



universität
wien

MASTERARBEIT / MASTER'S THESIS

Titel der Masterarbeit / Title of the Master's Thesis

„Fairness und Deprivation Costs in der Humanitären
Logistik: ein Lösungsansatz mittels
Partikelschwarmoptimierung“

verfasst von / submitted by

Fischer Sophie BSc

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of
Master of Science (MSc)

Wien / Vienna, 2018

Studienkennzahl lt. Studienblatt /
degree programme code as it appears on
the student record sheet:

A 066 915

Studienrichtung lt. Studienblatt /
degree programme as it appears on
the student record sheet:

Masterstudium Betriebswirtschaft

Betreut von / Supervisor:

ao. Univ.-Prof. Mag. Dr. Walter Gutjahr

DANKSAGUNG

Ich danke meinen Eltern dafür, dass sie immer für mich da sind und mich immer unterstützen. Ich bin dankbar, dass sie mir mein Studium und alles andere ermöglicht haben.

Weiters möchte ich mich bei meinen Freunden dafür bedanken, dass sie jeden persönlichen Erfolg mit mir feiern.

Zuletzt möchte ich Professor Gutjahr für die gute Zusammenarbeit danken.

INHALTSVERZEICHNIS

Inhaltsverzeichnis	5
Abbildungsverzeichnis	6
Tabellenverzeichnis	7
1. Einführung	8
2. Fairness und Deprivation Costs in der Humanitären Logistik	9
2.1. Humanitäre Logistik	9
2.2. Fairness und Deprivation Costs	11
2.3. Modell Formulierung	14
3. Partikelschwarmoptimierung	22
3.1. Basisalgorithmus	22
3.2. Varianten des PSO-Algorithmus	29
3.2.1. Kommunikationsstrukturen	29
3.2.2. FIPS	32
4. Anwendungsbeispiel: Nepal Erdbeben 2015	33
4.1. Modellierung	34
4.2. Implementierung	39
4.3. Ergebnisse	39
4.3.1. Kalkulation der Lieferkosten nach Variante 1	39
4.3.2. Kalkulation der Lieferkosten nach Variante 2	45
4.3.3. Weitere experimentelle Analysen	50
5. Fazit	53
Literaturverzeichnis	55
Anhang	57
C++ Code	57
Abstrakt	64
Schlagwörter	64

ABBILDUNGSVERZEICHNIS

Abbildung 1: Beispiel einer Deprivation Intensitätsfunktion für 2 Lieferintervalle [9].	17
Abbildung 2: Bewegung eines Partikels.	25
Abbildung 3: (a) gbest-Topologie, (b) lbest-Topologie [5].	30
Abbildung 4: Epizentren des Hauptbebens (3 Kreise) und des größten Nachbebens (2 Kreise) [9].	34
Abbildung 5: Die 14 Bezirke, die von der Nepalesischen Regierung als am stärksten unterstützungsbedürftig eingestuft wurden [9].	35
Abbildung 6: Grafische Darstellung der Wertepaare aus Tabelle 2 für die verschiedenen λ -Einstellungen (von links nach rechts).	40
Abbildung 7: Grafische Darstellung der Wertepaare aus Tabelle 3 für die verschiedenen λ -Einstellungen (von links nach rechts).	42
Abbildung 8: Grafische Darstellung der Wertepaare aus Tabelle 5 für die verschiedenen λ -Einstellungen (von links nach rechts).	44
Abbildung 9: Gini Index der Deprivation Costs in Abhängigkeit von p für d_k nach Variante 1.	45
Abbildung 10: Grafische Darstellung der Wertepaare aus Tabelle 6 für die verschiedenen λ -Einstellungen (von links nach rechts).	46
Abbildung 11: Grafische Darstellung der Wertepaare aus Tabelle 7 für die verschiedenen λ -Einstellungen (von links nach rechts).	48
Abbildung 12: Grafische Darstellung der Wertepaare aus Tabelle 8 für die verschiedenen λ -Einstellungen (von links nach rechts).	49
Abbildung 13: Global bester Zielfunktionswert in Abhängigkeit der Schwarmgröße für 200 Iterationen.	51
Abbildung 14: CPU-Berechnungszeit in Sekunden in Abhängigkeit der verwendeten Schwarmgröße für 200 Iterationen.	51
Abbildung 15: Global bester Zielfunktionswert in Abhängigkeit der Programmlaufzeit in Sekunden für verschiedene Anzahl an Iterationen zwischen 100 und 1000 (von links nach rechts).	52
Abbildung 16: Global bester Zielfunktionswert in Abhängigkeit der Anzahl an Iterationen.	53

TABELLENVERZEICHNIS

Tabelle 1: Unterstützungsempfänger w_k , Parameter c_k und σ_k und Kosten d_k (2 Varianten) aller Nachfrageknoten $k= 1, \dots, 14$.	37
Tabelle 2: Mittlere Deprivation Costs μ und Gini Index der Deprivation Costs $G(\delta)$ mit Berechnung der Lieferkosten d_k nach Variante 1 für $p = 1$ und alle λ -Werte.	40
Tabelle 3: Mittlere Deprivation Costs μ , Gini Index der Deprivation Costs $G(\delta)$ sowie Gini Index der Lieferhäufigkeiten $G(x)$ mit Berechnung der Lieferkosten d_k nach Variante 1 für $p = 2$ und alle λ -Werte.	41
Tabelle 4: Lieferhäufigkeiten x_k für die 14 Bezirke mit Berechnung der Lieferkosten d_k nach Variante 1 für $p = 2$ und alle λ -Werte mit einem Budget von 1000 je Zeiteinheit.	43
Tabelle 5: Mittlere Deprivation Costs μ und Gini Index der Deprivation Costs $G(\delta)$ mit Berechnung der Lieferkosten d_k nach Variante 1 für $p = 3$ und alle λ -Werte.	44
Tabelle 6: Mittlere Deprivation Costs μ und Gini Index der Deprivation Costs $G(\delta)$ mit Berechnung der Lieferkosten d_k nach Variante 2 für $p = 1$ und alle λ -Werte.	46
Tabelle 7: Mittlere Deprivation Costs μ und Gini Index der Deprivation Costs $G(\delta)$ mit Berechnung der Lieferkosten d_k nach Variante 2 für $p = 2$ und alle λ -Werte.	47
Tabelle 8: Mittlere Deprivation Costs μ und Gini Index der Deprivation Costs $G(\delta)$ mit Berechnung der Lieferkosten d_k nach Variante 2 für $p = 3$ und alle λ -Werte.	49

1. EINFÜHRUNG

In der Humanitären Logistik spielt Fairness eine zentrale Rolle. Bei der nachgelagerten Analyse einiger Hilfsmissionen wurde kritisiert, dass die Hilfsgüter unter den Betroffenen nicht gleichmäßig verteilt wurden. Daraus ergibt sich die Notwendigkeit die bestehenden Forschungen auf dem Gebiet der Humanitären Logistik um die explizite Berücksichtigung des Fairnessaspekts zu erweitern. In der Literatur zu quantitativen Entscheidungshilfen für Hilfsoperationen existieren bereits diverse Ansätze, die den Fairnessaspekt in den Optimierungsmodellen berücksichtigen. In der vorliegenden Arbeit wird das Konzept der Deprivation Costs, das von Holguín-Veras et al. [10] entwickelt wurde, um den Fairnessaspekt erweitert und das sich daraus ergebende nichtkonvexe nichtlineare Optimierungsproblem mittels einer Metaheuristik gelöst. Dafür wurde der Algorithmus der Partikelschwarmoptimierung gewählt, da dieser für derart komplexe Entscheidungsprobleme gut geeignet ist. Das Ziel ist eine fairere Verteilung der Hilfsgüter unter den Unterstützungsbedürftigen nach einer Katastrophe. Anhand eines realen Beispiels wird gezeigt, dass das vorgestellte Optimierungsmodell in der Lage ist, Ungerechtigkeit bei Hilfsoperationen zu verringern und fairere Lösungen zu produzieren.

Die vorliegende Arbeit ist wie folgt strukturiert. Das zweite Kapitel bietet eine Einführung in die Humanitäre Logistik sowie in das Konzept der Deprivation Costs. Anschließend folgt die Modellformulierung des Entscheidungsproblems. Dabei werden sowohl das utilitaristische als auch das um den Fairnessaspekt erweiterte Modell vorgestellt. Im dritten Kapitel wird ein computerbasiertes Lösungsverfahren vorgestellt, das auf die mathematische Re-Formulierung des Entscheidungsmodells und die Partikelschwarmoptimierung-Metaheuristik aufbaut. Anschließend folgt ein Anwendungsbeispiel des Lösungsalgorithmus, für welches reale Daten des großen Erdbebens in Nepal im Jahr 2015 verwendet werden, durch das gezeigt wird, dass das vorgeschlagene Verfahren Lösungen findet, die zugleich effizient und fair sind. Abschließend werden die Ergebnisse der Arbeit analysiert und die wesentlichen Untersuchungsergebnisse zusammengefasst.

2. FAIRNESS UND DEPRIVATION COSTS IN DER HUMANITÄREN LOGISTIK

2.1. HUMANITÄRE LOGISTIK

Humanitäre Logistik bezeichnet die Logistik veranlasst durch unvorhersehbare und oft plötzlich auftretende Ereignisse, die großen Schaden, Zerstörung und menschliches Leiden verursachen und nationale oder internationale Hilfe erfordern. Anwendungsgebiete der Humanitären Logistik sind Armut, Naturkatastrophen, Terror, Epidemien, Hungersnöte, Kriege und andere Krisensituationen [1]. Die Hilfsmissionen können von schnell benötigter Soforthilfe nach einer Katastrophe bis zu langfristigen Projekten zum Wiederaufbau der Infrastruktur variieren [2].

Schlechte Entscheidungen in der Humanitären Logistik können fatale Auswirkungen auf die Katastrophenopfer haben und im schlimmsten Fall über Leben oder Tod entscheiden. Umso wichtiger ist die bestmögliche Planung, Koordination, Durchführung und Kontrolle einer Hilfsoperation. Das Ziel in der Humanitären Logistik ist die zeitgerechte Bereitstellung von Hilfsgütern zur Vermeidung von Krankheiten, Hungernöten und Tod. Dabei geht es vermehrt um die Erfüllung der Grundbedürfnisse, wie die Bereitstellung von Wasser, Nahrung, (Not-)Unterkünften und medizinischer Versorgung [1].

Die Vorbereitung einer Logistikoperation ist im Bereich der Humanitären Logistik erschwert, da die Planung für eine ungewisse Zeit und Situation erfolgt. Auslöser für den Bedarf ist eine Notsituation, deren Umstände vorher nicht genau abgeschätzt werden können. Erschwerend kommt hinzu, dass keine Katastrophe exakt einer anderen gleicht.

Speziell in der Humanitären Logistik gibt es im Vergleich zu anderen Logistik-Anwendungsgebieten viele zusätzliche Herausforderungen. Eine große Herausforderung stellt die Unsicherheit dar, beispielsweise hinsichtlich Verfügbarkeit von Ressourcen, benötigtem Bedarf sowie fehlende Informationen über das Ausmaß des Schadens. Des Weiteren erschwert die Zerstörung der Infrastruktur die Transport-

und Lagerlogistik. Ein weiterer Aspekt sind die häufig fehlenden humanitären Kapazitäten, wie zu wenige Mitarbeiter von Hilfsorganisationen und freiwillige Helfer. Erschwerend kommen ebenfalls begrenztes Kapital und begrenzte Ressourcen hinzu, vor allem da Kapital und Ressourcen meist nicht von Beginn an in vollem Umfang zur Verfügung stehen, sondern erst nach und nach bereitgestellt werden können. Eine besondere Herausforderung stellt auch die Dringlichkeit dar, da in Katastrophensituationen schnell Hilfe benötigt wird. Dadurch müssen Hilfsoperationen eine sehr hohe Reaktionsgeschwindigkeit aufweisen und bringen eine hohe Intensität mit sich.

Ein nicht zu unterschätzender Faktor in der Katastrophenhilfe ist die Möglichkeit, dass die Gefahr fortbesteht, wodurch die Helfenden selbst in Gefahrensituationen geraten können. Beispielsweise seien hier mögliche Nachbeben, Strahlenverseuchung nach Nuklearkatastrophen, Ansteckungsgefahr bei Epidemien und die generelle Gefahr in Kriegsgebieten erwähnt. Anhand dieser Beispiele ist auch deutlich eine Korrelation mit anderen Herausforderungen erkennbar, wie beispielsweise die erhöhte Hemmschwelle für freiwillige Helfer sich Gefahren auszusetzen, die dazu führt, dass die Zahl der Helfer sinkt. Politische Einflüsse können sich ebenfalls erschwerend auf eine Hilfsmission auswirken, da beispielsweise der Druck erhöht wird oder gewisse Beschränkungen der Möglichkeiten die Folge sein können [18].

Partner aus dem Privatsektor können Hilfsorganisationen unterstützen, beispielsweise mit sofort verfügbaren Gütern oder ihren Kenntnissen der lokalen Gegebenheiten. Dies kann entweder pro bono erfolgen oder mit Kosten verbunden sein. Partner aus dem Privatsektor können jedenfalls behilflich dabei sein, die gesamten Kosten der Hilfsmission gering zu halten und die Reaktionsgeschwindigkeit in den ersten Tagen nach einer Katastrophe zu erhöhen.

Da nach Hilfsoperationen immer wieder Ineffizienzen und Kritik öffentlich diskutiert werden, sind eine verstärkte Forschung im Bereich der Humanitären Logistik, Verbesserungen der Prozesse und Mechanismen sowie die Entwicklung neuer Algorithmen zur Optimierung von großer Wichtigkeit. Vorrangig ist dabei, dass die

Effizienzsteigerung von Hilfsmissionen im direkten Verhältnis zu geretteten Menschenleben und der Verminderung menschlichen Leidens steht.

2.2. FAIRNESS UND DEPRIVATION COSTS

In der Humanitären Logistik ist Kostenminimierung meist nicht das wichtigste Ziel, wie es in anderen Transportproblemen oft der Fall ist. Wichtigere Faktoren sind meist eine ausreichende Versorgung der Hilfsbedürftigen und schnelle Hilfe. Eine zentrale Rolle spielt auch Fairness. Beispielsweise sollen Hilfsgüter in der Katastrophenhilfe so fair wie möglich verteilt werden und keine Region soll aufgrund von Abgeschiedenheit oder erschwerten Transportverhältnissen benachteiligt werden. Gründe dafür sind nicht nur ethischer Natur, wie das durchaus intuitive Ziel niemanden zu diskriminieren und gleiches Recht für alle Menschen geltend zu machen, sondern speziell in der Katastrophenhilfe auch Tumulte zu vermeiden, die aufgrund ungerechter Versorgung der Hilfsbedürftigen entstehen können [9]. Der Begriff Fairness wird in der vorliegenden Arbeit verwendet, um eine gerechte Verteilung unter einer Gruppe von Individuen mit gleichen Bedürfnissen auszudrücken. Der Begriff Ungerechtigkeit wird verwendet, um eine unfaire Verteilung und somit das Gegenteil von Fairness auszudrücken.

Der Fairnessaspekt kann in Optimierungsmodellen auf zwei verschiedene Arten berücksichtigt werden. Die erste Möglichkeit ist die Berücksichtigung als Nebenbedingung. Hier wird Fairness in eine Nebenbedingung integriert um ein minimales Level an Fairness (oder auch ein maximales Level an Ungerechtigkeit) sicherzustellen. Die zweite Möglichkeit, Fairness in ein Optimierungsmodell einzubinden, ist die gezielte Maximierung von Fairness bzw. die gezielte Minimierung von Ungerechtigkeit. Hierbei wird Fairness als separates Ziel in einer Mehrzieloptimierung verwendet. Im vorliegenden Entscheidungsproblem wird die letztere Variante verwendet.

Der Begriff Deprivation Costs bezeichnet Kosten der Entbehrung oder des Mangels, wobei hier bewusst der englische Ausdruck verwendet wird, da eine Übersetzung nur unzureichend die Bedeutung des englischen Ausdrucks erfüllt. Deprivation Costs

können definiert werden als eine ökonomische Bewertung menschlichen Leidens aufgrund von mangelndem Zugang zu Gütern oder Dienstleistungen. Ziel ist es, die Deprivation Costs und somit das menschliche Leiden und den Mangel zu minimieren.

In der jüngeren Literatur wurde ein neuer Ansatz für die Optimierung in der Humanitären Logistik gefunden. Holguín-Veras et al. [10] schlagen vor, alle Ziele in der Humanitären Logistik zu einem Kriterium zusammenzufassen und dieses zu minimieren. Das sich dadurch ergebende Kriterium sind die sogenannten Sozialkosten. Die Sozialkosten setzen sich aus den Logistikkosten, also den tatsächlich entstehenden Kosten während einer Hilfsoperation, und den Deprivation Costs zusammen, denen die Hilfsbedürftigen nach einer Katastrophe ausgesetzt sind. Durch die Optimierung der Sozialkosten können sowohl die Kosten als auch das menschliche Leiden minimiert werden und auch der Fairnessaspekt wird implizit berücksichtigt. Dies kann anhand des folgenden Beispiels deutlich gemacht werden: Angenommen, es wird ein Optimierungsproblem behandelt, bei dem einige Versorgungsknoten abgeschieden oder schwer zu erreichen sind. In einem Optimierungsmodell, das rein auf die Maximierung der Versorgung abzielt, würden diese Knoten nur selten besucht werden. Durch eine Optimierung der Sozialkosten, die die Deprivation Costs inkludieren, anstatt der maximalen Versorgung würden diese Nachfrageknoten aufgrund der Konvexität der Zielfunktion häufiger besucht werden. Intuitiv wird angenommen, dass der Fairnessaspekt dadurch ausreichend berücksichtigt wird und eine gezielte Berücksichtigung überflüssig wird. Für das vorliegende Entscheidungsproblem wird der spezielle Fall angenommen, bei dem ein festgelegtes Budget zur Verfügung steht, das vollständig ausgeschöpft wird. Dadurch ist der Term für die Logistikkosten in der Zielfunktion konstant und die Minimierung der Sozialkosten reduziert sich zu der Minimierung der Deprivation Costs. Wie im Folgenden gezeigt wird, genügt die reine Minimierung der Deprivation Costs aber nicht, um ausreichend faire Lösungen zu finden.

Im behandelten Entscheidungsproblem wird Fairness bzw. Ungerechtigkeit mithilfe des Gini Indizes gemessen. Dieser wird insbesondere in der Wohlfahrtsökonomie häufig als Maß zur Darstellung von Ungleichverteilungen verwendet und ist daher für Anwendungen im Bereich der Humanitären Logistik eine gut geeignete Wahl. Der Gini

Index zeigt, dass durch die Minimierung der Deprivation Costs, entgegen der Intuition, beliebig ungerechte Lösungen zustande kommen können. Der Grund dafür liegt in der Krümmung der Zielfunktion. Für dringend benötigte Hilfsgüter, die nicht auf lokalen Märkten verfügbar sind, weist die Zielfunktion eine starke Krümmung auf, was dazu führt, dass selbst durch die Minimierung der Deprivation Costs anstatt der Maximierung der gesamten Versorgung keine fairen Lösungen erreicht werden können. Die später vorgestellten Ergebnisse der Analysen zeigen, dass die Tendenz von Deprivation Cost Minimierungsmodellen faire Lösungen zu produzieren nicht ausreicht um die durch stark gekrümmte Zielfunktionen erhöhte Ungerechtigkeit zu kompensieren. Um dem Fairnessaspekt besser gerecht zu werden, wird deshalb der Gini Index in die Optimierung integriert und nicht nur zur Messung der Fairness verwendet.

Das Bestreben des vorliegenden Optimierungsmodells ist die Minimierung der Deprivation Costs bei Maximierung der Fairness. Modelliert wird dieses Mehrzielproblem durch eine Zielfunktion, die eine gewichtete Summe, bestehend aus der Summe der Deprivation Costs und Gini's mittlerer absoluter Abweichung der Deprivation Costs, minimiert. Ohne die Berücksichtigung des Gini Indizes in der Zielfunktion läge ein utilitaristisches Optimierungsproblem vor, das den Fairnessaspekt außer Acht lässt und einzig und allein darauf abzielt, die größtmögliche Menge an Hilfsgütern an eine größtmögliche Menge von Hilfsbedürftigen zu verteilen. Dieses utilitaristische Modell wird im nächsten Abschnitt ebenfalls vorgestellt.

Die Deprivation Costs sind gegeben durch eine steigende Funktion in Abhängigkeit der vergangenen Zeit seit der letzten Lieferung von Hilfsgütern, die als Deprivation Intensitätsfunktion bezeichnet wird. Holguín-Veras et al. [11] entwickelten 2016 anhand einer empirischen Analyse eine Schätzung der Deprivation Costs, wofür die Zahlungsbereitschaft für lebensnotwendige Güter zu verschiedenen Zeitpunkten nach einer Katastrophe herangezogen wurde. Die Ergebnisse der Studie zeigen, dass eine exponentielle Deprivation Intensitätsfunktion die empirischen Daten am besten widerspiegelt. Durch exponentielle Funktionen wird am besten wiedergegeben, dass der Mangel immer stärker steigt, je länger die Zeit bis zur Versorgung dauert.

2.3. MODELL FORMULIERUNG

Basierend auf der Forschung von Gutjahr und Fischer [9] wird im Folgenden das Optimierungsproblem beschrieben, bei dem der Entscheidungsträger mit der Entscheidung konfrontiert ist, Zeitintervalle für Lieferungen von Hilfsgütern zu einer Menge von Nachfragepunkten nach einer Katastrophe zu bestimmen. Häufig ist die Erreichbarkeit mancher Nachfragepunkte nach einer Katastrophe deutlich erschwert und variiert stark zwischen den Nachfragepunkten. Die erschwerte Erreichbarkeit hat große Auswirkungen auf die Versorgungskosten, die aus der Lieferung zu einem Nachfragepunkt resultieren, wodurch eine ungleichmäßige Versorgung der Nachfragepunkte mit Hilfsgütern oder Dienstleistungen begünstigt wird.

Das um den Fairnessaspekt erweiterte Entscheidungsproblem ist ein nichtkonvexes nichtlineares Optimierungsproblem. Um den wesentlichen Interessen des Problems nachzukommen, wird das Transportproblem so einfach und simpel wie möglich gehalten. Dafür wird im Modell von einem einzelnen zu liefernden Wirtschaftsgut ausgegangen, wodurch allerdings auch Transportmodelle mit einem Portfolio an Wirtschaftsgütern mit konstanten proportionalen Anteilen für jeden Nachfrageknoten, abgedeckt sind. Weiters wird angenommen, dass es sich bei den nachgefragten Gütern um Verbrauchsgüter handelt, deren Bedarf wiederkehrend ist, beispielsweise Wasser, Nahrung oder Medizin. Unter dem Begriff Güter werden im weitesten Sinne ebenfalls nicht-materielle Leistungen verstanden, wie etwa medizinische Versorgung.

In der Katastrophenhilfe wird wiederkehrender Bedarf an Gütern typischerweise durch wiederholte Lieferung abgedeckt, meist durch periodische Lieferschemata. Typischerweise kann der gesamte zukünftige Bedarf nicht mit einer Lieferung abgedeckt werden. Geschuldet ist dies einigen wesentlichen Umständen, wie etwa der Tatsache, dass Lieferkapazitäten, beispielsweise LKW-Kapazitäten, begrenzt sind. Weiters sind selbst an den zentralen Depot-Standorten die Ressourcen zu Beginn begrenzt und die Hilfsgüter sind nicht von Anfang an verfügbar, sondern müssen erst nach und nach beschafft werden, beispielsweise durch die Unterstützung internationaler Hilfsorganisationen. Für gewisse Hilfsgüter, zum Beispiel verderbliche Lebensmittel, ist eine einmalige Lieferung des gesamten zukünftigen Bedarfs generell nicht möglich, da eine längere Lagerung in den Nachfrageknoten nicht möglich ist. Das

Problem ist hier vor allem die beschädigte Infrastruktur, beispielsweise fehlender Strom für Kühlschränke. Ebenfalls problematisch sind die nicht verfügbaren lokalen Lagerkapazitäten, wobei auch die Sicherheit dieser eine Rolle spielt, da in Katastrophensituationen die Gefahr von Plünderungen steigt.

Für das nachgefragte Hilfsgut werden deshalb folgende Annahmen getroffen:

- Das Hilfsgut kann nicht längere Zeit gelagert werden, sondern ist kurz nach der Lieferung zu verbrauchen.
- Wenn das Hilfsgut nicht geliefert wird/werden kann, ist die Nachfrage verloren. Das bedeutet, dass die Nachfrage nicht durch zusätzliche Lieferungen in der Zukunft abgedeckt werden kann.

Beispiele für Hilfsgüter, die die oben genannten Annahmen erfüllen, sind verderbliche Nahrungsmittel oder medizinische Hilfe.

Im Entscheidungsmodell werden folgende Annahmen getroffen: Die Lieferungen beziehen sich auf Hin-und-Zurück-Touren zu den Nachfrageknoten $k = 1, \dots, K$, ausgehend von einem zentralen Depot. In jedem Nachfrageknoten k ist die Anzahl der Hilfsbedürftigen, auch als Unterstützungsempfänger bezeichnet, durch w_k gegeben. Die Gesamtanzahl der Hilfsbedürftigen N ergibt sich als Summe aller Hilfsbedürftigen w_k über alle Nachfrageknoten $k \in K$.

Der Zeithorizont der Hilfsmission beträgt T Zeiteinheiten. Es soll ein Schema mit periodischen Zeitintervallen für jeden Nachfrageknoten ermittelt werden. Im Abstand von τ_k Zeiteinheiten findet eine Lieferung zum Nachfrageknoten k statt, wobei $\tau_k \in \mathbb{R}^+$. Durch die Annahme von $\tau_k \in \mathbb{R}^+$ werden ebenfalls Lieferschemata erlaubt, bei denen beispielsweise alle 2,5 Zeiteinheiten eine Lieferung erfolgt. Es wird angenommen, dass T im Vergleich zu τ_k groß ist und somit T/τ_k ein guter Näherungswert für die gesamte Anzahl an Lieferungen zum Nachfrageknoten k während der gesamten Hilfsmission ist. Für Lieferungen wird des Weiteren angenommen, dass sie immer die gesamte Nachfrage im Nachfrageknoten zum Zeitpunkt der Lieferung abdecken.

Ähnlich dem Konzept von Holguín-Veras et al. [10] wird angenommen, dass der Mangel, verursacht durch das fehlende Hilfsgut, t Zeiteinheiten nach der letzten Lieferung für die Hilfsbedürftigen w_k im Nachfrageknoten k durch die Deprivation Intensität $g(t)$ gegeben ist. Die Deprivation Intensitätsfunktion $g(t)$ repräsentiert z.B. die Intensität des Hungers, wachsend mit der vergangenen Zeit t seit der letzten Lieferung. Dabei handelt es sich immer um eine nicht-fallende Funktion im Intervall $[0, \tau_k]$, die sich periodisch wiederholt. Unter Deprivation Costs δ_k versteht man den kumulierten Wert der Deprivation Intensitätsfunktion zwischen zwei Lieferungen, also im Intervall $[0, \tau_k]$, somit das Integral von $g(t)$ im Intervall. Damit sind die mittleren Deprivation Costs für einen Unterstützungsempfänger im Nachfrageknoten k pro Zeiteinheit gegeben durch:

$$\delta_k = \frac{1}{\tau_k} \int_0^{\tau_k} g(t) dt$$

Es wird angenommen, dass eine Lieferung exakt ausreicht, um die Deprivation Costs auf null zu reduzieren. In Abbildung 1 ist dies veranschaulicht. Die blaue Kurve zeigt die Deprivation Intensität bei einem Lieferintervall von $\tau_k = 4$ Tagen, die rote Kurve zeigt die Deprivation Intensität bei einem Lieferintervall von $\tau_k = 1$ Tag für die Deprivation Intensitätsfunktion $g(t) = 0,4 t^2$.

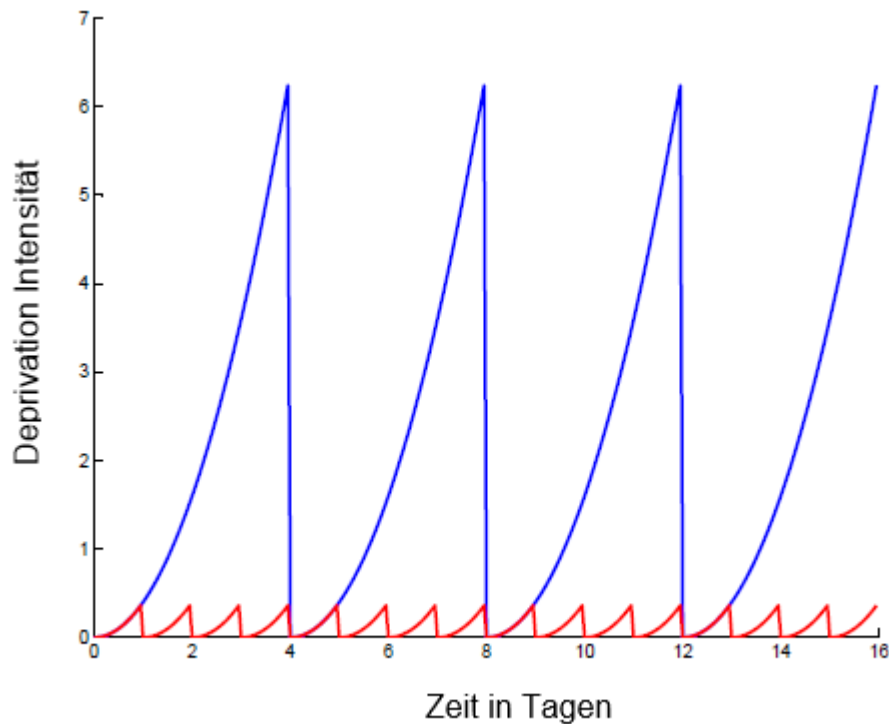


Abbildung 1: Beispiel einer Deprivation Intensitätsfunktion für 2 Lieferintervalle [9].

In Abbildung 1 ist ersichtlich, dass die Deprivation Intensität durch häufigere Lieferungen bzw. kürzere Lieferintervalle stark verringert werden kann. So ist gut erkennbar, dass um das Leiden der Hilfsbedürftigen zu verringern die Lieferintervalle so kurz wie möglich gehalten werden sollten, solange das verfügbare Budget dies zulässt. Ebenfalls kann in der Abbildung erkannt werden, dass die Deprivation Intensität $g(t)$ zum Zeitpunkt der Lieferung $t = \tau_k$ auf 0 zurückgeht, was auf die Annahme zurückzuführen ist, dass eine Lieferung die gesamte Nachfrage zum Lieferzeitpunkt abdeckt.

Aus den Zeitintervallen werden die Entscheidungsvariablen x_k abgeleitet. Die Entscheidungsvariablen $x_k = 1/\tau_k$ geben für jeden Nachfrageknoten k die mittlere Anzahl an Lieferungen pro Zeiteinheit t an.

Die Versorgungskosten d_k , auch Lieferkosten genannt, geben die Kosten für eine Hin- und-Zurück-Tour vom Depot zum Nachfrageknoten k an. Die Lieferkosten d_k , die anfallen um die Nachfrage in Nachfrageknoten k durch eine Lieferung zu befriedigen, setzen sich zusammen als

$$d_k = c_k + \gamma_k(w_k) \quad (k \in K),$$

wobei c_k die Fixkosten beschreibt und $\gamma_k(w_k)$ die variablen Kosten. Für die variablen Kosten wird eine nicht-fallende Funktion in Abhängigkeit der Hilfsbedürftigen w_k angenommen. Die Fixkosten c_k , und möglicherweise ebenfalls die Funktion der variablen Kosten, sind abhängig von der Erreichbarkeit des Nachfrageknotens k . Wenn der Nachfrageknoten k abgeschieden vom zentralen Depot liegt oder schwer zu erreichen ist, haben diese Parameter einen höheren Wert als für leicht zu erreichende Nachfrageknoten, und der Nachfrageknoten k somit höhere Kosten als diese.

Pro Zeiteinheit t steht ein Budget B zur Verfügung. Für den gesamten Zeithorizont T der Mission beträgt das Budget also $B \cdot T$.

$$T \sum_{k \in K} d_k x_k \leq B T$$

Das Budget wird in einer Nebenbedingung berücksichtigt, wonach die gesamten Lieferkosten zu allen Nachfrageknoten pro Zeiteinheit das Budget nicht überschreiten dürfen. Die obere Formel vereinfacht sich also zu:

$$\sum_{k \in K} d_k x_k \leq B$$

Ohne Berücksichtigung des Fairnessaspekts ergibt sich ein utilitaristisches Optimierungsmodell, das nur auf die Minimierung der gesamten oder der mittleren Deprivation Costs abzielt und durch Budget-Beschränkungen restringiert wird. Durch Einsetzen der Entscheidungsvariablen für die Lieferintervalle τ_k ergeben sich die mittleren Deprivation Costs für einen Unterstützungsempfänger im Nachfrageknoten k pro Zeiteinheit als:

$$\delta_k = x_k \int_0^{1/x_k} g(t) dt$$

Die mittleren Deprivation Costs aller Unterstützungsempfänger pro Zeiteinheit sind gegeben durch:

$$\mu = \frac{1}{N} \sum_{k \in K} w_k x_k \int_0^{1/x_k} g(t) dt$$

Beim utilitaristischen Optimierungsmodell wird μ als Zielfunktion minimiert, wobei die Budgetbedingung eingehalten werden muss und der zulässige Wertebereich für die Entscheidungsvariablen gegeben ist durch:

$$x_k \geq 0 \quad \forall k \in K$$

Für das Optimierungsmodell, bei dem der Fairnessaspekt berücksichtigt wird, wird die Zielfunktion durch einen Term, der proportional zum Gini Index ist, erweitert, wodurch Ungerechtigkeit in der Zielfunktion bestraft wird. Dies erfolgt durch die Linearkombination $\mu + \lambda \Delta$, mit Ginis mittlerer absoluter Abweichung $\Delta = 2\mu G$, wobei G für den Gini Index steht. Da der Gini Index eine dimensionslose Größe ist, muss er mit μ multipliziert werden um die beiden Terme sinnvoll kombinieren zu können. λ ist ein Parameter zur Einstellung der Einflussstärke der Fairness in der Zielfunktion, wobei λ einen Wert zwischen 0 und 0,5 annehmen kann. Bei $\lambda = 0$ wird der Fairnessaspekt nicht berücksichtigt und man erhält das utilitaristische Optimierungsmodell.

Der Gini Index kann Werte im Bereich $[0, 1]$ annehmen. $G = 0$ repräsentiert perfekte Fairness, das heißt, jedes Individuum erhält (oder verliert) genau gleich viel. $G = 1$ steht für vollkommene Ungerechtigkeit, das heißt, ein Individuum erhält alles, die anderen erhalten nichts. Der Gini Index wird wie folgt berechnet, wobei a_n für $n = 1, \dots, N$ den Verlust von Individuum n beschreibt und μ den mittleren Verlust über alle Individuen.

$$G = \frac{1}{2N \sum_{n=1}^N a_n} \sum_{n=1}^N \sum_{n'=1}^N |a_n - a_{n'}| = \frac{1}{2N^2 \mu} \sum_{n=1}^N \sum_{n'=1}^N |a_n - a_{n'}|$$

Für das vorliegende Optimierungsproblem, bei dem die Deprivation Costs δ_k die Verluste darstellen, kann der Gini Index der Deprivation Costs, aggregiert über alle Unterstützungsempfänger eines Nachfrageknotens, wie folgt berechnet werden.

$$G = \frac{1}{2N^2\mu} \sum_{k \in K} \sum_{k' \in K} w_k w_{k'} |\delta_k - \delta_{k'}|$$

Durch Einsetzen in die Linearkombination $\mu + 2\lambda\mu G$ ergibt sich das zweite Optimierungsproblem, bei dem der Fairnessaspekt berücksichtigt wird, als

$$\begin{aligned} \min_x \quad & \frac{1}{N} \sum_{k \in K} w_k x_k \int_0^{1/x_k} g(t) dt \\ & + \frac{\lambda}{N^2} \sum_{k, k' \in K} w_k w_{k'} \left| x_k \int_0^{1/x_k} g(t) dt - x_{k'} \int_0^{1/x_{k'}} g(t) dt \right| \end{aligned}$$

unter der bereits beschriebenen Budgetnebenbedingung und der Begrenzung des zulässigen Wertebereichs der Entscheidungsvariablen x_k .

Als Deprivation Intensitätsfunktion wird im vorliegenden Optimierungsmodell eine Potenzfunktion der Form $g(t) = g_1 t^p$ angenommen, wobei $p > 0$ und $t > 0$. Dadurch ergeben sich die mittleren Deprivation Costs für einen Unterstützungsempfänger in Nachfrageknoten k pro Zeiteinheit als

$$\delta_k = x_k \int_0^{1/x_k} g(t) dt = \frac{g_1}{(p+1) x_k^{p+1}}.$$

Die mittleren Deprivation Costs aller Unterstützungsempfänger pro Zeiteinheit sind definiert als

$$\mu = \frac{g_1}{N(p+1)} \sum_{k \in K} \frac{w_k}{x_k^p}$$

Der Gini Index der Deprivation Costs ist gegeben durch

$$G = \frac{g_1}{2N^2(p+1)\mu} \sum_{k,k' \in K} w_k w_{k'} \left| \frac{1}{x_k^p} - \frac{1}{x_{k'}^p} \right|.$$

Das Optimierungsproblem unter Verwendung von Potenzfunktionen als Deprivation Intensitätsfunktion ergibt sich somit als:

$$\min_x \frac{g_1}{N(p+1)} \sum_{k \in K} \frac{w_k}{x_k^p} + \frac{\lambda g_1}{N^2(p+1)} \sum_{k,k' \in K} w_k w_{k'} \left| \frac{1}{x_k^p} - \frac{1}{x_{k'}^p} \right|$$

$$\sum_{k \in K} d_k x_k \leq B$$

$$x_k \geq 0 \quad \forall k \in K$$

Für $\lambda = 0$ ergibt sich das utilitaristische Optimierungsmodell, bei dem die Zielfunktion sich auf die Minimierung von μ verkürzt. Durch das Weglassen der konstanten Faktoren vereinfacht sich die Zielfunktion des utilitaristischen Optimierungsmodells mit einer Potenzfunktion als Deprivation Intensitätsfunktion wie folgt.

$$\min_x \sum_{k \in K} w_k x_k^{-p}$$

Das sich daraus ergebende Optimierungsmodell ist konvex und lässt sich leicht unter Verwendung der Karush-Kuhn-Tucker Bedingungen lösen. Dadurch erhält man folgende optimale Lösung:

$$x_k = C \left(\frac{p w_k}{d_k} \right)^q$$

mit

$$C = \frac{B}{\sum_{k \in K} d_k^{1-q} (pw_k)^q}$$

und

$$q = \frac{1}{p+1}$$

Für das Optimierungsmodell, in dem der Fairnessaspekt berücksichtigt wird, gibt es keine einfach zu ermittelnde optimale Lösung wie für das utilitaristische Modell. Da die um den Gini Index erweiterte Zielfunktion nicht mehr konvex ist, kann das Optimierungsmodell mehrere lokale Optima haben. Zusätzlich ist die Funktion teilweise nicht-differenzierbar. Das macht die Suche nach dem Optimum anspruchsvoll. Um sich den Ansprüchen des nichtkonvexen nichtlinearen Optimierungsproblems zu stellen wurde folgender Lösungsansatz gewählt. Durch eine Variablentransformation wird das beschränkte Optimierungsproblem zu einem unbeschränkten Optimierungsproblem umformuliert, welches anschließend mittels einer Metaheuristik gelöst wird. Hierfür wurde die Partikelschwarmoptimierung gewählt, die im folgenden Abschnitt beschrieben wird.

3. PARTIKELSCHWARMOPTIMIERUNG

3.1. BASISALGORITHMUS

Nach dem Vorbild biologischen Schwarmverhaltens wurde 1995 die Partikelschwarmoptimierung, abgekürzt PSO, von Kennedy und Eberhart [13] entwickelt. Es handelt sich um ein naturanaloges Optimierungsverfahren, das eine Lösung für ein gegebenes Optimierungsproblem sucht. Das Verfahren wurde speziell zur Optimierung kontinuierlicher nichtlinearer Funktionen entwickelt. Eine wichtige Komponente des Konzepts ist, dass durch das Teilen von Informationen innerhalb des Schwarms ein evolutionärer Vorteil erzielt werden kann. Die initiale Idee dahinter war also, durch soziale Interaktion die Performance von ähnlichen Algorithmen, die rein auf individuelle kognitive Fähigkeiten aufbauen, zu übertreffen. Das Konzept beinhaltet Analogien zu Vogelschwärmen, die nach Korn suchen [17].

Bei der Partikelschwarmoptimierung bewegt sich eine Population von Lösungskandidaten, die so genannten Partikel, iterativ durch den Suchraum. Die Partikel werden im Suchraum platziert und für jeden Partikel wird der Zielfunktionswert, auch als Fitness bezeichnet, für die aktuelle Position ermittelt. Die Bewegung jedes Partikels wird durch eine Kombination aus dessen eigener Historie, im Speziellen dessen bisher bester Position, und der bisher besten Position eines oder mehrerer Schwarmmitglieder bestimmt. Zusätzlich beeinflussen zufällige Störungen die Flugbahnen. Mit großer Wahrscheinlichkeit bewegt sich der gesamte Schwarm nahe zum Optimum der Zielfunktion [17]. Ziel des Algorithmus ist es, bessere Positionen zu finden und die bisher beste Position upzudaten. Ein Partikel alleine besitzt nicht die Macht zu einer optimalen Lösung für das Problem zu gelangen, ein Fortschritt wird durch die Interaktion der Partikel erzielt.

Die bisher beste Position einer Gruppe von Partikeln oder aller Partikel wird auch als global beste Position bezeichnet. Wie viele Partikel miteinander kommunizieren ist ein Entscheidungsparameter, der vom Entscheidungsträger festzulegen ist. Diese Kommunikationsstruktur kann als soziales Netzwerk interpretiert werden. Mögliche Arten von Kommunikationsstrukturen sind statische und dynamische Strukturen. Es wird zwischen globaler Topologie und lokaler Topologie unterschieden, wobei bei der globalen Topologie alle Partikel miteinander kommunizieren, es also genau eine global beste Position gibt, die die Geschwindigkeit jedes einzelnen Partikels beeinflusst. Bei einer lokalen Topologie gibt es viele Möglichkeiten der Kommunikationsstruktur, z.B. eine Ringstruktur, bei der jeder Partikel genau zwei Nachbarn hat, mit denen er kommuniziert oder eine distanzbezogene Struktur, bei der jeder Partikel von den Partikeln beeinflusst wird, die ihm am nächsten sind. Es sind unzählige Formen von Kommunikationsstrukturen denkbar. Die oben genannten sind die in der Praxis meist verwendeten. In Unterabschnitt 3.2.1 wird näher auf die Varianten der möglichen Kommunikationsstrukturen eingegangen.

Jeder Partikel besteht aus drei D -dimensionalen Vektoren, wobei D die Dimension des Lösungsraums bezeichnet. Der erste D -dimensionale Vektor gibt die aktuelle Position x des Partikels an, also einen Punkt im Lösungsraum und somit eine mögliche Lösung

des Optimierungsproblems. Der zweite D -dimensionale Vektor gibt die bisher beste Position p des Partikels an, also den Punkt im Lösungsraum, der den bisher besten Zielfunktionswert des Partikels hat. Der dritte D -dimensionale Vektor gibt die Geschwindigkeit v an. Dieser Vektor ist ein Richtungsvektor und kein Ortsvektor wie die beiden anderen Vektoren. Der Geschwindigkeitsvektor kann als Schrittlänge interpretiert werden. Die Geschwindigkeit wird in jeder Iteration zur aktuellen Position addiert um zu einer neuen Position zu gelangen [17].

Der Algorithmus der Partikelschwarmoptimierung beginnt damit, dass die Partikel im Lösungsraum eines Optimierungsproblems platziert werden. Dies geschieht durch eine zufällige Platzierung innerhalb des zulässigen Raums. Zusätzlich wird bei der Initialisierung jedem Partikel eine zufällige Geschwindigkeit zugewiesen. Anschließend wird für jeden Partikel der Zielfunktionswert der aktuellen Position ermittelt. Die bisher beste Position eines Partikels wird mit dessen erster Position gleichgesetzt. Die global beste Position wird mit der besten Position einer Gruppe von Partikeln oder aller Partikel initialisiert.

Eine Iteration der Partikelschwarmoptimierung inkludiert eine Bewegung aller Partikel. Nach der Initialisierung beginnt die erste Iteration. Dabei wird für jeden Partikel eine Geschwindigkeit durch Kombination verschiedener Aspekte berechnet. Einfluss auf die Geschwindigkeit eines Partikels haben die aktuelle Position des Partikels, die bisher beste Position des Partikels, die bisher beste Position einer Gruppe von Partikeln oder aller Partikel und zufällige Störungen. Durch die Addition der Geschwindigkeit zur aktuellen Position gelangt ein Partikel zur nächsten Position [4]. Anschließend wird für jeden Partikel der Zielfunktionswert der aktuellen Position evaluiert. Sollte dieser besser sein als der Wert der bisher besten Position des Partikels, wird diese aktualisiert. Sollte dieser Wert ebenfalls besser als die bisher global beste Position sein, wird diese ebenfalls aktualisiert. Die Berechnung der Geschwindigkeit sowie der neuen Position eines Partikels geschieht im Basismodell durch die folgenden Formeln.

$$\vec{v}_i \leftarrow \vec{v}_i + \vec{U}(0, \phi_1) \otimes (\vec{p}_i - \vec{x}_i) + \vec{U}(0, \phi_2) \otimes (\vec{p}_g - \vec{x}_i)$$

$$\vec{x}_i \leftarrow \vec{x}_i + \vec{v}_i$$

$\vec{v}_i$ bezeichnet dabei die Geschwindigkeit von Partikel i , $\vec{x}_i$ bezeichnet die aktuelle Position von Partikel i . $\vec{p}_i$ steht für die bisher beste Position von Partikel i und $\vec{p}_g$ steht für die global beste Position einer Gruppe von Partikeln oder aller Partikel. $\vec{U}(0, \phi_1)$ und $\vec{U}(0, \phi_2)$ geben Vektoren mit zufälligen Zahlen an, gleichverteilt im Intervall $[0, \phi_1]$ bzw $[0, \phi_2]$, die für die zufälligen Störungen stehen. Das Symbol $\otimes$ bezeichnet die komponentenweise Multiplikation der Vektoren. ϕ_1 und ϕ_2 bezeichnen die Beschleunigungskoeffizienten. Die obere Formel dient zur Berechnung des Geschwindigkeitsvektors. Dabei setzt sich die neue Geschwindigkeit eines Partikels i aus einer Addition der alten Geschwindigkeit, eines Richtungsvektors zur bisher besten Position des Partikels, komponentenweise multipliziert mit einem zufälligen Vektor, und eines Richtungsvektors zur bisher global besten Position, ebenfalls komponentenweise multipliziert mit einem zufälligen Vektor, zusammen. Die untere Formel addiert die zuvor berechnete neue Geschwindigkeit zur aktuellen Position eines Partikels um zu einer neuen Position zu gelangen. Wenn die beschriebenen Schritte für alle Partikel durchgeführt wurden, folgt die nächste Iteration und die Schleife beginnt von vorne, es sei denn, das Abbruchkriterium ist erfüllt. Wenn das Abbruchkriterium erfüllt ist, gibt die bisher global beste Position die beste gefundene Lösung an.

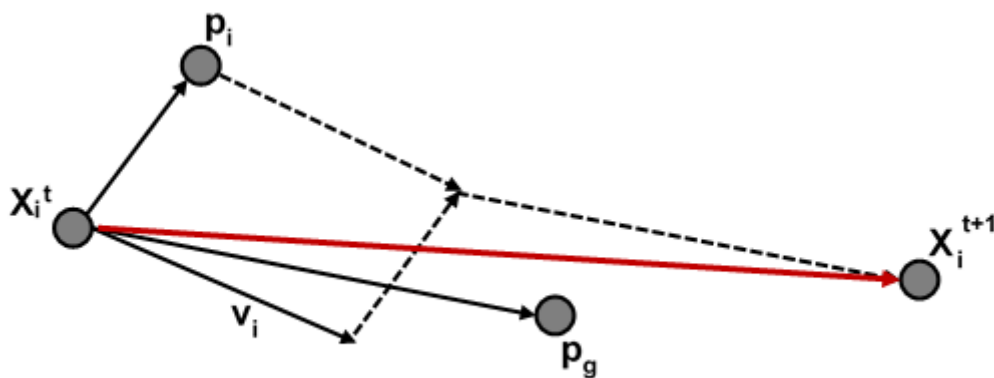


Abbildung 2: Bewegung eines Partikels.

In Abbildung 2 ist die Bewegung eines Partikels in einer Iteration skizziert. Dabei gibt x_i^t die aktuelle Position von Partikel i und x_i^{t+1} die nächste Position von Partikel i an, die sich aus der aktuellen Position und der errechneten Geschwindigkeit ergibt. Der rote Pfeil gibt die errechnete Geschwindigkeit an. Diese setzt sich aus einer Kombination der Vektoren der alten Geschwindigkeit v_i und den Richtungsvektoren zu der eigenen bisher besten Position p_i und der global besten Position p_g zusammen, die parallelverschoben als strichlierte Linien dargestellt sind.

Für die Partikelschwarmoptimierung sind einige Parameter vom Anwender zu wählen. Ein Parameter ist die Schwarmgröße, also die Anzahl an Partikeln. Häufig werden Schwarmgrößen zwischen 20 und 50 Partikeln gewählt. Möglich ist auch eine dynamische Schwarmgröße, die sich während der Laufzeit verändert, beispielsweise durch Eliminierung der schlechtesten und Klonung der besten Partikel. Ein weiterer festzulegender Parameter ist das Abbruchkriterium. Hier gibt es zwei Möglichkeiten: eine festgelegte Anzahl an Iterationen oder ein Abbruch des Algorithmus, wenn eine Lösung gefunden wurde, die nahe genug am Optimum liegt, sofern der Zielfunktionswert des Optimums bekannt ist, oder unter/über einer gewissen Grenze ist. Für bestimmte Anwendungen ist es ausreichend, dass die Lösung nahe genug am Optimum oder unter/über einem Schwellenwert liegt. Als Beispiel seien hier gesetzliche Vorgaben erwähnt, wie etwa die Abgaswerte unter einen vorgegebenen Schwellenwert zu senken.

Für die zufälligen Störungen sind ebenfalls Parameter festzulegen. Die Beschleunigungskoeffizienten ϕ_1 und ϕ_2 regeln die Einflussstärke der bisher besten Position des Partikels und der bisher global besten Position. Die Wahl der Beschleunigungskoeffizienten hat einen entscheidenden Einfluss auf das Verhalten des Partikelschwarmalgorithmus. Durch sie kann der Algorithmus mehr oder weniger reagierend bis hin zu instabil, wo die Geschwindigkeiten unkontrolliert zunehmen, gestaltet werden. In den Anfängen der PSO wurde für die Beschleunigungskoeffizienten häufig ein Wert von 2 gewählt, wodurch die Geschwindigkeiten relativ unkontrolliert zunahmen. Häufig war das nachteilig für die Suche, weshalb die Geschwindigkeiten besser kontrolliert werden sollten. Anfangs wurde die Geschwindigkeit eines Partikels durch einen Parameter V_{max} beschränkt,

der das Gleichgewicht zwischen Exploration und Exploitation beeinflusst. Die Geschwindigkeit musste immer im Intervall $[-V_{max}, +V_{max}]$ liegen. Die starren Grenzen und die problemabhängige Wahl des Parameters führten zu einigen Problemen und die Flugbahnen der Partikel konvergierten oft nicht, wenn der Parameter nicht optimal gewählt war. Für die Wahl von V_{max} waren keine festen Regeln gegeben und die eher willkürliche Entscheidung führte häufig zu schlechten Ergebnissen.

Mittlerweile existieren Ansätze, die den willkürlichen Parameter V_{max} eliminieren und die Beschleunigung der Partikel durch andere Parameter kontrollieren. Das Trägheitsgewicht ω , das 1998 von Shi und Eberhart [6] entwickelt wurde, hilft, das Ausmaß der Suche zu kontrollieren. Interpretiert werden kann ω als Fluidität des Mediums, in dem sich die Partikel bewegen. Ein hoher Wert, nahe 1, für ω entspricht einem dünnflüssigen Medium und begünstigt schnellere Bewegungen der Partikel und damit Exploration. Im Gegensatz dazu entspricht ein Trägheitsgewicht von beispielsweise 0,4 einem zähflüssigen Medium, in dem sich die Partikel langsamer bewegen, wodurch Exploitation begünstigt wird und die Partikel sich um lokale Optima zentrieren. Dadurch ist erkennbar, dass es zu Beginn der Suche von Vorteil sein kann, mit einem hohen Trägheitsgewicht zu beginnen um den Suchraum weitläufig zu erkunden und den Parameter später zu reduzieren, damit die Suche zentrierter wird und zum Optimum konvergiert. Andere Möglichkeiten der Anwendung eines dynamischen Trägheitsgewichts sind die Integration einer zufälligen Komponente, beispielsweise $\omega = U(0,5, 1)$ wie von Eberhart und Shi 2001 getestet [8], und die Anwendung eines unscharfen Trägheitsgewichts [7].

Bei der Anwendung des Trägheitsgewichts ω wird die alte Geschwindigkeit mit diesem multipliziert. Dadurch ergeben sich die Formeln für die Geschwindigkeit sowie für die aktuelle Position von Partikel i wie folgt.

$$\vec{v}_i \leftarrow \omega \vec{v}_i + \vec{U}(0, \phi_1) \otimes (\vec{p}_i - \vec{x}_i) + \vec{U}(0, \phi_2) \otimes (\vec{p}_g - \vec{x}_i)$$

$$\vec{x}_i \leftarrow \vec{x}_i + \vec{v}_i$$

Um die Kontrollmechanismen zu verbessern, entwickelten Clerc und Kennedy [3] das Konzept der Verengungskoeffizienten. Dabei kontrollieren die Verengungskoeffizienten die Konvergenz der Partikel und sind eine gut fundierte Methode um Explosion zu verhindern und Konvergenz zu sichern. Durch ihre Anwendung kann der willkürliche Parameter V_{max} eliminiert werden. Zusätzlich konkretisieren die Analysen der Autoren die Parameterwahl der Beschleunigungskoeffizienten ϕ_1 und ϕ_2 . Die Anwendung der Verengungskoeffizienten kann auf verschiedene Arten erfolgen. Eine einfache Form der Implementierung ist folgende:

$$\vec{v}_i \leftarrow \chi(\vec{v}_i + \vec{U}(0, \phi_1) \otimes (\vec{p}_i - \vec{x}_i) + \vec{U}(0, \phi_2) \otimes (\vec{p}_g - \vec{x}_i))$$

$$\vec{x}_i \leftarrow \vec{x}_i + \vec{v}_i$$

mit

$$\chi = \frac{2}{\phi - 2 + \sqrt{\phi^2 - 4\phi}}$$

und

$$\phi = \phi_1 + \phi_2 > 4$$

In der Praxis wird typischerweise $\phi = 4,1$ mit $\phi_1 = \phi_2$ verwendet, womit der konstante Multiplikationsfaktor χ in etwa einem Wert von 0,7298 entspricht. Mathematisch entspricht das Konzept der Verengungskoeffizienten dem Konzept der Partikelschwarmoptimierung mit Trägheitsgewicht. Mit einem simplen Mapping erhält man die von Clerc und Kennedy empfohlenen Parametereinstellungen $\omega = 0,7298$ und $\phi_1 = \phi_2 = 1,49618$ [17].

3.2. VARIANTEN DES PSO-ALGORITHMUS

3.2.1. KOMMUNIKATIONSSTRUKTUREN

Die ersten Überlegungen in der Partikelschwarmoptimierung [13] entstanden aus Vogelschar-Simulationen. In einigen dieser Modelle wurde die Flugbahn eines Vogels durch andere Vögel beeinflusst, die dem Vogel geographisch am nächsten waren. In den Anfängen der Partikelschwarmoptimierung wurde das soziale Netzwerk eines Partikels also durch physische Nähe im Suchraum definiert. Neben dem hohen Berechnungsaufwand waren bei derartigen Strukturen unerwünschte Konvergenzeigenschaften zu beobachten, weshalb die Anwendung euklidischer Nachbarschaften schnell eingestellt wurde und bessere Konzepte entwickelt wurden.

Abhilfe schaffte die Entwicklung der globalen Topologie. Wie zuvor bereits kurz beschrieben beeinflusst dabei die bisher global beste Position die Flugbahn jedes Partikels. Konkret bedeutet das, dass der beste Nachbar im gesamten Schwarm die Geschwindigkeit aller Partikel beeinflusst. Der beste Nachbar ist definiert als jener, der die bisher beste Position, also die Position mit dem besten Zielfunktionswert, lokalisiert hat. In einem Graph dargestellt wären dabei alle Partikel miteinander verbunden, zu sehen in Abbildung 3. In der Praxis verringert dieser Ansatz den Rechenaufwand erheblich, da das Programm lediglich die aktuell beste Position, den Zielfunktionswert an dieser Position und den Index des besten Nachbarn aufzeichnen muss. Bei der globalen Topologie, in der Literatur auch als gbest-Topologie (global-best Topologie) bezeichnet, handelt es sich um eine statische Kommunikationsstruktur. Bei statischen Kommunikationsstrukturen bleiben die Nachbarn und die Nachbarschaft eines Partikels über die gesamte Laufzeit des Programms unverändert.

Eine weitere statische Topologie ist die in der Literatur als lbest-Topologie (local-best Topologie) bezeichnete Struktur. Dabei handelt es sich um eine simple Ringstruktur, wie in Abbildung 3 ersichtlich, bei der jeder Partikel mit 2 Nachbarn verbunden ist und mit diesen kommuniziert. Der Vorteil bei dieser lokalen Kommunikationsstruktur ist die Möglichkeit der parallelen Suche. Hierbei können Subschwärme in verschiedenen Regionen des Suchraums konvergieren, wobei eine Stabilisierung der Population bei gleich guten lokalen Optima möglich ist. Falls eine Region allerdings besser ist als eine andere, ist es wahrscheinlich, dass diese die Partikel stärker anzieht. Durch diese

parallele Suche wird der Suchraum gründlicher abgesucht und die lbest-Topologie ist weniger anfällig für die Anziehung lokaler Optima. Allerdings konvergiert die Suche bei dieser lokalen Topologie weniger schnell als bei einer globalen Topologie [17].

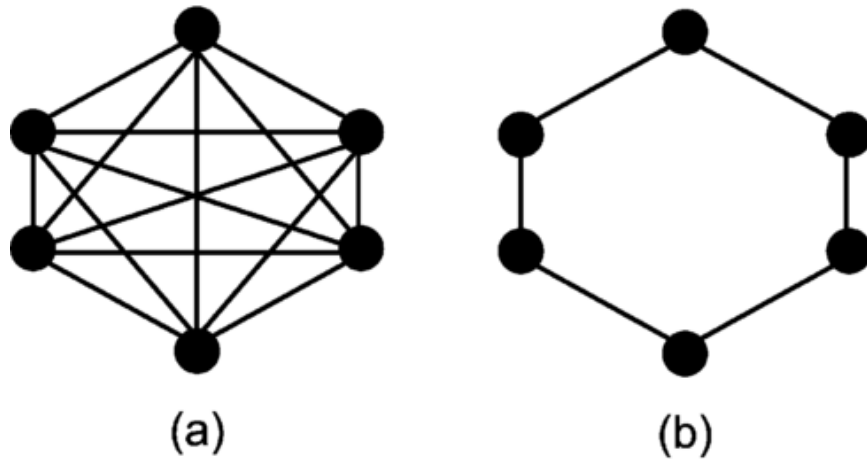


Abbildung 3: (a) gbest-Topologie, (b) lbest-Topologie [5].

Neben der ringförmigen lbest-Topologie sind viele andere Formen möglich. Beispielsweise wurde mit stern- und pyramidenförmigen Topologien experimentiert. Ein weiteres Beispiel einer lokalen Kommunikationsstruktur ist die von Neumann Topologie, bei der die Nachbarschaft auf einer Gitterstruktur aufgebaut ist und jeder Partikel mit 4 Nachbarn kommuniziert [14].

Durch die Kommunikationsstruktur wird die Geschwindigkeit des Informationsflusses beeinflusst. Die traditionelle gbest-Topologie stellt die unmittelbarste Art der Kommunikation dar, da alle Partikel direkt verbunden sind und somit Informationen zu besseren Positionen sofort weitergegeben werden. Bei der lbest-Topologie ist der Informationsfluss am langsamsten. Die Information einer besseren Position muss dabei von Nachbar zu Nachbar weitergegeben werden und braucht viele Iterationen um zu dem Partikel zu gelangen, der dem besten Partikel in der Ringstruktur gegenüberliegt. Es ist erkennbar, dass die Geschwindigkeit des Informationsflusses starke Auswirkungen darauf hat, ob der Partikelschwarm sich rasch in Richtung der früh entdeckten lokal besten Region bewegt oder ob Partikel sich planlos durch den Suchraum bewegen. Letzteres führt einerseits zu einer gründlicheren Exploration, andererseits zu ineffektiver Verlängerung der Programmlaufzeit [14].

Eine andere mögliche Form der Kommunikationsstruktur bilden dynamische Topologien. Durch nicht statische Strukturen lassen sich die Vorteile von verschiedenen Topologien nutzen. Beispielsweise fördert eine lbest-Topologie die gründliche Exploration des Suchraums, wohingegen eine gbest-Topologie schneller konvergiert. Bei einer Kombination der beiden Kommunikationsstrukturen, in der zuerst mit einer lbest-Topologie begonnen wird und die Anzahl der Nachbarn eines Partikels in jeder Iteration erhöht wird bis schließlich alle Partikel miteinander kommunizieren, kann die Performance des Algorithmus verbessert werden.

Eine andere Möglichkeit von dynamischen Kommunikationsstrukturen ist eine zufällige Generierung von Subpopulationen einer bestimmten Größe in jeder oder jeder x-ten Iteration, also die zufällige Zuordnung von Nachbarn. Ein weiterer Ansatz ist die performanceabhängige Änderung der Nachbarschaft. Bei diesem Ansatz verändert sich die Anzahl der Nachbarn eines Partikels je nach dessen Performance. Ist ein Partikel beispielsweise der beste in seiner Nachbarschaft, reduziert sich die Anzahl seiner Nachbarn. Wenn ein Partikel jedoch schlechte Lösungen findet, wird die Anzahl seiner Nachbarn erhöht, damit der Partikel von den Informationen der anderen profitieren kann [17].

3.2.2. FIPS

Die bisher vorgestellte traditionelle Form der Partikelschwarmoptimierung wird auch als kanonische Form der PSO bezeichnet. Mendes et al. [15] analysierten eine andere Form der Partikelschwarmoptimierung, bei der im Gegensatz zur kanonischen Form, bei der ein Partikel nur durch seine eigene und die beste Position seiner Nachbarschaft beeinflusst wird, alle Partikel in der Nachbarschaft die Flugbahn eines Partikels beeinflussen. Diese Form der Partikelschwärme wurde später als FIPS (Fully Informed Particle Swarms) bezeichnet. Die Autoren zielen darauf ab, dass die Informationen der Partikel, die nicht die besten der Nachbarschaft sind, nicht ungenutzt bleiben. Bei den Experimenten konnten gute Resultate erzielt werden und mit guten Parametereinstellungen wurden die Lösungen in weniger Iterationen gefunden als mit der kanonischen Form der Partikelschwarmoptimierung. Allerdings hängt die Performance verstärkt von der verwendeten Kommunikationsstruktur ab. Die Berechnung der Geschwindigkeit bei der Partikelschwarmoptimierung mit voll informierten Partikeln erfolgt durch

$$\vec{v}_i \leftarrow \chi(\vec{v}_i + \frac{1}{K_i} \sum_{n=1}^{K_i} \vec{U}(0, \phi) \otimes (\vec{p}_{nbr_n} - \vec{x}_i))$$

wobei K_i die Anzahl der Nachbarn von Partikel i darstellt und nbr_n der n -te Nachbar von Partikel i ist. Dabei nutzt ein Partikel einen stochastischen Durchschnitt der besten Positionen aller Nachbarn. Bei der Verwendung einer gbest-Topologie wird ein Partikel somit durch den gesamten Schwarm beeinflusst, was die Geschwindigkeit nahezu zufällig macht. Es werden 2 Ansätze unterschieden, bei denen Partikel i entweder zu seiner eigenen Nachbarschaft gehört oder nicht. Beim letzteren Ansatz wird die eigene beste Position des Partikels bei der Berechnung der neuen Position nicht berücksichtigt [17].

Detaillierte Forschungsergebnisse zu FIPS und anderen Weiterentwicklungen der Partikelschwarmoptimierung wurden von Mendes [16] in dessen Dissertation festgehalten.

4. ANWENDUNGSBEISPIEL: NEPAL ERDBEBEN 2015

Für das Anwendungsbeispiel werden reale Daten des großen Erdbebens in Nepal im Jahr 2015 verwendet. Dieses Fallbeispiel wurde gewählt, weil die Hilfsoperation von internationalen Organisationen kritisiert wurde. Im Speziellen wurde kritisiert, dass die Hilfsorganisationen die Hilfsgüter ungleichmäßig verteilt hätten und leichter zu erreichende Regionen bevorzugt behandelt wurden. Die nachgelagerten Bewertungen der Hilfsmission zeigten, dass die schwer zu erreichenden Gebirgsregionen in der Belieferung benachteiligt wurden und im Vergleich zu anderen Regionen weniger Hilfsgüter erhielten [12]. Dadurch ist dieses Anwendungsbeispiel gut geeignet, um das vorgestellte Optimierungsmodell mit Berücksichtigung des Fairnessaspekts zu implementieren. Ungerechte Verteilungen entstehen nicht zwangsläufig beabsichtigt mit der Absicht der Diskriminierung, sondern können bereits aus der Wahl eines Optimierungsansatzes resultieren, der den Fairnessaspekt nicht berücksichtigt. Durch die gezielte Beachtung des Fairnessaspekts, und sei es auch nur mit einer vergleichsweise minimalen Gewichtung, kann der Gini Index der Deprivation Costs, und damit die Ungerechtigkeit, bereits erheblich reduziert werden. Dies kann bereits durch Inkaufnahme eines überraschend kleinen Anstiegs in den Deprivation Costs umgesetzt werden.

Am 25. April 2015 ereignete sich in Nepal ein Erdbeben mit einer Magnitude von 7,8 M_w , das am 12. Mai 2015 gefolgt wurde von einem starken Nachbeben mit Magnitude 7,2 M_w . Die Beben wurden in den Medien auch als Erdbeben im Himalaya bezeichnet. Die Erdbeben gelten als tödlichste Katastrophe Nepals, bei der 8.800 Menschen ums Leben kamen und über 22.000 verletzt wurden. Die Hilfsmission wurde von der Nepalesischen Regierung koordiniert und hauptsächlich von der Nepalesischen Armee durchgeführt. Unterstützt wurde das Militär durch Hilfstruppen aus anderen Ländern sowie durch verschiedene internationale Hilfsorganisationen. Das Epizentrum des Hauptbebens befand sich etwa 80 km nordwestlich der Hauptstadt Kathmandu und jenes des stärksten Nachbebens etwa 83 km östlich der Hauptstadt, wie in Abbildung 4 zu erkennen ist. Für das Anwendungsbeispiel werden die realen Daten aus Johnsson [12] verwendet.

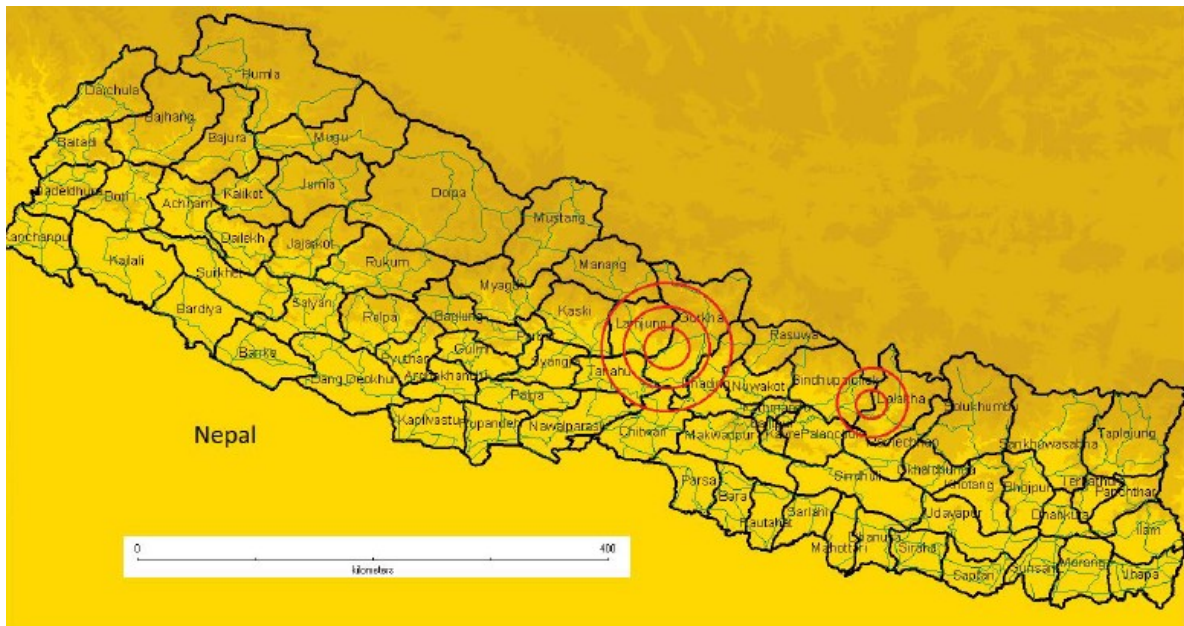


Abbildung 4: Epizentren des Hauptbebens (3 Kreise) und des größten Nachbebens (2 Kreise) [9].

4.1. MODELLIERUNG

Durch die Erdbeben wurden viele Gesundheitszentren in Nepal vollständig oder teilweise zerstört. Es wird angenommen, dass die Versorgungsteams in der Hauptstadt Kathmandu stationiert sind und von dort die Bezirke in regelmäßigen Zeitintervallen besuchen. Während eines Besuchs in einem Bezirk bieten die Versorgungsteams medizinische Hilfe an einem zentralen Standort an und machen in dringenden Fällen auch Hausbesuche. Im Fallbeispiel sind die Entscheidungsträger mit einer Situation konfrontiert, in welcher festzulegen ist, wie häufig medizinische Versorgungsteams die zentralen Bezirksstandorte besuchen.

Für die Implementierung des Optimierungsmodells werden die 14 Bezirke, die von der Nepalesischen Regierung als am stärksten unterstützungsbedürftig eingestuft wurden, herangezogen. Diese Bezirke werden als Nachfrageknoten bezeichnet und sind rund um die Hauptstadt Kathmandu, in der sich das Depot befindet, angesiedelt. Auch Kathmandu selbst zählt zu diesen 14 Nachfrageknoten, was in Abbildung 5 grafisch dargestellt ist.

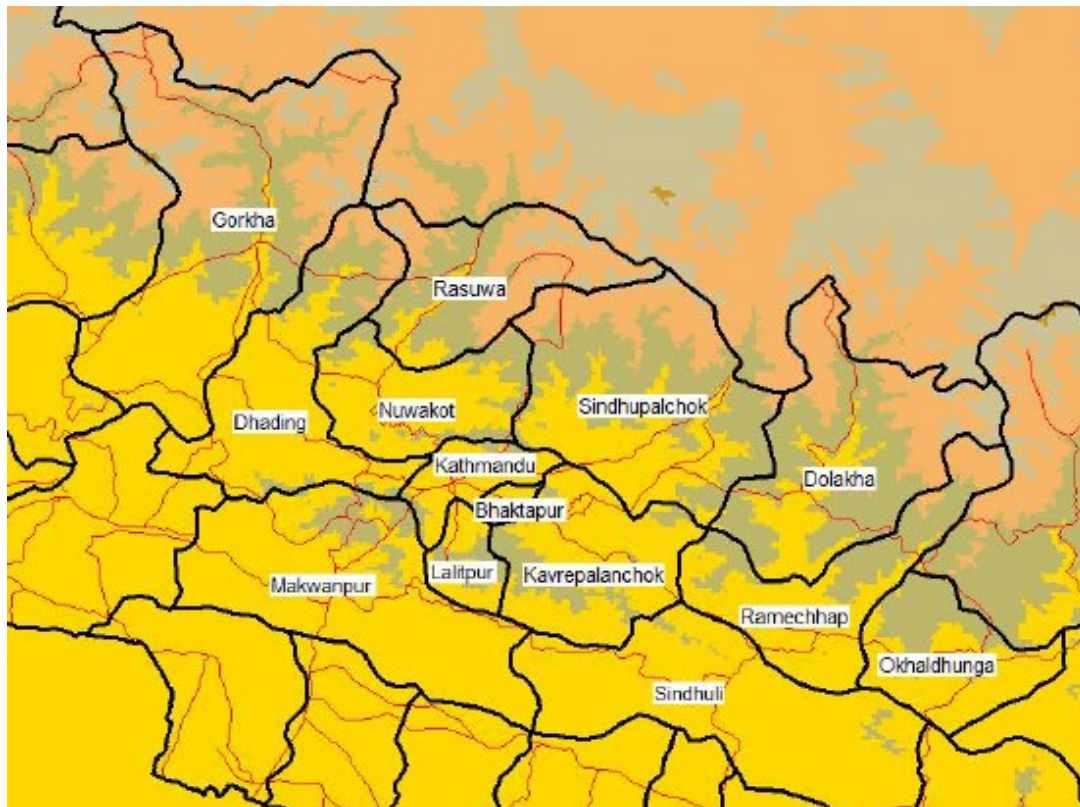


Abbildung 5: Die 14 Bezirke, die von der Nepalesischen Regierung als am stärksten unterstützungsbedürftig eingestuft wurden [9].

Die 14 Bezirke bilden die Menge K der Nachfrageknoten. Für die Nachfrage in Knoten k , also die Anzahl der Unterstützungsempfänger w_k , wird angenommen, dass diese proportional zu der Anzahl der vollständig oder teilweise zerstörten Gesundheitszentren ist. Da sowohl die optimale Lösung als auch der Gini Index invariant unter einer Multiplikation mit einer positiven Konstante sind, kann die Anzahl der Unterstützungsempfänger w_k gleich der Anzahl an vollständig oder teilweise zerstörten Gesundheitszentren gesetzt werden. Im vorliegenden Fall ergeben sich dadurch die Werte aus Tabelle 1 für die Nachfrage in Knoten k . Einige der Nachfrageknoten liegen in der nördlichen Bergregion und sind schwer zu erreichen, andere sind einfacher zu erreichen. Dadurch ergeben sich Kosten, die zwischen den Bezirken stark variieren.

Die Kosten, die ein Besuch von Bezirk k verursacht, sind gegeben durch d_k und werden mithilfe einer Formel geschätzt. Einbezogen werden unter anderem die Entfernung zum Depot (Kathmandu), die mittlere geographische Steigung σ_k in Graden und

variable Kosten je zerstörtem Gesundheitszentrum (pro Unterstützungsempfänger). Die Kosten sind gegeben durch:

$$d_k = c_k + \chi * (1 + \beta * (\tan \sigma_k)^\rho) * w_k$$

In der Formel bezeichnet $c_k = \bar{c}_k + c_{bas}$ die Entfernung, dabei steht $\bar{c}_k$ für die Entfernung zwischen Bezirk k und der Hauptstadt Kathmandu in Kilometern und c_{bas} ist ein festzulegender konstanter Basiswert, der zu der individuellen Entfernung von Knoten k addiert wird. χ gibt die variablen Kosten pro zerstörtem Gesundheitszentrum an, ausgedrückt in Kilometern. Die Parameter β und ρ sind zu wählende Parameter, die den Zusammenhang der Kosten mit der Steigung des Geländes angeben. β gibt dabei einen Gewichtungsfaktor für die erhöhten Kosten von Hausbesuchen im Vergleich zu stationärer Versorgung an den Bezirksstandorten an und ρ repräsentiert den Potenzfaktor der steigenden Unerreichbarkeit für den Tangens der mittleren geographischen Steigung σ_k .

Aus Johnsson [12] werden die Werte für w_k , $\bar{c}_k$ und σ_k verwendet. Für c_{bas} wird ein Wert von 10 km angenommen, der als Basiswert zu der Entfernung addiert wird. Für die Parametereinstellungen für χ , β und ρ wurden zwei Varianten getestet. In Variante 1 wurden die Parameter $\chi = 2$, $\beta = 200$ und $\rho = 3$ gewählt. In Variante 2 wurden die Parameter $\chi = 1$, $\beta = 100$ und $\rho = 2$ gewählt, was im Vergleich zu Variante 1 eine schwächere Gewichtung der Steilheit des Geländes ausdrückt. Die verwendeten Werte aus Johnsson [12] sowie die berechneten Kosten d_k für beide Varianten sind in Tabelle 1 für alle Bezirke aufgelistet.

Bezirk	w_k	$\overline{c_k}$	σ_k	d_k Variante 1	d_k Variante 2
Bhaktapur	25	16	3,98	79,40	63,10
Dhading	106	90	10,79	605,50	591,00
Dolakha	83	183	13,86	857,70	781,29
Gorkha	79	144	15,70	1013,80	857,18
Kathmandu	63	0	4,79	150,80	117,24
Kavrepalanchok	131	38	8,62	492,50	480,04
Lalitpur	39	7	8,12	140,30	135,39
Makwanpur	59	88	5,89	241,90	219,79
Nuwakot	99	61	10,39	513,10	502,82
Okhaldhunga	30	217	10,67	367,30	363,49
Ramechhap	66	154	11,14	497,60	485,92
Rasuwa	27	87	16,88	452,80	372,61
Sindhuli	59	134	5,57	283,90	259,11
Sindhupalchok	97	67	13,20	771,60	707,62

Tabelle 1: Unterstützungsempfänger w_k , Parameter $\overline{c_k}$ und σ_k und Kosten d_k (2 Varianten) aller Nachfrageknoten $k= 1, \dots, 14$.

Es kann angenommen werden, dass in der optimalen Lösung das zur Verfügung stehende Budget voll ausgeschöpft wird. Das bedeutet, dass die Ungleichheitsnebenbedingung durch die Gleichheitsnebenbedingung $\sum_{k \in K} d_k x_k = B$ ersetzt werden kann. Durch die Variablentransformation

$$x_k = \frac{B}{d_k} * \frac{\exp(z_k)}{\sum_{r \in K} \exp(z_r)} \quad (k \in K)$$

mit $z_k \in \mathbb{R}$ können die Budgetbedingung und die Variablenbeschränkung beseitigt werden. Setzt man in die Budgetbedingung und die Variablenbeschränkungen den oben beschriebenen Ausdruck für x_k ein, sind diese automatisch erfüllt und das beschränkte Optimierungsproblem kann zu einem unbeschränkten Optimierungsproblem transformiert werden, bei dem die Hilfsvariablen z_k die Entscheidungsvariablen darstellen. Somit wird die Partikelschwarmoptimierung mit den Positionen $z = (z_k) \in \mathbb{R}^m$ im Suchraum durchgeführt, die Zielfunktion allerdings nach Transformation für Lösung $x = (x_k) \in \mathbb{R}^m$ in Position z evaluiert. Durch die

Transformation in ein unbeschränktes Optimierungsproblem wird die Anwendung der Partikelschwarmoptimierung vereinfacht. Für ein beschränktes Optimierungsproblem wären zusätzliche Schritte zur Restriktion der Bewegungen der Partikel im Algorithmus zu implementieren, was den Algorithmus allerdings weniger effizient machen kann. Durch eine Restriktion werden beispielsweise die Schrittlängen begrenzt oder Partikel, die den zulässigen Lösungsraum verlassen, neu positioniert. Dadurch kann die Konvergenz verschlechtert werden.

Der Parameter λ , der die Einflussstärke des Fairnessaspekts in der Zielfunktion steuert, wird in Schritten von 0,05 im Bereich $[0, 0,5]$ variiert, um einige pareto-effiziente Lösungen zu erhalten. Für $\lambda = 0$ erhält man eine Lösung, die den Fairnessaspekt völlig außer Acht lässt und nur die mittleren Deprivation Costs minimiert. Diese stellt die optimale Lösung des utilitaristischen Entscheidungsmodells dar. Mit steigendem λ steigt die Berücksichtigung des Fairnessaspekts. Das bedeutet, dass mit steigendem λ die Fairness steigt und der Gini Index sinkt.

Für die Optimierung des Problems mittels Partikelschwarmoptimierung sind ebenfalls einige Parameter zu wählen. Nach einigen Pre-Tests wurde eine Schwarmgröße von 35 Partikeln gewählt und das Abbruchkriterium mit 200 Iterationen festgelegt, was sich als sehr effizient erwies. In einer experimentellen Analyse wurden die beiden Parameter allerdings variiert und zusätzlich der Einfluss auf die Programmlaufzeit analysiert. Es wurde die gbest-Topologie gewählt, bei der alle Partikel Informationen austauschen. Diese globale Kommunikationsstruktur wurde gewählt, da sie schneller konvergiert als beispielsweise die lokale lbest-Topologie. Eine experimentelle Analyse mit anderen Topologien bietet Möglichkeiten für zukünftige Forschung.

Für die restlichen festzulegenden Parametereinstellungen wurde die Analyse von Clerc und Kennedy [3] herangezogen und die dort empfohlenen Werte verwendet. Die Autoren empfehlen die Anwendung eines bestimmten Verengungskoeffizienten. Für die Implementierung wurde dieser Verengungskoeffizient in Werte für das Trägheitsgewicht $\omega = 0,7298$ und die Beschleunigungskoeffizienten $\phi_1 = \phi_2 = 1,49618$ transformiert, die mathematisch äquivalent zu dem empfohlenen Verengungskoeffizienten sind.

4.2. IMPLEMENTIERUNG

Die Implementierung wurde unter Verwendung von C++ und dem Microsoft Visual Studio Community 2015 durchgeführt. Der vollständige Code findet sich im Anhang. Die Berechnungen wurden auf einem Computer mit Intel Core i5 3470 und 8 GB RAM durchgeführt.

4.3. ERGEBNISSE

Der Testfall wurde für verschiedene Potenzen in der Deprivation Intensitätsfunktion implementiert. In der Deprivation Intensitätsfunktion $g(t) = g_1 t^p$ wurde für g_1 der Wert 2 gewählt und die Potenz p variiert mit $p = 1$, $p = 2$ und $p = 3$. Die λ -Einstellungen wurden in Schritten von 0,05 im Intervall $[0, 0,5]$ variiert. Im Folgenden sind die Ergebnisse präsentiert.

4.3.1. KALKULATION DER LIEFERKOSTEN NACH VARIANTE 1

In Variante 1 wurden die Parameter $\chi = 2$, $\beta = 200$ und $\rho = 3$ für die Kalkulation der Lieferkosten d_k gewählt. In Tabelle 2 sind die Ergebnisse der Partikelschwarmoptimierung für alle Ausprägungen von λ für $p = 1$ angeführt. Diese sind in Abbildung 6 auch grafisch dargestellt.

λ	μ	$G(\delta)$
0,00	6070,20	0,1428
0,05	6075,10	0,1265
0,10	6089,84	0,1101
0,15	6115,02	0,0932
0,20	6151,54	0,0757
0,25	6198,80	0,0582
0,30	6254,17	0,0415
0,35	6304,24	0,0288
0,40	6348,82	0,0191
0,45	6383,19	0,0126
0,50	6417,19	0,0069

Tabelle 2: Mittlere Deprivation Costs μ und Gini Index der Deprivation Costs $G(\delta)$ mit Berechnung der Lieferkosten d_k nach Variante 1 für $p = 1$ und alle λ -Werte.

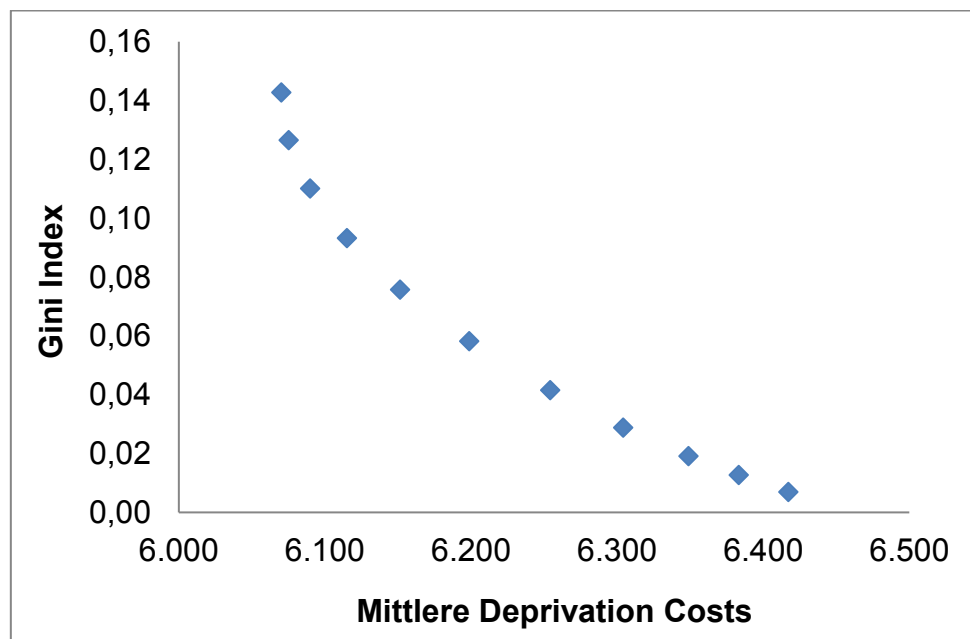


Abbildung 6: Grafische Darstellung der Wertepaare aus Tabelle 2 für die verschiedenen λ -Einstellungen (von links nach rechts).

Aus Tabelle 2 und Abbildung 6 ist erkennbar, dass durch die Erhöhung der Gewichtung des Fairnessaspekts von $\lambda = 0$ zu $\lambda = 0,5$ eine Reduktion des Gini Indizes um mehr als 95% erzielt werden kann. Dies kann zum Preis eines einigermaßen kleinen Anstiegs der Deprivation Costs von 5,7% erreicht werden.

In Tabelle 3 sind die Ergebnisse der Partikelschwarmoptimierung für alle Ausprägungen von λ für $p = 2$ angeführt. Zusätzlich sind hier die Gini Indizes der Lösungen x_k angeführt. Die Ergebnisse sind in Abbildung 7 auch grafisch dargestellt.

λ	μ (Vielfache von 10^7)	$G(\delta)$	$G(x)$
0,00	2,5631	0,1899	0,0937
0,05	2,5658	0,1686	0,0829
0,10	2,5739	0,1470	0,0718
0,15	2,5877	0,1249	0,0605
0,20	2,6077	0,1020	0,0487
0,25	2,6365	0,0768	0,0361
0,30	2,6642	0,0571	0,0264
0,35	2,6949	0,0387	0,0179
0,40	2,7192	0,0262	0,0121
0,45	2,7399	0,0171	0,0079
0,50	2,7608	0,0088	0,0041

Tabelle 3: Mittlere Deprivation Costs μ , Gini Index der Deprivation Costs $G(\delta)$ sowie Gini Index der Lieferhäufigkeiten $G(x)$ mit Berechnung der Lieferkosten d_k nach Variante 1 für $p = 2$ und alle λ -Werte.

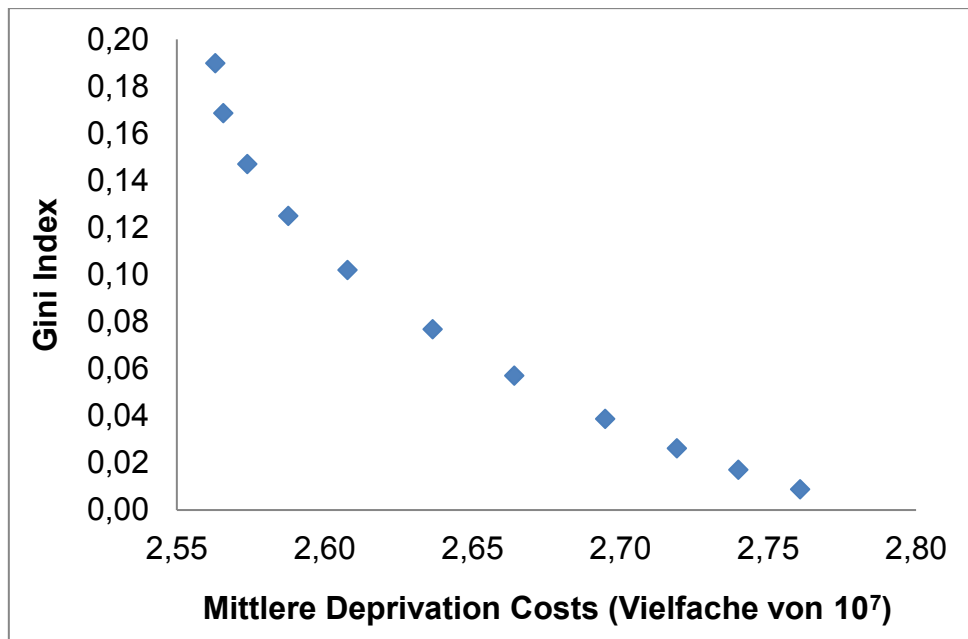


Abbildung 7: Grafische Darstellung der Wertepaare aus Tabelle 3 für die verschiedenen λ -Einstellungen (von links nach rechts).

In Tabelle 4 sind die Lösungen x_k für die verschiedenen λ -Einstellungen angegeben. Hierfür wurde exemplarisch ein Budget von 1000 pro Zeiteinheit unterstellt, was bei den gegebenen Lieferkosten d_k als verhältnismäßig erscheint. Dabei ist deutlich zu sehen, dass bei steigender Gewichtung des Fairnessaspekts die Lieferhäufigkeiten zwischen den einzelnen Regionen deutlich fairer werden und sich einander angleichen.

λ	0,00	0,05	0,10	0,15	0,20	0,25	0,30	0,35	0,40	0,45	0,50
x_1	0,204	0,197	0,190	0,182	0,175	0,167	0,164	0,161	0,159	0,158	0,156
x_2	0,168	0,167	0,167	0,167	0,167	0,166	0,164	0,161	0,159	0,158	0,156
x_3	0,138	0,140	0,142	0,144	0,146	0,148	0,150	0,152	0,154	0,155	0,156
x_4	0,128	0,131	0,134	0,136	0,139	0,141	0,144	0,146	0,148	0,151	0,153
x_5	0,224	0,216	0,207	0,198	0,188	0,178	0,167	0,161	0,159	0,158	0,156
x_6	0,193	0,188	0,183	0,178	0,172	0,167	0,164	0,161	0,159	0,158	0,156
x_7	0,196	0,189	0,183	0,178	0,172	0,167	0,164	0,161	0,159	0,158	0,156
x_8	0,187	0,184	0,180	0,177	0,172	0,167	0,164	0,161	0,159	0,158	0,156
x_9	0,173	0,172	0,170	0,169	0,167	0,166	0,164	0,161	0,159	0,158	0,156
x_{10}	0,130	0,133	0,135	0,137	0,139	0,142	0,144	0,146	0,148	0,151	0,153
x_{11}	0,153	0,153	0,154	0,155	0,156	0,157	0,158	0,159	0,159	0,158	0,156
x_{12}	0,117	0,120	0,123	0,126	0,128	0,131	0,133	0,136	0,139	0,140	0,143
x_{13}	0,178	0,175	0,172	0,170	0,168	0,166	0,164	0,161	0,159	0,158	0,156
x_{14}	0,150	0,152	0,153	0,154	0,156	0,157	0,158	0,159	0,159	0,158	0,156

Tabelle 4: Lieferhäufigkeiten x_k für die 14 Bezirke mit Berechnung der Lieferkosten d_k nach Variante 1 für $p = 2$ und alle λ -Werte mit einem Budget von 1000 je Zeiteinheit.

Aus Tabelle 3 und Abbildung 7 ist erkennbar, dass durch die Erhöhung der Gewichtung des Fairnessaspekts von $\lambda = 0$ zu $\lambda = 0,5$ eine Reduktion des Gini Indizes um mehr als 95% erzielt werden kann. Dies kann zum Preis eines einigermaßen kleinen Anstiegs der Deprivation Costs von 7,7% erreicht werden. Es ist klar, dass ein „Preis für Fairness“ immer in Kauf genommen werden muss, wenn Fairness als zusätzliches Ziel in die Zielfunktion integriert wird. Allerdings kann an den Ergebnissen abgelesen werden, dass der marginale Wert dieses Preises gering ist. Anders gesagt, die marginalen Kosten der Ungerechtigkeit zu Gunsten der optimalen Deprivation Costs (jene der utilitaristischen Lösung) sind exorbitant hoch. Vergleicht man die Deprivation Costs und die Gini Indizes der Lösungen für $\lambda = 0$ und $\lambda = 0,05$, ist zu sehen, dass für die letzten 0,1% der Deprivation Costs zum Optimum eine Zunahme von 11,2% der Ungerechtigkeit, gemessen am Gini Index, in Kauf genommen werden muss.

In Tabelle 5 sind die Ergebnisse der Partikelschwarmoptimierung für alle Ausprägungen von λ für $p = 3$ angeführt. Diese sind in Abbildung 8 auch grafisch dargestellt.

λ	μ (Vielfache von 10^{11})	$G(\delta)$
0,00	1,2304	0,2132
0,05	1,2319	0,1895
0,10	1,2362	0,1655
0,15	1,2436	0,1409
0,20	1,2546	0,1146
0,25	1,2686	0,0888
0,30	1,2858	0,0632
0,35	1,3040	0,0410
0,40	1,3146	0,0299
0,45	1,3279	0,0177
0,50	1,3355	0,0116

Tabelle 5: Mittlere Deprivation Costs μ und Gini Index der Deprivation Costs $G(\delta)$ mit Berechnung der Lieferkosten d_k nach Variante 1 für $p = 3$ und alle λ -Werte.

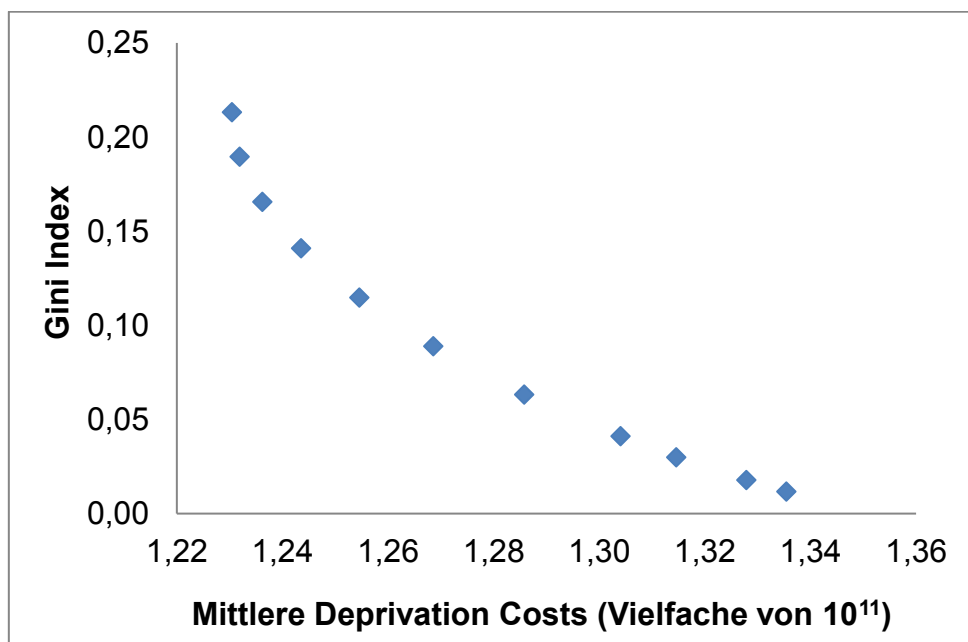


Abbildung 8: Grafische Darstellung der Wertepaare aus Tabelle 5 für die verschiedenen λ -Einstellungen (von links nach rechts).

Aus Tabelle 5 und Abbildung 8 ist erkennbar, dass durch die Erhöhung der Gewichtung des Fairnessaspekts von $\lambda = 0$ zu $\lambda = 0,5$ eine Reduktion des Gini Indizes um knapp

95% erzielt werden kann. Dies kann zum Preis eines einigermaßen kleinen Anstiegs der Deprivation Costs von 8,5% erreicht werden.

Wie an den Ergebnissen zu sehen ist, steigt der Gini Index der Deprivation Costs mit steigendem p , es steigt also die Ungerechtigkeit. Die gewählte Potenz beeinflusst außerdem die Größenordnung der Deprivation Costs.

In Abbildung 9 ist zu erkennen, dass die stärkere Gewichtung λ des Fairnessfaktors in der Zielfunktion die Fähigkeit besitzt, den negativen Einfluss des Exponenten p auf den Gini Index zu verringern. Dies wirkt also der steigenden Ungerechtigkeit mit steigendem p entgegen.

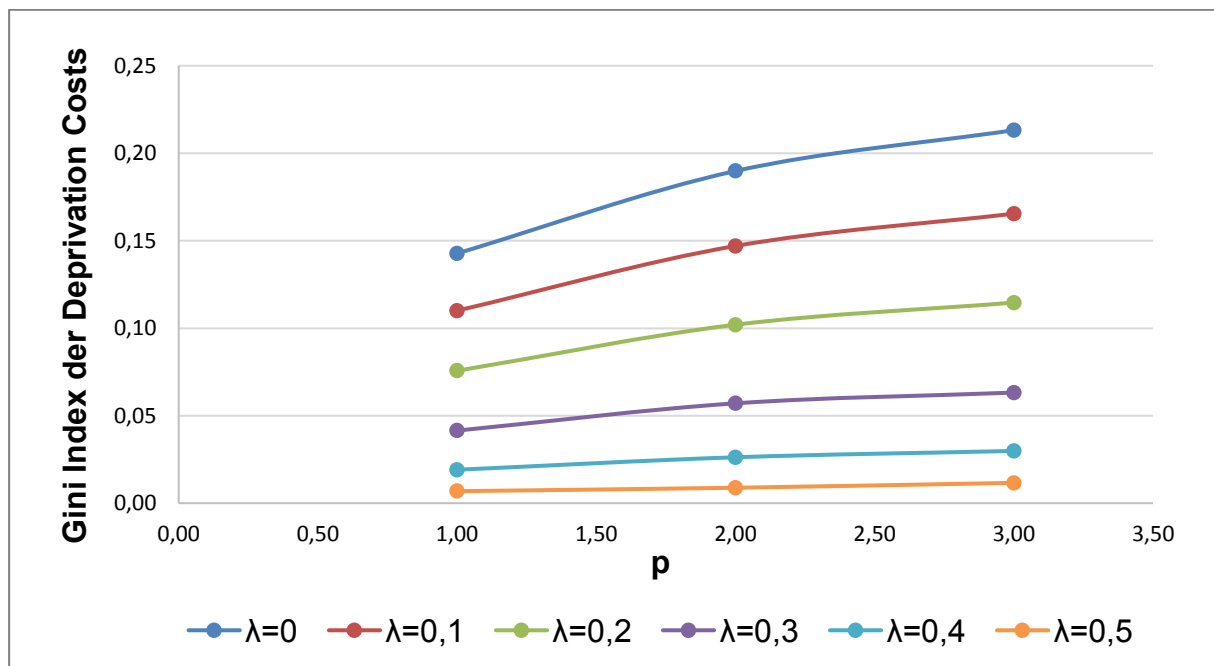


Abbildung 9: Gini Index der Deprivation Costs in Abhängigkeit von p für d_k nach Variante 1.

4.3.2. KALKULATION DER LIEFERKOSTEN NACH VARIANTE 2

In Variante 2 wurden die Parameter $\chi = 1$, $\beta = 100$ und $\rho = 2$ für die Kalkulation der Lieferkosten gewählt, was im Vergleich zu Variante 1 eine schwächere Gewichtung der Steilheit des Geländes ausdrückt. In Tabelle 6 sind die Ergebnisse der Partikelschwarmoptimierung für alle Ausprägungen von λ für $p = 1$ angeführt. Diese sind in Abbildung 10 auch grafisch dargestellt.

λ	μ	$G(\delta)$
0,00	5592,56	0,1398
0,05	5596,96	0,1239
0,10	5610,05	0,1080
0,15	5632,19	0,0918
0,20	5664,93	0,0748
0,25	5710,11	0,0567
0,30	5761,60	0,0399
0,35	5821,55	0,0236
0,40	5868,12	0,0126
0,45	5899,54	0,0063
0,50	5919,76	0,0026

Tabelle 6: Mittlere Deprivation Costs μ und Gini Index der Deprivation Costs $G(\delta)$ mit Berechnung der Lieferkosten d_k nach Variante 2 für $p = 1$ und alle λ -Werte.

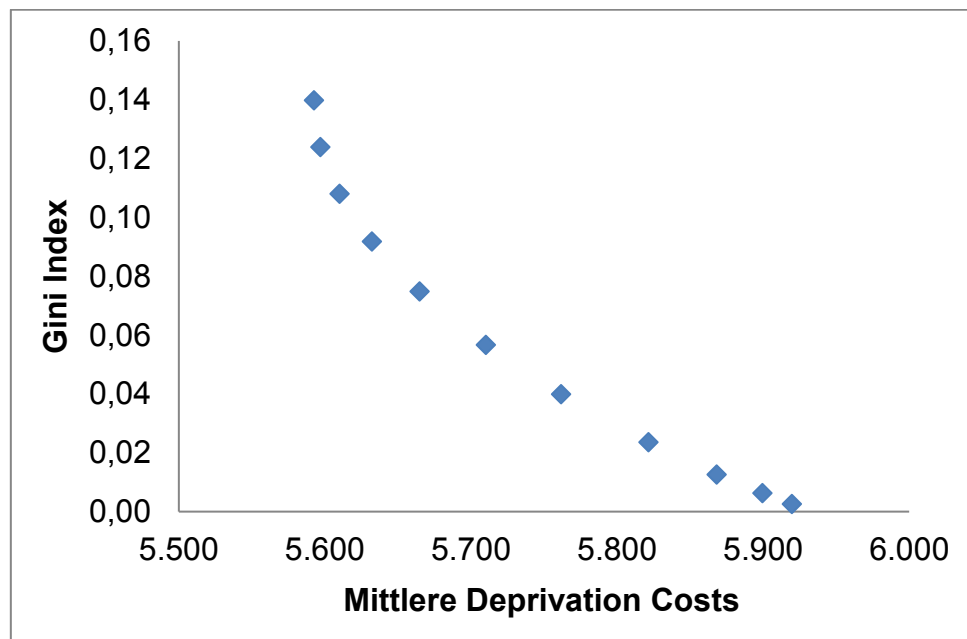


Abbildung 10: Grafische Darstellung der Wertepaare aus Tabelle 6 für die verschiedenen λ -Einstellungen (von links nach rechts).

Aus Tabelle 6 und Abbildung 10 ist erkennbar, dass durch die Erhöhung der Gewichtung des Fairnessaspekts von $\lambda = 0$ zu $\lambda = 0,5$ eine Reduktion des Gini Indizes

um mehr als 98% erzielt werden kann. Dies kann zum Preis eines einigermaßen kleinen Anstiegs der Deprivation Costs von 5,9% erreicht werden.

In Tabelle 7 sind die Ergebnisse der Partikelschwarmoptimierung für alle Ausprägungen von λ für $p = 2$ angeführt. Diese sind in Abbildung 11 auch grafisch dargestellt.

λ	μ (Vielfache von 10^7)	$G(\delta)$
0,00	2,1715	0,1846
0,05	2,1737	0,1638
0,10	2,1805	0,1427
0,15	2,1920	0,1211
0,20	2,2076	0,1001
0,25	2,2302	0,0765
0,30	2,2584	0,0528
0,35	2,2884	0,0319
0,40	2,3128	0,0172
0,45	2,3298	0,0085
0,50	2,3393	0,0041

Tabelle 7: Mittlere Deprivation Costs μ und Gini Index der Deprivation Costs $G(\delta)$ mit Berechnung der Lieferkosten d_k nach Variante 2 für $p = 2$ und alle λ -Werte.

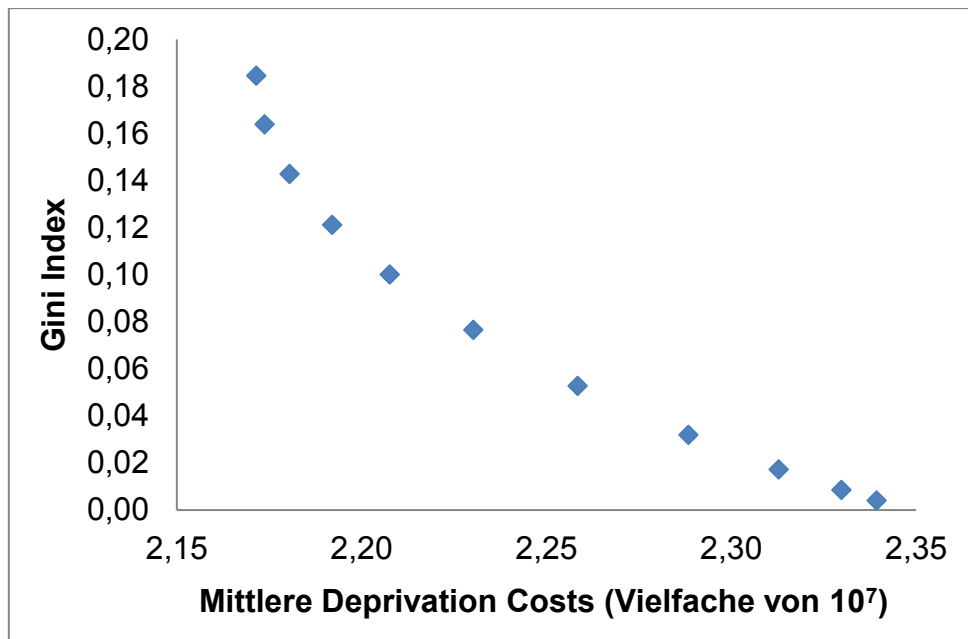


Abbildung 11: Grafische Darstellung der Wertepaare aus Tabelle 7 für die verschiedenen λ -Einstellungen (von links nach rechts).

Aus Tabelle 7 und Abbildung 11 ist erkennbar, dass durch die Erhöhung der Gewichtung des Fairnessaspekts von $\lambda = 0$ zu $\lambda = 0,5$ eine Reduktion des Gini Indizes um knapp 98% erzielt werden kann. Dies kann zum Preis eines einigermaßen kleinen Anstiegs der Deprivation Costs von 7,7% erreicht werden.

In Tabelle 8 sind die Ergebnisse der Partikelschwarmoptimierung für alle Ausprägungen von λ für $p = 3$ angeführt. Diese sind in Abbildung 12 auch grafisch dargestellt.

λ	μ (Vielfache von 10^{11})	$G(\delta)$
0,00	0,9578	0,2065
0,05	0,9589	0,1834
0,10	0,9621	0,1605
0,15	0,9677	0,1363
0,20	0,9754	0,1127
0,25	0,9864	0,0867
0,30	0,9998	0,0609
0,35	1,0162	0,0350
0,40	1,0273	0,0199
0,45	1,0363	0,0094
0,50	1,0411	0,0045

Tabelle 8: Mittlere Deprivation Costs μ und Gini Index der Deprivation Costs $G(\delta)$ mit Berechnung der Lieferkosten d_k nach Variante 2 für $p = 3$ und alle λ -Werte.

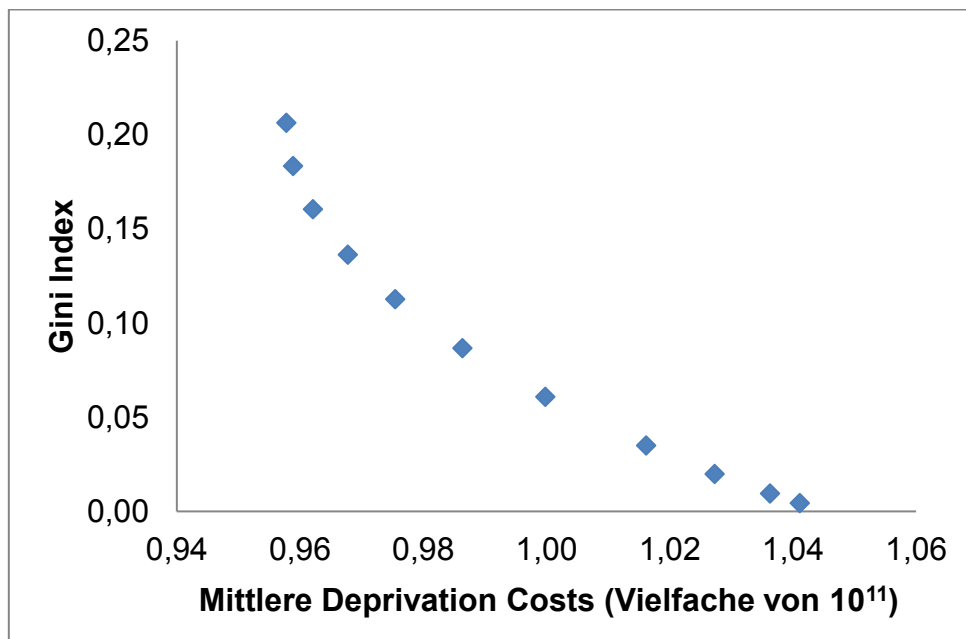


Abbildung 12: Grafische Darstellung der Wertepaare aus Tabelle 8 für die verschiedenen λ -Einstellungen (von links nach rechts).

Aus Tabelle 8 und Abbildung 12 ist erkennbar, dass durch die Erhöhung der Gewichtung des Fairnessaspekts von $\lambda = 0$ zu $\lambda = 0,5$ eine Reduktion des Gini Indizes

um knapp 98% erzielt werden kann. Dies kann zum Preis eines einigermaßen kleinen Anstiegs der Deprivation Costs von 8,7% erreicht werden.

Wie an den Ergebnissen erkennbar ist, hat die Berechnung der Kosten d_k einen Einfluss auf die Deprivation Costs und den Gini Index der Deprivation Costs. Bei der Berechnung mit Variante 2, also einer schwächeren Gewichtung der Steilheit des Geländes, sinken sowohl die Deprivation Costs, als auch der Gini Index, die Verteilung wird also fairer. Allerdings hat bei dieser Variante die Gewichtung des Fairnessaspekts in der Zielfunktion mit steigendem Exponenten p einen stärkeren Einfluss auf die relative Veränderung der Deprivation Costs und des Gini Indizes.

4.3.3. WEITERE EXPERIMENTELLE ANALYSEN

Um die Wahl der PSO-Parameter zu untermauern, wurden weitere Analysen durchgeführt, bei denen diese variiert wurden. Dabei wurden die restlichen Einstellungen des Optimierungsmodells gleich belassen, um vergleichbare Ergebnisse zu erhalten. Hierfür wurden exemplarisch die Lieferkosten d_k nach Variante 1, ein Exponent $p = 2$ und $\lambda = 0,5$ verwendet.

In Abbildung 13 sind die Ergebnisse für verschiedene Schwarmgrößen dargestellt. Das Abbruchkriterium wurde dabei bei 200 Iterationen belassen. Die Schwarmgrößen wurden im Intervall [5, 75] variiert, jeweils in Schritten von 5 Partikeln. Zu erkennen ist, dass die gewählte Schwarmgröße von 35 Partikeln sehr gute Ergebnisse erzielt und sehr nahe an den optimalen Zielfunktionswert kommt. Im speziellen Fall beträgt die Differenz absolut 200 und relativ 0,0007% zum besten Wert, der mit Schwarmgrößen von 55, 65 und 70 Partikeln gefunden wurde.

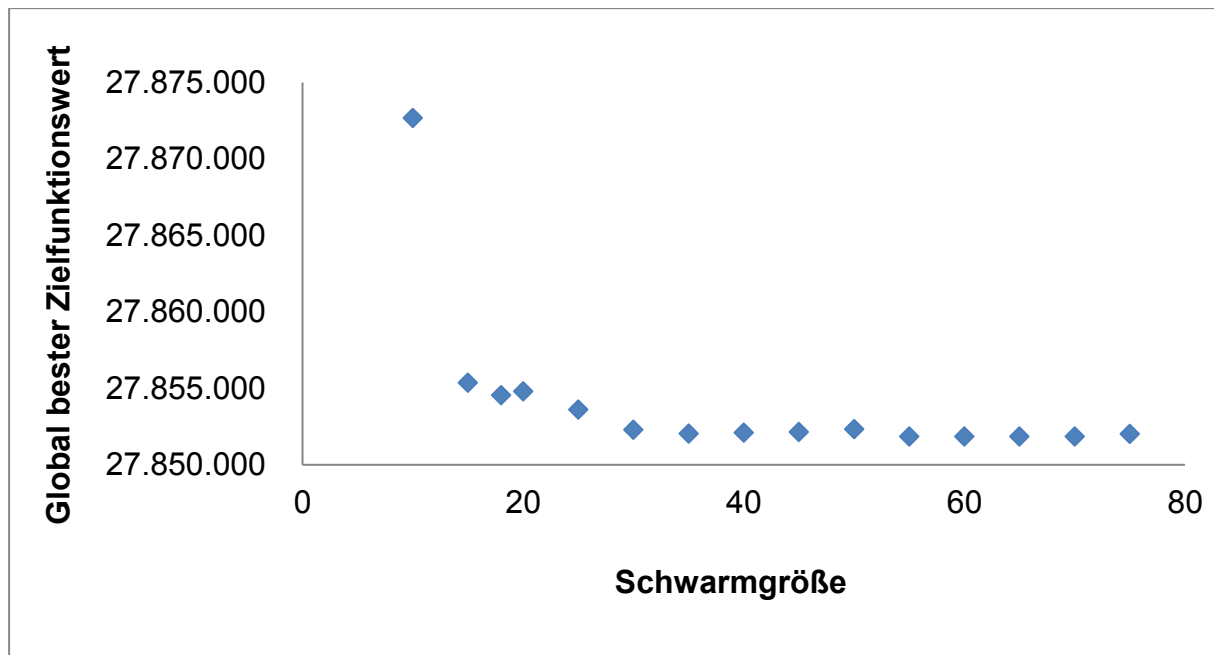


Abbildung 13: Global bester Zielfunktionswert in Abhängigkeit der Schwarmgröße für 200 Iterationen.

In Abbildung 14 ist die Dauer der Berechnungszeit in Abhängigkeit der Schwarmgröße skizziert. In Verbindung mit Abbildung 13 ist erkennbar, dass die gewählte Schwarmgröße von 35 Partikeln äußerst effizient und daher eine gute Wahl ist.

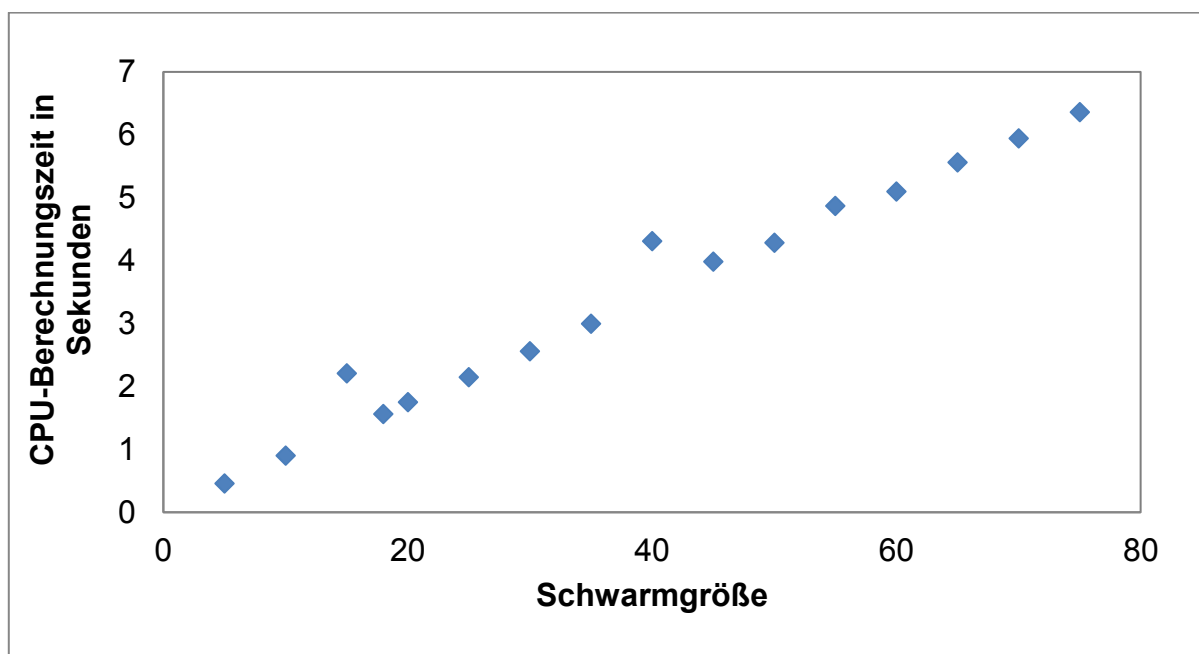


Abbildung 14: CPU-Berechnungszeit in Sekunden in Abhängigkeit der verwendeten Schwarmgröße für 200 Iterationen.

In Abbildung 15 und 16 sind die Ergebnisse für eine variierende Anzahl an Iterationen dargestellt. In Abbildung 15 ist der global beste Zielfunktionswert in Abhängigkeit der CPU-Berechnungszeit in Sekunden dargestellt, welche über die Modifikation der Anzahl an Iterationen gesteuert wurde. In Abbildung 16 ist der global beste Zielfunktionswert direkt in Abhängigkeit der Anzahl an Iterationen dargestellt. Die Schwarmgröße wurde dabei bei 35 Partikeln belassen. Die Anzahl der Iterationen wurde in verschiedenen großen Schritten in einem Bereich von [100, 1000] gewählt. Die optimale Lösung wurde bereits in einem Bereich zwischen 400 und 450 Iterationen gefunden.

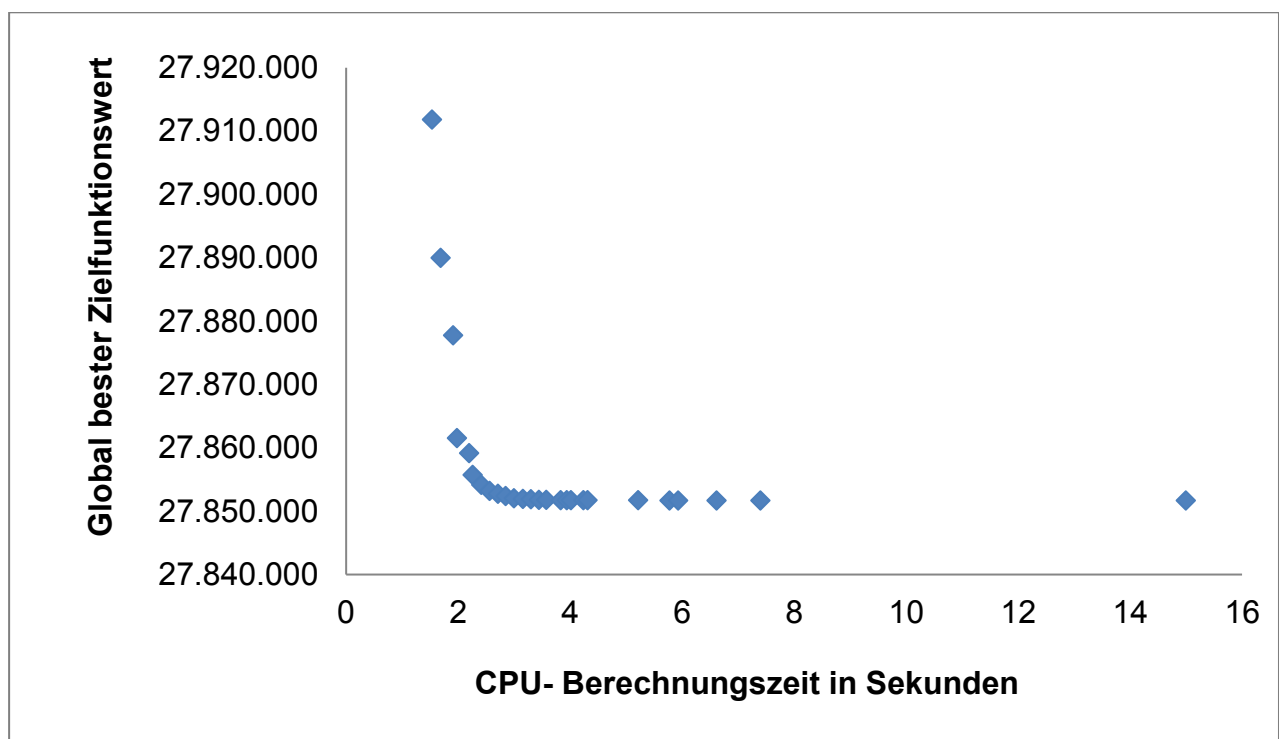


Abbildung 15: Global bester Zielfunktionswert in Abhängigkeit der Programmlaufzeit in Sekunden für verschiedene Anzahl an Iterationen zwischen 100 und 1000 (von links nach rechts).

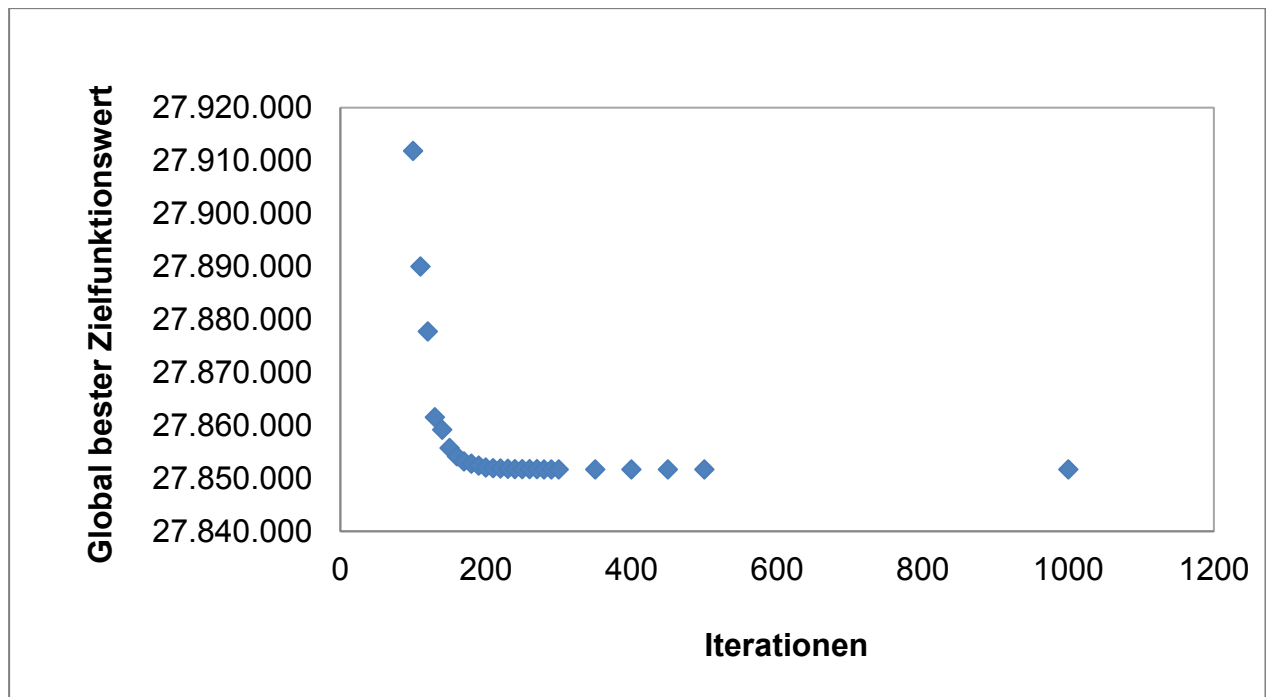


Abbildung 16: Global bester Zielfunktionswert in Abhängigkeit der Anzahl an Iterationen.

Aus Abbildung 16 ist zu erkennen, dass das gewählte Abbruchkriterium von 200 Iterationen sehr gute Ergebnisse erzielt und sehr nahe an den optimalen Zielfunktionswert kommt. Im speziellen Fall beträgt die Differenz absolut 360 und relativ 0,0013% zum besten Wert, der ab knapp 450 Iterationen gefunden wurde. In Verbindung mit Abbildung 15 zeigt sich, dass ein Abbruchkriterium von 200 Iterationen in Bezug auf die Programmlaufzeit sehr effizient ist und daher eine gute Wahl darstellt.

5. FAZIT

Durch das Konzept der Deprivation Costs von Holguín-Veras et al. [10] konnte die Forschung auf dem Gebiet der Humanitären Logistik einen wesentlichen Fortschritt verzeichnen. Allerdings zeigen die Ergebnisse des utilitaristischen Entscheidungsmodells, dass dadurch beliebig ungerechte Lösungen zustande kommen können. In der vorliegenden Arbeit wurde gezeigt, dass durch die Erweiterung des Konzepts um den Fairnessaspekt die Ergebnisse deutlich verbessert werden können und eine fairere Verteilung von Hilfsgütern ermöglicht wird. Die Lösung des komplexen Optimierungsproblems stellte sich als herausfordernd dar. Durch die

Implementierung der Partikelschwarmoptimierung konnte das Entscheidungsmodell allerdings effizient gelöst werden.

Zusammenfassend lässt sich sagen, dass das vorgestellte Optimierungsmodell in der Lage ist, Hilfsoperationen zu verbessern und die Verteilung gleichmäßiger zu gestalten. Außerdem zeigen die Ergebnisse, dass die Partikelschwarmoptimierung eine gut geeignete Methode ist, Optimierungsprobleme dieser Art zu lösen. Der präsentierte Lösungsalgorithmus ist in der Lage die Optima solch komplexer Zielfunktionen zu finden.

Unter den gewählten Annahmen für die Form der Deprivation Intensitätsfunktion und den Parametern des Logistikmodells erzielt das vorgestellte Optimierungsmodell sehr gute Ergebnisse und zeigt, dass durch verhältnismäßig geringe Zunahmen der Deprivation Costs erheblich fairere Lösungen generiert werden können. Ob das Modell für andere Formen von Deprivation Intensitätsfunktionen und Parameter ähnliche Ergebnisse erzielt, ist in künftigen experimentellen Analysen zu erforschen. Jedoch sind die gewonnenen Ergebnisse vielversprechend.

LITERATURVERZEICHNIS

- [1] H. Abidi, S. Zinnert & M. Klumpp. Humanitäre Logistik – Status quo und wissenschaftliche Systematisierung. *Schriftenreihe Logistikforschung Band 18, FOM Hochschule für Ökonomie & Management Essen*, 2011.
- [2] A. Blecken. Humanitarian Logistics: Modelling Supply Chain Processes of Humanitarian Organisations. *Haupt Verlag AG Berne, Kuehne Foundation book series on logistics*, 18, 2010.
- [3] M. Clerc & J. Kennedy. The particle swarm – explosion, stability, and convergence in a multidimensional complex space. *IEEE Transaction on Evolutionary Computation*, 6(1): 58-73, 2002.
- [4] M. Clerc. Standard Particle Swarm Optimisation. *Open access archive HAL*, 2012.
- [5] Y. del Valle, G.K. Venayagamoorthy, S. Mohagheghi, J.-C. Hernandez & R. G. Harley. Particle Swarm Optimization: Basic Concepts, Variants and Applications in Power Systems. *IEEE Transaction on Evolutionary Computation*, 12(2): 171-195, 2008.
- [6] R. C. Eberhart & Y. Shi. A modified particle swarm optimizer. *IEEE International Conference on Evolutionary Computation Proceedings*: 69-73, 1998.
- [7] R. C. Eberhart & Y. Shi. Comparing inertia weights and constriction factors in particle swarm optimization. *Proceedings of the Congress on Evolutionary Computation*, 1: 84-88, 2000.
- [8] R. C. Eberhart & Y. Shi. Tracking and optimizing dynamic systems with particle swarms. *Proceedings of the Congress on Evolutionary Computation*, 1: 94-100, 2001.
- [9] W. J. Gutjahr & S. Fischer. Equity and deprivation costs in humanitarian logistics. *European Journal of Operational Research*, 270: 185-197, 2018.

- [10] J. Holguín-Veras, N. Pérez, M. Jaller, L. N. Van Wassenhove & F. Aros-Vera. On the appropriate objective function for post-disaster humanitarian logistics models. *Journal of Operations Management*, 31(5): 262-280, 2013.
- [11] J. Holguín-Veras, J. Amaya-Leal, V. Cantillo, L. N. Van Wassenhove, F. Aros-Vera & M. Jaller. Econometric estimation of deprivation cost functions: A contingent valuation experiment. *Journal of Operations Management*, 45: 44-56, 2016.
- [12] I. Johnsson. Tracking relief Aid: A spatial analysis of aid distribution in Nepal after the 2015 Gorkha earthquake. *Thesis, Lund University, Sweden*, 2016.
- [13] J. Kennedy & R. C. Eberhart. Particle swarm optimization. *Proceedings of the ICNN - International conference on neural networks*, 4: 1942-1948, 1995.
- [14] J. Kennedy & R. Mendes. Population Structure and Particle Swarm Performance. *Proceedings of the Congress on Evolutionary Computation*, 2: 1671-1676, 2002.
- [15] R. Mendes, J. Kennedy & J. Neves. Watch thy neighbor or how the swarm can learn from its environment. *Proceedings of the IEEE Swarm Intelligence Symposium*: 88-94, 2003.
- [16] R. Mendes. Population Topologies and Their Influence in Particle Swarm Performance. *PhD Thesis, Departamento de Informática Escola de Engenharia Universidade do Minho, Portugal*, 2004.
- [17] R. Poli, J. Kennedy & T. Blackwell. Particle swarm optimization: An overview. *Swarm Intell*, 1: 33-57, 2007.
- [18] R. M. Tomasini & L. Van Wassenhove. Humanitarian Logistics. *Palgrave Macmillan, Basingstoke UK*, 2009.

ANHANG

C++ CODE

Nachfolgend ist der vollständige Code präsentiert. Die Partikelschwarmoptimierung wurde mit C++ programmiert und im Microsoft Visual Studio Community 2015 implementiert. Der Code sowie die Kommentare sind in englischer Sprache verfasst.

Im unten angeführten Code sind die Kosten d_k nach Variante 1 berechnet und Parametereinstellungen von einer Schwarmgröße = 35 Partikeln sowie ein Abbruchkriterium von 200 Iterationen verwendet.

Blaue, graue und violette Ausdrücke sind programmspezifische Ausdrücke bzw. Anweisungen. Bei dem grünen Text, der immer mit „//“ beginnt, handelt es sich um Kommentare, die dazu dienen, die Lesbarkeit und das Verständnis zu verbessern.

```
1  // library includes
2  #include <conio.h> //required for function _getch()
3  #include <iostream> //standard input/output stream library
4  #include <fstream>
5  #include <sstream>
6  #include <vector>
7  #include <cmath>
8  #include <random>
9  #include <ctime>
10 using namespace std;
11
12 // ***** global variables, structures, constants and types *****
13 // fill with problem specific data
14 int dimension_D = 14; // dimension of solution: demand points
15 int swarmsize = 35;
16 int termination_after_iterations = 200; // number of iterations - stopping criterion
17 ofstream ofile;
18
19 // parameters & weights for velocity update formula - chosen by the user
20 double inertia_weight = 0.7298; // inertia weight = 1 corresponds to the original PSO without
    inertia weight
21 double phi1 = 1.49618;
22 double phi2 = 1.49618;
23
24 vector<int> number_of_beneficiaries_w = { 25, 106, 83, 79, 63, 131, 39, 59, 99, 30, 66, 27, 59,
    97 }; // vector to store the number of beneficiaries for each demand point k (start with index 0)
25 int total_number_of_beneficiaries_N = 0; // total number of beneficiaries = sum over k
```

```

26 vector<double> driving_cost_d = {79.4, 605.5, 857.7, 1013.8, 150.8, 492.5, 140.3, 241.9, 513.1,
    367.3, 497.6, 452.8, 283.9, 771.6 }; // vector to store the driving costs from the depot for each
    demand point k (start with index 0)
27 double lambda = 0.5; // between 0 and 1/2 (0 = no consideration of equity); to tune the degree of
    influence given to the equity consideration
28 double budget = 1; // available budget per time unit
29
30 // deprivation function  $g(t) = g_1 * t^p$ 
31 double g1 = 2; // constant multiplier in deprivation function
32 double p = 3; // power in the deprivation function
33
34 struct Particle {
35     int index_i; // index of the particle (start with 0; < swarmsize)
36     vector<double> position_x; // current position of the particle - D-dimensional vector
37     vector<double> velocity_v; // current velocity of the particle - D-dimensional vector
38     vector<double> bestposition_pi; // best position found so far by the particle – D-dimensional
        vector
39     vector<double> particlesbestsolution_pbest; // fitness value [0], deprivation costs [1] & gini
        index [2] for the best position pi of the particle
40
41 };
42
43 struct Global_best {
44     vector<double> globalbest_pg; // best position found so far by all particles - D-dimensional
        vector
45     vector<double> globalbestsolution_pbestg; // fitness value [0], deprivation costs [1] & gini
        index [2] for the best overall position so far
46     int index_particle_globalbest; // index of the particle that found the global best
47
48 } g_best;
49
50 // ***** functions and function prototypes *****
51 double calcFitness(vector<double> auxiliary_variables_zk, double & my, double & gini_index) {
52     vector<double> deliveries_xk (dimension_D, 0);
53     double fitness = 0; // fitness = my + 2 * lambda * my * gini index
54     my = 0;
55     gini_index = 0;
56
57     // transform auxiliary variables zk to the delivery variables xk
58     for (int k = 0; k < dimension_D; k++) {
59         for (int r = 0; r < dimension_D; r++) deliveries_xk[k] +=
            exp(auxiliary_variables_zk[r]);
60         deliveries_xk[k] = (budget / driving_cost_d[k]) * (exp (auxiliary_variables_zk[k]) /
            deliveries_xk[k]);
61         ofile << deliveries_xk[k] << "\t";
62     }
63
64     // calc OF value for a given solution
65     for (int k = 0; k < dimension_D; k++)
66         my += (number_of_beneficiaries_w[k] / pow(deliveries_xk[k], p));
67     my = my * (g1 / (total_number_of_beneficiaries_N * (p + 1)));
68

```

```

69     for (int k = 0; k < dimension_D; k++) {
70         for (int k_prime = 0; k_prime < dimension_D; k_prime++)
71             gini_index += number_of_beneficiaries_w[k] *
                        number_of_beneficiaries_w[k_prime] * abs(1 / pow(deliveries_xk[k], p) - 1
                        / pow(deliveries_xk[k_prime], p));
72     }
73     gini_index = gini_index * (g1 / (2 * total_number_of_beneficiaries_N *
total_number_of_beneficiaries_N * (p + 1) * my));
74
75
76     fitness = my + 2 * lambda * my * gini_index;
77     return fitness;
78 }
79
80 // ***** main function *****
81 int main() {
82
83     clock_t start = clock();
84     // write results into a text file
85     ofile.open("Nepal_lambda_0_5_p_3_original_dk_time_measure.txt", ios::app);
86     ofile << "Dimension:\t" << dimension_D << endl;
87     ofile << "nSwarmsize:\t" << swarmsize << endl;
88     ofile << "Iterations:\t" << termination_after_iterations << endl;
89
90     ofile << "Inertia Weight: \t" << inertia_weight << endl;
91     ofile << "Phi 1: \t" << phi1 << endl;
92     ofile << "Phi 2: \t" << phi2 << endl;
93     ofile << "Number of beneficiaries w[k] (vector): \t";
94
95     for (int k = 0; k < dimension_D; k++) {
96         ofile << number_of_beneficiaries_w[k] << "\t";
97         total_number_of_beneficiaries_N += number_of_beneficiaries_w[k];
98     }
99
100    ofile << "nTotal number of beneficiaries N:\t" << total_number_of_beneficiaries_N << endl;
101    ofile << "Driving costs d[k] (vector): \t";
102    for (int k = 0; k < dimension_D; k++) {
103        ofile << driving_cost_d[k] << "\t";
104    }
105    ofile << "nLambda: \t" << lambda << endl;
106    ofile << "Budget: \t" << budget << endl;
107    ofile << "Objective function: g1 = " << g1 << ", p = " << p << endl;
108
109
110
111    default_random_engine generator;
112    uniform_real_distribution<double> distribution_phi1(0, phi1);
113    uniform_real_distribution<double> distribution_phi2(0, phi2);
114
115    vector<Particle> particles;
116    Particle particle;
117
118    double fitnessvalue = 0;

```

```

119     double deprivation_costs = 0;
120     double gini_index = 0;
121
122     // Initialization of Particles
123     for (int i = 0; i < swarmsize; i++) {
124         particle.index_i = i;
125         ofile << "\nInitial position of particle " << i << "\t";
126         for (int k = 0; k < dimension_D; k++) {
127             uniform_real_distribution<double> position_distrib_k(-1, 1);
128             particle.position_x.push_back(position_distrib_k(generator)); // randomly
                                   generated (uniformly distributed)
129             uniform_real_distribution<double> velocity_distrib_k(-1, 1);
130             particle.velocity_v.push_back(velocity_distrib_k(generator)); // randomly
                                   generated (uniformly distributed)
131             ofile << particle.position_x[k] << "\t";
132         }
133
134         particle.bestposition_pi = particle.position_x; // best position so far is initialized with
                                   the current position
135         particle.particlesbestsolution_pbest.push_back(calcFitness(particle.bestposition_pi,
                                   deprivation_costs, gini_index)); // calc fitness value of current position
136         particle.particlesbestsolution_pbest.push_back(deprivation_costs); // store
                                   according deprivation costs
137         particle.particlesbestsolution_pbest.push_back(gini_index); // store according gini
                                   index
138         ofile << "Fitness value: " << particle.particlesbestsolution_pbest [0] << endl;
139         ofile << "Deprivation costs: " << particle.particlesbestsolution_pbest[1] << endl;
140         ofile << "Gini index: " << particle.particlesbestsolution_pbest[2] << endl;
141
142
143         particles.push_back(particle); // add particle to the vector of all particles
144         particle.position_x.clear();
145         particle.velocity_v.clear();
146         particle.bestposition_pi.clear();
147         particle.particlesbestsolution_pbest.clear();
148
149     }
150
151     // Initialization of Global_best
152     for(int a = 0; a < 3; a++) g_best.globalbestsolution_pbestg.push_back(particles[0].
                                   particlesbestsolution_pbest[a]); // set global best to pbest of the first particle (fitness,
                                   deprivation costs and gini index)
153     g_best.globalbest_pg = particles[0].bestposition_pi;
154     g_best.index_particle_globalbest = 0;
155     for (int i = 1; i < swarmsize; i++) {
156         if (particles[i].particlesbestsolution_pbest[0] <
                                   g_best.globalbestsolution_pbestg[0]) {
157             // update global best
158             for (int a = 0; a < 3; a++) g_best.globalbestsolution_pbestg[a] =
                                   particles[i]. particlesbestsolution_pbest[a]; // update fitness,
                                   deprivation costs and gini index of the best solution
159             g_best.globalbest_pg = particles[i].bestposition_pi;

```

```

160         g_best.index_particle_globalbest = i;
161     }
162 }
163
164 ofile << "\nGlobal best position of particle " << g_best.index_particle_globalbest << " : ";
165 for (int k = 0; k < dimension_D; k++) ofile << g_best.globalbest_pg[k] << "\t";
166 ofile << "\nGlobal best fitness: " << g_best.globalbestsolution_pbestg[0] << endl;
167 ofile << "\nGlobal best deprivation costs: " << g_best.globalbestsolution_pbestg[1] << endl;
168 ofile << "\nGlobal best gini index: " << g_best.globalbestsolution_pbestg[2] << endl;
169
170
171 // iterations
172 for (int counter = 0; counter < termination_after_iterations; counter++) {
173     ofile << "\nIteration " << counter + 1 << " : " << endl;
174     // for each particle
175     for (int i = 0; i < swarmsize; i++) {
176         ofile << "Particle " << i << ":\t";
177         // component-wise
178         for (int k = 0; k < dimension_D; k++) {
179             // update velocity in component k
180             particles[i].velocity_v[k] = inertia_weight *
particles[i].velocity_v[k] + distribution_phi1(generator)*
(particles[i].bestposition_pi[k] - particles[i].position_x[k]) +
distribution_phi2(generator) * (g_best.globalbest_pg[k]
particles[i].position_x[k]);
181
182             // update position in component k
183             particles[i].position_x[k] += particles[i].velocity_v[k];
184
185             ofile << particles[i].position_x[k] << "\t";
186         } ofile << endl;
187
188
189
190         // check if fitness value of new position is better than pbest
191         fitnessvalue = calcFitness(particles[i].position_x, deprivation_costs,
gini_index);
192
193         ofile << "Fitness value:\t" << fitnessvalue << endl;
194         ofile << "Deprivation costs:\t" << deprivation_costs << endl;
195         ofile << "Gini index:\t" << gini_index << endl << endl;
196
197         if (fitnessvalue < particles[i].particlesbestsolution_pbest[0]) {
198             // update pbest
199             particles[i].particlesbestsolution_pbest[0] = fitnessvalue;
200             particles[i].particlesbestsolution_pbest[1] = deprivation_costs;
201             particles[i].particlesbestsolution_pbest[2] = gini_index;
202             particles[i].bestposition_pi = particles[i].position_x;
203             ofile << "Pbest updated. " << endl << endl;
204             // check if new pbest is better than gbest
205             if (particles[i].particlesbestsolution_pbest[0] <
g_best.globalbestsolution_pbestg[0]) {

```

```

206         for(int a=0; a < 3; a++)
            g_best.globalbestsolution_pbestg[a] =
207             particles[i].particlesbestsolution_pbest[a];
208         g_best.globalbest_pg = particles[i].bestposition_pi;
209         g_best.index_particle_globalbest = i;
210         ofile << "Gbest updated. " << endl << endl;
211     }
212 }
213
214 }
215 }
216
217 ofile << "\nBest solution found: " << endl;
218 ofile << "\nGlobal best position of particle " << g_best.index_particle_globalbest << " : ";
219 fitnessvalue = calcFitness(g_best.globalbest_pg, deprivation_costs, gini_index);
220 for (int k = 0; k < dimension_D; k++) ofile << g_best.globalbest_pg[k] << "\t";
221
222 ofile << "\nGlobal best fitness: " << std::scientific << g_best.globalbestsolution_pbestg[0]
    << endl;
223 ofile << "\nGlobal best deprivation costs: " << g_best.globalbestsolution_pbestg[1] << endl;
224 ofile << "\nGlobal best gini index: " << g_best.globalbestsolution_pbestg[2] << endl;
225
226 double gini_index_xk = 0;
227 double average = 0;
228 vector<double> bestsolution_xk_values(dimension_D, 0);
229 // transform auxiliary variables zk to the delivery variables xk
230 for (int k = 0; k < dimension_D; k++) {
231     for (int r = 0; r < dimension_D; r++) bestsolution_xk_values[k] += exp
        (g_best.globalbest_pg[r]);
232     bestsolution_xk_values[k] = (budget / driving_cost_d[k]) *
        (exp(g_best.globalbest_pg[k]) / bestsolution_xk_values[k]);
233 }
234
235 for (int k = 0; k < dimension_D; k++) {
236     average += bestsolution_xk_values[k];
237     for (int k_prime = 0; k_prime < dimension_D; k_prime++)
238         gini_index_xk += abs(bestsolution_xk_values[k] -
            bestsolution_xk_values[k_prime]);
239 }
240 average /= dimension_D;
241 gini_index_xk = gini_index_xk * (1 / (2 * total_number_of_beneficiaries_N *
    total_number_of_beneficiaries_N * average));
242
243 ofile << "\nGini Index of the best solution (of xk):" << gini_index_xk << endl << endl;
244
245 clock_t clockcount = clock() - start;
246 ofile << "Time in Seconds: " << clockcount / (double)CLOCKS_PER_SEC << endl << endl;
247
248 ofile.close();
249
250

```

```
251
252     cout << "Execution successful";
253
254
255
256     _getch(); // read from keyboard to show output until any key is pressed
257     return EXIT_SUCCESS; // report success to operating system
258 }
```

ABSTRAKT

Das Konzept der Deprivation Costs stellt einen Fortschritt in der Humanitären Logistik dar und trägt wesentlich dazu bei, Hilfsmissionen fairer zu gestalten um keine Bevölkerungsgruppen, beispielsweise aufgrund erschwerter Erreichbarkeit, zu benachteiligen. Durch die explizite Berücksichtigung von Fairness in der Zielfunktion kann die Ungerechtigkeit stark reduziert werden. Dies wird durch Minimierung einer gewichteten Summe der Deprivation Costs und Ginis mittlerer absoluter Abweichung der Deprivation Costs unter Budgetnebenbedingungen erreicht. Daraus resultiert ein nichtkonvexes nichtlineares Optimierungsproblem, das mittels Partikelschwarmoptimierung gelöst wird. Der Lösungsalgorithmus wird für einen realen Anwendungsfall implementiert. Dafür werden Daten des Erdbebens in Nepal im Jahr 2015 verwendet, da hier die ungerechte Verteilung von Hilfsgütern kritisiert wurde. Die Ergebnisse zeigen, dass das Modell in der Lage ist die Ungerechtigkeit in der Humanitären Logistik zu verringern und effiziente Lösungen zu produzieren.

SCHLAGWÖRTER

Humanitäre Logistik, Fairness, Deprivation Costs, Humanitarian Logistics, Equity, Gini Index, Partikelschwarmoptimierung, Particle Swarm Optimization, C++ Code