Modul Metaheuristiken

Projektbericht über die Anwendung und Weiterentwicklung eines evolutionären Algorithmus zur Lösung eines Travelling Salesman Problems

Maximilian Stumpf und Markus Wünsche

Abstract
Evolutionäre Algorithmen sind eines von vielen Beispielen für Metaheuristiken, die von der Natur inspiriert wurden. Innerhalb dieser Arbeit wird ein vorgegebener Algorithmus zur Lösung eines zweidimensionalen Travelling Salesman Problems (TSP) in zwei Phasen angewandt und weiterentwickelt. Dazu werden in Phase I über Anpassung der Parameter möglichst gute Individuallösungen für die jeweiligen Dimensionen erfasst und in Phase II auf deren Grundlage einer Pareto-Front über beide Zielsetzungen angenähert. Zur Evaluation der Ergebnisse erfolgen dabei Anpassungen am Code, sowohl zur Strukturierung der Auswertung als auch der Integration alternativer Selektionsmethoden und Annäherung an den Strength Pareto Evolutionary Algorithm 2 (SPEA2, Zitler et al., 2002). Dabei werden neben den gesuchten Lösungen und Ergebnissen aus der Anwendung des Algorithmus auf das Ausgangsproblems auch weiteres Anpassungspotential des Codes identifiziert.

1 Aufgabenstellung und Zielsetzung

Metaheuristiken können als übergeordnete, nicht problemspezifische Methoden definiert werden, mit denen der Suchprozess in heuristischen Verfahren zur Lösung von Optimierungsproblemen gesteuert wird. Sie kommen immer dann zur Anwendung, wenn sehr komplexe Probleme mit einer großen Anzahl an Lösungsmöglichkeiten oder sehr restriktiven Bedingungen gegeben sind. Dabei soll der Lösungsraum effizient durchsucht werden und mit möglichst geringem Zeit- und Rechenaufwand eine hinreichend gute Lösung gefunden werden. Verschiedene Prinzipien werden hierbei angewendet, um lokale Optima zu überwinden und bessere Lösungen zu ermitteln. Durch Diversifikation, dem Erforschen eines möglichst großen Bereichs des Lösungsraums, und Intensivierung, dem Ausbeuten eines Teilbereichs zur Suche möglichst guter Lösungen, werden fortlaufend bessere Lösungen erzielt.

1.1 Problemstellung und Projektziele

Im Rahmen dieses Projektberichtes wird die Anwendung und Weiterentwicklung einer Metaheuristik in Form eines evolutionären Algorithmus dokumentiert und erläutert sowie die Ergebnisse bei der Applikation auf ein Travelling Salesman Problem (TSP) vorgestellt. Die Problemstellung beinhaltet ein symmetrisches TSP mit 25 Orten und zwei Zielkriterien. Dabei sollen bei der Erstellung der Rundreise als Lösung sowohl die zurückgelegte Distanz als auch ein Stresslevel auf Basis des zu erwarteten Staus zwischen den einzelnen Punkten minimiert werden. Da bereits bei 25 Orten ein sehr großer Lösungsraum existiert, eignet sich dieses Problem hervorragend für die Anwendung eines heuristischen Verfahrens. Ein vorgegebener Algorithmus aus einem bestehenden Lösungsframework (https://github.com/ezstoltz/genetic-algorithm) wird in einer ersten Phase benutzt, um für die Problemstellung in beiden Zielkriterien möglichst gute Lösungen zu finden. Dabei werden verschiedene Parameter des evolutionären Algorithmus iterativ angepasst und deren Auswirkungen auf die Lösungen

untersucht. In der zweiten Phase des Projekts wird das bestehende Framework erweitert, um beide Zieldimensionen in einem mehrkriteriellen Verfahren zu optimieren und eine Lösungsmenge an Alternativlösungen in Form einer Pareto-Front zu erstellen. Auch hier werden Parameteränderungen und deren Auswirkungen untersucht. Durch einen differenzierteren Umgang mit den initialen Populationen sowie einer Anpassung der Selektionsmethode als turnierbasiertes Vorgehen wird der vorliegende Algorithmus zudem weiter angepasst und analysiert. Das Ziel des Projekts ist schlussendlich, den Algorithmus möglichst optimal für die Aufgabenstellung zu gestalten und gute Ergebnisse mit den heuristischen Verfahren zu erzielen.

1.2 Evolutionäre Algorithmen

Häufig sind Metaheuristiken von der Natur abgeleitet, so auch evolutionäre Algorithmen, die Gegenstand dieses Projektes sind. Hier werden, inspiriert von der Biologie, mehrere Schritte eines evolutionären Prozesses auf eine initiale Lösungsmenge (der Population) angewendet, um immer wieder neue Lösungen zu generieren und diese auf ihre Nähe zur Optimalität zu überprüfen. Vorteilhaft ist hier die gleichzeitige Betrachtung von mehreren Lösungsansätzen im Verlauf des Verfahren, anstatt der Fokussierung auf nur eine Lösung. Diese Individuen der Population werden in mehreren Schritten miteinander kombiniert, verändert und bewertet. Bei der *Selektion* wird bestimmt, welche Lösungen für die nächsten Stufen des Algorithmus übernommen werden. Individuen mit einer besseren Lösung, welche über eine Fitness-Funktion bestimmt werden kann, haben bessere Chancen übernommen zu werden und Nachkommen für die nächste Generation der Population zu bilden. Dabei sind verschiedene Verfahren möglich, unter anderem auch eine turnierbasierte Selektion, die in Phase II des Projekts angewendet wird. Nach *Selektion* der besten Individuen werden durch die *Rekombination* Nachkommen gebildet. Auch hier kann durch verschiedene Ansätze der Algorithmus angepasst werden und die Art und Weise der Bildung von neuen Lösungsansätzen gesteuert werden. Basierend auf einem oder mehreren *Crossover*-Punkten werden die Elemente zweier Lösungen hierbei kombiniert und eine neue Lösung kreiert. Durch *Mutation* der so neu erstellten Lösungen kann mit einer bestimmten Wahrscheinlichkeit der Lösungsraum zudem weiter diversifiziert werden. Hier werden Elemente einer Lösung angepasst oder getauscht. Nach jedem Durchlauf dieser drei Schritte ist eine neue Generation erzeugt. Die daraus entstandenen Lösungen werden anschließend bewertet und je nach Erreichen einer Terminierungsbedingung weiter im evolutionären Prozess benutzt oder als finale Lösungen ausgegeben.

1.3 Umsetzung des Algorithmus

Das vorliegende Programm für den evolutionären Algorithmus ist in der Programmiersprache Python umgesetzt und kann jeweils für beide Zielkriterien, Distanz und Stress, als auch in einer erweiterten Form für die Optimierung beider Zielsetzungen gleichzeitig angewendet werden. Die Fitness-Funktion zur Evaluation der Lösungen berechnet sich dabei als 1/Distanz bzw. 1/Stress. Die Selektion der Individuen für die Rekombination erfolgt mit einer Fitness-basierten Roulette-Wheel-Selection, es ist jedoch auch ein Elitismus implementiert, mit dem eine bestimmte Anzahl der besten Lösungen immer übernommen wird. Beim Crossover-Verfahren wird nach Bestimmung zweier zufälliger Stellen ein Subset des ersten Parent-Strings gebildet, und die neue Lösung dann durch Auffüllen mit den übrigen Werten in der Reihenfolge des zweiten Parent-Strings erstellt. Als Mutation wird ein Swap verwendet, hierbei tauschen mit einer bestimmten geringen Wahrscheinlichkeit zwei Städte den Platz in einer Lösung. Die neuen Generationen werden jeweils gebildet, indem die aktuellen Routen mit der Fitness-Funktion evaluiert werden, das Selektionsverfahren und Elitismus auf die Population angewendet werden und so der „Mating Pool“ erstellt wird. Anschließend werden durch Applikation des Crossover-Verfahrens die

Nachkommen gebildet und die Mutation wird mit der definierten Wahrscheinlichkeit auf sie angewendet.

Bei Ausführung des Programms können die folgenden Parameter angepasst werden, um das Verhalten des evolutionären Algorithmus anzupassen und unterschiedliche Lösungsräume zu erstellen. Durch Definition der *popSize* wird festgelegt, wie viele Individuen jeweils Teil einer Population sind und den evolutionären Prozess durchlaufen. Mit *eliteSize* wird bestimmt, wie viele der jeweils besten Lösungen unabhängig vom Selektionsverfahren für die Rekombination übernommen werden. Durch Anpassen der *mutationRate* kann definiert werden, mit welcher Wahrscheinlichkeit nach der Bildung einer neuen Generation die Mutation, also der Tausch zweier Städte in der Lösung, angewendet wird. Mit dem Parameter *generations* kann abschließend festgelegt werden, wie viele Generationen durch den Algorithmus erstellt werden, bzw. wie oft der evolutionäre Prozess iterativ durchlaufen wird.

2 Studienteil Phase I – Untersuchung mit bestehendem Lösungsframework

Der erste Schritt in der Modellierung des Problems lag in der Anwendung des Algorithmus mit vorgegebenen Parametern auf die jeweils einzelnen Dimensionen des Problems, Distanz und Stress.

2.1 Anpassung des Programmcodes

Die Abhängigkeit des Ergebnisses von den Aufrufparametern *popSize*, *eliteSize*, *mutationRate* und *generations* konnte dabei nicht eingeschätzt werden, weshalb eine heuristische Untersuchung der einzelnen Parameter nötig wurde. Dabei zeigte sich, dass aufgrund der Laufzeit des Algorithmus eine Automatisierung des Prozesses hilfreich sein würde. Um dies zu erreichen wurden mehrere Anpassungen am Code vorgenommen: Um grafische Ausgaben beizubehalten und mit den restlichen Ergebnissen zu vereinen wurde für jede Konfiguration von Parametern eine PDF-Datei automatisiert angelegt, um jeden Schritt zu dokumentieren. Zusätzlich wurde eine Logdatei erstellt, in der immer die besten Lösungen gespeichert wurden. Im nächsten Schritt konnte dann der Kernalgorithmus über verschachteltes Setzen der Parameter innerhalb festgelegter Bandbreiten in Schleifen mit unterschiedlichen Kombinationen ausgeführt werden ohne zusätzlichen Userinput zu benötigen. Da die *print* Ausgabe für die Klasse City viele Informationen enthielt, die für die weitere Bearbeitung nicht nötig waren, wurde sie außerdem so angepasst, dass sie nur den Index des jeweiligen Objektes ausgibt. Um die Laufzeit zu verbessern, wurden zudem die Distanz- und Stressabstände zwischen den Städten nur einmal zu Beginn des Programmaufrufs berechnet und in einem entsprechenden Array gespeichert.

2.2 Parametrisierung des Algorithmus

Nach anfangs heuristischer Bestimmung eines Rahmens für die Setzung der Parameter, wurde mit Rücksicht auf die Laufzeit folgende Konfiguration gewählt:

popSize	**eliteSize**	**mutationRate**	**generations**
100, 125, 150, 200	10, 20, 30	0,015, 0.01, 0.001	500, 1000

Tabelle 1: Konfiguration der Parameter

Die Laufzeit wurde dabei in erster Linie von den Werten in *popSize* beeinflusst, wo oberhalb von 300 Individuen kaum bessere Ergebnisse erzielt wurden bei deutlich längerer Dauer. Gleichzeitig stellte sich schon nach ungefähr 100-300 Generationen das endgültige Ergebnis ein. Um zu testen ob in Verbindung mit einer kleineren Mutationsrate genauere Ergebnisse erzielt werden können, wurden *generations* auf

1000 und *mutationRate* auf 0,001 gesetzt. Zwar wurden aufgrund der randomisierten Erstellung neuer Lösungen nicht immer dieselbe Lösungsqualität erreicht, allerdings zeigte sich dennoch, dass eine Mutationsrate von 0,1 gute Ergebnisse liefert, eine Erhöhung hatte hier schlechtere Resultate zur Folge.

2.3 Ergebnisse

Die Distanzminimierung konvergierte schnell in Richtung eines bestimmten Wertes, nämlich 862.618, bei der Stressminimierung wurde sich nur langsam einem Minimum angenähert, hier lag der kleinste gefundene Wert bei 2684.4. Im späteren Verlauf konnte im Rahmen der *multiObjective* Auswertung ein Wert von 2680 ermittelt werden, allerdings stellte sich hier kein wiederkehrender Minimalwert ein. Grundsätzlich konnten mit höherer Generationen- und Populationsanzahl bessere Ergebnisse erreicht werden, allerdings zu Lasten stark steigender Laufzeiten. Die *eliteSize* trug unter sonst gleichen Bedingungen mit einem Wert von etwa 10% der Population am meisten zu einer guten Lösung bei, die Mutationsrate wurde nach mehreren Tests bei den vorgegebenen 0.01 belassen.

Objective	Min Wert	*popSize*	*eliteSize*	*mutationRate*	*generations*
1	862.618	150	20	0.01	1000
2	2682.4	200	20	0.01	1000

Tabelle 2: Beste Lösungen und deren Parameter nach Phase I

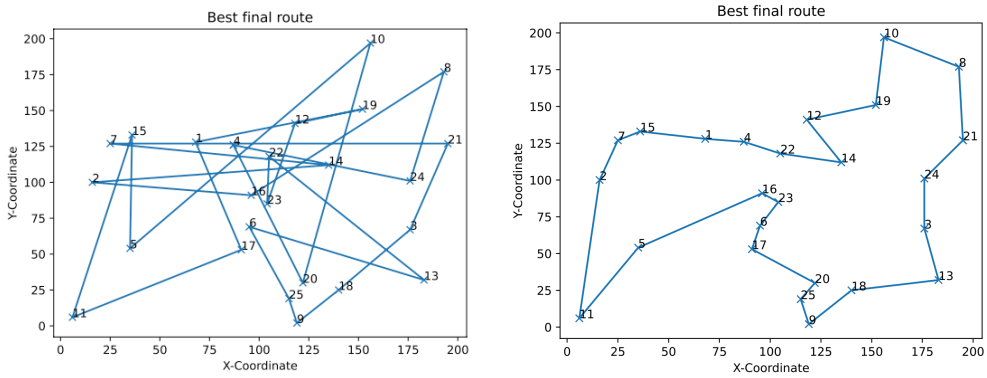

Abbildung 1: Beste Routen für Stress- und Distanzminimierung

3 Studienteil Phase II – Anpassung des bestehenden Frameworks

Der Code für die multiObjective Zielsetzung wurde umfangreich angepasst: Um den Lösungsraum zu initialisieren wurden die beiden besten ermittelten Routen aus Phase I dem Algorithmus zu Beginn übergeben. Außerdem konnte die Distanzberechnung ebenfalls ausgelagert werden, um die Laufzeit zu verbessern. Ein fehlender Aufruf der Fitnessberechnung wurde ergänzt, um einen ersten Durchlauf mit Vorgabeparametern starten zu können. Um eine alternative Selektionsmethode zu testen, wurde zusätzlich eine turnierbasierte Selektion implementiert, die über einen Parameter *k* der die Anzahl der Teilnehmer an jedem Turnier darstellt. Es wurde weiterhin versucht ein Archiv einzubauen, wie es im SPEA2 vorgesehen ist, da die allerdings zu viele Änderungen am Gesamtcode zur Folge gehabt hätte wurde davon

abgesehen. Dennoch wurde die Ausgabe der besten Routen am Ende des Algorithmus so verändert, dass sie nur die nicht-dominierten Lösungen plotted, um die Paretofront hervorzuheben.

```
if selectionMethod == 2:
    while len(selectionResults)<len(popRanked):
        competition = random.sample(popRanked, k)
        winner = competition[0]
        for i in range(0,k):
            if competition[i][1] > winner[1]:
                winner = competition[i]
        selectionResults.append(winner[0])
```

Abbildung 2: Implementierung der turnierbasierten Selektion

4 Studienteil Phase II – Untersuchung mit erweitertem Framework

Der erste Aufruf des erweiterten Frameworks mit Vorgabeparametern deutete die Paretofront schon erkennbar an. Im nächsten Schritt wurde erneut eine Parameterkomposition ermittelt die möglichst gute Ergebniss liefert und diese Paretofront verfeinert. Aufbauend auf den Erkenntnissen aus Phase I wurde die Mutationsrate bei 0.01 belassen. Da der Code aber weniger häufig ausgeführt wurde, konnten für *popSize* und *generations* deutlich höhere Werte verwendet werden, nämlich 500 und 1000. Deshalb wurde auch die *eliteSize* auf 50 gesetzt, was 10% einer *popSize* von 500 entspricht. Mit dieser Konfiguration erreichte das Programm in angemessener Laufzeit (~15 Minuten) ein Ergebnis. Dabei ist anzumerken, dass die fitnessbasierte Selektion langsamer läuft, dafür allerdings bessere Ergebnisse hinsichtlich der resultierenden Fitness erzielt. Der beste erreichte Wert belief sich hier auf 282.79 .

Beste Fitness	**popSize**	**eliteSize**	**mutationRate**	**generations**
282.79	500	50	0.01	1000

Tabelle 3: Parameterkonfiguration für die beste gefundene Fitness mit fitnessbasierter Selektion

Bei einer genaueren Untersuchung des Frameworks stellte sich heraus, dass der Algorithmus stark vom SPEA2 abweicht. Neben dem fehlenden festen Archiv zeigte sich, dass in der Selektion Lösungen auch mehrfach aufgenommen werden können. Dies hat zur Folge, dass ich der Breeding Phase Lösungen mit sich selbst gekreuzt werden können. Aus diesem Grund wurde die turnierbasierte Selektion ähnlich behandelt und Dopplungen ebenfalls zugelassen.

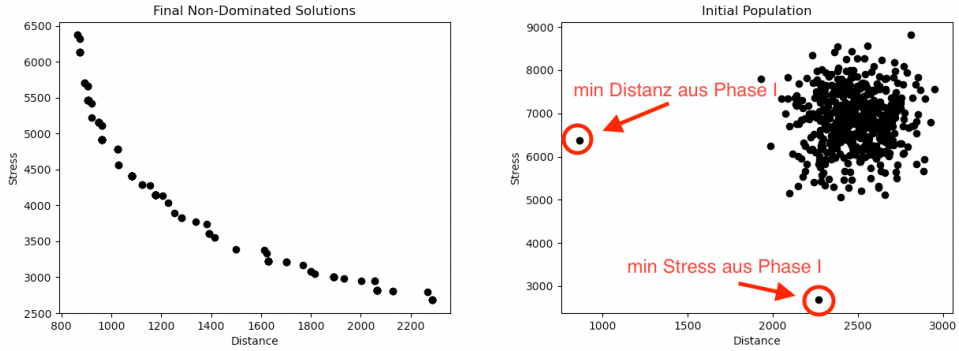

Abbildung 3: Paretofront der finalen nicht-dominierten Lösungen und initiale Population

5 Diskussion der Erkenntnisse zu Phase I und Phase II

Aufgrund der eingänglichen Suche nach möglichst guten Lösungen in Phase I konnten sich die entsprechenden bestmöglichen Werte für Stress und Distanz in der multiobjective Betrachtung nicht weiter verbessern. Es lässt sich in Phase I durch Erhöhung von *popSize* und *generations* keine oder kaum eine Verbesserung erzielen, während die Laufzeit entschieden steigen würde, was darauf schließen lässt, dass die gefundenen Lösungen nah am jeweiligen Optimum sind.

Anders hingegen in Phase II: wie sich zeigte, konnte eine Erhöhung von *population* und *generations* hier eine immer deutlichere Paretofront erzeugen und die Fitness der besten Lösung verbesserte sich ebenso. Da aber gleichzeitig die Laufzeit mehr als linear steigt, muss bezweifelt werden, dass es sinnvoll ist weiter zu erhöhen. Mit den gefundenen Lösungen aus Phase II lässt sich bereits eine sehr gute Auswahl an annähernd optimalen Lösungen für die Rundreise in und um Quebec City erreichen.

Die Erweiterung der Selektionsmethode um eine turnierbasierte Selektion konnte die Laufzeit der Algorithmen merklich verringern, allerdings auf Kosten der Güte der erreichten Lösungen. Im Rahmen der Arbeit war der Unterschied nicht groß genug um auf die besseren Lösungen zu verzichten, bei einer Anwendung auf ein komplexeres Problem könnten hier bei zeitkritischer Auswertung allerdings Vorteile bestehen.

6 Zusammenfassung und Fazit

Im Rahmen dieses Projektes konnte in zwei Phasen der vorliegende evolutionäre Algorithmus angewendet und erweitert werden. In Phase I wurden zunächst die Parametereinstellungen so vorgenommen, dass mit Hinblick auf Laufzeit und Konvergenz der Ergebnisse möglichst effizient gute Lösungen gefunden werden konnten. Der Programmcode wurde dabei geringfügig angepasst, um das Setzen der Parameter zu automatisieren sowie die Lösungen in PDF-Form auszugeben. Dies ermöglichte eine produktive Vorgehensweise bei der Bestimmung der besten Rahmenwerte für die Parameterdefinitionen.

Herausforderungen ergaben sich im Verlauf des Projekts vor allem bei der Weiterentwicklung des Programmcodes zur Integration fortgeschrittener Funktionalitäten. Die Umsetzung der turnierbasierten Selektion konnte erfolgreich gestaltet werden, nachdem Diskrepanzen des Frameworks zum SPEA2 ermittelt wurden. Die Implementation eines Archivs zur Speicherung der dominierenden Lösungen in Phase II, wie es im SPEA2 vorgesehen ist, wurde aus Komplexitätsgründen nach eingängiger Beschäftigung mit dem Problem nicht mehr umgesetzt, der entsprechende Code, der wahrscheinlich nicht weit von einer Lösung entfernt ist, ist im Anhang dieser Arbeit zu finden. Dennoch konnten durch die vorgenommenen Anpassungen am Programm die Projektziele erreicht werden und gute Lösungen effizient gebildet werden.

Von den Ergebnissen eines solchen Projekts können Entscheidungsträger in der Planung, Gestaltung und Operation von Transportnetzwerken profitieren, indem die Anwendung eines evolutionären Algorithmus praxisnah dargelegt wird. Ebenso wird aufgezeigt, welche Anpassungen an bestehenden Frameworks für spezifische Problemstellungen und erweiterte Funktionalitäten möglich sind.

7 Anhang

Array mit minimaler Distanz: [10,19,12,14,22,4,1,15,7,2,11,5,16,23,6,17,20,25,9,18,13,3,24,21,8]

Array mit minimalem Stress: [4,20,10,5,15,11,17,1,19,12,23,22,13,6,25,9,18,3,21,7,14,2,16,8,24]

Array mit maximaler Fitness: [C3_(176,67)_(T:7), C24_(176,101)_(T:12), C21_(195,127)_(T:31), C8_(193,177)_(T:23), C10_(156,197)_(T:11), C19_(152,151)_(T:24), C12_(118,141)_(T:6), C14_(135,112)_(T:24), C22_(105,118)_(T:3), C4_(87,126)_(T:27), C1_(68,128)_(T:8), C15_(36,133)_(T:9), C7_(25,127)_(T:12), C2_(16,100)_(T:14), C11_(6,6)_(T:9), C5_(35,54)_(T:38), C16_(96,91)_(T:12), C23_(104,85)_(T:25), C6_(95,69)_(T:1), C17_(91,53)_(T:16), C20_(122,30)_(T:15), C25_(115,19)_(T:26), C9_(119,2)_(T:4), C18_(140,25)_(T:37), C13_(183,32)_(T:39)]

8 Literaturverzeichnis

Zitzler, E., Laumanns, M., und Thiele, L. (2002). Spea2: Improving the strength pareto evolutionary algorithm for multiobjective optimization. Evolutionary Methods for Design, Optimisation, and Control, Seiten 19-26. Barcelona, Spain. CIMMNE.