Über die Analyse randomisierter Suchheuristiken und den Entwurf spezialisierter Algorithmen im Bereich der kombinatorischen Optimierung

Carsten Witt

FB Informatik, LS 2
Universität Dortmund
44221 Dortmund
carsten.witt@cs.uni-dortmund.de

Abstract: Dieser Beitrag ist eine Zusammenfassung der gleichnamigen Dissertation. Der Schwerpunkt der Arbeit liegt auf der theoretischen Analyse randomisierter Suchheuristiken wie evolutionärer Algorithmen, insbesondere in Bezug auf ihre Laufzeit bei der Lösung kombinatorischer Optimierungsprobleme. Neben einfachen randomisierten Suchheuristiken, die zu jedem Zeitpunkt nur einen Suchpunkt betrachten, werden auch evolutionäre Algorithmen mit größeren Populationen untersucht. Dabei werden jeweils neue und allgemeine Methoden zur Analyse randomisierter Suchheuristiken entwickelt. Abschließend wird ein spezialisierter Algorithmus für ein aktuelles kombinatorisches Optimierungsproblem vorgestellt.

1 Einleitung

Nicht nur in der industriellen Praxis, sondern auch im täglichen Leben müssen fortwährend schwierige Optimierungsaufgaben gelöst werden. Bespielsweise soll bei der Planung des eigenen Tagesablaufs eine möglichst gute Verteilung der wahrzunehmenden Termine und Aufgaben gefunden werden. Der Versuch, einen möglichst guten Tagesplan zu erstellen, entspricht in abstrakter Form einem Problem, dem in der Informatik eine große Bedeutung zukommt. Neben solchen Problemen der Ablaufplanung fallen auch andere, leicht zu beschreibende Aufgaben wie die Ermittlung kürzester Wege in Straßennetzen, die Bildung möglichst gut harmonierender Arbeitsteams, die optimale Bepackung von Transportmitteln u. v. a. m. in das Arbeitsgebiet der sogenannten *kombinatorischen Optimierung*, das eine lange Tradition in der Mathematik und Informatik besitzt. Das Attribut „kombinatorisch“ können wir damit erklären, dass es bei den geschilderten Problemen immer direkt oder indirekt darum geht, eine optimale Kombination von Objekten zu finden.

Eine Aufgabe der Informatik besteht darin, *Algorithmen*, d. h. Verfahren, bereitzustellen, die kombinatorische Optimierungsprobleme *effizient* lösen. Einen Algorithmus nennen wir effizient, wenn er bei Eingabelänge n, d. h. bei im Wesentlichen n zu behandelnden Objekten, mit einer Anzahl von Rechenschritten auskommt, die polynomiell in n ist, z. B. n^2,

n^3, also insbesondere nicht exponentiell wachsend, wie es z. B. bei 2^n Rechenschritten der Fall wäre. Durch bloßes Ausprobieren aller Kombinationen gelangt man niemals zu effizienten Algorithmen für ein Optimierungsproblem. Tatsächlich stellt der Entwurf effizienter Algorithmen für kombinatorische Optimierungsprobleme einen der wichtigsten Zweige der theoretischen Informatik dar, in dem fortwährend Forschungsergebnisse benötigt und erzielt werden.

An dieser Stelle wird die Kluft zwischen Theorie und Praxis offenkundig. Die Vielfalt der besonders in der industriellen Praxis auftretenden Optimierungsprobleme schließt es aus, dass für jedes neu zu lösende Problem bereits ein effizienter Algorithmus in der Literatur der theoretischen Informatik vorliegt. Der Entwurf maßgeschneiderter Algorithmen für das neue Optimierungsproblem erfordert dann Expertenwissen sowie finanzielle und zeitliche Ressourcen, die häufig nicht gegeben sind. Bei solchen Engpässen bedarf es dann eines Kompromisses, der nicht so zeit- und kostenintensiv wie der Entwurf spezieller Algorithmen ist, sich aber andererseits effizienter darstellt als das mechanische Ausprobieren aller Kombinationen. Aus dieser Warte stellt die praktische Informatik zahlreiche allgemeine Optimierungsverfahren zur Verfügung. Diese sind nicht oder nur begrenzt auf spezielle Problemklassen zugeschnitten und können mit geringem zeitlichen Aufwand und begrenztem theoretischen Hintergrundwissen programmiert oder an das vorliegende Problem angepasst werden. Solche allgemeinen Verfahren bezeichnen wir als *Suchheuristiken*.

Trotz der Vielzahl an allgemeinen Suchheuristiken, die bis heute vorgeschlagen worden sind, sticht eine Klasse von Suchheuristiken aufgrund ihrer Einfachheit und ihres Erfolgs in der Praxis stets hervor, nämlich die *evolutionären Algorithmen*. Ursprünglich mit dem Ziel, die natürliche Evolution nachzubilden, konzipiert, vgl. z. B. [Sch95, Rec94, Hol75], handelt es sich bei evolutionären Algorithmen um die heutzutage vielleicht erfolgreichsten und meistangewandten Suchheuristiken. Ein typischer evolutionärer Algorithmus für ein kombinatorisches Optimierungsproblem, beispielhaft für ein Problem der Lastverteilung auf Maschinen, besitzt folgende Eigenschaften. Er hält zu jedem Zeitpunkt eine *Population* genannte Menge von Lösungen vor, d. h. in unserem Beispiel eine Menge von Belegungsplänen der Maschinen. Jeder Lösung der Population ist ein Wert zugeordnet, den man meist *Fitness* nennt. Wenn eine möglichst gleichmäßige Auslastung der Maschinen gewünscht ist, könnte eine beispielhafte Fitness(funktion) die Auslastung der am wenigsten ausgelasteten Maschine angeben. Der evolutionäre Algorithmus versucht dann in mehreren Runden eine Lösung mit möglichst hohem Fitnesswert zu finden. Der im Folgenden beschriebene, typische Ablauf einer Runde ist ein zentrales Merkmal evolutionärer Algorithmen.

Zu Beginn einer Runde wird aus der aktuellen Population eine Menge von Lösungen ausgewählt, um *mutiert* zu werden. Unter Mutation ist dabei eine geringfügige Veränderung der Lösung zu verstehen, bei der Lastverteilung beispielsweise die Verlagerung einer Aufgabe von einer Maschine auf eine andere. Aus den durch Mutation erzeugten Lösungen werden dann in der Regel Lösungen mit hohem Fitnesswert ausgewählt, um Lösungen der aktuellen Population zu ersetzen. Damit entsteht die Population für die nächste Runde. Komplexere evolutionäre Algorithmen können pro Runde noch weitere Operationen durchführen, aber im Rahmen der Dissertation beschränken wir uns auf verschiedene Varianten der Mutation und der *Selektion*, d. h. der Auswahl der zu mutierenden Lösungen und

des Verfahrens, gemäß dem die mutierten Lösungen in die nächste Population eingehen. Insbesondere die Mutation geht üblicherweise *randomisiert*, d. h. mit Zufallsentscheidungen, vonstatten. Anstelle des eher ideologisch begründeten Begriffs der evolutionären Algorithmen sprechen wir daher meist von *randomisierten Suchheuristiken*. Die Klasse der randomisierten Suchheuristiken umfasst die evolutionären Algorithmen vollständig und wird zuweilen sogar als allgemeiner angesehen.

Obschon randomisierte Suchheuristiken seit Jahrzehnten erfolgreich auf kombinatorische Optimierungsprobleme angewendet werden und eine unüberschaubare Menge von Publikationen diesen Erfolg dokumentiert, sind bei der theoretischen Analyse randomisierter Suchheuristiken erst in den vergangenen Jahren bedeutsame Fortschritte zu verzeichnen. Einen Einblick in die Ergebnisse, die bis zum Jahr 2000 gewonnen wurden, bieten u. a. die Dissertationen von Droste und Jansen [Dro00, Jan00]. Dennoch reichen die theoretischen Einsichten weiterhin noch bei weitem nicht aus, um das Verhalten randomisierter Suchheuristiken bei der Lösung kombinatorischer Optimierungsprobleme befriedigend zu verstehen. Die hier besprochene Dissertation soll einen weiteren Beitrag bei der Errichtung eines theoretischen Fundaments zur Analyse randomisierter Suchheuristiken leisten.

2 Aufbau und Ziele der Dissertation

In der Dissertation widmen wir uns hauptsächlich der theoretischen Analyse randomisierter Suchheuristiken zur kombinatorischen Optimierung, führen aber zum Abschluss ebenfalls beispielhaft vor, auf welche Weise der Entwurf maßgeschneiderter Algorithmen für gut verstandene kombinatorische Optimierungsprobleme vonstatten gehen kann. Die Dissertation ist sogar dreigeteilt angelegt. Im ersten Teil betrachten wir sehr einfache randomisierte Suchheuristiken mit Populationsgröße 1, d. h. randomisierte Suchheuristiken, die in jeder Runde lediglich eine Lösung vorhalten. Der zweite Teil beschäftigt sich mit dem Einfluss größerer Populationen auf die Effizienz randomisierter Suchheuristiken und der dritte Teil befasst sich schließlich mit dem oben erwähnten Entwurf effizienter spezialisierter Algorithmen.

Alle in der Arbeit betrachteten randomisierten Suchheuristiken verwenden dasselbe Verfahren, um Lösungen für kombinatorische Optimierungsprobleme abzuspeichern. Eine Lösung wird stets als Folge von n Bits abgespeichert, die entweder den Wert 0 oder den Wert 1 annehmen. Im eingangs erwähnten Beispiel eines Lastverteilungsproblems können wir den Fall, dass n Aufgaben auf zwei Maschinen zu verteilen sind, offensichtlich auf diese Weise kodieren. Jedoch werden auch Lösungen komplexerer kombinatorischer Optimierungsprobleme in Rechnern stets als Folge von Bits abgespeichert, sodass wir uns in der gesamten Dissertation auf solche Darstellungen beschränken. Wir betrachten also Fitnessfunktionen, die jeder n-stelligen Bitfolge einen Wert zuweisen. Aus rein mathematischer Sicht sind derartige Funktionen bereits intensiv untersucht worden und verkörpern ebenfalls die übliche Gestalt von Fitnessfunktionen in populären evolutionären Algorithmen. Die von uns untersuchten randomisierten Suchheuristiken sollen eine Bitfolge finden, die den größtmöglichen Fitnesswert besitzt. Damit die Verfahren möglichst allgemein anwendbar sind, treffen sie keine weiteren Annahmen über die Fitnessfunktion.

Die im ersten Teil der Dissertation betrachteten Suchheuristiken heißen (1+1)-EA und RLS (randomisierte lokale Suche). Sie unterscheiden sich nur in der Mutation, d. h. der eingangs erwähnten zufälligen Veränderung der Lösung, und laufen wie folgt ab. Vor Beginn der ersten Runde wird eine zufällige Lösung erzeugt, in der jedes Bit mit gleicher Wahrscheinlichkeit den Wert 0 und den Wert 1 annimmt. In jeder Runde wird die aktuelle Lösung mutiert und die damit erzeugte neue Lösung mit der bisherigen Lösung verglichen. Ist die Fitness der neuen Lösung mindestens so groß wie die der alten, macht die Suchheuristik in der nächsten Runde mit der neuen und sonst mit der alten Lösung weiter. RLS erzeugt die neue Lösung, indem genau ein zufällig ausgesuchtes Bit gedreht wird, d. h. sein Wert von 0 in 1 bzw. umgekehrt geändert wird. Der (1+1)-EA hingegen betrachtet hier jedes Bit für sich und dreht es mit einer Wahrscheinlichkeit von $1/n$. Die Mutation von RLS erzeugt also nur eine lokale Veränderung an genau einem Bit, während der (1+1)-EA lediglich im Durchschnitt pro Runde ein Bit dreht, aber in manchen Runden auch mehrere Bits verändert. Die Analyse von RLS ist aufgrund der lokalen Mutationen häufig einfacher als beim (1+1)-EA. Auf die komplexeren Suchheuristiken mit großen Populationen, die im zweiten Teil der Dissertation relevant sind, werden wir später eingehen.

Wenn wir spezielle randomisierte Suchheuristiken auf kombinatorischen Optimierungsproblemen untersuchen, versuchen wir mithilfe mathematischer Beweise auszurechnen, wie viele Schritte (d. h. Runden) der Suchheuristik erforderlich sind, bis eine optimale Lösung gefunden ist. Diese Zahl nennen wir *Laufzeit*. Aufgrund der zufälligen Entscheidungen der Suchheuristik ist die Laufzeit selbst ein zufälliger Wert und kann sich ändern, wenn wir die Suchheuristik ein zweites Mal starten. Somit werden wir lediglich Durchschnittswerte für die Laufzeit beweisen können, aber solche Durchschnittswerte sind ein guter und akzeptierter Indikator für die Effizienz. Ein zentrales Ziel der Dissertation besteht darüber hinaus darin, allgemeine, wiederverwertbare Methoden zur theoretischen Analyse der Laufzeit randomisierter Suchheuristiken zu entwickeln. In den folgenden Abschnitten werden wir beschreiben, auf welchen Optimierungsproblemen der (1+1)-EA, RLS sowie die populationsbasierten Suchheuristiken untersucht wurden, welche Ergebnisse erzielt und welche Analysemethoden dazu erarbeitet wurden.

3 Randomisierte Suchheuristiken für Funktionenklassen

Die n Bits, die den Wert unserer Fitnessfunktionen bestimmen, können wir in der Form von n Variablen $x_1, \ldots, x_n$ formalisieren. Jede Fitnessfunktion lässt sich dann als sogenanntes *Polynom* schreiben, d. h. als Summe und Differenz von Produkten, die aus Variablen und Zahlen bestehen. Ein sehr einfaches Beispiel ist die von drei Bits abhängige Funktion $f(x_1, x_2, x_3) = x_1 + 2 \cdot x_1 \cdot x_2 - 3 \cdot x_1 \cdot x_2 \cdot x_3$.

Man kann beweisen, dass die von uns betrachteten Suchheuristiken nicht alle Fitnessfunktionen auf n Variablen effizient optimieren können. Daher sind wir daran interessiert, ob sie effizient sind, wenn nur ein Teil aller denkbaren Funktionen erlaubt ist. Eine naheliegende und mathematisch sinnvolle Einschränkung besteht darin, die Anzahl der Variablen in jedem Produkt einzuschränken. Die größte Anzahl von Variablen nennt man den *Grad* der Funktion, der im obigen Beispiel 3 beträgt, da das letzte Produkt drei Variablen ent-

hält. Droste, Jansen und Wegener gelang als Ersten der Nachweis, dass der (1+1)-EA alle Funktionen vom Grad 1 effizient optimiert [DJW02].

Es warf sich die Frage auf, wie effizient Suchheuristiken auf Funktionen von größerem Grad sind. Weil wiederum bewiesen werden kann, dass sie nicht auf allen Funktionen vom Grad 2 oder größer effizient sind, haben wir weitere Einschränkungen vorgenommen. Die Klasse der *monotonen Polynome* enthält alle Funktionen, bei denen in der obengenannten Darstellung keine negativen Vorzeichen auftreten; unser Beispiel ist also kein monotones Polynom. In der Dissertation beweisen wir, dass RLS auf allen monotonen Polynomen vom Grad d im Durchschnitt eine Laufzeit benötigt, die in der Größenordnung von $(n/d)\log(n/d+1)2^d$ liegt, für kleine d also effizient ist. Für große d gibt es wiederum den Beweis, dass die Suchheuristiken nicht auf allen betrachteten Funktionen effizient sein können. Bei der Analyse des (1+1)-EA auf monotonen Polynomen beweisen wir, dass die durchschnittliche Laufzeit höchstens in der Größenordnung $N(n/d)2^d$ ist, wenn N die Zahl der Summanden in der Darstellung des Polynoms ist. Da N durchaus nichtpolynomielle Werte annehmen kann, lässt sich die Laufzeit für manche Funktionen vermutlich besser abschätzen. Ein offenes Problem besteht darin nachzuweisen, dass auch für den (1+1)-EA die oben erwähnte Größenordnung $(n/d)\log(n/d+1)2^d$ gilt. Diese Vermutung erhärten wir in der Dissertation, indem wir genau diese Laufzeit für eine weitere randomisierte Suchheuristik, die dem (1+1)-EA bereits sehr ähnlich ist, nachweisen.

Um die durchschnittlichen Laufzeiten abzuschätzen, entwickeln wir neue und in allgemeineren Rahmen anwendbare Beweistechniken. Der zufällige Prozess, den die Suchheuristik RLS beschreibt, ist, wie oben erwähnt, viel einfacher zu analysieren als der des (1+1)-EA, da RLS in jedem Schritt nur eine lokale Änderung vornimmt. Die neuen Beweistechniken gestatten es, aus den Laufzeitschranken für RLS Laufzeitschranken für den (1+1)-EA herzuleiten, ohne den zufälligen Prozess des (1+1)-EA vollständig untersuchen zu müssen. Die Techniken eignen sich darüber hinaus dafür, solche Übertragungen von Laufzeitanalysen auch bei anderen randomisierten Suchheuristiken durchzuführen.

4 Randomisierte Suchheuristiken für ein Lastverteilungsproblem

Im ersten Teil der Dissertation wenden wir uns nach der strukturellen Frage, die die Untersuchung der monotonen Polynome verkörpert, einem konkreten kombinatorischen Optimierungsproblem zu. Dabei handelt es sich um folgendes Lastverteilungsproblem, das dem in der Einleitung geschilderten Beispiel sehr ähnlich ist. Gegeben sind n Aufgaben mit Bearbeitungszeiten $w_1,\ldots,w_n$. Ziel ist es, die Aufgaben so auf zwei Maschinen zu verteilen, dass die größere Laufzeit der beiden Maschinen möglichst klein wird. Im Extremfall besitzen beide Maschinen dann dieselbe Laufzeit.

Obwohl einfach zu formulieren, ist kein effizienter Algorithmus bekannt, der dieses Lastverteilungsproblem für alle möglichen Spezifikationen der Bearbeitungszeiten $w_1,\ldots,w_n$ immer exakt löst. Aus der Sicht der theoretischen Informatik zählt es sogar zu den am schwersten zu lösenden Problemen. Allerdings gibt es für das Problem sehr gute effiziente Näherungswertalgorithmen. Diese geben Lösungen aus, deren Wert nur um einen kleinen

Faktor, beispielsweise nur um 1 %, vom Wert einer optimalen Lösung abweicht. Wir interessieren uns daher dafür, ob der (1+1)-EA und RLS gute Näherungswertalgorithmen für das Lastverteilungsproblem repräsentieren.

Es gelingt der Nachweis, dass sowohl der (1+1)-EA als auch RLS stets in effizienter durchschnittlicher Laufzeit Lösungen für das Lastverteilungsproblem finden, deren Wert um höchstens 33,3 % vom Wert einer optimalen Lösung abweicht. Es besteht aber sogar eine gute Wahrscheinlichkeit, dass die Suchheuristiken deutlich bessere Lösungen finden. Daher ist es hilfreich, sie mehrfach neu zu starten und die beste gefundene Lösung zu verwenden. Beispielsweise wollen wir damit für ein festes p Lösungen erhalten, die um höchstens p % schlechter als optimale Lösungen sind. Ein weiterer Beweis zeigt, dass man mit einer hohen Wahrscheinlichkeit nach wenigen Neustarts und mit insgesamt effizienter Laufzeit tatsächlich solche Lösungen findet.

Aus der Sicht der theoretischen Informatik wird ein Algorithmus für ein Optimierungsproblem als ineffizient bezeichnet, wenn es auch nur eine Eingabe für das Problem gibt, auf der der Algorithmus eine ineffiziente Laufzeit besitzt. Bei exakten Algorithmen für unser Lastverteilungsproblem lassen sich z. B. meist einfach Bearbeitungszeiten $w_1, \ldots, w_n$ spezifizieren, die zu einer ineffizienten Laufzeit führen. Allerdings sind solche Eingaben eher unrealistisch und stellen aus der Sicht von Praktiker(inne)n meist pathologische Fälle dar, die vielleicht niemals auftreten. Wir begegnen dem, indem wir diese sogenannte Worst-Case-Analyse abmildern. Pathologischen Eingaben werden nun geringe Auftrittswahrscheinlichkeiten zugewiesen und die Laufzeiten auf solchen Eingaben mit deren Wahrscheinlichkeiten gewichtet. Dieses Vorgehen bezeichnet man als Average-Case-Analyse. Zum Abschluss des ersten Teils der Dissertation führen wir erstmals eine Average-Case-Analyse für randomisierte Suchheuristiken durch und erhalten, dass unsere Suchheuristiken selbst ohne Neustarts in effizienter Zeit Lösungen finden, deren Qualität mit der, die einfache auf das Lastverteilungsproblem zugeschnittene Algorithmen erzielen, vergleichbar ist. Diese Analyse umfasst eine neue Technik, um Ergebnisse aus der Wahrscheinlichkeitstheorie auf die Analyse von randomisierten Suchheuristiken zu übertragen. In diesem Zusammenhang wurden sortierte zufällige Objekte, sogenannte Ordnungsstatistiken, untersucht, um den Wert von Lösungen, die der (1+1)-EA durch lokale Änderungen verbessern kann, abzuschätzen.

5 Der grundsätzliche Nutzen von Populationen

Im ersten Teil der Dissertation haben wir gesehen, dass einfache randomisierte Suchheuristiken, deren Populationsgröße 1 beträgt, oft erstaunlich effizient sind. Demgegenüber ist es in der Praxis üblich, evolutionäre Algorithmen mit größeren Populationen zu verwenden. Wir wollen die Auswirkungen der Populationsgröße auf die Effizienz randomisierter Suchheuristiken theoretisch untersuchen.

Ziel dieses Abschnitts der Dissertation ist es, eine populationsbasierte randomisierte Suchheuristik und eine Fitnessfunktion mit den folgenden Eigenschaften anzugeben. Bei einer geringen Populationsgröße soll die durchschnittliche Laufzeit der Suchheuristik exponen-

tiell, d. h. sehr ineffizient, sein und auch ein Neustart der Suchheuristik darf keine nennenswerten Vorteile bringen. Vergrößert man die Population jedoch um einen konstanten (von n unabhängigen) Faktor, soll die durchschnittliche Laufzeit polynomiell und somit effizient sein. Da dieser Nachweis gelingt, ist eine theoretisch fundierte Erklärung für den Nutzen von Populationen gefunden.

Im Folgenden bezeichnen wir die Lösungen, die Element einer Population sind, gemäß dem üblichen Sprachgebrauch als *Individuen* und die Größe der Population nennen wir μ. Die populationsbasierte Suchheuristik, die wir hier zugrunde legen, ist eine Verallgemeinerung des (1+1)-EA und läuft im Groben wie folgt ab. Zunächst werden alle Individuen der Startpopulation jeweils unabhängig zufällig wie im (1+1)-EA ausgesucht. Es schließt sich eine Schleife an, die zunächst ein Individuum zur Mutation wählt, dieses dann mutiert und aus den danach vorliegenden $\mu+1$ Individuen eines auswählt, das zu entfernen ist. In jeder Runde der Schleife wird also höchstens ein Element der Population verändert. Die Mutation läuft hier genauso wie beim (1+1)-EA ab. Die Selektion zur Mutation bevorzugt Individuen mit großer Fitness und die Selektion des zu entfernenden Individuums solche mit geringer Fitness, aber mit geringer Wahrscheinlichkeit werden auch schlechte Individuen mutiert und gute entfernt. Diese sogenannte fitnessbasierte Selektion ist in praktischen evolutionären Algorithmen sehr geläufig.

Die spezielle Fitnessfunktion, die wir in diesem Teil der Dissertation betrachten, ist sorgfältig gewählt, um den Nutzen von Populationen nachweisen zu können. Allerdings eignet sich die zugrundeliegende Idee, um weitere Funktionen definieren zu können, bei denen die Laufzeit unserer populationsbasierten Suchheuristik entscheidend von der Populationsgröße abhängt. Zum einen lässt sich die Populationsgröße, bei der die durchschnittliche Laufzeit von einem exponentiellen auf einen polynomiellen Wert umschlägt, in ihrer Größenordnung einstellen. Bei der zunächst genannten Funktion liegt diese Größenordnung bei $\sqrt{n}$, aber auch Größenordnungen von $n\sqrt{n}$, $n^2\sqrt{n}$, $n^3\sqrt{n}$, $\ldots$ können durch Redefinition der Funktion vorgegeben werden. Darüber hinaus geben wir eine Funktion an, auf der eine Population schädlich ist, d. h., dass eine kleine Populationsgröße zu einer polynomiellen durchschnittlichen Laufzeit und eine nur geringfügig größere Population zu einer exponentiellen durchschnittlichen Laufzeit führt.

Zur Analyse des Laufzeitverhaltens der populationsbasierten Suchheuristik stellen wir ein zentrales neues Werkzeug vor, die sogenannten *Stammbäume*. Für jedes Individuum aus der Startpopulation lässt sich im Laufe der Zeit auf folgende, naheliegende Weise ein Stammbaum aufbauen. Wenn das Individuum zur Mutation gewählt wird, hängen wir das durch Mutation entstandene Individuum an das ursprüngliche an. Wenn das neue Individuum ebenfalls mutiert wird, hängen wir das damit erzeugte Individuum wiederum an das gewählte an usw. Auf diese Weise entsteht ein Baum aus Individuen, die alle direkt oder indirekt von einem Ursprungsindividuum aus der Startpopulation abstammen. Man kann dem Baum sogar entnehmen, wie viele Mutationsschritte stattfanden, um ein gerade betrachtetes Individuum aus dem Ursprungsindividuum zu erzeugen. Die Zahl dieser Schritte, die gerade der Zahl der Vorgänger auf dem Weg zurück zum Ursprungsindividuum entspricht, bezeichnen wir als *Tiefe* des Individuums. Eine mathematische Analyse führt zutage, dass auch nach vielen Schritten der randomisierten Suchheuristik mit hoher Wahrscheinlichkeit kein Individuum eine große Tiefe besitzt. Eine geringe Tiefe eines

Individuums bedeutet, dass wenige Mutationen zwischen dem Ursprungsindividuum und dem betrachteten Individuum liegen. Da in wenigen Mutationen nur verhältnismäßig geringe Änderungen eintreten, können wir damit zeigen, dass die aktuelle Population immer noch ähnlich zur Startpopulation ist. Da die Startpopulation typischerweise kein optimales Individuum enthält, gilt dies mit hoher Wahrscheinlichkeit also auch für die aktuelle Population. Folglich übersetzt sich eine Tiefenbeschränkung für Stammbäume direkt in eine Mindestlaufzeit für die populationsbasierte Suchheuristik. Diese Technik scheint wegweisend für weitere Analysen populationsbasierter Suchheuristiken zu sein.

6 Die Analyse einer einfachen Suchheuristik mit einer Population

Die strukturelle Frage nach dem Nutzen von Populationen haben wir im obenerwähnten Teil der Dissertation erfolgreich geklärt. Dabei sind Fitnessfunktion und Suchheuristik jedoch passend gewählt worden, um die Analysen zu ermöglichen. Auch wenn die dort betrachtete populationsbasierte Suchheuristik einem bekannten Paradigma entspricht, existiert in der Praxis eine große Klasse von ebenso wichtigen populationsbasierten evolutionären Algorithmen, für die noch keine Laufzeitanalysen bekannt sind. Diese Klasse trägt den Namen *Evolutionsstrategien*. Wir wollen eine einfache Evolutionsstrategie, die auch in der Praxis eingesetzt werden kann und den Namen (μ+1)-EA trägt, auf Beispielproblemen untersuchen und dabei den Zusammenhang zwischen Problemgröße, Populationsgröße μ und Laufzeit erhellen. Der (μ+1)-EA zeichnet sich dadurch aus, dass er in jedem Schritt gleich wahrscheinlich jedes Individuum aus der aktuellen Population zur Mutation wählt, wie der (1+1)-EA mutiert und aus den $\mu+1$ anschließend vorhandenen Individuen eins mit geringstem Fitnesswert löscht.

Weil die theoretische Analyse populationsbasierter randomisierter Suchheuristiken immer noch in den Anfängen steckt, betrachten wir diesmal kein komplexes kombinatorisches Optimierungsproblem wie das obengenannte Lastverteilungsproblem, sondern drei sehr einfach zu beschreibende Fitnessfunktionen. Die erste Funktion zählt die Zahl der Einsen in der n-stelligen Bitfolge, die zweite die Zahl der führenden Einsen und die dritte liefert einen guten, konstanten Funktionswert, wenn rechts der am weitesten links stehenden Null keine Eins mehr auftritt, sowie den optimalen Funktionswert, wenn nur Einsen vorhanden sind. Als die ersten Schritte zu einer Analyse des (1+1)-EA gemacht wurden, betrachtete man ebenfalls diese oder ähnliche Funktionen (vgl. [DJW02]), da sie klar strukturiert sind und damit die Entwicklung von Beweismethoden vereinfachen. In der Dissertation gelingt die Analyse der Laufzeit des (μ+1)-EA auf den genannten drei Funktionen beinahe vollständig. In allen drei Fällen stellt sich heraus, dass die durchschnittlichen Laufzeiten mindestens proportional zu μ wachsen und die Wahl $\mu = 1$ zur geringsten Laufzeit führt (in der üblichen Sicht, sich auf die Größenordnungen von Laufzeiten zu beschränken). Bei den letzten beiden Funktionen sind obere und untere Schranken für die durchschnittliche Laufzeit nur um einen kleinen Faktor $\log n$ voneinander entfernt.

Bei den drei Beispielfunktionen hat die Population des (μ+1)-EA also in Bezug auf die Größenordnung der Laufzeit keinen Nutzen. Da Evolutionsstrategien in der Praxis aber eingesetzt werden, ist es wiederum wünschenswert, Fälle zu identifizieren, bei denen eine

kleine Erhöhung von μ die durchschnittliche Laufzeit drastisch senken kann. Zum Abschluss dieses Teils der Dissertation konstruieren wir eine solche Beispielfunktion und erweitern damit unser Verständnis von der Arbeitsweise des (μ+1)-EA.

Viele der Analysen im Zusammenhang mit dem (μ+1)-EA greifen auf die obenerwähnten Stammbäume zurück. Wir können die neue Beweismethode also auch auf andere populationsbasierte randomisierte Suchheuristiken übertragen und damit ihre Laufzeit nach unten beschränken. Darüber hinaus erläutern wir mit den sogenannten *Potenzialfunktionen*, wie man die Laufzeit solcher Suchheuristiken bequem nach oben abschätzen kann.

7 Der Entwurf von Algorithmen für ein kombinatorisches Optimierungsproblem

In den ersten beiden Teilen der Dissertation ist es uns gelungen, die Arbeitsweise allgemeiner randomisierter Suchheuristiken im Bereich der kombinatorischen Optimierung besser zu verstehen. Es erscheint erfreulich, dass diese auch aus theoretischer Sicht oft sehr effizient sind. Nichtsdestoweniger können wir von spezialisierten Algorithmen, die auf ein vorliegendes Optimierungsproblem zugeschnitten sind, noch bessere Lösungen und Laufzeiten erwarten. Zum Abschluss führen wir beispielhaft vor, wie der Entwurf solcher spezialisierter Algorithmen vor sich gehen kann, zeigen damit aber auch, dass der dazu nötige Aufwand in der Praxis vermutlich nicht immer betrieben werden kann. Somit beleuchten wir Entwurf und Analyse von spezialisierten Algorithmen einerseits und randomisierten Suchheuristiken andererseits aus zwei verschiedenen Blickwinkeln.

Das kombinatorische Optimierungsproblem, für das wir im dritten Teil der Dissertation Algorithmen entwerfen wollen, entstammt Optimierungsaufgaben im Rahmen der Rechnerarchitektur. In modernen Rechnern liegt die Zugriffszeit für Festplattenspeicher um ein Vielfaches über der für den flüchtigen Hauptspeicher. Daher ist es hilfreich, möglichst viele benötigte Speicherinhalte im Hauptspeicher zu behalten, um möglichst selten Inhalte von Festplatten lesen zu müssen. Dieses Konzept heißt gewöhnlich *Caching*. Wenn man die Folge der angefragten Speicherinhalte im Vorhinein kennt und die aktuell benötigten Inhalte bereits im Hauptspeicher liegen, darf man das Einbringen eines weiteren Inhalts der Festplatte schon weit vor dem Zeitpunkt starten, an dem der Inhalt benötigt wird. Diese Technik, die Wartezeiten bei Festplattenzugriffen abmildert, heißt *Prefetching*.

Bislang existieren wenige Algorithmen, um effizient für eine gegebene Anfragesequenz zu entscheiden, wann Prefetching betrieben wird und nach welcher Cachingstrategie Inhalte im Hauptspeicher gehalten werden. Wir entwickeln in der Dissertation einen neuen effizienten Algorithmus, der Prefetching und Caching integriert. Gegenüber dem einzigen bisher bekannten Ansatz, der das Problem effizient und exakt löst, zeichnet sich unser Algorithmus dadurch aus, dass er auf gebräuchlichen und nicht zu komplizierten Datenstrukturen, sogenannten Flussnetzwerken, operiert. Weiterhin entwickeln wir für ein verallgemeinertes Optimierungsproblem verbesserte Näherungswertalgorithmen.

Mehrere Jahre Forschung waren erforderlich, ehe der erste effiziente Algorithmus für das Problem des integrierten Prefetchings und Cachings entwickelt werden konnte. Obschon

wir damit vermutlich bessere Ergebnisse als bei der Anwendung von Suchheuristiken erzielen, mussten wir zuvor mit solchen Kompromissen vorlieb nehmen. Für die Theoretikerin und den Theoretiker schließen wir mit der beruhigenden Botschaft, dass sich die randomisierten Suchheuristiken unserem theoretischen Verständnis nicht mehr gänzlich entziehen.

Literatur

[DJW02] S. Droste, T. Jansen und I. Wegener. On the analysis of the (1+1) Evolutionary Algorithm. *Theoretical Computer Science*, 276:51–81, 2002.

[Dro00] S. Droste. *Zu Analyse und Entwurf evolutionärer Algorithmen*. Dissertation, Universität Dortmund, 2000.

[Hol75] J. H. Holland. *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, Ann Arbor, MI, USA, 1975.

[Jan00] T. Jansen. *Theoretische Analyse evolutionärer Algorithmen unter dem Aspekt der Optimierung in diskreten Suchräumen*. Dissertation, Universität Dortmund, 2000.

[Rec94] I. Rechenberg. *Evolutionsstrategie '94*. Frommann-Holzboog, Stuttgart, 1994.

[Sch95] H.-P. Schwefel. *Evolution and Optimum Seeking*. Wiley, 1995.

Carsten Witt wurde am 7. März 1975 in Bochum geboren. Nach dem Abitur (Note 1,0) in Recklinghausen im Jahr 1994 und anschließendem Zivildienst studierte er ab Oktober 1995 Informatik an der Universität Dortmund und schloss das Studium im April 2000 als Diplom-Informatiker mit Auszeichnung ab. Carsten Witt erhielt für das beste Informatik-Diplom seines Jahrgangs den Hans-Uhde-Preis zur Förderung der Wissenschaft. Seit Juli 2000 arbeitet er als wissenschaftlicher Mitarbeiter am Lehrstuhl für effiziente Algorithmen und Komplexitätstheorie am Fachbereich Informatik der Universität Dortmund. Seine Forschungsschwerpunkte sind der Entwurf und die Analyse von Algorithmen, insbesondere von randomisierten Suchheuristiken. Carsten Witt promovierte im Dezember 2004 bei Prof. Dr. Ingo Wegener mit Auszeichnung.