSO. Eine Kombination beider Vorgehensweisen könnte hier die Vorteile beider Methoden vereinen und einen schnellen Anstieg mit einem guten Maximalwert verbinden.
Eine grundlegende Untersuchung der Eigenschaften, Fähigkeiten und Grenzen von GRA stellt ein weiteres mögliches Forschungsobjekt dar. Hierbei ist die Einsetzbarkeit für andere Probleme als Distanzminimierungsprobleme eine interessante Fragestellung.
Für die Erweiterung des hier vorgestellten dynamischen Distanzminimierungsproblems besteht beispielsweise die Möglichkeit jedem Zielpunkte eine eigene Zielfunktion zuzuordnen. Dies hätte einen realen Anwendungsfall, wenn beispielsweise ein Punkt innerhalb einer Stadt gefunden werden soll, der einen minimalen Abstand zu bestimmten Gebäuden (Summennorm) und Funkeinrichtungen (euklidische Norm) hat. Auch die Auswertung nur bestimmter Variablen wäre eine mögliche Erweiterung. In der jetzigen Variante werden die Variablen von 0 bis $D_{act} - 1$ ausgewertet, ohne dass eine Möglichkeit besteht eine Variable zu überspringen.
Das Finden von Lösungen innerhalb des pareto-optimalen Bereichs ist wie im vorherigen Abschnitt beschrieben eine schwierige Aufgabe, die jedoch deutlich die Qualität der Ergebnisse verbessern würde und damit ein relevantes Forschungsobjekt für weitere Untersuchungen darstellt.
Bei der Änderung von multikriteriellen Optimierungsproblemen in Wirtschaft und Alltag wirken die Änderungszyklen und -auswirkungen für den Menschen unter Umständen chaotisch und unvorhersehbar. Die Untersuchung dieser Änderungen mittels künstlicher Intelligenz und maschinellem Lernen könnte dazu beitragen Muster und Strukturen darin zu erkennen. Aufgrund dieser Erkenntnisse könnte Änderungen proaktiv begegnet werden und damit Zeit und Geld bei der Suche nach neuen, geänderten optimalen Lösungen gespart werden.

Literaturverzeichnis

[1] J. R. Schott. Fault tolerance design using single and multi-criteria genetic algorithms. Master's thesis, Departement of Aeronautics and Astronautics Massachusetts Institute of Technology, 1995.

[2] K. Deb, S. Agarwal, A. Pratap, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *Tech. Report 200001*, 2000.

[3] A. J. Nebro, J. J. Durillo, J. García-Nieto, C. A. Coello Coello, F. Luna, and E. Alba. SMPSO: A new pso metaheuristic for multi-objective optimization. Technical report, Departamento de Lenguajes y Ciencias de la Computación, University of Málaga, Spain and Department of Computer Science, CINVESTAVIPN, Mexico, 2009.

[4] A. S. Masud C.-L. Hwang. *Multiple objective decision making, methods and applications*. Springer, 1979.

[5] H. Zille and S. Mostaghim. Properties of scalable distance minimization problems using the manhattan metric. *IEEE Congress on Evolutionary Computation (CEC)*, 2015.

[6] T. Bäck. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. Oxford Univ. Press, 1996.

[7] W. Banzhaf, P. Nordin, R. Keller, and F. Francone. *Genetic Programming - An Introduction*. Morgan Kaufmann, San Francisco, 1998.

[8] T. Bäck, D. Fogel, and Z. Michalewicz. *Handbook of Evolutionary Computation*. Oxford Univ. Press, 1997.

[9] M. Clerc and J. Kennedy. The particle swarm - explosion, stability, and convergence in a multidimensional complex space. *IEEE Transactions on Evolutionary Computation*, 2002.

[10] C. W. Reynolds. Flocks, herds, and schools: A distributed behavioral model. In *Computer Graphics, 21(4) (SIGGRAPH '87 Conference Proceeding)*.

[11] J. Kennedy and R.C. Eberhart. Particle swarm optimization. In *Proceedings of IEEE International Conference on Neural Networks*.

[12] K. Deb. *Multi-Objective Optimization Using Evolutionary Algorithms*. John Wiley and Sons, 2001.

[13] E. Zitzler and L. Thiele. Multiobjective evolutionary algorithms; a comparative case study and the strength pareto approach. *IEEE Transactions on Evolutionary Computation*, 1999.

[14] D.A. van Veldhuizen. *Multiobjective evolutionary algorithms: classification, analyses, and new innovations*. PhD thesis, Graduate School of Engineering Air University, 1999.

[15] A. Shirazi, B. Najafi, M. Aminyavari, F. Rinaldi, and R. A. Taylor. Thermal–economic–environmental analysis and multi-objective optimization of an ice thermal energy storage system for gas turbine cycle inlet air cooling. *Energy*, 2014.

[16] B. Najafi, A. Shirazi, M. Aminyavari, F. Rinaldi, and R. A. Taylor. Exergetic, economic and environmental analyses and multi-objective optimization of an sofc-gas turbine hybrid cycle coupled with an msf desalination system. *Desalination*, 2014.

[17] K. Miettinen. *Nonlinear Multiobjective Optimization*. Springer, 1999.

[18] J. Moore and R. Chapman. Application of particle swarm to multiobjective optimization. Technical report, Departement of Computer Science and Software Engineering, Auburn University, 1999.

[19] D. Ashlock. *Evolutionary Computation for Modeling and Optimization*. Springer, 2006.

[20] E. Zitzler, M. Laumanns, and L. Thiele. SPEA2: Improving the strength pareto evolutionary algorithm. Technical report, Computer Engineering and Networks Laboratory (TIK), Swiss Federal Institute of Technology (ETH), Zurich, Switzerland, 2001.

[21] A. J. Nebro, F. Luna, E. Alba, B. Dorronsoro, J.J. Durillo, and A. Beham. AbY-SS: Adapting scatter search to multiobjective optimization. *IEEE Transactions on Evolutionary Computation*, 2008.

[22] M. Reyes Sierra and C. A. Coello Coello. Improving pso-based multiobjective optimization using crowding, mutation and ε-dominance. In *Evolutionary Multi-Criterion Optimization.(EMO 2005)*.

[23] J.J. Durillo, A. J. Nebro, F. Luna, B. Dorronsoro, and E. Alba. jMetal: A java framework for developing multi-objective optimization metaheuristics. Technical report, Departamento de Lenguajes y Ciencias de la Computación, University of Málaga, Spain, 2006.

[24] MOEA framework. `http://moeaframework.org/`. Aufgerufen: 21.10.2016.

[25] Java evolutionary computation toolkit. `http://cs.gmu.edu/~eclab/projects/ecj/`. Aufgerufen: 21.10.2016.

[26] Paradiseo. `http://paradiseo.gforge.inria.fr/`. Aufgerufen: 21.10.2016.

[27] K. Deb. Multiobjective genetic algorithms: Problem difficulties and construction of test problems. *Evolutionary Computation Journal*, 1999.

[28] E. Zitzler, K. Deb, and L. Thiele. Comparison of multiobjective evolutionary algorithms: Empirical results. *Evolutionary Computation Journal*, 2000.

[29] S. Huband, L. Barone, L. While, and P. Hingston. A scalable multi-objective test problem toolkit. In *Evolutionary Multi-Criterion Optimization.(EMO 2005)*.

[30] K. Deb, L. Thiele, M. Laumanns, and E-Zitzler. Scalable multi-objective optimization test problems. In *Evolutionary Computation 2002 (CEC '02) Proceedings of the 2002 Congress*.

[31] M. Farina, K. Deb, and P. Amato. Dynamic multiobjective optimization problems: test cases, approximations and applications. *IEEE Transactions on Evolutionary Computation*, 2003.

[32] J. Mehnen, G. Rudolph, and T. Wagner. Evolutionary optimization of dynamic multiobjective functions. Technical report, Universität Dortmund, Germany, 2006.

[33] C.-K. Goh and K. C. Tan. A competitive-cooperative coevolutionary paradigm for dynamic multiobjective optimization. *IEEE Transactions on Evolutionary Computation 13*, 2009.

[34] L. Huang, I. H. Suh, and A. Abraham. Dynamic multi-objective optimization based on membrane computing for control of time-varying unstable plants. *Information Sciences 181*, 2011.

[35] M. Helbig and A. P. Engelbrecht. Benchmarks for dynamic multi-objective optimisation algorithms. *ACM Computing Surveys, Vol. 46*, 2014.

[36] D.A. van Veldhuizen. *Multiobjective Evolutionary Algorithms: Classifications, Analyses, and New Innovations*. PhD thesis, Graduate School of Engineering, Air Force Institute of Technology, Wright-Patterson AFB, Ohio, USA, 1999.

[37] M. Helbig and A. P. Engelbrecht. Performance measures for dynamic multi-objective optimisation algorithms. *Information Science*, 2013.

[38] M. Köppen and K. Yoshida. *Many-objective particle swarm optimization by gradual leader selection*. Springer, 2007.

[39] H. Ishibuchi, Y. Hitotsuyanagi, N. Tsukamoto, and Y. Nojima. Many-objective test problems to visually examine the behaviour of multiobjective evolution in a decision space. *Parallel Problem Solving from Nature*, 2010.

[40] H. Ishibuchi, N. Akedo, and Y. Nojima. A many-objective test problem for visually examining diversity maintenance behaviour in a decision space. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*.

[41] H. Ishibuchi, M. Yamane, N. Akedo, and Y. Nojima. Many-objective and many-variable test problems for visual examination of multiobjective search. In *Evolutionary Computation 2013*.

[42] O. Schütze, A. Lara, and C. A. Coello Coello. On the influence of the number of objectives on the hardness of a multiobjective optimization problem. In *Evolutionary Computation, IEEE Transactions on*.

[43] K Deb and R. B. Agrawal. Simulated binary crossover for continuous search space. *Complex Systems 9*, 1995.

[44] K. Deb and S. Agrawal. A niched-penalty approachfor constraint handling in genetic algorithms. In *Proceedings of the International Conference on Artificial Neural Networks and Genetic algorithms*.

[45] K. Deb, U. Bhaskara Rao N., , and S. Karthik. Dynamic multi-objective optimization and decision-making using modified NSGA-II: A case study on hydro-thermal power scheduling. Technical report, Kanpur Genetic Algorithms Laboratory (KanGAL) Indian Institute of Technology Kanpur, 2006.

[46] A. J. Nebro, J.J. Durillo, F. Luna, B. Dorronsoro, and E. Alba. Design issues in a multiobjective cellular genetic algorithm. In *Evolutionary Multi-Criterion Optimization. 4th International Conference EMO 2007*.

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Magdeburg, den 14. Dezember 2016

André Kottenhahn