Theoretische Analyse evolutionärer Algorithmen unter dem Aspekt der Optimierung in diskreten Suchräumen

Dissertation
zur Erlangung des Grades eines
Doktors der Naturwissenschaften
der Universität Dortmund
am Fachbereich Informatik
von

Thomas Jansen

Dortmund

2000

Tag der mündlichen Prüfung: 9. Oktober 2000
Dekan: Bernd Reusch
Gutachter: Ernst-Erich Doberkat, Ingo Wegener

Vorwort

Diese Arbeit beschäftigt sich mit einigen (hoffentlich) wichtigen und interessanten Aspekten der Analyse evolutionärer Algorithmen unter dem Aspekt der Optimierung in diskreten Suchräumen, namentlich in $\{0,1\}^n$. Sie ist eine Zusammenfassung und gegliederte Darstellung einer Reihe von Überlegungen und Arbeiten, die von Januar 1997 bis März 2000 im Zusammenhang mit meiner Arbeit im Sonderforschungsbereich 531 mit dem Titel „Design und Management komplexer technischer Prozesse und Systeme mit Methoden der Computational Intelligence" (gerne auch kurz „SFB CI") entstanden sind. Ohne die freundliche und großzügige Unterstützung der DFG im Rahmen dieses SFB wäre die Erstellung der Arbeit sicher nicht möglich gewesen. Es versteht sich heutzutage vermutlich von selbst, dass wesentliche Ergebnisse nicht alleine, sondern in Zusammenarbeit mit Kollegen erarbeitet worden sind. Das ergibt sich in aller Klarheit aus der Liste der Arbeiten, in denen zum Teil Ergebnisse, die hier erstmals im Überblick und Bezug zu einander (und auch erstmals in deutscher Sprache) dargestellt werden, vorab veröffentlicht worden sind. Wir führen diese Veröffentlichungen hier konkret auf und geben an, welche Ergebnisse man in welchen Kapiteln wiederfindet.

Droste, Jansen und Wegener (1998a): On the analysis of the (1+1) evolutionary algorithm. Reihe CI 21/98, SFB 531, Universität Dortmund. Eingereicht: Theoretical Computer Science, in den Kapiteln 3, 4, 5, 6 und 9,

Droste, Jansen und Wegener (1998b): On the optimization of unimodal functions with the (1+1) evolutionary algorithm. In A. E. Eiben, T. Bäck, M. Schoenauer und H.-P. Schwefel (Hrsg.): Proceedings of the 5th Parallel Problem Solving from Nature (PPSN V), LNCS 1498, 47–56, Springer, Berlin, in Kapitel 5,

Droste, Jansen und Wegener (1998d): A rigorous complexity analysis of the (1+1) evolutionary algorithm for separable functions with Boolean inputs. Evolutionary Computation 6(2):185–196, 1998 (auch Droste, Jansen und Wegener (1998c)), in Kapitel 4,

Jansen (1998): Introduction to the theory of complexity and approximation algorithms. In E. W. Mayr, H. J. Prömel und A. Steger (Hrsg.): Lectures on Proof Verification and Approximation Algorithms, LNCS 1367, 5–28, Springer, Berlin,

Droste, Jansen und Wegener (1999): Perhaps not a free lunch but at least a free appetizer. In W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela und R. E. Smith: Proceedings of the First Genetic and Evolutionary Computation Conference (GECCO ’99), 833–830, Morgan Kaufman, San Francisco, CA, in Kapitel 2,

Jansen und Wegener (1999): On the analysis of evolutionary algorithms — a proof that crossover really can help. In J. Nešetřil (Hrsg.): Proceedings of the 7th Annual European Symposium on Algorithms, LNCS 1643, 184–193, Springer, Berlin, in Kapitel 9,

Jansen (2000): On classifications of fitness functions. In L. Kallel, B. Naudts, A. Rogers (Hrsg.): Theoretical Aspects of Evolutionary Computing. Natural Computing, 377–390, Springer, Berlin, in Kapitel 2,

Jansen und Wegener (2000b): On the choice of the mutation probability for the (1+1) EA. In M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J. J. Merelo und H.-P. Schwfel (Hrsg.): Proceedings of the 6th Parallel Problem Solving from Nature (PPSN VI), LNCS 1917, 89–98, Springer, Berlin, in Kapitel 7,

Droste, Jansen und Wegener (2000a): Dynamic parameter control in simple evolutionary algorithms (extended abstract). Akzeptiert für: FOGA 2000, in den Kapiteln 7 und 8 und

Droste, Jansen und Wegener (2000b): A natural and simple function which is hard for all evolutionary algorithms. Akzeptiert für: SEAL 2000, in Kapitel 10.

Im Gegensatz zu dem, was vielleicht für manche andere Dissertationen am Fachbereich Informatik der Universität Dortmund gilt, liegen dieser Arbeit Überlegungen zu Grunde, die ihren Ursprung oft in einer fruchtbaren Zusammenarbeit mit Kollegen von anderen Lehrstühlen haben. Stellvertretend für alle sei an dieser Stelle Hans-Paul Schwefel mit seinen Mitarbeitern vom Lehrstuhl für Systemanalyse gedankt. Noch wesentlich enger und fruchtbarer war natürlich die Zusammenarbeit am eigenen Lehrstuhl, hier gilt mein Dank in erster Linie meinem Betreuer Ingo Wegener und meinem Kollegen Stefan Droste, mit dem mich außer einem gemeinsamen Büro auch sonst noch viel verbindet. Für die gute Arbeitsatmosphäre, die vielen ungezählten Freundlichkeiten, den fachlichen Rat und ihre Anteilnahme gebührt allen Mitarbeiterinnen und Mitarbeitern am Lehrstuhl für effiziente Algorithmen und Komplexitätstheorie mein Dank. In noch viel grundlegenderem Sinne wichtig für mich waren aber Unterstützung und Verständnis meiner Familie, darum danke ich an dieser Stelle vor allem meiner Frau Sylvie und unseren Kindern Nikolas, Lena-Marie und Nora-Alina.

Inhaltsverzeichnis

1 Einleitung

Evolutionäre Algorithmen sind allgemeine, randomisierte Suchheuristiken, die in vielen verschiedenen Bereichen zu unterschiedlichen Zwecken eingesetzt werden können. Ihre Ursprünge liegen im Grunde in der Bionik, also in dem Versuch, erfolgreiche technische Neuentwicklungen durch Naturimitation im weitesten Sinne zu erreichen. Es gibt aber auch über die erfolgreiche Nachahmung natürlicher Prozesse und „Ausbeutung“ des Verständnisses dieser Prozesse in technischen Anwendungen hinaus stärker an der Biologie orientierte Interessen an evolutionären Algorithmen. Es geht darum, einen natürlichen Prozess als evolutionären Algorithmus zu modellieren, um durch erfolgreiche Anwendung des Algorithmus eine Bestätigung des Naturverständnisses zu erlangen. Zeigt dieser Algorithmus nun ein Verhalten, das nicht mehr als naturanalog bezeichnet werden kann, so deutet das auf unvollständige Modellierung oder lückenhaftes Verständnis hin, kann sogar als Falsifizierung der Modellbildung gesehen werden. In diesem Sinne ist das Ziel der Erforschung evolutionärer Algorithmen ein Verständnis, wobei Verstehen der Algorithmen und „der Natur“ sich gegenseitig bedingen und in enger Beziehung zu einander stehen. Wir wollen der damit verknüpften schwierigen philosophischen Frage, was verstehen eigentlich bedeutet, aus dem Wege gehen und definieren unsere Zielsetzung gänzlich anders.

Evolutionäre Algorithmen sind eine seit vielen Jahren bekannte und in Anwendungen etablierte Algorithmenklasse. Wir betrachten sie darum losgelöst von ihrem biologischen Ursprung als Forschungsgegenstand „klassischer“ Informatikforschung in den Gebieten Algorithmen und Komplexität. Ingenieure, die evolutionäre Algorithmen in der ingenieurwissenschaftlichen Praxis einsetzen, haben einen großen Fundus praktischer Erkenntnisse. Aus diesem reichhaltigen Erfahrungsschatz resultiert umfangreiches empirisches Wissen über diese Algorithmen. Ziel dieser Arbeit ist es, dieses empirische Wissen in analytisches Wissen umzuwandeln, dem praktisch Gewussten eine solide, auf mathematischen Beweisen fußende Grundlage zu geben. Für den praktisch erfahrenen Wissenschaftler enthält diese Arbeit kaum Unerwartetes oder gar Neues. Wir präsentieren zu „bekannten Resultaten“ formale Beweise. Wir haben den festen Glauben, dass dem Gebiet auf diese Weise ein Dienst erwiesen wird, dass eine wichtige Qualität wenn nicht neu hinzugefügt, so doch zumindest wesentlich gestärkt wird. Formale Beweise für scheinbar Offensichtliches – oder auch Gegenbeispiele zu verbreiteten Irrtümern – sind wichtige Bausteine einer exakten Wissenschaft, die unserer Überzeugung nach gerade auf dem Gebiet evolutionärer Algorithmen wesentlich vor allem die Lehrbarkeit des Fachs verbessern.

Die noch heute gängigen Hauptvarianten genetische Algorithmen (Goldberg 1989), evolutionäre Programmierung (Fogel 1995) und Evolutionsstrategien (Schwefel 1995) haben alle ihre (von einander unabhängigen) Ursprünge in den 60er Jahren (Fogel 1999; Holland 1992; Rechenberg 1994). Es geht uns hier nicht um die historische Entwicklung von diesen Anfängen zu modernen evolutionären Algorithmen. Insbesondere werden wir eine in gewissen Grenzen mögliche Trennung der drei Traditionen unter Betonung ihrer jeweiligen Besonderheiten außer acht lassen. Für eine zeitgemäße Darstellung von Gemeinsamkeiten und Unterschieden der verschiedenen Varianten verweisen wir auf das Buch von Bäck (1996). Wir schließen uns in unserer Arbeit dem in letzter Zeit deutlich erkennbar werdenden Trend an, alle drei Algorithmenklassen als Ausprägungen ein und desselben Paradigmas zu sehen und den vereinheitlichenden Begriff „evolutionäre Algorithmen“ zu verwenden.

Die Grundidee ist, den Prozess der natürlichen Evolution, wie man ihn in der Natur zu erkennen glaubt, als verbessernden Prozess zu verstehen. Man erhofft sich, durch Imitation dieses Prozesses in algorithmischer Form zu verbessernden Algorithmen in weitestem Sinne zu gelangen. Dem auf dem Gebiet der evolutionären Algorithmen verwendeten Vokabular merkt man seine in der Bionik liegenden Wurzeln noch deutlich an.

Ziel eines evolutionären Algorithmus ist es, in einem gegebenen Suchraum S möglichst gute Suchpunkte zu finden, wobei die Güte eines Suchpunkts anhand einer Funktion $f\colon S \to V$ definiert ist. Um einen Vergleich zu ermöglichen, muss auf V offensichtlich zumindest eine partielle Ordnung definiert sein. Häufig findet man $V = \mathbb{R}$ und hat dann eine total geordnete Menge vor sich. Der evolutionäre Algorithmus sucht manchmal in einem anderen Raum C, wobei es zwischen C und S ein (nicht immer bijektives) Mapping gibt, also auf jeden Fall eine Funktion $d\colon C \to S$, oft auch eine Funktion $c\colon S \to C$. In Anlehnung an die Biologie nennt man den Suchraum C auch *Genotypraum* und den Suchraum S *Phänotypraum*, eine Kollektion von Punkten im Suchraum *Population*, einen Punkt $s \in S$ *Individuum* und seinen Wert $f(s)$ seine *Fitness*. Es gibt wie schon erwähnt verschiedene evolutionäre Algorithmen, die in extremer Variantenvielfalt an die aktuell betrachteten Probleme angepasst werden. Das macht es schwierig, genau zu beschreiben, was ein evolutionärer Algorithmus eigentlich ist. Im folgenden versuchen wir eine grobe Darstellung der Grundprinzipien, die so praktisch allen evolutionären Algorithmen zu Grunde liegen. Das schließt nicht aus, dass einzelne Algorithmen, die in dem einen oder anderen Punkt abweichend verfahren, nicht auch noch als Ausprägung des sehr allgemeinen Paradigmas „evolutionärer Algorithmus“ verstanden werden können.

Ein evolutionärer Algorithmus verläuft in Runden, die in Anlehnung an das natürliche Vorbild *Generationen* genannt werden. Wir untersuchen im Rahmen dieser Arbeit nicht die Problematik geeigneter Codierungen des Suchraums und nehmen stets $C = S$ an, so dass wir für die Codierung und Decodierung stets einfach $c = d = \mathrm{id}$ annehmen. Anfangs wird (in der Regel zufällig) eine Menge von Punkten ausgewählt, die so genannte *Anfangspopulation*. In jeder Runde wählt man aus der aktuellen Menge von Punkten einige aus, auf denen basierend neue Punkte im Suchraum berechnet werden. Diese Auswahl bezeichnet man als *Selektion zur Reproduktion*. Man unterscheidet zwei verschiedene Operatoren, die neue Punkte im Suchraum erzeugen, nämlich Mutation und Rekombination. Die Mutation operiert auf einem Punkt im Suchraum und erzeugt (in der Regel zufällig) einen anderen Punkt im Suchraum. Bei der Rekombination, die häufig auch Crossover genannt wird, wird ein neuer Punkt im Suchraum auf Basis von mindestens zwei anderen Punkten (wiederum in der Regel zufällig) generiert. In Anlehnung an den Prozess der Evolution bezeichnet man die so neu generierten Punkte als Kinder, die ursprünglich aktuelle Menge als Eltern. Am Ende einer solchen Runde steht die Auswahl der Punkte, die am Anfang der nächsten Runde die aktuelle Menge von Punkten bilden. Weil typischerweise die Größe der Population während eines gesamten Laufs konstant bleibt, werden bei dieser abschließenden Selektion einige Individuen durch andere ersetzt. Man nennt darum diese Auswahl auch *Selektion zur Ersetzung*. Üblicherweise hängen Mutation und Rekombination nur von den Individuen selbst, nicht aber ihrer Fitness ab. Diese spielt dann bei der Selektion eine wichtige Rolle, in vielen Algorithmen hängt die Selektion auch ausschließlich von der Fitness ab.

Die wohl naheliegendste Anwendung evolutionärer Algorithmen ist die Optimierung. In einem Suchraum sucht man nach möglichst guten Punkten, wobei die Güte durch die Fitness f gegeben ist. Wählt man als Fitness die Zielfunktion und findet ein evolutionärer Algorithmus einen besten Punkt, so hat man die Funktion erfolgreich optimiert. Wir werden uns in dieser Arbeit ausschließlich um diesen Aspekt kümmern, die Optimierung statischer Zielfunktionen. Dabei beschränken wir uns auf Funktionen $f\colon \{0,1\}^n \rightarrow \mathbb{R}$. Die Wahl von $V = \mathbb{R}$ bedeutet, dass wir den Problemkreis der multikriteriellen Optimierung aus unseren Betrachtungen ausschließen. Die Wahl von $S = \{0,1\}^n$, also im wesentlichen das Festlegen auf einen diskreten Suchraum, hat erhebliche methodische Konsequenzen. Wir halten es in Bezug auf praktische Fragestellungen aber nicht für eine wesentliche Einschränkung. Bei der Implementierung auf Digitalcomputern werden nicht-diskrete Suchräume notwendig auf diskrete Suchräume abgebildet, insbesondere werden reelle

Zahlen (meistens implizit, oft naiv ohne Abschätzung der Folgen) auf Bitstrings abgebildet. Insofern sind alle in der Praxis auftretenden Fragen prinzipiell unter diesen Bedingungen beantwortbar.

Es gibt inzwischen eine schier unüberschaubar große Anzahl von Veröffentlichungen, die in erster Linie den erfolgreichen Einsatz evolutionärer Algorithmen demonstrieren. Begleitend dazu gibt es eine große Anzahl theoretisch orientierter Arbeiten, wobei allerdings von Anfang an heuristische Analysen, die auf zwar einsichtig erscheinenden, allerdings unbewiesenen Annahmen beruhen, im Vordergrund stehen. Typisch für solche Arbeiten ist auch eine naheliegende aber ungerechtfertigte Interpretation der tatsächlich bewiesenen Aussagen weit über ihrem Rahmen hinaus. Die Diskussionen um das Schema Theorem (Holland 1992) und die Building Block Hypothese sind das deutlichste Beispiel hierfür.

Diese Arbeit ist anders angelegt. Es geht hier nicht um eine Theorie evolutionärer Algorithmen, welche diese Algorithmenklasse vollständig durchdringt und erklärt. Selbstverständlich ist ein derart umfassendes Verständnis ein erstrebenswertes visionäres Fernziel und es ist zu hoffen, dass diese Arbeit einen Schritt in Richtung eines solchen Fernziels darstellt. Als sichere Fußtritte auf einem solchen Weg sollen kleine Beispiele dienen, an denen grundlegende Eigenschaften bestimmter evolutionärer Algorithmen exemplarisch verdeutlicht werden. Wir werden also stets über ganz bestimmte Algorithmen und ganz bestimmte Funktionen oder Funktionenfamilien sprechen. Diese Beschränkung hilft uns wesentlich, zu klaren und klar bewiesenen Aussagen zu kommen, die einem noch zu errichtenden größerem Theoriegebäude als sicheres Fundament dienen können.

Unser Augenmerk gilt der Frage, wie lange der betrachtete evolutionäre Algorithmus braucht, um ein globales Optimum der betrachteten Funktion zu finden. Dabei verlangen wir insbesondere nicht, dass der Algorithmus „weiß“, dass er ein Optimum gefunden hat. Weil evolutionäre Algorithmen randomisierte Suchverfahren sind, gibt es unterschiedliche Typen von Antworten, die wir auf diese Frage geben können. Man kann die Anzahl der Generationen oder Schritte, die der betrachtete evolutionäre Algorithmus bis zum Erreichen eines globalen Optimums braucht, als Zufallsvariable T auffassen. Eine naheliegende Antwort auf die Frage nach der Zeit zum Optimum ist die Angabe der Anzahl der erwarteten Generationen $\mathrm{E}(T)$. Wir werden gelegentlich tatsächlich genau so antworten. Allerdings hat der Erwartungswert eine Reihe von Nachteilen, die dazu führen, dass wir mitunter stärkere Aussagen machen. Das wesentliche Problem ist, dass der Erwartungswert keine Aussagen darüber macht, mit welcher Wahrscheinlichkeit die Laufzeit erheblich vom

Erwartungswert abweicht. Zwar folgern wir mittels Markov-Ungleichung (für einige mathematische Grundlagen siehe Anhang A), dass die Wahrscheinlichkeit, mehr als $t \cdot \mathrm{E}(T)$ Generationen zu brauchen, nach oben durch $1/t$ beschränkt ist. Auf der anderen Seite ist aber überhaupt nicht klar, dass nicht auch wesentlich kleinere Laufzeiten mit großer Wahrscheinlichkeit möglich sind. Wir werden in Kapitel 6 ein konkretes Beispiel diskutieren, bei dem die erwartete Laufzeit zwar exponentiell groß ist, aber mit Wahrscheinlichkeit beinahe 1 nur polynomiell viele Generationen benötigt werden.

Eine Antwort anderen Typs auf die Frage nach der Laufzeit ist die Angabe einer Wahrscheinlichkeit $p(n)$, mit der eine Laufzeit T nicht überschritten wird. Man kann ausgehend von einer Aussage dieses Typs direkt eine Aussage über die erwartete Laufzeit eines evolutionären Algorithmus machen. Wenn Algorithmus A mit Wahrscheinlichkeit $p(n)$ nach höchstens $T(n)$ Schritten das Optimum einer Funktion $f\colon \{0,1\}^n \to \mathbb{R}$ gefunden hat, so kann man offenbar einen Algorithmus A' mit erwarteter Laufzeit $(1/p(n))T(n)$ angeben, indem man einfach Algorithmus A nach $T(n)$ Schritten abbricht und neu startet. Wenn wir annehmen, dass $T(n)$ nach oben und $p(n)$ nach unten beschränkt sind durch Polynome in n (der Dimension des Suchraums), so ist die erwartete Laufzeit von A' polynomiell beschränkt.

Wir sehen also, dass die beiden Aussagetypen miteinander verknüpft sind. Eine Aussage über den Erwartungswert $\mathrm{E}(T(n))$ liefert eine Aussage, dass mit Wahrscheinlichkeit $1/t$ nicht mehr als $t \cdot \mathrm{E}(T(n))$ Schritte benötigt werden. Und eine Aussage, dass mit Wahrscheinlichkeit $p(n)$ nicht mehr als $T(n)$ Schritte benötigt werden, liefert einen Algorithmus mit erwarteter Schrittzahl $T(n)/p(n)$. Wie erwähnt kann es Funktionen geben, bei denen ein evolutionärer Algorithmus zwar keine polynomielle erwartete Laufzeit hat, er aber in polynomieller Zeit mit Wahrscheinlichkeit $1/p(n)$ für ein Polynom p ein globales Optimum findet. Der zweite Aussagetyp erlaubt also in diesem Sinne, eine größere Klasse von Funktionen als effizient optimierbar zu bezeichnen und deckt sich auch besser mit dem Einsatz evolutionärer Algorithmen in der Praxis, wo ein Abbruch nach einer gewissen Zeit oder einer augenscheinlichen Stagnation des Optimierungsprozesses und ein anschließender Neustart üblich sind.

Ziel dieser Arbeit ist es, ein theoretisch gesichertes Verständnis evolutionärer Algorithmen zu gewinnen. Wir werden uns dazu auf einfache evolutionäre Algorithmen beschränken, hauptsächlich betrachten wir den (1+1) evolutionären Algorithmus, den man als einfachsten evolutionären Algorithmus überhaupt bezeichnen kann. Eine zentrale Rolle spielen Beispielfunktionen, die wichtige Eigenschaften im überspitzten Sinne haben und dadurch Phänomene, von denen wir annehmen, dass sie von allgemeiner Bedeutung sind, klar erkennbar und verständlich machen.

Wir beginnen unsere Untersuchungen mit der Einordnung evolutionärer Algorithmen in den allgemeinen Kontext der Black Box Optimierung in Kapitel 2. Wir diskutieren dabei grundsätzliche Beschränkungen von Algorithmen für Black Box Optimierung. Ausgehend davon beschreiben wir verschiedene Szenarien, in denen Black Box Optimierung in praktisch relevanter Weise betrieben werden kann und rechtfertigen den Einsatz und die Untersuchung evolutionärer Algorithmen auf sehr grundsätzliche Weise. Neben dem hilfreichen Blickwinkel auf das Problemfeld aus einer übergeordneten, allgemeinen Perspektive findet man in diesem Abschnitt eine Motivation für die an einfachen Algorithmen und Beispielen orientierte Vorgehensweise in den folgenden Kapiteln, in denen dann jeweils sehr konkrete Fragestellungen und Analysen im Vordergrund stehen.

In Kapitel 3 führen wir formal den (1+1) evolutionären Algorithmus ein, dessen Analyse den größten Teil dieser Arbeit einnimmt. Ein erstes wichtiges Ergebnis für diesen Algorithmus ist die erwartete Laufzeit auf der Klasse der linearen Funktionen, das wir uns schrittweise in den Kapiteln 3 und 4 erarbeiten werden. Daran schließt sich die Betrachtung der Klasse von unimodalen Funktionen an, die eine echte Obermenge der vollständig nicht-trivialen linearen Funktionen darstellt. Wir diskutieren in Kapitel 5 allgemeine obere Schranken für die erwartete Laufzeit und geben eine exponentielle untere Schranke für eine unimodale Funktion an. Exponentielle erwartete Laufzeiten und somit schwierige Funktionen beschäftigen uns auch in Kapitel 6, wo wir verschiedene wichtige schwierige Funktionen betrachten. Anschließend beginnen wir in Kapitel 7, uns allgemeineren evolutionären Algorithmen zu widmen. Als ersten Schritt auf diesem Weg betrachten wir Algorithmen, die durch Variation der Mutation des (1+1) EA entstehen. Dabei werden wir einerseits die Wahl der Mutationswahrscheinlichkeit untersuchen. Andererseits betrachten wir auch einen strukturell einfacheren Mutationsoperator. In Kapitel 8 führen wir dieselbe Idee in etwas anderer Weise weiter, indem wir uns Variationen der Selektion ansehen. Unter anderem greifen wir dabei auch die zuvor betrachtete einfachere Mutation wieder auf und untersuchen in diesem Kontext die Unterschiede zwischen den bekannten Suchheuristiken Simulated Annealing (Kirkpatrick, Gelatt und Vecchi 1983) und dem Metropolis-Algorithmus (Metropolis, Rosenbluth, Rosenbluth, Teller und Teller 1953). Wir stellen strukturell sehr einfache Beispiele vor, an denen sich leicht und anschaulich beweisen lässt, dass Simulated Annealing in gewissen Situationen erheblich besser sein kann als ein optimal eingestellter Metropolis-Algorithmus. Dem bis dahin nicht betrachteten in der Praxis aber wichtigen Operator Crossover nähern wir uns in Kapitel 9. Wir diskutieren ein Beispiel, bei dem ein evolutionärer Algorithmus mit Rekombination rein mutationsbasierte evolutionäre Algorithmen um Größenordnungen schlägt. In Kapitel 10 schließlich findet man zum einen eine kurze Zusammenfassung der Arbeit, wie man sie üblicherweise zu finden erwartet. Wir gehen allerdings über diesen Rahmen hinaus und beschäftigen uns mit möglichen zukünftigen Forschungsrichtungen, wobei wir eine mögliche Richtung konkretisieren und exemplifizieren. Wir zeigen konkret und beispielhaft, wie man an Stelle der im Rahmen dieser Arbeit betrachteten „künstlichen Beispielfunktionen“ auch „natürliche Probleme“ in Zusammenhang mit evolutionären Algorithmen stellen kann. Die Analyse evolutionärer Algorithmen rückt auf diese Weise näher an das etablierte Gebiet der Analyse randomisierter Algorithmen heran. In den Anhängen findet man dann noch einerseits einige mathematische Grundlagen und Hilfsmittel (Anhang A), andererseits eine Übersicht über die betrachteten Beispielfunktionen (Anhang B), von denen wir annehmen, dass sie zum Teil auch in Zukunft bei weiterer Erforschung evolutionärer Algorithmen nützlich sein können.

2 Möglichkeiten und Grenzen evolutionärer Algorithmen

Evolutionäre Algorithmen gelten gemeinhin als allgemeine, robuste Suchheuristiken, die bei einer großen Vielzahl unterschiedlicher Funktionen zur Optimierung eingesetzt werden können, ohne dass eine Anpassung schwierig oder auch nur notwendig ist. Es ist kaum etwas bekannt über Grenzen der Einsetzbarkeit evolutionärer Algorithmen, mitunter wird sogar bezweifelt, dass es überhaupt seriöse Beispiele für Funktionen gibt, auf denen evolutionäre Algorithmen versagen. Meist wird dabei implizit unterstellt, dass die Optimierung in einem allgemeinen Black Box Szenarium abläuft, das man wie folgt etwas formaler fassen kann. Die zu optimierende Zielfunktion $f\colon S \to V$ ist vor dem Algorithmus in einer Black Box verborgen. Der Zugriff auf die Funktion geschieht ausschließlich über ein Orakel. Der Algorithmus kann Orakelfragen stellen, indem er einen Punkt $s \in S$ als Eingabe an die Black Box übergibt. Als Antwort erhält er den zugehörigen Funktionswert $f(s)$.

Wenn man annimmt, dass der Algorithmus keinerlei weitere Information über die zu optimierende Funktion besitzt, ist leicht einzusehen, dass nicht viel erreicht werden kann. Um diesen Punkt auch formal einsehbar zu machen, ist es hilfreich, einige vereinfachende Annahmen zu machen. Zunächst nehmen wir an, dass S und V endliche Mengen sind. Die Annahme, dass S endlich ist, deckt sich mit unserer Grundannahme $S = \{0,1\}^n$. Für jede praktisch vorkommende Situation ist das offenbar auch gegeben. Die Einschränkung von V auf eine endliche Menge ist ebenfalls in praktischer Hinsicht nicht wesentlich: Viele evolutionäre Algorithmen betrachten nur die Ordnung der Punkte im Suchraum und gar nicht deren konkrete Funktionswerte. Für solche Algorithmen kann man offenbar ohne Einschränkung $|V| \leq |S|$ annehmen. Man kann zum Beispiel $V = \{1, 2, \ldots, |S|\}$ wählen. Zusätzlich wird vereinbart, als einzige Ressource die Anzahl *unterschiedlicher* Orakelanfragen zu berücksichtigen. Evolutionäre Algorithmen stellen in der Regel Orakelanfragen für ein und denselben Suchpunkt im Laufe einer Optimierung auch mehrfach. Da nur die Orakelanfragen selbst gezählt werden, kann das aber offensichtlich durch Verwendung eines Wörterbuches, in dem alle Orakelanfragen und die zugehörigen Funktionswerte gespeichert werden, ohne gemessenen Mehraufwand vermieden werden. In diesem nun formal festgelegten Rahmen wollen wir die durchschnittliche Effizienz von Optimierungsalgorithmen vergleichen. Dabei ergibt sich ein einfaches und umfassendes Resultat, das in gewisser Weise nicht überraschend ist, welches in der Gemeinde aber für erhebliche Aufregung und geradezu ausufernde Diskussionen gesorgt hat. Die

Rede ist hier vom so genannten No Free Lunch Theorem (oder auch kurz NFL-Theorem), das von Wolpert und Macready (1997) formuliert und bewiesen wurde.

Satz 2.1 (NFL-Theorem (Wolpert und Macready 1997)). *Es seien S und V zwei endliche Mengen. Sei V total geordnet. Sei F die Menge aller Funktionen $f\colon S \to V$. Für alle (randomisierten und deterministischen) Algorithmen ist die durchschnittliche Anzahl unterschiedlicher Zielfunktionsaufrufe bis zum ersten Erreichen eines globalen Maximums gemittelt über alle Funktionen $f \in F$ gleich.*

Beweis. Wir führen den Beweis zunächst nur für deterministische Algorithmen und überlegen uns dann im Anschluss, warum sich die Aussage leicht auf randomisierte Algorithmen verallgemeinern lässt. Seien also A_1 und A_2 zwei beliebige deterministische Algorithmen, die Funktionen $f \in F$ maximieren.

Man kann einen solchen Algorithmus als $|V|$-ären Baum modellieren: Jeder Knoten des Baumes ist mit einem Element $s \in S$ markiert, alle inneren Knoten haben genau $|V|$ ausgehende Kanten, die mit den $|V|$ verschiedenen Elementen aus V markiert sind. Wir betrachten die $|V|$ Kanten als angeordnet, dabei ist die i-te Kante mit dem i-ten Element aus V markiert. Auf jedem Pfad von der Wurzel zu einem Blatt kommt jedes Element $s \in S$ genau einmal vor. Die Interpretation ist die folgende. An der Wurzel des Baumes steht das Element $s \in S$, für das als erstes der Funktionswert nachgefragt wird. Für jeden möglichen Ausgang der Orakelanfrage gibt es einen eindeutigen Nachfolger des Knotens, der das nächste anzufragende Element definiert.

Zu einer beliebig gewählten Funktion $f \in F$ gibt es nun genau einen Pfad im Baum, der angibt, in welcher Reihenfolge die Zielfunktionsaufrufe für diese spezielle Funktion f durchgeführt werden. Wir interessieren uns für die Anzahl unterschiedlicher Zielfunktionsaufrufe bis zum ersten Erreichen eines globalen Maximums. Darum definieren wir als Länge dieses Pfades die Anzahl der Knoten, die man an der Wurzel beginnend entlang laufen muss, bis man erstmals einen mit $s \in S$ markierten Knoten erreicht, für den $f(s) = \max\{f(x) \mid x \in S\}$ gilt. Die Pfadlänge liegt zwischen 1 und $|S|$. Die Behauptung ist, dass für jeden deterministischen Algorithmus (also für jeden solchen Baum) die durchschnittliche Pfadlänge gemittelt über alle Funktionen gleich ist. Wir zeigen das mittels vollständiger Induktion über die Größe des Suchraums $|S|$.

Für $|S| = 1$ besteht jeder Baum nur aus der Wurzel, für jede Funktion ist die Pfadlänge 1 und die Aussage ist trivialerweise richtig. Sei die Behauptung zutreffend für alle Suchräume S' mit $|S'| < |S|$. Wir betrachten zwei deterministische Algorithmen A_1 und A_2, die wir mit den zugehörigen Bäumen identifizieren. Seien $s_1, s_2 \in S$ die Markierungen an den Wurzelknoten. Weil F *alle* Funktionen $f\colon S \to V$ enthält, ist die Anzahl der Funktionen, für die $s_1 \in S$ maximal ist, genauso groß wie die Anzahl der Funktionen, für die s_2 maximal ist. Diese Funktionen liefern also zur durchschnittlichen Pfadlänge von A_1 und A_2 gleichen Anteil. Wir betrachten jetzt also nur noch die Funktionen f, für die weder $f(s_1)$ noch $f(s_2)$ maximal ist. Dann ergibt sich sowohl für A_1 als auch für A_2 als Pfadlänge 1 zuzüglich der Pfadlänge im einschlägigen Teilbaum, dessen Wurzel durch $v_1 := f(s_1)$ bzw. $v_2 := f(s_2)$ bestimmt ist. Die Werte v_1 und v_2 hängen natürlich nicht nur von A_1 bzw. A_2, sondern auch von f ab. Diese Teilbäume können wir interpretieren als deterministische Programme A'_1 und A'_2, die Funktionen $f_1\colon S_1 \to V$ bzw. $f_2\colon S_2 \to V$ maximieren, dabei sind $S_1 := S \setminus \{s_1\}$ und $S_2 := S \setminus \{s_2\}$. Als „Restfunktionen“ kommen die Funktionen aus

$$F_1 := \{f_1\colon S_1 \to V \mid \exists f' \in F\colon\ (\forall x \in S_1\colon f_1(x) = f'(x)) \wedge (f'(s_1) = v_1)\}$$

bzw.

$$F_2 := \{f_2\colon S_2 \to V \mid \exists f' \in F\colon\ (\forall x \in S_2\colon f_2(x) = f'(x)) \wedge (f'(s_2) = v_2)\}$$

in Frage. Weil F *alle* Funktionen $f'\colon S \to V$ enthält, sind

$$F_1 := \{f_1\colon S_1 \to V\} \qquad \text{und} \qquad F_2 := \{f_2\colon S_2 \to V\}\,.$$

Man sieht durch Umbenennung sofort ein, dass F_1 und F_2 isomorphe Funktionen enthalten, also wesentlich gleich sind. Es ist $|S_1| = |S_2| = |S|-1$, wir sind also in der gleichen Situation wie anfangs mit in der Größe um 1 reduzierten Suchraum. Folglich gilt gemäß Induktionsannahme die Behauptung.

Wir wollen jetzt zeigen, dass die Aussage sogar für alle randomisierten Algorithmen stimmt. Ein randomisierter Algorithmus kann ähnlich wie ein deterministischer Algorithmus als ein Baum beschrieben werden. Allerdings hängt sein Verhalten zusätzlich vom Ausgang von Zufallsexperimenten ab. Wir haben deterministische Algorithmen als Bäume beschrieben, wobei wir alle Knoten, die Tiefe i haben, zu der i-ten Ebene zusammenfassen. Man kann jetzt einen randomisierten Algorithmus zur Optimierung von Funktionen aus F beschreiben als einen solchen Baum, wobei oberhalb jeder Ebene noch eine Ebene eingeschoben wird, auf der Zufallsexperimente durchgeführt werden können. Es ist klar, dass der Zeitpunkt der Durchführung der Zufallsexperimente für die Ausführung des Algorithmus unerheblich ist. Wir können uns darum vorstellen, dass alle Zufallsexperimente am Anfang des Algorithmus durchgeführt werden und der Algorithmus dann deterministisch (aber in Abhängigkeit von den anfangs festgelegten Zufallsbits) arbeitet. Wir können folglich einen randomisierten Algorithmus in diesem Fall als Wahrscheinlichkeitsverteilung auf den deterministischen Algorithmen beschreiben. Weil für alle deterministischen Algorithmen die durchschnittliche Pfadlänge gleich ist, gilt das natürlich auch für jede gewichtete Summe von deterministischen Algorithmen, also auch für jeden randomisierten Algorithmus. □

Im Grunde ist die Aussage, dass man aus bisher betrachteten Punkten im Suchraum nichts weiter über die zu optimierende Funktion erfahren kann, wenn tatsächlich *alle* Funktionen gleichwahrscheinlich sind. Das ist zwar anschaulich klar, hat aber manche im ersten Moment vielleicht überraschend klingende Konsequenzen. Zum einen kann kein Algorithmus im Durchschnitt besser sein als eine Durchmusterung des Suchraums in irgendeiner Reihenfolge oder eine rein zufällige Irrfahrt durch den Suchraum. Zum zweiten ergeben sich für klar strukturierte Suchverfahren wie etwa die lokale Suche geradezu absurd anmutende Schlussfolgerungen. Eine lokale Suche beruht auf der Definition einer Nachbarschaft im Suchraum. In jedem Schritt gibt es einen aktuellen Punkt, dessen Funktionswert bereits durch Orakelanfrage festgestellt wurde. Für alle seine Nachbarn wird das gleiche getan, dann wählt man nach einer festen Regel einen der Nachbarn als neuen aktuellen Punkt aus. Das wird so lange iteriert, bis keine Veränderung mehr stattfindet. Aus dem NFL-Theorem folgt, dass es beim Einsatz der lokalen Suche für die Maximierung keine Rolle spielt, ob aus der durchsuchten Nachbarschaft ein Punkt mit maximalem oder minimalem Funktionswert als nächster Punkt gewählt wird. Für evolutionäre Algorithmen ergibt sich als Konsequenz, dass diese Algorithmen im Durchschnitt nicht besser sein können, als beliebige andere Verfahren.

Das NFL-Theorem ist zum einen wegen dieser augenscheinlich absurden Folgerungen kritisiert worden. Zum anderen widerspricht es aber auch in eklatanter Art und Weise praktischen Erfahrungen. Es ist eine elementare empirische Tatsache, dass es „gute“ und „weniger gute“ Optimierverfahren gibt. Dieser scheinbare Widerspruch ist aber in der Tat leicht zu erklären, wenn man sich die dem NFL-Theorem zugrundeliegenden Annahmen etwas genauer ansieht (Droste, Jansen und Wegener 1999).

Eine für die Korrektheit des NFL-Theorems zentrale Grundannahme ist, dass alle Funktionen $f\colon S \to V$ mit gleicher Wahrscheinlichkeit als Zielfunktion auftreten. Wir wollen zunächst von der allgemeinen, abstrakten Grundsituation weg zu einem realistischen Beispiel. Nehmen wir an, wir wollen eine

Funktion f maximieren, die Bitstrings der Länge 100 auf eine reelle Zahl abbildet, die intern durch einen Bitstring der Länge 32 codiert wird. Wir haben also $S = \{0,1\}^{100}$ und $V = \{0,1\}^{32}$. Die Anzahl aller Funktionen $g\colon \{0,1\}^{100} \to \{0,1\}^{32}$ beträgt dann $2^{32 \cdot 2^{100}}$. Wenn wir annehmen, dass wir die Zielfunktion im Hauptspeicher unseres Computers repräsentieren wollen und dieser Hauptspeicher eine Größe von 1 Gigabyte = 2^{33} Bit hat, dann kann nur ein Anteil von weniger als 2^{-65}% aller Funktionen tatsächlich vorkommen. Offensichtlich ist also die Annahme, dass alle Funktionen gleichermaßen als Zielfunktionen in Frage kommen schon für relativ moderate Größen von Such- und Werteraum unsinnig. Tatsächlich kann nur ein verschwindend geringer Bruchteil von Funktionen als Zielfunktion vorkommen. Es gilt also, gegenüber dem völlig uneingeschränkten Black Box Szenarium des NFL-Theorems realistisch eingeschränkte Black Box Szenarien zu entwerfen. Dann ist zu prüfen, ob in einem derart eingeschränkten Szenarium eine (eventuell angepasste) Fassung des NFL-Theorems bewiesen werden kann.

Wir wollen hier zwei grundsätzlich verschiedene Arten, wie man zu eingeschränkten Black Box Szenarien für die Optimierung kommen kann, unterscheiden. Zum einen kann man rein syntaktisch die Ressourcen Rechenzeit oder Speicherplatz beschränken. Wir werden diese Möglichkeiten gleich diskutieren. Ein anderer Weg besteht in der Einschränkung der für die Optimierung in Frage kommenden Funktionen, die in gewisser Weise auf die Semantik der Funktionen Bezug nimmt. Wir geben zunächst einige konkrete Beispiele für derartige Restriktionen an.

Optimierung einer Zielfunktion: Als Zielfunktion zugelassen ist nur eine ganz konkrete Funktion $f\colon \{0,1\}^n \to \mathbb{R}$. Das ist in der Praxis vermutlich der am häufigsten auftretende Fall, der aber von keinerlei theoretischem Interesse ist. Offensichtlich kann man ein Optimierverfahren angeben, dass als ersten Punkt im Suchraum ein globales Maximum testet. Ein optimaler Algorithmus stellt in diesem Fall also nur genau eine Anfrage an das Orakel und ist trivial.

Optimierung eines Problems: Als Zielfunktionen zugelassen sind Funktionen, die Instanzen eines bestimmten Problems repräsentieren. Ein Problem hat in der Regel eine klare Semantik, was in gewisser Weise Strukturwissen ermöglicht. Konkrete Beispiele sind zum Beispiel das Traveling Salesperson Problem (TSP) oder Erfüllbarkeitsprobleme (SAT bzw. MAXSAT). Das ist der auf dem Gebiet der effizienten Algorithmen vermutlich am häufigsten untersuchte Fall. Auch für evolutionäre Algorithmen ist dieses Szenarium häufig realistisch. In der

Regel werden dann spezielle Mutations- und Crossoveroperatoren verwendet, die auf das Problem zugeschnitten sind und problemspezifisches Wissen verwenden. Ein spezieller evolutionärer Algorithmus für das TSP von Tao und Michalewicz (1998) ist ein gutes Beispiel dafür.

Optimierung eines Funktionstyps: Als Zielfunktionen zugelassen sind Funktionen, die bestimmte syntaktische oder semantische Eigenschaften haben. Konkrete Beispiele sind zum Beispiel Funktionen, die sich durch ein Polynom kleinen Grades darstellen lassen, oder unimodale Funktionen. Solche Funktionenklassen haben häufig den Vorteil, dass sie einer analytischen Betrachtung gut zugänglich sind.

Weniger drastisch als diese drei semantischen Einschränkungen sind rein syntaktisch definierte Szenarien, die lediglich die für das Orakel zur Verfügung stehenden Ressourcen Speicherplatz und Rechenzeit beschränken. Ein derart definiertes Black Box Szenarium kann als realistisch eingeschränkt betrachtet werden, wenn es nur noch solche Zielfunktionen zulässt, die mit den zur Optimierung zur Verfügung stehenden Ressourcen repräsentiert und ausgewertet werden können. Die Forderung nach Repräsentierbarkeit entspricht im wesentlichen einer Begrenzung der Größe der Funktion in der gewählten Darstellung. Die Forderung nach der Auswertbarkeit entspricht im wesentlichen einer Begrenzung der Rechenzeit, die für die Berechnung eines Funktionswertes benötigt wird. Natürlich besteht im allgemeinen zwischen der Größe der Darstellung und der Zeit für eine Zielfunktionsauswertung ein Zusammenhang. Wir skizzieren hier kurz verschiedene mögliche Restriktionen, wobei wir uns auf prinzipielle Grenzen beschränken wollen. Eine konkrete Festlegung ist im einzelnen oft schwierig und sollte auch die konkret zur Verfügung stehenden Ressourcen berücksichtigen.

zeitbeschränkte Black Box Optimierung: Als Zielfunktionen zugelassen sind Funktionen, bei denen eine Auswertung (also eine Orakelanfrage) in einem fest gewählten Berechnungsmodell höchstens t Rechenschritte erfordert. Dabei ist t eine fest gewählte Größe.

größenbeschränkte Black Box Optimierung: Als Zielfunktionen zugelassen sind Funktionen, deren Repräsentation in einer fest gewählten Darstellungsart eine Größe von höchstens s hat. Dabei ist s eine fest gewählte Größe.

Kolmogorov-komplexitätsbeschränkte Black Box Optimierung: Als Zielfunktionen zugelassen sind Funktionen, deren Kolmogorov-Komplexität höchstens k ist. Dabei ist k eine fest gewählte Größe. Die

> Kolmogorov-Komplexität eines Bitstrings x ist die minimale Länge eines Programms für eine fest gewählte universelle Turingmaschine, die den Bitstring x als Ausgabe produziert.

Den drei durch syntaktische Ressourcenbeschränkungen definierten Black Box Szenarien ist gemeinsam, dass sie zunächst für beliebige Funktionen definiert sind. Wir werden uns im Rahmen dieser Arbeit niemals mit einzelnen Funktionen beschäftigen. Es werden ausschließlich Funktionsfamilien $f = (f_n)_{n\geq 1}$ definiert und betrachtet, bei denen also für alle (oder zumindest für unendlich viele) Werte von n eine Instanz f_n definiert ist. Weil wir einzelne Funktionen nicht betrachten, werden wir eine solche Funktionsfamilie f im folgenden kurz als Funktion bezeichnen. Weil die Funktion f für verschiedene Werte von n definiert ist, wird es im allgemeinen sinnvoll sein, die verwendeten Schranken auch von n abhängen zu lassen. Etwas weniger konkrete aber sicher sinnvolle Minimalforderungen sind dann etwa, dass die Auswertung eines Zielfunktionsaufrufs deterministisch in Polynomialzeit erfolgen kann, oder dass die Größe der Repräsentation der Funktion polynomiell beschränkt ist. Dabei verwenden wir Polynome in n, der Dimension des Suchraums $\{0,1\}^n$.

Der Kern des No Free Lunch Theorems ist, dass das Zulassen aller Funktionen impliziert, dass der Funktionswert einer rein zufällig gewählten Funktion an einer Stelle unabhängig von den Funktionswerten an anderen Punkten im Suchraum ist. Wir haben uns schon oben überlegt, dass diese vollständige Unabhängigkeit ihren Preis hat; bei realistisch eingeschränkten Szenarien muss es Abhängigkeiten geben. Es ist also grundsätzlich möglich, dass ein Optimierungsverfahren diese Abhängigkeiten, die eine gewisse Struktur implizieren, ausnutzt, um mit überdurchschnittlicher Effizienz ein globales Maximum zu finden. Leider ist es beim derzeitigen Stand des Wissens nicht möglich, für beliebige Festlegungen von $|S|$ und $|V|$ und zufällig gewählte Funktionen allgemeine Aussagen zu machen. Wir beschränken uns darum auf eine bestimmte Situation, bei der wir für S und V sehr kleine Mengen wählen. Mittels vollständiger Durchmusterung weisen wir an diesem kleinen Beispiel exakt nach, dass es für eingeschränkte Szenarien im allgemeinen kein No Free Lunch Theorem geben kann.

Wir setzen $S := \{0,1\}^3$ und $V := \{0,1\}^2$, die lexikographische Ordnung benutzen wir als Ordnung auf V. Es gibt insgesamt $2^{2\cdot 2^3} = 65536$ verschiedene Funktionen $f\colon S \to V$. Es ist nicht ganz einfach, vernünftige Ressourcenbeschränkungen zu definieren, die eine derart kleine Funktionenklasse sinnvoll einschränken. Wir betrachten darum sieben verschiedene Restriktionen, für die wir jeweils zeigen, dass es zwischen verschiedenen Algorithmen tatsächlich Unterschiede gibt.

Eine naheliegende mögliche Repräsentation der Funktionen sind Schaltkreise. Wir werden tatsächlich größenbeschränkte Schaltkreise als eine Möglichkeit betrachten, allerdings sind Schaltkreise schon in diesen geringen Größen nur schlecht minimierbar, was unsere Problemstellung aber erfordert. Wir verwenden darum ganz wesentlich OBDDs, die von Bryant (1986) als Datenstruktur für boolesche Funktionen eingeführt worden sind und wesentlich günstigere Eigenschaften haben.

Ein OBDD (ordered binary decision diagram) ist ein gerichteter, azyklischer Graph, bei dem außer den Senken jeder Knoten genau zwei ausgehende Kanten hat, von denen eine mit 0 und eine mit 1 markiert ist. Die Senken sind mit 0 oder 1 markiert, jeder andere Knoten ist mit einer Variablen markiert. Es gibt zu einem OBDD eine Ordnung auf der Menge der vorkommenden Variablen, so dass für jeden gerichteten Pfad im OBDD gilt, dass die Reihenfolge der auf dem Pfad vorkommenden Variablen der Ordnung genügt. Jeder Knoten im OBDD stellt eine boolesche Funktion dar. Eine mit 0 markierte Senke repräsentiert die konstante Nullfunktion, eine mit 1 markierte Senke die Einsfunktion. Wenn ein Knoten mit der Variablen x_i markiert ist und als Nullnachfolger einen Knoten hat, an dem die Funktion f_0 repräsentiert wird und als Einsnachfolger einen Knoten, der die Funktion f_1 repräsentiert, so wird an diesem Knoten die Funktion $\overline{x_i} f_0 \vee x_i f_1$ dargestellt.

Um die Unabhängigkeit der Ergebnisse vom verwendeten Komplexitätsmaß nachzuweisen, verwenden wir neben Schaltkreisen und OBDDs noch fünf andere Restriktionsmöglichkeiten. Wir stellen die insgesamt sieben verschiedenen Szenarien vor und geben anschließend eine kurze Motivation für die verschiedenen Möglichkeiten. Wir benötigen verschiedene, in der Informatik wohlbekannte Begriffe und Modellierungen, um die verschiedenen Definitionen zu erklären, die wir stets zumindest kurz erläutern.

- Die Menge C enthält alle Funktionen, die mit Schaltkreisen über der Basis $\{\wedge, \vee, \neg\}$ und höchstens drei Bausteinen dargestellt werden können.
- Die Menge B_i enthält alle Funktionen, die mit OBDDs mit höchstens i inneren Knoten dargestellt werden können. Die Menge B_0 enthält also genau die vier konstanten Funktionen.
- Die Menge M_i enthält alle Funktionen, die eine Minimalpolynomdarstellung der Größe höchstens i haben. Die Größe der Minimalpolynomdarstellung einer Funktion $f\colon \{0,1\}^3 \rightarrow \{0,1\}^2$ mit $f = (f_1, f_2)$ ist

die Summe der Größen der Minimalpolynomdarstellungen ihrer beiden Subfunktionen f_1 und f_2. Eine Polynomdarstellung einer Funktion $f'\colon \{0,1\}^3 \to \{0,1\}$ ist eine Disjunktion von Konjunktionen von Literalen über $\{x_1, x_2, x_3\}$. Die Größe einer solchen Polynomdarstellung ist die Anzahl insgesamt vorkommender Literale. Ein Minimalpolynom ist eine solche Polynomdarstellung minimaler Größe. Die Konstanten 0 und 1 haben Größe 0, darum enthält M_0 ebenfalls genau die vier konstanten Funktionen.

- Die Menge G_i enthält alle Funktionen, bei denen folgendes „Glattheitskriterium“ erfüllt ist. Alle Suchpunkte $x, y \in \{0,1\}^n$, deren Hammingabstand $H(x, y)$ höchstens 1 beträgt, haben Funktionswerte $f(x)$ und $f(y)$, die sich höchstens um i unterscheiden. Die Menge G_0 enthält also ebenfalls genau die vier konstanten Funktionen. Der Hammingabstand $H(x, y)$ zweier Strings $x, y \in \{0,1\}^n$ ist die Anzahl der Stellen, an denen sich x und y unterscheiden.

- Die Menge E_i enthält alle Funktionen, die höchstens Partitionszahl i haben. Die Partitionszahl einer Funktion $f\colon \{0,1\}^3 \to \{0,1\}^2$ mit $f = (f_1, f_2)$ ist die Summe der Partitionszahlen ihrer beider Subfunktionen f_1 und f_2. Die Partitionszahl einer Funktion $f'\colon \{0,1\}^3 \to \{0,1\}$ ist die Anzahl von Äquivalenzklassen, die f' auf $\{1,2,3\}$ induziert. Zwei Indizes i und j mit $1 \leq i \leq j \leq 3$ heißen äquivalent, wenn für alle $x \in \{0,1\}^3$ gilt, dass der Austausch von Bit i und j in x den Funktionswert unter f' nicht ändert. Die Menge E_2 enthält also genau alle Funktionen, deren Funktionswert nur von der Anzahl der Einsen in der Eingabe abhängen, also gerade alle symmetrischen Funktionen.

- Die Menge A_i enthält alle Funktionen, die von höchstens i Variablen essenziell abhängen. Eine Funktion $f\colon \{0,1\}^3 \to \{0,1\}^2$ hängt von einer Variablen x_i mit $1 \leq i \leq 3$ essenziell ab, wenn es eine Eingabe $x \in \{0,1\}^3$ gibt, so dass eine Änderung des i-ten Bits in x den Funktionswert von f ändert. Die Menge A_0 enthält also genau die vier konstanten Funktionen.

- Die Menge R_i enthält alle Funktionen, die eine RLE-Darstellung der Länge höchstens i haben. Die Länge der RLE-Darstellung (run length encoding) einer Funktion $f\colon \{0,1\}^3 \to \{0,1\}^2$ ist um eins größer als die Anzahl von Bitstrings $x \in \{0,1\}^3 \setminus \{111\}$, so dass sich der Funktionswert von f für x von dem Funktionswert für den direkten Nachfolger von x bezüglich der Ordnung auf $\{0,1\}^3$ unterscheidet. Die Menge R_1 enthält also ebenfalls genau die vier konstanten Funktionen.

Schaltkreise (C) sind wie erwähnt die vermutlich naheliegendste Möglichkeit, allerdings nicht gut handhabbar. Das motiviert die Einführung von OBDDs (B_i), die vor allem effizient minimierbar sind (Wegener 2000). Die Darstellung als Minimalpolynom (M_i) ist eine alternative, naheliegende Möglichkeit der Darstellung. Die Mengen G_i sollen evolutionären Algorithmen entgegen kommen. Eine Grundvorstellung ist, dass „natürliche Funktionen" in gewissen Sinne „glatt" sind. Diese Vorstellung versuchen wir formal zu fassen, indem wir die Größe des Wechsels im Funktionswert bei einem kleinen Schritt (bezüglich Hammingabstand) nach oben beschränken. Die Einführung der über die Partitionszahl definierten Mengen E_i klassifiziert Funktionen in gewisser Weise nach dem Maß an Symmetrie, das sie aufweisen. Die Begrenzung der Anzahl von Variablen, von denen die Zielfunktion essenziell abhängt (A_i), kann als Beschränkung der Rechenzeit, die für eine Zielfunktionsauswertung zur Verfügung steht, verstanden werden. Wenn die Anzahl der Variablen, von denen eine Funktion essenziell abhängt, klein ist, brauchen nur wenige Variable getestet werden, was eine Auswertung potentiell schneller macht. Die Begrenzung der Länge der RLE-Darstellung ist in gewisser Weise ebenso wie die Begrenzung der Schaltkreisgröße und der OBDD-Größe eine Beschränkung des für die Repräsentation zur Verfügung stehenden Speicherplatzes. Die Idee bei der RLE-Darstellung ist, als Darstellung einfach den Wertevektor der Funktion zu verwenden und den benötigten Speicherplatz durch Anwendung einer einfachen Kompressionsmethode (run length encoding) zu verringern.

Wir haben jetzt insgesamt 53 (verschiedene) Funktionenmengen definiert, für die wir jetzt die durchschnittliche Effizienz verschiedener Optimierverfahren ansehen wollen. Wir betrachten drei verschiedene Arten von Algorithmen.

Die einfachsten denkbaren Optimierverfahren sind nicht-adaptive Algorithmen. Ein solcher Algorithmus durchsucht den Suchraum in einer fest vorgegebenen Reihenfolge, ohne dabei auf die auftretenden Funktionswerte zu reagieren. Der Suchraum ist $\{0,1\}^3$, enthält also genau 8 Elemente. Folglich gibt es $8! = 40320$ verschiedene nicht-adaptive Algorithmen. Wir untersuchen alle und stellen die Ergebnisse für den besten nicht-adaptiven Algorithmus (NA min), den schlechtesten nicht-adaptiven Algorithmus (NA max) sowie den Durchschnitt über alle nicht-adaptiven Algorithmen auf (NA avg). Letzteres entspricht einem randomisierten nicht-adaptiven Algorithmus, der anfangs rein zufällig eine Permutation wählt und den Suchraum dann in der so bestimmten Reihenfolge durchsucht. Wir bilden Minimum, Maximum und Durchschnitt dabei für jede Funktionenklasse neu. Es gibt also nicht einen festen nicht-adaptiven Algorithmus, der zum Beispiel die unter „NA min" angegebenen Werte für alle Funktionenklassen erreicht.

Als zweite Algorithmenklasse betrachten wir evolutionäre Algorithmen. Wir verwenden ausschließlich den (1+1) evolutionären Algorithmus (EA), den wir formal im nächsten Kapitel einführen und im Anschluss auch ausführlich untersuchen. Für den Moment genügt es zu wissen, dass der (1+1) EA ein einfacher, mutations- und selektionsbasierter evolutionärer Algorithmus ist.

Schließlich sehen wir uns noch problemspezifische Heuristiken an. Wir verwenden zwei verschiedene Heuristiken, die beide vom Wissen um die Funktionenklasse, aus der die Zielfunktion stammt, intensiven Gebrauch machen. Allerdings entwickeln wir keine Heuristiken, die auf bestimmte (semantische) Eigenschaften der Funktionenklassen Bezug nehmen. Der Algorithmus MaxOpt kennt zu jedem Zeitpunkt die Menge der Funktionen, die noch als Zielfunktion in Frage kommen. Als nächsten Punkt im Suchraum, für den er den Funktionswert anfragt, wählt er stets einen, der mit größter Wahrscheinlichkeit ein globales Maximum ist. Es sei F_i die dem Algorithmus bekannte Klasse von Funktionen, aus der die dem Algorithmus unbekannte Zielfunktion f stammt.

Algorithmus 2.2. MaxOpt

1. *$G := F_i$.*
2. *Für alle noch nicht angefragten $x \in \{0,1\}^3$*
   $m_x :=$ Anzahl Funktionen $g \in G$, so dass
   x ein globales Optimum für g ist.
3. *Frage den Funktionswert für ein x mit maximalem m_x an.*
4. *Entferne alle Funktionen $g \in G$ für die x global maximal ist oder für die $g(x) \neq f(x)$ gilt.*
5. *Weiter bei* 2.

Der andere Algorithmus MinMax verfolgt einen pessimistischeren Ansatz. Auch er weiß zu jedem Zeitpunkt um die noch in Frage kommenden Zielfunktionen. Als nächsten Punkt im Suchraum, für den er den Funktionswert anfragt, wählt er stets einen, bei dem, falls der Punkt nicht global maximal ist, die Menge von noch in Frage kommenden Funktionen möglichst klein wird. Es sei wiederum F_i die dem Algorithmus bekannte Klasse von Funktionen, aus der die dem Algorithmus unbekannte Zielfunktion f stammt.

Algorithmus 2.3. MinMax

1. *$G := F_i$.*
2. *Für alle noch nicht angefragten $x \in \{0,1\}^3$*
   Für alle $y \in \{0,1\}^2$
   $m_{x,y} :=$ Größe von G, wenn alle Funktionen g entfernt sind, bei denen $g(x)$ global maximal ist oder $g(x) \neq y$ gilt.
3. *Für alle $x \in \{0,1\}^3$*
   $m_x := \max\{m_{x,y} \mid y \in \{0,1\}^3\}$.
4. *Frage den Funktionswert für ein x mit minimalem m_x an.*
5. *Entferne alle Funktionen $g \in G$ für die x global maximal ist oder für die $g(x) \neq f(x)$ gilt.*
6. *Weiter bei 2.*

Vor dem konkreten Vergleich der Algorithmen stellen wir noch einige allgemeine Überlegungen an. Die im folgenden angegebenen Werte sind die exakten Durchschnitte für die deterministischen Algorithmen und die exakten erwarteten Durchschnitte für den evolutionären Algorithmus. Alle Rechnungen sind ohne Rundung vorgenommen worden, nur für die tabellarische Darstellung werden die Werte auf fünf Stellen gerundet. Unterschiedliche Durchschnitte bedeuten also, dass die Algorithmen auf der betroffenen Funktionenklasse tatsächlich unterschiedlich abschneiden. Für jedes der sieben Szenarien betrachten wir alle möglichen Teilmengen: für die parametrisierten Funktionenklassen lassen wir also den Parameter vom kleinstmöglichen Wert so lange wachsen, bis alle 65536 Funktionen enthalten sind. In diesem Fall gilt natürlich das No Free Lunch Theorem: alle Algorithmen schneiden exakt gleich ab. Speziell für den evolutionären Algorithmus sei noch einmal darauf hingewiesen, dass wir tatsächlich nur die Anzahl *unterschiedlicher* Orakelanfragen zählen. Das für evolutionäre Algorithmen typische mehrfache Anfragen eines Suchpunktes wird also nicht gezählt. In der folgenden Tabelle finden sich nun die konkreten Ergebnisse, die wir im Anschluss kurz diskutieren. Die Spalten enthalten die Bezeichnung der Menge (Bez.), die Anzahl der darin enthaltenen Funktionen (Anz.), und die (erwartete) durchschnittliche Anzahl angefragter Punkte im Suchraum für den (1+1) EA (EA), den Durchschnitt aller nicht-adaptiven Algorithmen (NA avg), den besten nicht-adaptiven Algorithmus (NA min) und den schlechtesten nicht-adaptiven Algorithmus (NA max), außerdem die Anzahlen für die beiden heuristischen Algorithmen MaxOpt und MinMax. Für den besten und schlechtesten nicht-adaptiven Algorithmus gilt, dass jeweils minimale und maximale Anzahl für die jeweilige Menge bestimmt wurde. Es handelt sich also wie erwähnt nicht um eine bestimmte Aufzählungsreihenfolge, die für alle Mengen besonders gut oder besonders schlecht ist.

Bez.	Anz.	EA	NA avg	NA min	NA max	MaxOpt	MinMax
C	1290	2.76497	2.74710	2.26744	3.55814	2.13953	2.15736
B_0	4	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000
B_1	34	1.79594	1.70588	1.44118	2.02941	1.44118	1.44118
B_2	280	2.29443	2.24408	1.96071	2.52143	1.94643	2.00714
B_3	2166	2.50887	2.48673	2.26362	2.69575	2.24331	2.23223
B_4	8908	2.87085	2.86225	2.69185	3.00707	2.64111	2.64042
B_5	25694	2.98287	2.97920	2.86974	3.05943	2.83518	2.84600
B_6	51520	3.02609	3.02528	2.98453	3.04926	2.96555	2.97977
B_7	65144	3.06387	3.06392	3.06126	3.06499	3.05928	3.06219
B_8	65536	3.06369	3.06369	3.06369	3.06369	3.06369	3.06369
M_0	4	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000
M_1	28	1.77320	1.68571	1.42857	2.00000	1.42857	1.42857
M_2	160	2.21372	2.16571	1.90000	2.41250	1.86875	1.96250
M_3	608	2.46543	2.44098	2.23684	2.64803	2.19408	2.16447
M_4	1880	2.69165	2.66711	2.45638	2.86277	2.40319	2.40691
M_5	4592	2.83897	2.82440	2.63415	2.98301	2.57056	2.58580
M_6	9424	2.89481	2.88402	2.72475	3.01295	2.67339	2.71647
M_7	16192	2.94332	2.93339	2.78662	3.04101	2.75926	2.74864
M_8	24772	3.01065	3.00525	2.87829	3.08744	2.81980	2.86602
M_9	34036	3.02776	3.02422	2.92687	3.08277	2.86879	2.91530
M_{10}	43060	3.04190	3.03982	2.95639	3.08349	2.90748	2.94745
M_{11}	50164	3.06442	3.06355	2.99478	3.09242	2.96179	2.99326
M_{12}	55884	3.05766	3.05768	3.00870	3.07449	2.98713	3.01020
M_{13}	60132	3.06465	3.06498	3.03093	3.07545	3.01766	3.03037
M_{14}	61956	3.06777	3.06760	3.04142	3.07481	3.03431	3.04117
M_{15}	63780	3.06438	3.06431	3.05122	3.06784	3.04704	3.05220
M_{16}	64764	3.06511	3.06506	3.05769	3.06692	3.05573	3.05772
M_{17}	64908	3.06577	3.06558	3.05941	3.06714	3.05839	3.05948
M_{18}	65372	3.06376	3.06372	3.06211	3.06413	3.06159	3.06266
M_{19}	65468	3.06424	3.06418	3.06330	3.06440	3.06328	3.06345
M_{21}	65532	3.06377	3.06377	3.06369	3.06379	3.06369	3.06374
M_{22}	65536	3.06369	3.06369	3.06369	3.06369	3.06369	3.06369
G_0	4	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000
G_1	1238	2.56553	2.60727	2.57108	2.62601	2.34733	2.34653
G_2	19522	3.04156	3.07265	3.04871	3.08478	2.81636	2.81595
G_3	65536	3.06369	3.06369	3.06369	3.06369	3.06369	3.06369
E_2	256	2.63825	2.66523	2.08594	3.97656	2.08594	2.08594
E_3	3328	2.91037	2.91560	2.54387	3.57903	2.55078	2.56881
E_4	17152	2.99356	2.99543	2.81705	3.31238	2.80370	2.82194
E_5	44800	3.06366	3.06266	2.98283	3.13435	2.98500	3.01571
E_6	65536	3.06369	3.06369	3.06369	3.06369	3.06369	3.06369
A_0	4	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000

Bez.	Anz.	EA	NA avg	NA min	NA max	MaxOpt	MinMax
A_1	40	1.81186	1.72000	1.45000	2.05000	1.45000	1.45000
A_2	724	2.53598	2.46046	2.11878	2.71547	2.11878	2.11878
A_3	65536	3.06369	3.06369	3.06369	3.06369	3.06369	3.06369
R_1	4	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000
R_2	88	2.19792	2.15373	1.47727	2.90909	1.47727	1.47727
R_3	844	2.63363	2.60968	2.24171	3.18957	2.19668	2.23460
R_4	4624	2.85272	2.83867	2.55839	3.24654	2.51341	2.54758
R_5	15964	2.97277	2.96530	2.79341	3.21523	2.82448	2.82924
R_6	36376	3.03233	3.02955	2.93226	3.15296	2.92352	2.95923
R_7	56788	3.05695	3.05685	3.01698	3.09389	3.01317	3.03948
R_8	65536	3.06369	3.06369	3.06369	3.06369	3.06369	3.06369

Wir halten zunächst als grundsätzliches Fazit fest, dass es in *allen* Szenarien Unterschiede zwischen den verschiedenen Algorithmen gibt. Man kann also prinzipiell in allgemeinen nur durch Ressourcenbeschränkung definierten Black Box Szenarien durch geschickte Ausnutzung vorhandener Abhängigkeiten von vornherein etwas gegenüber einer „nur durchschnittlichen“ Performanz gewinnen. Es ist selbstverständlich, dass aus dem Miniaturbeispiel $\{0,1\}^3 \rightarrow \{0,1\}^2$ keine allgemeingültigen Schlüsse für Such- und Werteräume mit realistischeren Größen gezogen werden können. Wir wollen trotzdem einige Auffälligkeiten erwähnen und kommentieren.

- Die beiden heuristischen Algorithmen, die zwei verschiedene Greedy-Ansätze realisieren, zeigen für alle betrachteten Beispiele das beste Verhalten. Das stützt die verbreitete Meinung, dass vorhandenes Wissen über die zu optimierende Zielfunktion ausgenutzt und bei der Erstellung eines Algorithmus berücksichtigt werden sollte. Im Vergleich zu solchen angepassten Algorithmen schneiden reine allgemeine Suchverfahren typischerweise schlecht ab.

- Bei den Mengen G_i schneidet der evolutionäre Algorithmus besser ab als der Durchschnitt der nicht-adaptiven Algorithmen. Das deckt sich mit unserer Erwartung. Die Mengen G_i sind gerade Mengen von Funktionen, die „glatt“ sind, bei denen sich also die Funktionswerte zweier direkter Nachbarn im Suchraum um höchstens i unterscheiden. Das kommt dem Suchverhalten evolutionärer Algorithmen entgegen, wie wir später noch sehen werden.

- Bei fast allen anderen Mengen schneidet der Durchschnitt der nicht-adaptiven Algorithmen besser ab als der evolutionäre Algorithmus. Das ist ein überraschendes Resultat. Das durchschnittliche Abschneiden aller nicht-adaptiver Algorithmen entspricht dem Abschneiden eines randomisierten, nicht-adaptiven Algorithmus, der die Aufzählungsreihenfolge rein zufällig wählt. Das ist ohne Zweifel ein praktisch gut zu realisierender Algorithmus. Allerdings sollte die Situation nicht überbewertet werden. Zum einen sind wie bereits klargestellt die Beispiele zu klein, um seriöse relevante Schlussfolgerungen zuzulassen. Zum zweiten findet man in unseren Beispielen M_{12} und M_{13} Gegenbeispiele.

- Eine andere allgemeine Beobachtung, die bei fast allen Mengen und sogar für alle Algorithmen zutrifft, ist, dass mit zunehmender Anzahl von Funktionen die durchschnittlich zur Optimierung benötigte Anzahl von Zielfunktionsauswertungen zunimmt. Funktionen aus kleinen Funktionenklassen scheinen tendenziell leichter optimierbar zu sein. Allerdings stellen die Mengen E_i im Zusammenhang mit dem schlechtesten nicht-adaptiven Algorithmus ein klares Gegenbeispiel dar.

Wir haben gesehen, dass keine vernünftigen Aussagen über die Qualität eines Algorithmus möglich sind, solange man ohne jede Einschränkung über alle Funktionen spricht. Es sind aber in eingeschränkten Szenarien durchaus sinnvolle Aussagen möglich, wobei die in der Praxis ohnehin implizit immer vorgenommenen Einschränkungen durch Ressourcenbeschränkung vermutlich schon ausreichen. Tatsächlich gibt es eine Reihe von Ansätzen, die versuchen zu bestimmen, welche Zielfunktionen effizient durch evolutionäre Algorithmen optimiert werden können und welche nur schlecht zu optimieren sind. In der Regel sind diese Klassifikationen nicht sehr erfolgreich, was vor allem auf unrealistische Zielsetzung zurückzuführen ist. Wie werden im folgenden vier verschiedene bekannte Klassifikationen betrachten und erklären, worin die spezifischen Schwierigkeiten bestehen (Jansen 2000). Ziel ist es, klarer zu machen, wo Grenzen und Möglichkeiten solcher Klassifikationen gefunden werden können. Schließlich sollen die in der Summe eher ernüchternden Ergebnisse im Zusammenhang motivieren, warum der Rest der Arbeit sich auf die Analyse konkreter Beispiele und konkreter, stark vereinfachter evolutionärer Algorithmen konzentriert.

Wir unterscheiden zwei grundlegend verschiedene Arten der Klassifikation, die allerdings in enger Beziehung zueinander stehen. Eine *deskriptive Klassifikation* definiert eine Menge von Zielfunktionen, die gemeinsame Eigenschaften haben und bezüglich Optimierung mit evolutionären Algorithmen (oder auch einem konkreten evolutionärem Algorithmus) vergleichbaren Schwierigkeitsgrad haben. Sie vermittelt in gewisser Weise Strukturinformation über eine Menge von Funktionen, ist aber nicht unmittelbar praktisch. Zu einer gegebenen Zielfunktion wird es im allgemeinen ein nur schwer zu lösendes Problem sein zu entscheiden, ob sie zu einer durch eine Klassifikation bestimmten Menge gehört.

Eine *analytische Klassifikation* ist ein Algorithmus, der zu einer Zielfunktion, auf die er ganz im Sinne eines Black Box Szenariums Zugriff nur durch ein Orakel hat, eine Ausgabe berechnet, die ein Maß für die Schwierigkeit der Funktion bezüglich Optimierung durch evolutionäre Algorithmen ist. Im allgemeinen wird die Ausgabe eine reelle Zahl sein, aber es sind auch komplexere Ausgaben denkbar und kommen tatsächlich vor. Offenbar ist eine analytische Klassifikation in ihrem Wesen unmittelbar praktisch. Darum verwundert es nicht, dass man in der Literatur im Bereich evolutionärer Algorithmen überwiegend analytische Klassifikationen findet. Drei der vier im folgenden betrachteten Klassifikationen sind analytisch. Eine „perfekte“ analytische Klassifikation sollte in etwa das folgende leisten. Zu einer Menge von Optimieralgorithmen $\{A_1, A_2, \dots, A_m\}$ und einer Zielfunktion f soll sie einen Index i so berechnen, dass keiner der m Algorithmen ein globales Maximum von f schneller findet als Algorithmus A_i. Dabei soll gelten, dass im Durchschnitt die Anwendung der Klassifikation und die anschließende Optimierung mit einem optimalen Algorithmus zusammen weniger Zeit in Anspruch nehmen soll als die Anwendung eines festen Algorithmus. Dank dem No Free Lunch Theorem wissen wir natürlich unmittelbar, dass diese Wunschvorstellung in dieser Allgemeinheit nicht realisiert werden kann. Wir werden später einsehen, dass es tatsächlich noch erheblich enger gesteckte Grenzen für reine analytische Klassifikationen gibt.

Als erstes konkretes Beispiel für eine Klassifikation betrachten wir das so genannte NK-Modell, eine bekannte deskriptive Klassifikation, die von Kauffman (1993) eingeführt wurde (Altenberg 1997). Ursprünglich wurde das NK-Modell eingeführt, um eine Art probabilistische Definition von Funktionen $f\colon \{0,1\}^N \to [0;1]$ zu geben, wobei die Funktionen einen einstellbaren Grad an „Rauigkeit“ oder Epistasie haben sollen. Die Idee ist, dass mit zunehmender „Rauigkeit“ (im Gegensatz zu „glatten“ Funktionen) die Optimierung schwieriger wird. Wir bleiben im folgenden bei der Bezeichnung NK-Modell, verwenden aber statt der Parameter N für die Dimension des Suchraums und K für das erlaubte Maß an Interaktion zwischen Bits die in unserer Notation konsistenteren Bezeichnungen n und k.

Definition 2.4. *Eine Funktion* $f\colon \{0,1\}^n \to \mathbb{R}$ *heißt* NK-Funktion *mit* $K = k \in \{0,1,\ldots,n-1\}$, *wenn es* $N = n$ *Funktionen* $g_1\colon \{0,1\}^{k+1} \to \mathbb{R}$, $g_2\colon \{0,1\}^{k+1} \to \mathbb{R}$, … $g_n\colon \{0,1\}^{k+1} \to \mathbb{R}$ *und* n *Mengen* $I_1 = \{i_{1,1},\ i_{1,2},\ \ldots,\ i_{1,k}\}$, … , $I_n = \{i_{n,1},\ i_{n,2},\ \ldots,\ i_{n,k}\}$ *gibt, so dass*

$$f(x) = g_1\left(x_1, x_{i_{1,1}}, \ldots, x_{i_{1,k}}\right) + \cdots + g_n\left(x_n, x_{i_{n,1}}, \ldots, x_{i_{n,k}}\right)$$

für alle $x = x_1x_2\cdots x_n \in \{0,1\}^n$ *gilt. Die Funktion* f *heißt* adjazente NK-Funktion, *wenn zusätzlich* $I_1 = \{2,\ 3,\ \ldots,\ 1+k \bmod n\}$, … , $I_n = \{1,\ 2,\ \ldots,\ k\}$ *gilt.*

Wir bemerken zunächst, dass das NK-Modell eine Partitionierung der Menge aller Zielfunktionen $f\colon \{0,1\}^n \to \mathbb{R}$ in n disjunkte Mengen S_k definiert, wobei S_k alle Funktionen enthält, die NK-Funktionen aber nicht N(K − 1)-Funktionen sind.

Wie schon erwähnt steuert der Parameter k in gewisser Weise das Maß an Interaktion, das zwischen den Bits zugelassen wird. Für $k = 0$ etwa gibt es gar keine Interaktion. Solche Funktionen heißen linear oder bitweise separabel. Sie sind in der Tat auch für evolutionäre Algorithmen einfach zu optimieren, wie wir in Kapitel 4 nachweisen werden. Für $k = n - 1$ sind offenbar alle Funktionen zugelassen, also auch schwierigste. Das alleine reicht aber natürlich nicht aus, um das NK-Modell als sinnvolle Klassifikation zu rechtfertigen. Dazu ist es nötig, dass der Schwierigkeitsgrad der Funktionen mit wachsendem k allmählich zunimmt. Wir fassen kurz zusammen, was über NK-Funktionen aus komplexitätstheoretischer Sicht bekannt ist (Thompson und Wright 1996; Weinberger 1996).

Satz 2.5. *Man kann zu jeder adjazenten NK-Funktion den global maximalen Funktionswert deterministisch in Zeit* $O\left(2^k n\right)$ *bestimmen.*

Beweis. Es lässt sich recht direkt ein Optimierungsalgorithmus angeben, der dem Paradigma der dynamischen Programmierung folgt (siehe etwa Cormen, Leiserson und Rivest (1990), Kapitel 16, für eine Einführung).

Wir definieren $F_i^{b_i\cdots b_{i+k}}$ als größten erreichbaren Wert, der mit $\sum\limits_{1\leq j\leq i} g_j$ erreicht werden kann, wenn die Bits x_i, … , x_{i+k} auf die Werte b_i, … , b_{i+k} festgelegt sind. Offenbar gilt $F_1^{b_1\cdots b_{k+1}} = g_1(b_1, \ldots, b_k)$, es ist also $F_1^{b_1\cdots b_{k+1}}$ ein konstanter Wert, für jede feste Wahl der Bits $b_1, \ldots, b_{k+1}$ ist die Bestimmung in Zeit $O(1)$ möglich. Alle Werte von $F_1^{b_1\cdots b_{k+1}}$ können in Zeit $O\left(2^k\right)$ berechnet und einer Tabelle abgelegt werden. Offensichtlich gilt

$$F_i^{b_i\cdots b_{i+k}} = \max\left\{F_{i-1}^{x_{i-1}b_i\cdots b_{i+k}} + g_i(b_i, \ldots, b_{i+k}) \mid x_i \in \{0,1\}\right\}.$$

Wenn die Werte $F_{i-1}^{b_{i-1}\cdots b_{i+k}}$ zur Verfügung stehen, kann der Wert für $F_i^{b_i\cdots b_{i+k+1}}$ in Zeit $O(1)$ berechnet werden. Insgesamt können alle Werte in Zeit $O\left(2^k n\right)$ berechnet werden. Man sieht direkt, dass das globale Maximum den Wert $\max\left\{F_n^{b_n b_1\cdots b_k} \mid b_n b_1 \cdots b_k \in \{0,1\}^{k+1}\right\}$ hat. $\square$

Man kann für adjazente NK-Funktionen also zumindest die Hoffnung haben, dass ihre Schwierigkeit mit wachsenden Werten von k allmählich zunimmt. Wir sehen jedenfalls, dass die Optimierung (sogar deterministisch) in Zeit $O\left(2^k n\right)$ möglich ist. Diese obere Schranke wächst mit k, für $k = O(\log n)$ ist eine Optimierung deterministisch in Polynomialzeit möglich. Allerdings haben wir ihre Schwierigkeit hier unter klassischen komplexitätstheoretischen Gesichtspunkten untersucht. Dabei haben wir uns mit der Angabe einer oberen Schranke zufrieden gegeben. Die Frage, ob diese Schwierigkeit auch der Schwierigkeit entspricht, die evolutionäre Algorithmen bei der Optimierung von NK-Funktionen haben, ist zunächst noch offen. Bevor wir uns dieser Frage stellen, betrachten wir noch bekannte Resultate über allgemeine NK-Funktionen (Thompson und Wright 1996; Weinberger 1996).

Satz 2.6. *Man kann zu jeder N1-Funktion ein globales Optimum deterministisch in Polynomialzeit bestimmen.*

Beweis. Sei f eine N1-Funktion mit den Variablen $x_1, \ldots, x_n$ und den Funktionen $g_1, \ldots\, g_n$, zu denen die Mengen $I_1, \ldots, I_n$ gehören. Wir nehmen ohne Beschränkung der Allgemeinheit an, dass alle Indexmengen $I_1, \ldots, I_n$ verschieden sind. Andernfalls modifizieren wir die Instanz: Sei etwa $I_i = I_j$ mit $1 \leq i \neq j \leq n$. Dann ersetzen wir g_i durch $g_i + g_j$ und g_j durch die konstante Nullfunktion. Offenbar ändert sich dadurch der maximal zu erreichende Funktionswert nicht und die Funktion bleibt $N1$-Funktion.

Die verwendete algorithmische Idee besteht darin, die Funktion f durch eine äquivalente adjazente N1-Funktion f' zu ersetzen, die dann gemäß Satz 2.5 in Zeit $O(n)$ optimiert werden kann. Wir modellieren das Problem zunächst als ungerichteten Graph. Zu jeder Variablen x_i definieren wir einen Knoten v_i. Zwei Knoten v_i, v_j werden durch eine Kante verbunden, wenn entweder die Variable x_i in der Menge I_j oder die Variable x_j in der Menge I_i vorkommt. Beides gleichzeitig ist gemäß unseren Voraussetzungen nicht möglich. Der Graph zerfällt in Zusammenhangskomponenten, die im Rahmen einer Tiefensuche bestimmt werden können. Die Kanten modellieren Abhängigkeiten zwischen den Variablen. Die Variablen, die Knoten verschiedener Zusammenhangskomponenten entsprechen, können offensichtlich getrennt behandelt werden. Wir betrachten nur noch eine Zusammenhangskomponente. Wir haben also einen zusammenhängenden Graph mit k Knoten und k Kanten. Wenn G ein Kreis ist, entspricht diese Teilinstanz offenbar gerade dem adjazenten Fall und wir sind fertig. Andernfalls gibt es mindestens einen Knoten mit Grad 1. Sei v_i dieser Knoten, sei $I_i = \{x_i, x_j\}$. Dann modifizieren wir f, indem wir g_j durch die Funktion $g_j + \max\{g_i(b, x_j) \mid b \in \{0,1\}\}$ ersetzen und die Variable x_i sowie die Funktion g_i streichen. Offenbar ändert sich der dadurch maximal erreichbare Funktionswert nicht und eine Kante wird aus dem Graphen entfernt. Diese Schritte werden iteriert, bis entweder nur noch Kreise vorliegen oder der Graph keine Kanten mehr enthält. □

Im Vergleich zu N0-Funktionen, die wie erwähnt leicht zu optimieren sind, weisen unter komplexitätstheoretischen Aspekten N1-Funktionen also tatsächlich eine nicht zu drastische Steigerung in der Schwierigkeit auf. Allerdings setzt sich dieser mäßige Anstieg im Schwierigkeitsgrad nicht fort.

Satz 2.7. *Die Optimierung von N2-Funktionen ist NP-hart.*

Beweis. Der Beweis ist einfach, man erkennt direkt, dass sich MAX-2-SAT ganz leicht als Optimierung von N2-Funktionen formulieren lässt. Beim Problem MAX-2-SAT ist zu einer Menge von Klauseln, die jeweils höchstens zwei Literale enthalten, und einer Zahl r zu entscheiden, ob es eine Belegung der Variablen gibt, so dass mindestens r Klauseln gleichzeitig erfüllt sind. MAX-2-SAT ist NP-vollständig (Garey und Johnson 1979). Wir führen eine lokale Ersetzung durch. Wir definieren $l := \max\{n, m\}$ Variablen $y_1, \ldots, y_l$, wobei n die Anzahl der Variablen und m die Anzahl der Klauseln der MAX-2-SAT Instanz ist. Jetzt müssen wir eine $l2$-Funktion definieren, also l Funktionen $g_1, \ldots, g_l$, wobei g_i nur von y_i und zwei weiteren Variablen abhängt. Für die Funktion g_i sind das die zwei Variablen in der Menge I_i, wobei wir genau die Variablen wählen, die in der Klausel c_i vorkommen. Falls c_i nur ein Literal enthält, wählen wir eine beliebige zweite Variable für I_i. Enthält c_i das Literal x_j, so ist $1 - y_j$ ein Term der Funktion. Enthält c_i das Literal $\overline{x_k}$, so ist y_k ein Term der Funktion. Enthält c_i zwei Literale und sind t_1 und t_2 die beiden Terme, dann ist $g_i = 1 - t_1 \cdot t_2$, andernfalls ist $g_i = 1 - t$, wenn t der einzige Term ist. Offensichtlich haben wir auf diese Weise eine $l2$-Funktion definiert, wobei zu jeder Klausel c_i der Eingabe genau eine Funktion g_i gehört. Die $l2$-Funktion hängt essenziell nur von den Variablen $y_1, \ldots, y_n$ ab. Ist $m > n$, so kommen die Variablen $y_{n+1}, \ldots, y_m$ gar nicht vor, wir definieren $g_{n+1} := g_{n+2} := \cdots := g_m := 0$. Sei eine beliebige Belegung der Variablen $x_1, \ldots, x_n$ gegeben. Wir übernehmen diese Belegung für die Variablen $y_1, \ldots, y_n$. Genau dann, wenn die Klausel c_i durch diese Belegung erfüllt wird, hat g_i unter dieser Belegung den Funktionswert 1. Andernfalls hat g_i den

Funktionswert 0. Es folgt, dass für jede Belegung die Anzahl erfüllter Klauseln gleich dem Funktionswert der $l2$-Funktion ist. Also ist die MAX-2-SAT Instanz genau dann erfüllbar, wenn das globale Maximum der $l2$-Funktion mindestens r beträgt. □

Offenbar verhalten sich (allgemeine) NK-Funktionen bezüglich steigender Schwierigkeit mit wachsendem k nicht vernünftig, sie definieren also keine brauchbare Klassifikation. Es ist offensichtlich, dass sich im allgemeinen jede Funktion $f\colon \{0,1\}^n \to \mathbb{R}$ mit Grad k in der Polynomdarstellung als NK-Funktion mit $K = k$ und passend gewähltem N darstellen lässt. Allgemeine NK-Funktionen tragen also im wesentlichen nicht mehr zur Klassifikation bei als der Polynomgrad. Die Frage, ob adjazente NK-Funktionen, die aus komplexitätstheoretischer Sicht vielversprechende Kandidaten für Definition einer Klassifikation von Zielfunktionen sind, einen sinnvollen Bezug zu evolutionären Algorithmen aufweisen, liegt zur Zeit außerhalb dessen, was analytisch erfasst und theoretisch fundiert beantwortet werden kann. In solchen Situation ist es vernünftig, empirische Resultate als Hinweise zu verwenden. Kauffman (1993) beschreibt eine Reihe von Versuchen, die darauf hinweisen, dass zwischen allgemeinen und adjazenten NK-Funktionen für evolutionäre Algorithmen kein nennenswerter Unterschied besteht. Das deutet darauf hin, dass auch adjazente NK-Funktionen keine geeignete Klassifikation liefern. Einen weiteren Hinweis in dieser Richtung liefert Weinberger (1996), der Korrelationen zwischen Bits betrachtet und wenig Hinweise auf Unterschiede zwischen allgemeinen und adjazenten NK-Funktionen findet. Außerdem weisen Naudts und Kallel (1999) empirisch nach, dass schon adjazente N1-Funktionen für einen einfachen genetischen Algorithmus extrem schwierig sein können. Wir vermuten also, dass das NK-Modell die Aufgabe, eine sinnvolle Klassifikation aller Zielfunktionen für evolutionäre Algorithmen zu liefern, nicht erfüllen kann.

Eine gänzlich andere Art von Klassifikation sind analytische Klassifikationen. Das vermutlich am besten bekannte Beispiel dafür ist die so genannte Epistasie, wie sie von Davidor (1991) eingeführt wurde.

Definition 2.8. *Die* Epistasie *einer Funktion f wird mit ε_f bezeichnet und ist definiert durch* (*Naudts, Suys und Verschoren 1997*)

$$\varepsilon_f = \sqrt{\frac{\sum\limits_{x\in\{0,1\}^n}\left(f(x) - 2^{1-n}\left(\sum\limits_{1\leq i\leq n}\sum\limits_{\substack{y\in\{0,1\}^n\\ y_i=x_i}} f(y)\right) + \frac{n-1}{2^n}\sum\limits_{y\in\{0,1\}^n} f(y)\right)^2}{2^n}}.$$

Der Begriff der Epistasie stammt ursprünglich aus der Genetik. Dort nimmt man in erster Näherung an, dass die Gesamtwirkung einer Anzahl verschiedener Gene durch eine Summenbildung ihrer Einzelwirkungen bestimmt werden kann. Man kennt aber auch Phänomene, die von diesem einfachen Schema abweichen und die durch Interaktion verschiedener Gene entstehen. So kann etwa ein Gen das Sichtbarwerden der Wirkung eines anderen überdecken, oder ein Gen kann durch solche Interaktionen überhaupt nicht zur Wirkung gelangen. In diesen Fällen spricht man von Epistasie.

Die Vorstellung im Kontext der Klassifikation ist im Grunde, dass man misst, wie weit eine Zielfunktion von einer linearen Funktion, bei der die „Wirkung“ einzelner Bits als Summe der einzelnen „Wirkungen“ errechnet werden kann, abweicht. Je größer diese Abweichung ist, desto schwieriger wird die Funktion eingeschätzt, desto größer wird der Epistasie-Wert. Die exakte Berechnung benötigt exponentielle Zeit in n, in der Praxis wird man darum auf Schätzungen zurückgreifen, die auf einer polynomiellen Anzahl von Funktionswerten beruhen. Sich daraus ergebende Schwierigkeiten wollen wir hier nicht diskutieren.

Die Epistasie ist zunächst einmal nach oben unbeschränkt. Man kann aber eine normalisierte Epistasie definieren, so dass man Funktionen mit maximaler Epistasie bestimmen kann. Im Sinne einer Klassifikation sollte eine solche Funktion dann für evolutionäre Algorithmen bezüglich Optimierung maximale Schwierigkeit haben. Naudts, Suys und Verschoren (1997) geben als Beispiel für eine solche Funktion eine Funktion $f_{\mathrm{C},a}$ an, die sie Camel nennen.

Definition 2.9. *Die Funktion $f_{C,a}\colon \{0,1\}^n \to \mathbb{R}$ ist definiert durch*

$$f_{C,a}(x) := \left(\prod_{\substack{1\leq i\leq n\\ a_i=1}} x_i \cdot \prod_{\substack{1\leq i\leq n\\ a_i=0}} (1-x_i)\right) + \left(\prod_{\substack{1\leq i\leq n\\ a_i=1}} (1-x_i) \cdot \prod_{\substack{1\leq i\leq n\\ a_i=0}} x_i\right)$$

für jedes $a \in \{0,1\}^n$.

Offensichtlich ist $f_{\mathrm{C},a}(a) = f_{\mathrm{C},a}(\overline{a}) = 1$ und $f_{\mathrm{C},a}(x) = 0$ für alle anderen Werte von x. Dabei bezeichnet $\overline{a}$ das bitweise Komplement von a. Es ist intuitiv klar, dass ein Algorithmus ohne Kenntnis von a die Funktion $f_{\mathrm{C},a}$ nicht effizient optimieren kann. Gleichgültig, wie er die Punkte im Suchraum, für die er Orakelanfragen stellt, aussucht, nach t getesteten Punkten kann er eines der zwei globalen Maxima mit Wahrscheinlichkeit höchstens $t/2^{n-1}$ gefunden haben. Die erwartete Laufzeit für alle Algorithmen beträgt also $\Omega\left(2^n\right)$. Insofern ist $f_{\mathrm{C},a}$ tatsächlich ein guter Kandidat für eine schwierigste Funktion.

Wir berechnen konkret die Epistasie von $f_{\mathrm{C},\mathbf{1}}$, wobei $\mathbf{1}$ den Einsstring bezeichnet. Es gilt

$$\sum_{y \in \{0,1\}^n} f_{\mathrm{C},\mathbf{1}}(y) = 2$$

sowie

$$\sum_{\substack{y \in \{0,1\}^n \\ y_i = x_i}} f_{\mathrm{C},\mathbf{1}}(y) = 1$$

für alle $x \in \{0,1\}^n$ und alle $i \in \{1,2,\dots,n\}$. Also haben wir

$$\begin{aligned} \varepsilon_{f_{\mathrm{C},\mathbf{1}}} &= \left(2^{-n}\left(2 \cdot \left(1 - \frac{n}{2^{n-1}} + \frac{n-1}{2^{n-1}}\right)^2\right.\right. \\ &\qquad \left.\left. + \left(2^n - 2\right) \cdot \left(0 - \frac{n}{2^{n-1}} + \frac{n-1}{2^{n-1}}\right)^2\right)\right)^{1/2} \\ &= \sqrt{2^{-n}\left(2 \cdot \left(1 - \frac{1}{2^{n-1}}\right)^2 + \left(2^n - 2\right) \cdot \frac{1}{2^{2n-2}}\right)} \end{aligned}$$

für die Epistasie von $f_{\mathrm{C},\mathbf{1}}$. Uns interessiert hier gar nicht der konkrete Wert, den wir erhalten haben. Wir haben $\varepsilon_{f_{\mathrm{C},\mathbf{1}}}$ berechnet, um diesen Wert mit der Epistasie einer anderen Funktion zu vergleichen.

Definition 2.10. *Die Funktion* $\overline{f_{C,a}}\colon \{0,1\}^n \to \mathbb{R}$ *ist definiert durch*

$$\overline{f_{C,a}}(x) := 1 - f_{C,a}(x)$$

für alle $a \in \{0,1\}^n$.

Die Funktion $\overline{f_{\mathrm{C},a}}$ hat also genau überall da ein globales Maximum, wo $f_{\mathrm{C},a}$ keines hat. Folglich wird jeder Algorithmus spätestens beim dritten noch ungefragten Punkt im Suchraum, für den er den Funktionswert erfragt, ein globales Maximum finden. Wir sehen, dass $\overline{f_{\mathrm{C},a}}$ extrem einfach zu optimieren ist und erwarten von einer analytischen Klassifikation, dass sie für $\overline{f_{\mathrm{C},a}}$ eine wesentlich geringere Schwierigkeit vorhersagt als für $f_{\mathrm{C},a}$. Wir berechnen jetzt also $\varepsilon_{\overline{f_{\mathrm{C},\mathbf{1}}}}$ und sehen, dass

$$\sum_{y \in \{0,1\}^n} \overline{f_{\mathrm{C},\mathbf{1}}}(y) = 2^n - 2$$

sowie

$$\sum_{\substack{y \in \{0,1\}^n \\ y_i = x_i}} \overline{f_{\mathrm{C},\mathbf{1}}}(y) = 2^{n-1} - 1$$

für alle $x \in \{0,1\}^n$ und alle $i \in \{1, 2, \dots, n\}$ gilt. Also haben wir

$$\begin{aligned}
\varepsilon_{\overline{f_{\mathrm{C},\mathbf{1}}}} &= \left(2^{-n}\left(2 \cdot \left(0 - \frac{n\,(2^{n-1}-1)}{2^{n-1}} + \frac{(n-1)\,(2^{n-1}-1)}{2^{n-1}}\right)^2\right.\right. \\
&\qquad \left.\left. + (2^n - 2) \cdot \left(1 - \frac{n\,(2^{n-1}-1)}{2^{n-1}} + \frac{(n-1)\,(2^{n-1}-1)}{2^{n-1}}\right)^2\right)\right)^{1/2} \\
&= \sqrt{2^{-n}\left(2 \cdot \left(\frac{1}{2^{n-1}} - 1\right)^2 + (2^n - 2) \cdot \frac{1}{2^{2n-2}}\right)}
\end{aligned}$$

für die Epistasie von $\overline{f_{\mathrm{C},\mathbf{1}}}$. Offensichtlich gilt $\varepsilon_{f_{\mathrm{C},\mathbf{1}}} = \varepsilon_{\overline{f_{\mathrm{C},\mathbf{1}}}}$. Es gibt also zwei Funktionen, die exakt gleiche Epistasie haben, obwohl ihr Schwierigkeitsgrad extrem verschieden ist. Das disqualifiziert Epistasie als Kandidaten für eine brauchbare Klassifikation.

Eine andere analytische Klassifikation, die von Jones und Forrest (1995) vorgestellt worden ist, heißt Fitness Distance Correlation (FDC) und benutzt die Korrelation zwischen dem Funktionswert eines Punktes im Suchraum und seinem Abstand zu einem nächsten globalen Maximum. Von zentraler Bedeutung ist dabei die formale Definition des Abstands zweier Punkte im Suchraum. Es ist klar (Jones 1995), dass die Klassifikation nur dann sinnvoll sein kann, wenn das verwendete Abstandsmaß etwas mit der Art und Weise zu tun hat, wie der evolutionäre Algorithmus den Suchraum behandelt. Es muss Abstände so behandeln, dass der evolutionäre Algorithmus kleine

Schritte im Suchraum mit größerer Wahrscheinlichkeit unternimmt als größere. Für mutationsbasierte evolutionäre Algorithmen genügt in aller Regel der Hammingabstand dieser Forderung. Tatsächlich benutzen die von Jones und Forrest (1995) präsentierten Beispiele den Hammingabstand als Abstandsmaß.

Definition 2.11. *Sei $f\colon \{0,1\}^n \to \mathbb{R}$ eine Funktion. Es gebe $d\colon \{0,1\}^n \to \mathbb{R}^+$ den Abstand von einem Punkt zum nächsten globalen Maximum an. Sei $\overline{f}$ bzw. $\overline{d}$ der Durchschnitt über alle Funktionswerte bzw. Abstände, seien s_f und s_d die Standardabweichungen. Bezeichne C_{fd} die Kovarianz mit*

$$C_{fd} = 2^{-n} \sum_{x \in \{0,1\}^n} \left(f(x) - \overline{f}\right)\left(d(x) - \overline{d}\right).$$

Dann ist die Fitness Distance Correlation *von f gegeben durch*

$$FDC_f := \frac{C_{fd}}{s_f \cdot s_d}.$$

Offensichtlich ist der Rechenaufwand zur exakten Bestimmung von FDC_f exponentiell in n. Wir gehen auch hier nicht auf Schwierigkeiten ein, die dadurch entstehen, dass man sich in der Praxis auf Schätzungen beschränkt, die auf einer nur polynomiell großen Anzahl von Funktionswerten beruhen. Erschwerend kommt hinzu, dass man die Anzahl und Positionen der globalen Maxima kennen muss. Es gibt allerdings Ansätze, um mittels zufälliger Auswahl von einigen Punkten und anschließender lokaler Suche sogar alle lokalen Maxima zu bestimmen, allerdings nur für bestimmte Funktionenklassen (Garnier und Kallel 2000). Und selbst für die bleibt die Frage nach dem Rechenzeitaufwand im wesentlichen unbeantwortet.

Quick, Rayward-Smith und Smith (1998) präsentieren eine Funktion f_{R}, die sie Ridge nennen, als Gegenbeispiel. Die Funktion f_{R} ist leicht zu optimieren, zeigt aber praktisch keine positive Korrelation zwischen guten Funktionswerten und geringer Distanz zum eindeutigen globalen Maximum.

Definition 2.12. *Die Funktion $f_R\colon \{0,1\}^n \to \mathbb{R}$ ist definiert durch*

$$f_R(x) := \begin{cases} n + 1 + \|x\|_1 & \text{falls } \exists i \in \{0,1,\dots,n\} : x = 1^i 0^{n-i}, \\ n - \|x\|_1 & \text{sonst,} \end{cases}$$

wobei $\|x\|_1$ die Anzahl von Einsen in x bezeichnet. Mit $1^i 0^{n-i}$ bezeichnen wir den Bitstring der Länge n, der aus i führenden Einsen gefolgt von $n-i$ Nullen besteht.

Die Funktion f_{R} ist klar strukturiert und in ihren Eigenschaften gut zu verstehen. Das eindeutige globale Maximum ist $\mathbf{1}$, der Einsstring, mit $f_{\mathrm{R}}(\mathbf{1}) = 2n+1$. Für alle 2^n Strings außer den $n+1$ Strings der Form 1^i0^{n-i} sind größere Funktionswerte korreliert mit einer kleineren Anzahl von Einsen, also größerem Abstand zum einzigen globalen Maximum. Weil zur Bestimmung von FDC_f über alle 2^n Strings gemittelt wird, spielen die $n+1$ Strings mit klarer positiver Korrelation von hohem Funktionswert und geringem Abstand keine Rolle. Wir erhalten also einen extrem kleinen Wert für $\mathrm{FDC}_{f_{\mathrm{R}}}$, was eine extrem schwierige Funktion signalisiert. Tatsächlich ist f_{R} aber leicht zu optimieren, wie man sich leicht klarmacht. Bei Wahl eines zufälligen Startpunktes findet man mit überwältigender Wahrscheinlichkeit keinen der $n+1$ ausgezeichneten Strings 1^i0^{n-i}. Darum werden evolutionäre Algorithmen rasch den „Hinweisen" folgen und den Nullstring $\mathbf{0}$ finden. Auch eine einfache lokale Suche, die als Nachbarschaft alle Punkte mit Hammingabstand 1 zum aktuellen Suchpunkt verwendet, verfährt ebenso. Dann gibt es aber immer genau einen direkten Nachbarpunkt, der besseren Funktionswert hat und den Hammingabstand zum globalen Maximum verringert. Dieser kann mit $O(n)$ Orakelanfragen gefunden werden. Nach n solcher Schritte ist das globale Maximum gefunden, so dass f_{R} in $O(n^2)$ Schritten mittels einer einfachen lokalen Suche optimiert wird. Für den (1+1) EA, den wir im nächsten Kapitel vorstellen, gilt die gleiche obere Schranke für die erwartete Rechenzeit. Somit ist die Hypothese, dass Fitness Distance Correlation eine sinnvolle Klassifikation definiert, falsifiziert.

Als letztes Beispiel für den Versuch, eine sinnvolle analytische Klassifikation anzugeben, betrachten wir bitweise Epistasie, die von Fonlupt, Robilliard und Preux (1998) eingeführt worden ist. Wie schon aus der Bezeichnung zu schließen ist, wird auch hier die Epistasie, also das Maß an Interaktion zwischen den Bits, als Gradmesser für die Schwierigkeit der Funktion verwendet. Anders als bei der Epistasie (Definition 2.8) wird hier aber jedem Bit ein Wert zugewiesen.

Definition 2.13. *Sei $f\colon \{0,1\}^n \to \mathbb{R}$ eine Funktion. Für jedes $i \in \{1, 2, \dots, n\}$ bezeichnet $\sigma^2_{f,i}$ die* bitweise Epistasie *für Bit i. Sei $\Sigma_i \subseteq \{0,1,*\}$ die Menge aller Strings der Länge n, die genau an Position i den Wert $*$ und sonst nur Werte 0 oder 1 haben. Offensichtlich enthält Σ_i genau 2^{n-1} solcher Strings. Für $x \in \Sigma_i$ bezeichnen wir mit x_0 bzw. x_1 den String aus $\{0,1\}^n$, der mit x an allen Stellen außer der i-ten Stelle übereinstimmt und dort den*

Wert 0 *bzw.* 1 *trägt. Es ist*

$$\sigma^2_{f,i} := 2^{-(n-1)} \sum_{x \in \Sigma_i} \left(2^{-(n-1)} \left(\sum_{y \in \Sigma_i} (f(y_0) - f(y_1)) \right) - (f(x_0) - f(x_1)) \right)^2 .$$

Jansen (2000) zeigt anhand verschiedener Beispiele, dass auch bitweise Epistasie keine hilfreiche Klassifikation definiert. Die Situation ist bei bitweiser Epistasie anders als bei den anderen bisher besprochenen Klassifikationsversuchen. Fonlupt, Robilliard und Preux (1998) sehen explizit vor, dass die bitweise Epistasie praktisch eingesetzt wird, um evolutionären Algorithmen bei der Suche nach Maxima zu unterstützen. Weil (wie üblich) die exakte Berechnung exponentiell lange braucht, zieht man sich in der Praxis auf Schätzungen zurück, die auf einer zufällig gewählten Menge von polynomiell vielen Suchpunkten beruhen. Weil bitweise Epistasie ein relativ neuer und in der Literatur noch nicht diskutierter Klassifikationsansatz ist (abgesehen von Jansen (2000)), gehen wir hier auf drei Aspekte ein: den Nutzen der exakten Kenntnis der bitweisen Epistasie bei der Suche nach einem globalen Maximum, die Frage, ob bitweise Epistasie eine fehlerfreie Klassifikation liefert, sowie die Abweichung der Schätzung vom exakten Wert.

Definition 2.14. *Die Funktion* $f_{N,a}\colon \{0,1\}^n \to \mathbb{R}$ *ist definiert durch*

$$f_{N,a}(x) := \prod_{\substack{1 \le i \le n \\ a_i = 1}} x_i \cdot \prod_{\substack{1 \le i \le n \\ a_i = 0}} (1 - x_i)$$

für alle $a \in \{0,1\}^n$.

Die Funktion $f_{\mathrm{N},a}$ ist (meist in der Gestalt $f_{\mathrm{N},\mathbf{1}}$) eine der bekanntesten Beispielfunktionen. Sie wird in der Regel als „Nadel im Heuhaufen“ bezeichnet, was ihren Charakter tatsächlich gut widergibt. Der Funktionswert ist überall 0, nur für a wird der globale Maximalwert 1 angenommen. Offensichtlich ist $f_{\mathrm{N},a}$ noch etwas schlechter zu optimieren als $f_{\mathrm{C},a}$. Wir werden jetzt die bitweise Epistasie von $f_{\mathrm{N},a}$ exakt bestimmen, um zu sehen, in wie weit die exakte Kenntnis der bitweisen Epistasie bei der Suche nach einem globalen Maximum hilfreich sein kann.

Satz 2.15. *Die bitweise Epistasie des i-ten Bits für* $f_{N,a}\colon \{0,1\}^n \to \mathbb{R}$ *beträgt*

$$\sigma^2_{f_{N,a},i} = \frac{1}{2^{n-1}} - \frac{1}{2^{2n-2}}$$

für alle Werte von $i \in \{1,\dots,n\}$ *und* $a \in \{0,1\}^n$.

Beweis. Gemäß Definition 2.13 ist

$$\sigma^2_{f_{\mathrm{N},a},i} = \frac{1}{2^{n-1}} \sum_{x\in\Sigma_i} \left(\frac{\sum_{y\in\Sigma_i} (f_{\mathrm{N},a}(y_0) - f_{\mathrm{N},a}(y_1))}{2^{n-1}} - (f_{\mathrm{N},a}(x_0) - f_{\mathrm{N},a}(x_1)) \right)^2$$

für alle $i \in \{1, \ldots, n\}$. Offensichtlich ist

$$\sum_{y\in\Sigma_i} (f_{\mathrm{N},a}(y_0) - f_{\mathrm{N},a}(y_1)) \in \{-1, 1\}$$

für alle Werte von a und i. Also ist

$$\begin{aligned} \sigma^2_{f_{\mathrm{N},a},i} &= \frac{2^{n-1}-1}{2^{n-1}} \cdot \frac{1}{2^{2n-2}} + \frac{1}{2^{n-1}} \cdot \left(1 - \frac{2}{2^{n-1}} + \frac{1}{2^{2n-2}}\right) \\ &= \frac{1}{2^{n-1}} - \frac{1}{2^{2n-2}} \end{aligned}$$

□

Wir sehen, dass $\sigma^2_{f_{\mathrm{N},a},i}$ für alle Werte von a und i den gleichen Wert annimmt. Folglich nützt bei $f_{\mathrm{N},a}$ die exakte Kenntnis der bitweisen Epistasie bei der Suche nach dem globalen Optimum a gar nichts.

Als zweiten Punkt betrachten wir mögliche Abweichungen von $\sigma^2_{f,i}$, wenn man Schätzungen auf Basis einer polynomiell großen Anzahl von zufällig gewählten Suchpunkten anstellt. Dazu betrachten wir die Funktionen g_k mit

$$g_k(x) := k \cdot f_{\mathrm{N},\mathbf{1}}(x),$$

die für alle $k \in \mathbb{N}$ definiert sind.

Satz 2.16. *Die bitweise Epistasie des i-ten Bits für $g_k\colon \{0,1\}^n \to \mathbb{R}$ beträgt*

$$\sigma^2_{g_k,i} = k^2 \cdot \left(\frac{1}{2^{n-1}} - \frac{1}{2^{2n-2}} \right).$$

Beweis. Analog zum Beweis von Satz 2.15 haben wir

$$\begin{aligned} \sigma^2_{g_k,i} &= \frac{2^{n-1}-1}{2^{n-1}} \cdot \frac{k^2}{2^{2n-2}} + \frac{k^2}{2^{n-1}} \cdot \left(\frac{1}{2^{n-1}} - 1\right)^2 \\ &= k^2 \cdot \left(\frac{1}{2^{n-1}} - \frac{1}{2^{2n-2}} \right) \end{aligned}$$

□

Wir sehen, dass die bitweise Epistasie für jedes Bit proportional zu k^2 wächst. Für evolutionäre Algorithmen, die nur die Ordnung auf dem Suchraum beachten, nicht aber die absoluten Funktionswerte, spielt aber k überhaupt keine Rolle. Und für alle Algorithmen ist der Wert von k bei der Frage nach dem ersten Auffinden des globalen Maximums $\mathbf{1}$ irrelevant. Verschieden große Werte von $\sigma^2_{f,i}$ sollten verschieden schwierige Funktionen signalisieren. Für die Funktionen g_k nimmt aber trotz gleicher Schwierigkeit $\sigma^2_{g_k,i}$ beliebig unterschiedliche Werte an.

Schätzt man $\sigma^2_{g_k,i}$ basierend auf $p(n)$ zufällig gewählten Punkten im Suchraum, so erhält man eine von 0 abweichende Schätzung nur dann, wenn der Punkt $\mathbf{1}$ gewählt wird. Wie gezeigt ist der tatsächliche Wert von $\sigma^2_{g_k,i}$ proportional zu k^2 und somit beliebig groß. Die Wahrscheinlichkeit für eine in diesem Sinne gute Schätzung ist durch $p(n)/2^n$ nach oben beschränkt, also durch $2^{-\Omega(n)}$ für jedes Polynom p. Man erhält folglich mit überwältigender Wahrscheinlichkeit beliebig falsche Schätzungen.

Wir haben jetzt insgesamt vier Beispiele gesehen, die alle keine zufriedenstellende Klassifikation der Funktionen $f\colon \{0,1\}^n \to \mathbb{R}$ liefern. Die drei analytischen Klassifikationsversuche waren dabei von den jeweiligen Autoren ausdrücklich als Klassifikation vorgesehen. Es ist ganz klar, dass es nicht sinnvoll sein kann, weiter kreativ neue Klassifikationen vorzuschlagen und dann darauf zu warten, dass jemand dazu passende Gegenbeispiele konstruiert, um ihre Unzulänglichkeiten nachzuweisen. Wir wollen darum einige grundsätzliche Überlegungen anstellen, um die Grenzen und Möglichkeiten von solchen Klassifikationen zu ermitteln. Dazu beschäftigen wir uns zunächst mit analytischen Klassifikationen.

Wir haben bei allen drei untersuchten analytischen Klassifikationen bemerkt, dass der Rechenzeitaufwand zur exakten Berechnung exponentiell in der Dimension des Suchraums ist. Es ist leicht einzusehen, dass das auch so sein muss, wenn alle Funktionen korrekt klassifiziert werden sollen. Um die Funktion $f_{\mathrm{N},a}$, die allen Punkten im Suchraum außer dem globalen Maximum a den Funktionswert 0 zuordnet, von der trivial zu optimierenden konstanten Nullfunktion zu unterscheiden, muss die Funktion genau am Punkt a ausgewertet werden. Wenn a nicht bekannt ist und nur polynomiell viele Punkte betrachtet werden, wird mit Wahrscheinlichkeit $1 - 2^{-\Omega(n)}$ der Punkt a nicht gefunden. Wir sehen also,

1. dass eine allgemeine fehlerlose Klassifikation für exponentiell viele Punkte im Suchraum den Funktionswert anfragen muss und

2. dass bei einer auf nur polynomiell vielen Punkten beruhenden Schätzung mit sehr großer Wahrscheinlichkeit eine sehr leichte mit einer sehr schwierigen Funktion verwechselt wird.

Bevor wir Auswege diskutieren, wollen wir uns erst deskriptiven Klassifikationen zuwenden.

Eine sinnvolle deskriptive Klassifikation beschreibt eine Klasse von Funktionen, die bezüglich Optimierung durch evolutionäre Algorithmen (oder auch einen bestimmten evolutionären Algorithmus) ähnlichen Schwierigkeitsgrad haben. Das vermittelt ohne Zweifel wichtiges Strukturwissen über die so erfassten Funktionen. Wenn man annehmen darf, dass die so beschriebenen Funktionen in der Praxis tatsächlich vorkommen, hat man sogar praktisch relevantes Wissen, das zumindest bei der Entscheidung, welches Optimierverfahren eingesetzt werden soll, hilfreich ist. Wir werden in Kapitel 4 eine in diesem Sinne gelungene deskriptive Klassifikation kennenlernen. In Kapitel 5 finden wir ein weniger gelungenes Beispiel: es handelt sich dort um semantisch verwandte Funktionen, die allerdings von extrem unterschiedlicher Schwierigkeit sein können. Im Idealfall wird durch eine deskriptive Klassifikation nicht nur eine Teilmenge von Funktionen erfasst. Es wird günstigenfalls eine sinnvolle Partitionierung der Menge aller Funktionen definiert, die eine Hierarchie der Schwierigkeit der Funktionen bezüglich Optimierung darstellt. Wir haben das NK-Modell als Versuch einer solchen Klassifikation betrachtet, wobei allerdings wie beim Polynomgrad, über den wir in Kapitel 6 ausführlich sprechen, bezweifelt werden darf, dass in diesem Fall tatsächlich eine sinnvolle Partitionierung stattfindet.

Eine deskriptive Klassifikation alleine gibt aber nicht nur in dem beschriebenen Sinne Antwort, sie wirft auch direkt eine neue und in der Regel nur schwer zu beantwortende Frage auf. Zu einer gegebenen Funktion möchte man natürlich entscheiden, ob sie denn zu einer durch die deskriptive Klassifikation beschriebenen Klasse gehört und welche Klasse das konkret ist. Ein solcher Entscheidungsalgorithmus ist natürlich eine analytische Klassifikation. Wir sehen also von diesem Ende aus, dass eine Verbindung von deskriptiver und analytischer Klassifikation hilfreich sein kann.

Wir kommen nun zurück zu analytischen Klassifikationen. Wir hatten bereits festgestellt, dass eine allgemeine analytische Klassifikation entweder fehlerhaft sein muss oder zumindest für einige Funktionen exponentiell viele Punkte im Suchraum untersuchen muss. Diese Aussage gilt aber nicht für eingeschränkte analytische Klassifikationen, also solche analytischen Klassifikationen, die nur für eine bestimmte Teilmenge aller Funktionen definiert sind. Die Beschreibung einer solchen Teilmenge ist offensichtlich eine deskriptive Klassifikation. Wir sehen also auch von dieser Seite her einen Bedarf für die Kombination von analytischer und deskriptiver Klassifikation. Es gibt durchaus Ansätze, die man als Versuche in dieser Richtung interpretieren darf. Garnier und Kallel (2000) untersuchen die Anzahl zufällig gewählter Startpunkte für eine lokale Suche, die man braucht, um mit großer Wahrscheinlichkeit alle lokalen Optima einer Funktion zu finden. Das kann als ein Maß für die Schwierigkeit der Funktion betrachtet werden, da die Identifikation aller lokalen Optima offenbar auch alle globalen Maxima enthüllt. Es werden für einige Funktionenklassen scharfe Schranken für die benötigte Anzahl angegeben, zusätzlich wird anhand statistischer Tests das Problem der Entscheidung, ob eine gegebene Funktion zu einer derart spezifizierten Funktionenklasse gehört, angegangen. Der Ansatz lässt zunächst das Problem einer akzeptablen Laufzeit offen, insbesondere wird die Frage nach der Laufzeit der lokalen Suche, die von jedem Startpunkt aus tatsächlich durchgeführt wird, nicht beantwortet. Weil diese Frage prinzipiell schwer zu beantworten ist (Papadimitriou, Schäffer und Yannakakis 1990), ist es fraglich, ob der Ansatz in dieser Form praxistauglich gemacht werden kann.

Es ist klar, dass nicht jeder Wunsch, den man bezüglich einer Klassifikation haben kann, erfüllbar ist. Die Unterscheidung so ähnlicher Funktionen wie $f_{\mathrm{N},a}$ und der konstanten Nullfunktion ist allein aufgrund zufälliger Untersuchung einer polynomiell großen Anzahl von Punkten im Suchraum nicht möglich. Auch das No Free Lunch Theorem setzt Grenzen, vor allem was uneingeschränkte Klassifikationen angeht. Es ist keineswegs so, dass die Aufgabe, eine sinnvolle, fehlerfreie und praxisrelevante Klassifikation zu finden, völlig unmöglich ist. Es ist jedoch beim gegenwärtigen Stand der Forschung auf dem Gebiet evolutionärer Algorithmen noch zu früh, um das schon zu leisten. Darum ist unser Ansatz in dieser Arbeit sehr viel bescheidener.

Wir beschränken uns zum einen auf die Untersuchung sehr einfacher evolutionärer Algorithmen, die allerdings wesentliche Aspekte evolutionärer Algorithmen, wie sie in der Praxis tatsächlich eingesetzt werden, aufweisen. Wir erhoffen uns von diesem Vorgehen, dass diese wesentlichen Aspekte in dieser einfacheren und klareren Form begreiflich werden. Zum zweiten argumentieren wir vor allem anhand konkreter Beispielfunktionen. Dabei geht es natürlich nicht um die Funktionen selbst, die häufig konstruiert, rein künstlich und nicht praxisrelevant sind. Wir nutzen aber die starke erklärende Kraft gut strukturierter und dadurch verständlicher Beispiele, die paradigmatisch wichtige Effekte, die wir für verallgemeinerbar halten, veranschaulichen und erklären helfen.

Allen Beispielfunktionen ist gemeinsam, dass sie

1. kurze Definitionen haben,
2. effizient ausgewertet werden können, in der Regel in Zeit $\Theta(n)$,
3. eine einfache, verständliche Struktur haben und
4. Eigenschaften aufweisen, die bestimmte Aspekte des Suchverhaltens des untersuchten evolutionären Algorithmus besonders deutlich herausstellen und so gewissermaßen exemplifizieren.

Als sehr gewünschte Nebeneffekte ergeben sich dabei manchmal deskriptive Klassifikationen: wir identifizieren Klassen von Funktionen, die (tatsächlich oder auch nur vermeintlich) ähnliche Schwierigkeit bezüglich Optimierung durch den untersuchten evolutionären Algorithmus haben.

3 Der (1+1) evolutionäre Algorithmus

Wir werden unsere Untersuchungen im wesentlichen auf den so genannten (1+1) evolutionären Algorithmus stützen, der mit gewissem Recht als einfachster evolutionärer Algorithmus überhaupt bezeichnet werden kann. Wesentliche Einschränkungen gegenüber evolutionären Algorithmen, wie sie tatsächlich eingesetzt werden, ist vor allem die Beschränkung der Populationsgröße auf ein einziges Individuum. Dadurch entfallen die Möglichkeiten, Selektion zur Reproduktion und Rekombination einzusetzen. Es bleiben Mutation und Selektion zur Ersetzung als Suchoperatoren übrig. Der (1+1) EA verwendet eine einfache bitweise Mutation mit einer festen Mutationswahrscheinlichkeit $p(n)$ und einen fast deterministischen Selektionsoperator, der bei Evolutionsstrategien eingesetzt wird und dort Plus-Selektion genannt wird. Dabei werden von den Eltern und Kindern die Individuen mit größtem Funktionswert selektiert, bei Gleichheit erhalten die Kinder den Vorzug vor den Eltern, bei Gleichheit unter den Kindern wählt man zufällig gleichverteilt. Weil beim (1+1) EA aus einem Elter ein Kind generiert wird, ist die Plus-Selektion hier vollkommen deterministisch. Trotz seiner Einfachheit ist die Analyse des Algorithmus weder einfach noch uninteressant. Vor allem aber lassen sich viele interessante und für evolutionäre Algorithmen allgemein typische Effekte beobachten und gut erklären. So ist es nicht verwunderlich, dass der (1+1) EA schon mehrfach Gegenstand intensiver Untersuchungen war (Mühlenbein 1992; Rudolph 1997a; Garnier, Kallel und Schoenauer 1999). Der Algorithmus soll eine Funktion $f\colon \{0,1\}^n \to \mathbb{R}$ maximieren, dabei verlangen wir nicht, dass der Algorithmus erkennt, dass ein Maximum gefunden wurde. Wir vernachlässigen das Problem, ein angemessenes Stopp- oder Abbruchkriterium zu definieren und lassen das Verfahren formal endlos lange laufen. Wir untersuchen die Anzahl von Generationen oder Schritte, bis das Verfahren erstmals den Funktionswert für einen global maximalen Punkt im Suchraum anfragt.

Algorithmus 3.1. (1+1) evolutionärer Algorithmus ((1+1) EA)

1. *Wähle* $p(n) \in (0; 1/2]$.
2. *Wähle* $x \in \{0,1\}^n$ *zufällig gleichverteilt.*
3. $y := x$
   Für alle $i \in \{1, 2, \ldots, n\}$
      Mit Wahrscheinlichkeit $p(n)$*:*
         Ersetze das i*-te Bit in* y *durch sein Komplement.*
4. *Falls* $f(y) \geq f(x)$ *gilt, setze* $x := y$
5. *Weiter bei* 3.

In der ersten Zeile wird die Mutationswahrscheinlichkeit gewählt. Dann wird in Zeile 2 die Population, die nur aus dem Individuum x besteht, rein zufällig initialisiert. In der dritten Zeile findet die Mutation statt. Dazu wird von x zunächst eine Kopie y angelegt. Dann wird unabhängig für jedes Bit ein Zufallsexperiment durchgeführt. Mit Wahrscheinlichkeit $p(n)$ wird der Wert des Bits in y geändert. In Zeile 4 findet dann die Selektion zur Ersetzung statt, der Funktionswert $f(y)$ des Kindes y wird mit dem des Elters verglichen. Wenn der Funktionswert des Kindes mindestens genauso groß ist, ersetzt das Kind sein Elter, andernfalls wird x beibehalten.

Offensichtlich beeinflusst die Wahl der Mutationswahrscheinlichkeit $p(n)$ in Zeile 1 die Mutation in Zeile 3. Die erwartete Anzahl Bits, die verändert werden — wir sagen kurz mutierte Bits — beträgt $n \cdot p(n)$. Die aus der Biologie entlehnte Vorstellung ist, dass Mutationen eher unwahrscheinlich und „klein“ sind. Die Metrik, in der wir Abstände messen, ist der Hammingabstand. Das ist angemessen, weil mit kleinem $p(n)$ dann größere Abstände tatsächlich schwieriger zu überbrücken sind als kleine. Wir legen basierend auf diesen Überlegungen $1/2$ als Obergrenze für den Wert von $p(n)$ fest. Wir werden uns im Rahmen dieser Arbeit im wesentlichen auf eine feste Wahl für die Mutationswahrscheinlichkeit $p(n)$ festlegen. Diese Entscheidung motivieren und diskutieren wir etwas später in diesem Kapitel. In Kapitel 7 überlegen wir, wie Situationen aussehen können, in denen diese feste Wahl nicht optimal ist.

Neben einer anderen Initialisierung als der rein zufälligen sind noch eine andere Mutation und eine andere Selektion denkbar. Mit alternativen Initialisierungen, die dann in aller Regel problemabhängig sind, beschäftigen wir uns nicht. Etwas andere Mutations- und Selektionsverfahren werden wir in den Kapiteln 7 und 8 untersuchen. Der Rahmen dessen, was wir untersuchen, wird im wesentlichen durch die (ingenieurwissenschaftliche) Praxis bestimmt. In der Praxis eingesetzte alternative Operatoren motivieren für uns eine theoretisch orientierte Untersuchung. Auf der anderen Seite entwickeln solche Analysen mitunter auch ein Eigenleben, regen sie von sich aus zum Design neuer Varianten evolutionärer Algorithmen an. Ein Beispiel dafür findet man in dieser Arbeit in Kapitel 7.

Der (1+1) EA ist der erste evolutionäre Algorithmus, den wir konkret betrachten. Wir wollen sein Verhalten auf einer Reihe einfacher und wichtiger Funktionen untersuchen. Dabei geht es uns um die Anzahl der Orakelanfragen, die der Algorithmus stellt, bis er erstmalig ein globales Maximum findet. Im Falle des (1+1) EA ist das identisch mit der Anzahl der Generationen. Wie bereits angekündigt werden wir dabei eine feste Wahl für die Mutationswahrscheinlichkeit $p(n)$ treffen und begründen, die wir auch in den folgenden Kapiteln beibehalten. Es handelt sich um die am häufigsten empfohlene konstante Mutationswahrscheinlichkeit (Mühlenbein 1992; Bäck 1993). Die erste Funktion, die wir näher betrachten, ist die vermutlich bekannteste (und „beliebteste") Funktion im Zusammenhang mit der Untersuchung evolutionärer Algorithmen. Sie ist unter den Namen OneMax oder auch CountingOnes-Problem (dann auch kurz COP) bekannt und zählt einfach die Anzahl von Einsen im Bitstring. Wir nennen sie hier f_1.

Definition 3.2. *Die Funktion* $f_1\colon \{0,1\}^n \to \mathbb{R}$ *ist definiert durch*

$$f_1(x) := \|x\|_1.$$

Die Funktion f_1 ist symmetrisch, das heißt ihr Funktionswert hängt nur von der Anzahl der Einsen im Eingabestring ab. Solche symmetrischen Funktionen haben den unmittelbaren Vorteil, dass man sie als Funktion in Abhängigkeit von der Anzahl der Einsen auffassen und dann auch so graphisch darstellen kann. Aus solchen Funktionsgraphen lässt sich oft schon ein zutreffendes, intuitives Verständnis der Funktion und des Verhaltens des (1+1) EA auf dieser Funktion gewinnen. Im Falle von f_1 ist das vielleicht unnötig, weil die Funktion so einfach ist. Wir wollen aber der Vollständigkeit halber auch hier den Funktionsgraphen für $n = 40$ angeben (Abbildung 1).

Wir wollen anhand dieser Funktion Hinweise für eine geeignete Wahl der Mutationswahrscheinlichkeit finden. Dazu stellen wir einige einfache Überlegungen an. Offensichtlich ist $\mathbf{1}$ das eindeutige Maximum. Die Wahrscheinlichkeit, nicht genau diesen Punkt als initialen Startpunkt zu wählen ist $1 - 2^{-n}$. Dann gibt es natürlich eine letzte Mutation, bei der dann schließlich das Maximum gefunden wird. Wenn der aktuelle String bei diesem letzten Schritt genau b Nullen enthält, beträgt die Wahrscheinlichkeit für die Mutation genau $p(n)^b(1 - p(n))^{n-b}$, weil genau die b Bits mit dem Wert Null gleichzeitig mutieren müssen. Weil wir $p(n) \leq 1/2$ voraussetzen, wird dieser Ausdruck maximal für $b = 1$. Die Mutationswahrscheinlichkeit für diesen letzten Schritt beträgt dann $p(n)(1 - p(n))^{n-1}$ und wird maximal für $p(n) = 1/n$: Um das einzusehen, substituieren wir $p(n)$ durch x und leiten nach x ab. Wir erhalten $(1 - x)^{n-1} - (n - 1)x(1 - x)^{n-2}$, was offenbar $x = 1/n$ als einzige Nullstelle hat. Man macht sich leicht (etwa durch Einsetzen von $x = 2/n$ und $x = 1/(2n)$) klar, dass $p(n) = 1/n$ Maximalstelle ist. Wir untersuchen jetzt in einem ersten Schritt die erwartete Laufzeit des (1+1) EA, also die erwartete Anzahl Generationen bis zum Erreichen des globalen Maximums, für die Wahl $p(n) = 1/n$. Das Resultat ist seit langem wohlbekannt, Mühlenbein (1992) leitet es näherungsweise her, Rudolph (1997a) gibt eine korrekte Abschätzung.

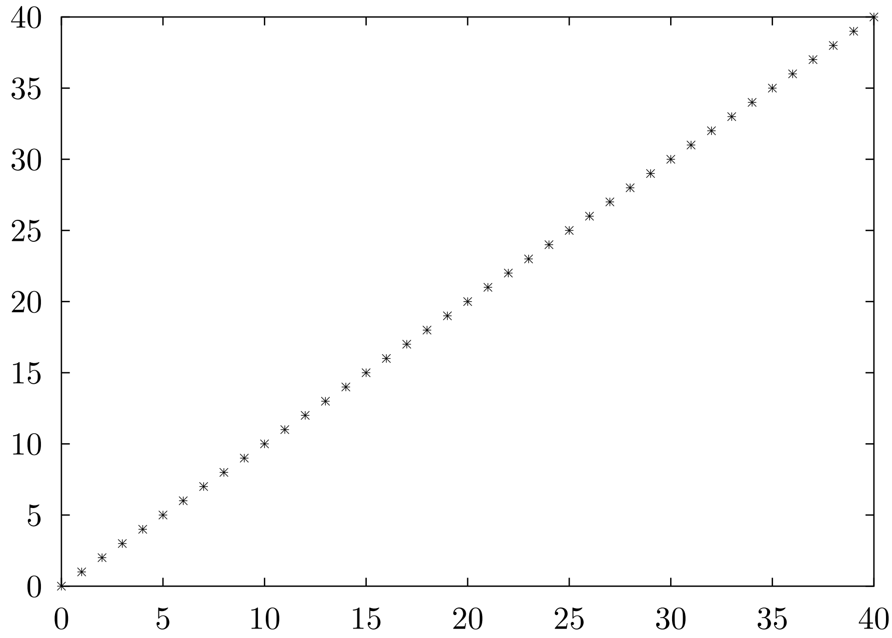

Abbildung 1: Die Funktion $f_1 \colon \{0,1\}^{40} \to \mathbb{R}$.

Die Beweisidee ist sehr simpel und beruht auf der strikten, deterministischen Selektion in Zeile 4 des (1+1) EA. Wenn bekannt ist, dass der aktuelle Zustand x Funktionswert $f(x) \geq w$ hat, so kann kein Zustand x' mit $f(x) < w$ jemals aktueller Zustand werden. Es ist darum oft hilfreich, Teilmengen $A \subseteq \{0,1\}^n$ bezüglich der Funktionswerte $f(a)$ der Elemente $a \in A$ zu klassifizieren. Diese Klassifikation hängt natürlich von der aktuell betrachteten Zielfunktion f ab. Wir definieren nun zunächst diese Klassifikation in Form einer Relation $<_f$ und überlegen uns dann, wie diese Relation beim Nachweis oberer Schranken hilfreich verwendet werden kann.

Definition 3.3. *Für eine Funktion $f \colon \{0,1\}^n \to \mathbb{R}$ und zwei nicht leere Mengen $A, B \subseteq \{0,1\}^n$, schreiben wir $A <_f B$ genau dann, wenn für alle $a \in A$ und alle $b \in B$ stets $f(a) < f(b)$ gilt. Wir bezeichnen A und B dann auch als* Ranggruppen *und sagen, dass B eine höhere Ranggruppe ist als A.*

Nehmen wir an, wir haben zu einer Zielfunktion f eine Partitionierung P_1, P_2, ... , P_s gegeben, so dass

$$P_1 <_f P_2 <_f \cdots <_f P_s$$

und

$$\bigcup_{1 \le i \le s} P_i = \{0,1\}^n$$

gilt. Nehmen wir weiter an, wir kennen Übergangswahrscheinlichkeiten p_1, p_2, ..., p_{s-1}, so dass $0 < p_i \le 1$ für alle $i \in \{1, 2, \ldots, s-1\}$ gilt und p_i eine untere Schranke für die Wahrscheinlichkeit ist, in einer Generation von einem Zustand $x \in P_i$ zu einem Zustand $x' \in P_j$ mit $j > i$ zu kommen, also

$$p_i \le p(n)^h (1 - p(n))^{n-h}$$

mit

$$h = \max \left\{ \min \left\{ H(x, x') \mid x' \in \bigcup_{i<j \le s} P_j \right\} \mid x \in P_i \right\}$$

gilt. Dann ist $1/p_i$ eine obere Schranke für die erwartete Anzahl Generationen, von P_i mindestens nach P_{i+1} zu kommen und

$$\sum_{1 \le i < s} \frac{1}{p_i}$$

ist eine obere Schranke für die erwartete Laufzeit. Wir verwenden diese Methode, um eine obere Schranke für die erwartete Laufzeit des (1+1) EA auf der Funktion f_1 nachzuweisen.

Satz 3.4. *Die erwartete Laufzeit des* $(1+1)$ *EA auf der Funktion* f_1 *mit* $p(n) = 1/n$ *beträgt* $\Theta(n \log n)$.

Beweis. Wir beginnen mit der oberen Schranke und verwenden die oben skizzierte Beweismethode. Wir partitionieren den Suchraum $\{0,1\}^n$ auf triviale Art und Weise in die $n+1$ Ranggruppen $P_0, P_1, \ldots, P_n$ mit

$$P_i := \{x \in \{0,1\}^n \mid f_1(x) = i\}$$

für alle $i \in \{0, 1, \ldots, n\}$. Es ist klar, dass für alle $i \in \{0, 1, \ldots, n-1\}$ stets $P_i <_{f_1} P_{i+1}$ gilt. Es ist $f_1(x) = \|x\|_1$ für alle $x \in \{0,1\}^n$, darum enthält P_i genau die $|P_i| = \binom{n}{i}$ Strings, die genau i Bits mit Wert Eins enthalten. Es genügt also für jeden String $x \in P_i$ mit $i \in \{0, 1, \ldots, n-1\}$, die Anzahl Einsbits in x um 1 zu erhöhen, um zu P_{i+1} zu gelangen. Dafür reicht es aus, wenn genau eines der $n-i$ Bits mit Wert Null mutiert und die anderen $n-1$ Bits nicht mutieren. Wir können folglich

$$p_i = \binom{n-i}{1}\frac{1}{n}\left(1-\frac{1}{n}\right)^{n-1} \geq \frac{n-i}{en}$$

als untere Schranke für die Übergangswahrscheinlichkeiten verwenden. Daraus ergibt sich für die erwartete Laufzeit

$$\sum_{0\leq i<n} \frac{en}{n-i} = en \sum_{1\leq i\leq n} \frac{1}{i} \leq en(1+\ln n) = O\left(n \log n\right)$$

als obere Schranke.

Für die untere Schranke überlegen wir uns, dass mit Wahrscheinlichkeit mindestens $1/2$ mindestens $\lfloor n/2 \rfloor$ der Bits den Wert Null haben und wenigstens einmal mutieren müssen. Die Wahrscheinlichkeit, nach t Schritten unter diesen Bits noch eins zu finden, dass gar nicht mutiert ist, beträgt

$$1-\left(1-\left(1-\frac{1}{n}\right)^t\right)^{\lfloor n/2 \rfloor},$$

wie man sich leicht klar macht. Die Wahrscheinlichkeit, dass ein bestimmtes Bit in einem Schritt mutiert, beträgt $p(n) = 1/n$. Die Wahrscheinlichkeit, dass das Bit nicht mutiert beträgt also $1-1/n$. Offenbar beträgt die Wahrscheinlichkeit, dass ein bestimmtes Bit in t Schritten gar nicht mutiert $(1-1/n)^t$. Mit Wahrscheinlichkeit $1-(1-1/n)^t$ mutiert also ein bestimmtes Bit in t Schritten mindestens einmal. Folglich beträgt die Wahrscheinlichkeit, dass $\lfloor n/2 \rfloor$ Bits in t Schritten alle mindestens einmal mutieren $(1-(1-1/n)^t)^{\lfloor n/2 \rfloor}$. Für die uns eigentlich interessierende Wahrscheinlichkeit, dass es unter $\lfloor n/2 \rfloor$ Bits mindestens eines gibt, dass in t Schritten gar nicht mutiert, ergibt sich folglich wie behauptet $1-(1-(1-1/n)^t)^{\lfloor n/2 \rfloor}$. Wir setzen $t = n \ln n$ und erhalten insgesamt, dass mit Wahrscheinlichkeit mindestens

$$\frac{1}{2}\cdot\left(1-\left(1-\frac{1}{n}\right)^{n/2}\right) \geq \frac{1}{2}\cdot\left(1-\frac{1}{e^{1/2}}\right) = \Omega(1)$$

nach $n \ln n$ Schritten mindestens ein Bit, das anfangs den Wert Null hatte, gar nicht mutiert ist und folglich immer noch den Wert Null hat. Das gibt $\Omega(n \log n)$ als untere Schranke. $\square$

Wir haben gesehen, dass bei passender Wahl der Mutationswahrscheinlichkeit $p(n)$ die Funktion f_1 mit dem (1+1) EA in erwarteter Zeit $O\left(n \log n\right)$ optimiert werden kann. Es stellen sich natürlich sofort die Fragen, ob $p(n) = 1/n$

auch über den Rahmen von f_1 hinaus eine allgemein vernünftige Wahl ist und ob für andere Werte von $p(n)$ eine substanziell kürzere Laufzeit möglich ist. Wir kommen zunächst nochmal auf unsere Überlegungen bezüglich des allerletzten Schritts zurück. Nehmen wir an, dass $p(n) = 2(\ln\ln n)/n$ gewählt wird. Allgemein könnte man auch $p(n) = \alpha(n)/n$ mit einer Funktion $\alpha\colon \mathbb{N} \to \mathbb{R}^+$ mit $\lim\limits_{n\to\infty} \alpha(n) \to \infty$ betrachten. Wir werden uns in Kapitel 7 mit dem Thema noch einmal eingehender beschäftigen. Bei der konkreten Wahl $p(n) = 2(\ln\ln n)/n$ ergibt sich jedenfalls allein für den letzten Schritt eine untere Schranke von

$$\frac{n}{2\ln\ln n}\left(1-\frac{2\ln\ln n}{n}\right)^{1-n} = \Omega\left(\frac{n}{\ln\ln n}\cdot e^{2\ln\ln n}\right) = \Omega\left(n\log n\cdot\frac{\log n}{\log\log n}\right),$$

was substanziell über $n\log n$ liegt. Für größere Wahlen von $p(n)$ ergeben sich noch größere untere Schranken. Andererseits sind substanziell kleinere Werte für die Mutationswahrscheinlichkeit $p(n)$ als $1/n$ auch ungünstig.

Satz 3.5. *Der* $(1+1)$ *EA mit* $p(n) = \alpha(n)/n$*, wobei* $\lim\limits_{n\to\infty}\alpha(n) = 0$ *gilt, braucht* $\Omega((n\log n)/\alpha(n))$ *Schritte, um die Funktion* $f_1\colon \{0,1\}^n \to \mathbb{R}$ *zu optimieren.*

Beweis. Wir hatten beim Beweis von Satz 3.4 überlegt, dass anfangs mindestens $\lfloor n/2\rfloor$ Bits mit Wahrscheinlichkeit mindestens $1/2$ den Wert Null haben und folglich mindestens einmal mutieren müssen. Die Wahrscheinlichkeit, dass das nach t Schritten nicht der Fall ist, beträgt

$$1-\left(1-\left(1-\frac{\alpha(n)}{n}\right)^t\right)^{\lfloor n/2\rfloor}.$$

Wir setzen $t = (n\ln n)/\alpha(n)$ ein und sehen, dass mit Wahrscheinlichkeit mindestens

$$\frac{1}{2}\cdot\left(1-\left(1-\left(1-\frac{\alpha(n)}{n}\right)^{(n\ln n)/\alpha(n)}\right)^{n/2}\right) \geq \frac{1}{2}\left(1-\frac{1}{e^{1/2}}\right) = \Omega(1)$$

das globale Maximum noch nicht erreicht ist. Das gibt die untere Schranke $\Omega((n\log n)/\alpha(n))$ für die erwartete Laufzeit. □

Intuitiv ist Satz 3.5 jedenfalls qualitativ unmittelbar einsichtig. Wird $p(n)$ mit $p(n) = \alpha(n)/n$ substanziell kleiner als $1/n$ gewählt, mutieren in einem Schritt im Erwartungswert nur $\alpha(n)$ Bits, wobei $\lim_{n\to\infty} \alpha(n) = 0$ gilt. Es geschieht also zu häufig überhaupt nichts, was einer raschen Optimierung offenbar nicht zuträglich sein kann. Andererseits haben wir gesehen, dass schon die Wahl $p(n) = 2(\ln\ln n)/n$ zu groß ist. Wir machen uns hier nicht die Mühe, anhand der Funktion f_1 die Lücke zwischen $1/n$ und $2(\ln\ln n)/n$ weiter zu schließen. Stattdessen betrachten wir eine zweite Funktion, bei der auf leichtere Art deutlich wird, worauf es uns ankommt.

Die Funktion f_B ist ebenfalls nicht unbekannt, sie wird von Rudolph (1997a) als Beispiel verwendet, um Grenzen der von ihm für die obere Schranke der erwarteten Laufzeit des (1+1) EA auf f_1 verwendeten Beweismethode aufzuzeigen. Droste, Jansen und Wegener (1998d) dient sie als wichtiger Meilenstein auf dem Weg zur Analyse des (1+1) EA auf der Klasse der linearen Funktionen.

Definition 3.6. *Die Funktion $f_B \colon \{0,1\}^n \to \mathbb{R}$ ist definiert durch*

$$f_B(x) := \sum_{1\leq i\leq n} 2^{n-i} x_i.$$

Die Funktion f_B wird mitunter auch Bin oder BinVal genannt. Zu einem Binärstring x ist der Funktionswert $f_\mathrm{B}(x)$ gerade die nicht-negative Zahl, die x interpretiert als Standardbinärcodierung darstellt. Offenbar ist f_B keine symmetrische Funktion. Der Funktionswert ist die gewichtete Summe der einzelnen Bits, wobei das i-te Bit mit Gewicht 2^{n-i} eingeht. Weil für alle $i \in \{1, 2, \ldots, n-1\}$

$$2^{n-i} > \sum_{i<j\leq n} 2^{n-j}$$

gilt, kommt es im Rahmen des (1+1) EA bei einer Mutation nur darauf an, ob unter den mutierenden Bits das mit dem kleinsten Index seinen Wert von Null nach Eins oder umgekehrt wechselt. Wird dieses Bit von einer Null zu einer Eins, so wächst der Funktionswert insgesamt und die Mutation wird akzeptiert: der neue Bitstring y wird aktueller Bitstring x. Andernfalls wird die Mutation verworfen. Diese starke Eigenschaft kann leicht für den Nachweis einer oberen Schranke für die erwartete Laufzeit des (1+1) EA nach bekanntem Muster verwendet werden. Die Anzahl ununterbrochener führender Einsen kann offenbar nie abnehmen. Solange es mindestens eine Null im aktuellen Bitstring x gibt, gibt es eine Mutation, welche die Anzahl führender Einsen um mindestens 1 vergrößert. Wählen wir die Mutationswahrscheinlichkeit $p(n) = 1/n$, so hat eine solche Mutation mindestens Wahrscheinlichkeit $(1/n)(1-1/n)^{n-1}$, so dass sich en als obere Schranke für die erwartete

Zeit bis zum Eintreffen einer solchen Mutation ergibt. Das entspricht der Partitionierung des Suchraums $\{0,1\}^n$ in $P_0, P_1, \dots, P_n$ mit

$$P_i = \left\{ x \in \{0,1\}^n \mid \sum_{1 \leq j \leq n} \prod_{1 \leq k \leq j} x_k = i \right\}$$

für alle $i \in \{0, 1, \dots, n\}$. Es ist klar, dass

$$\sum_{1 \leq j \leq n} \prod_{1 \leq k \leq j} x_k$$

genau die Anzahl führender Bits mit Wert Eins in x zählt. Damit haben wir

$$\sum_{1 \leq j \leq i} 2^{n-j} \leq f_{\mathrm{B}}(x) < \sum_{1 \leq j \leq i+1} 2^{n-j}$$

für alle $x \in P_i$ und damit ist $P_i <_{f_{\mathrm{B}}} P_{i+1}$ für alle $i \in \{0, 1, \dots, n-1\}$ offensichtlich. Wir haben uns überlegt, dass wir

$$p_i = \frac{1}{n}\left(1 - \frac{1}{n}\right)^{n-1} \geq \frac{1}{en}$$

als untere Schranke für die Übergangswahrscheinlichkeiten verwenden können. Damit haben wir

$$\sum_{0 \leq i < n} \frac{1}{p_i} \leq \sum_{0 \leq i < n} en = n \cdot en = O\left(n^2\right)$$

als obere Schranke für die erwartete Laufzeit. Das ist aber zu pessimistisch. Tatsächlich kann die Funktion f_{B} vom (1+1) EA mit $p(n) = 1/n$ im wesentlichen genauso effizient optimiert werden wie die Funktion f_1.

Satz 3.7. *Die erwartete Laufzeit des* $(1+1)$ *EA mit Mutationswahrscheinlichkeit* $p(n) = 1/n$ *auf der Funktion* $f_B\colon \{0,1\}^n \to \mathbb{R}$ *beträgt* $\Theta(n \log n)$.

Beweis. Die untere Schranke $\Omega(n \log n)$ folgt auf die gleiche Art wie im Beweis zu Satz 3.4. Wir beweisen darum nur die obere Schranke.

Ohne Beschränkung der Allgemeinheit nehmen wir an, dass n gerade ist. Wir bezeichnen die Anzahl Generationen, bis nach der zufälligen Initialisierung erstmals $x = \mathbf{1}$ gilt als T. Wir teilen den Zufallsprozess, der schließlich dazu führt, gedanklich in zwei Teile: die Anzahl Generationen, bis erstmals die linke Hälfte von x nur noch Einsen enthält (also $x_1 = x_2 = \cdots = x_{n/2} = 1$

gilt), bezeichnen wir mit T_1. Die Anzahl restlicher Generationen heißt T_2. Damit gilt $T = T_1 + T_2$. Wir geben nun getrennt obere Schranken für $\mathrm{E}(T_1)$ und $\mathrm{E}(T_2)$ an und beginnen mit T_1. Wir analysieren einen Zufallsprozess, der ähnliche Eigenschaften wie der (1+1) EA auf der Funktion f_{B} hat und beweisbar nicht schneller ist. Wir zerlegen die erste Phase, die T_1 Generationen dauert, in $n/2$ Teilphasen. Es bezeichne X_i (mit $i \in \{0, \dots, n/2\}$) den ersten Zeitpunkt, zu dem die linke Hälfte von x erstmals mindestens i Einsen enthält (also $\|x_1 x_2 \cdots x_{n/2}\|_1 \geq i$ gilt). Insbesondere ist natürlich $X_0 = 0$. Damit haben wir also

$$T_1 = X_{n/2} = \sum_{1 \leq i \leq n/2} (X_i - X_{i-1}).$$

Wir wollen eine obere Schranke für die erwartete Differenz zwischen den Zeitpunkten angeben, zu denen die linke Hälfte erstmals i und erstmals $i-1$ Einsen enthielt. Zu diesem Zweck unterscheiden wir zwei verschiedene Typen von Generationen. Wir nennen eine Generation (also einen Schleifendurchlauf des (1+1) EA) erfolgreich, wenn eine Mutation eintritt, die in der linken Hälfte von x eine Änderung bewirkt und akzeptiert wird. Alle anderen Schritte heißen erfolglos. Wir bezeichnen die Anzahl erfolgreicher Schritte im Intervall $[X_{i-1}+1, \dots, X_i]$ mit Y_i. Die Anzahl erfolgloser Schritte in diesem Zeitraum bezeichnen wir mit Z_i. Jetzt haben wir also

$$T_1 = \sum_{1 \leq i \leq n/2} (Y_i + Z_i)$$

und wir können erfolgreiche und erfolglose Schritte getrennt betrachten.

Wir beginnen mit den erfolgreichen Schritten und behaupten, dass $\mathrm{E}(Y_i) \leq 2$ für alle $i \in \{1, \dots, n/2\}$ gilt. Am Anfang des Intervalls haben wir einen Bitstring z_0, der mindestens i Einsen enthält. Der (1+1) EA durchläuft ausgehend von z_0 eine Folge von aktuellen Zuständen z_1, z_2, …, bis erstmals ein Zustand mit mindestens $i+1$ Einsen aktueller Zustand wird. Wir behaupten, dass die erwartete Länge dieser Folge höchstens 2 ist. Dazu betrachten wir die Differenz in der Anzahl der Einsen zwischen den Zuständen z_j und z_{j-1} und bezeichnen sie mit D_j, also $D_j = \|z_j\|_1 - \|z_{j-1}\|_1$. Es bezeichnet also Y_i den ersten Zeitpunkt t, zu dem

$$\sum_{1 \leq i \leq t} D_i \geq 1$$

gilt. Wir beschreiben die Wahrscheinlichkeitsverteilung von D_j unter der Annahme, dass

$$\sum_{1 \leq i < j} D_i < 1$$

gilt. Der String z_{j-1} enthält höchstens $i-1$ Einsen. Der nächste Schritt ist genau dann erfolgreich, wenn in der linken Hälfte von z_{j-1} mindestens ein Bit mutiert und unter allen mutierenden Bits das mit minimalem Index seinen Wert von Null auf Eins wechselt. Dieses Bit trägt 1 zum Wert von D_j bei. Es bezeichne k die Anzahl Bits rechts von diesem Bit, die den Wert Eins haben und l die Anzahl Bits rechts von diesem Bit, die den Wert Null haben, wobei nur Bits in der linken Hälfte von z_{j-1} zählen. Dann gilt $D_j = 1 - B_{j,0} + B_{j,1}$, wobei $B_{j,0}$ die Anzahl mutierender Bits bezeichnet, die ihren Wert von Eins auf Null ändern und $B_{j,1}$ die Anzahl mutierender Bits, die ihren Wert von Null auf Eins ändern. Dabei sind $B_{j,0}$ eine binomialverteilte Zufallsvariable mit den Parametern k und $1/n$ und $B_{j,1}$ eine binomialverteilte Zufallsvariable mit den Parameter l und $1/n$. Das liegt daran, dass das mutierende Bit mit minimalem Index echt kleineren Index hat als diese k bzw. l Bits. Darum hat der Umstand, dass wir bedingte Ereignisse betrachten, hier keinen Einfluss und es ergeben sich einfach Binomialverteilungen. Wir führen zur Vereinfachung der Notation Zufallsvariablen $B_j := B_{j,0} - B_{j,1}$ ein, die wir analysieren wollen.

Wir vereinfachen die Analyse, indem wir stattdessen unabhängige Zufallsvariable B_j^* einführen, die binomialverteilt sind mit den Parametern $(n/2)-1$ und $1/n$. Es sei $D_j^* = 1 - B_j^*$. Die wesentliche Eigenschaft dieser neu eingeführten Zufallsvariablen ist, dass für alle Werte und unabhängig von $B_1, \dots, B_{j-1}$ stets

$$\operatorname{Prob}\left(B_j^* \leq r\right) \leq \operatorname{Prob}\left(B_j \leq r\right)$$

gilt. Folglich gilt auch stets

$$\operatorname{Prob}\left(D_j^* \leq r\right) \geq \operatorname{Prob}\left(D_j \leq r\right)$$

und der von uns untersuchte Prozess ist tatsächlich höchstens langsamer als der (1+1) EA. Es bezeichne nun T^* den ersten Zeitpunkt t, zu dem

$$\sum_{1 \leq i \leq t} D_i^* \geq 1$$

gilt. Offenbar gilt dann für alle Werte von t^* stets

$$\operatorname{Prob}\left(T^* \leq t^*\right) \leq \operatorname{Prob}\left(Y_i \leq t^*\right)$$

und somit $\operatorname{E}(T^*) \geq \operatorname{E}(Y_i)$. Es genügt folglich, 2 als obere Schranke für den Erwartungswert von T^* nachzuweisen.

Es ist $D_j^* \leq 1$ und

$$\mathrm{E}\left(D_j^*\right) = 1 - \frac{(n/2)-1}{n} > \frac{1}{2}.$$

Wir betrachten nun Zufallsvariablen S_t, die wir mittels

$$S_t := \sum_{1 \leq i \leq t} D_i^*$$

definieren. Die Zufallsvariablen $S_0 = 0, S_1, S_2, \ldots$ beschreiben einen zeithomogenen Random Walk. Das ist anders als beim (1+1) EA, bei dem die Wahrscheinlichkeit, die Anzahl der Einsen wieder zu erhöhen steigt, wenn die Anzahl der Einsen abnimmt. Unser zu pessimistisches Modell, in dem das eben nicht der Fall ist, wird durch die so erkaufte Homogenität wesentlich besser analysierbar. Weil $D_j^* \leq 1$ ist, erreichen wir 1 als ersten Punkt rechts des Startpunktes 0. Wir halten den Random Walk dann an und analysieren die Stoppzeit T^* dieses Prozesses. Der aktuelle Punkt nach dem ersten Schritt ist $1 - D_1^*$. Weil der Random Walk homogen ist, wissen wir, dass die erwartete Stoppzeit $d \cdot \mathrm{E}\left(T^*\right)$ beträgt, wenn der Prozess nicht bei 0 sondern $1-d$ gestartet wird. Es gilt also

$$\begin{aligned}
\mathrm{E}\left(T^*\right) &= \sum_{-(n/2)+2 \leq d \leq 1} \mathrm{Prob}\left(D_1^* = d\right)(1-d)\mathrm{E}\left(T^*\right) \\
&= 1 + \mathrm{E}\left(T^*\right) - \mathrm{E}\left(T^*\right) \sum_{-(n/2)+2 \leq d \leq 1} d \cdot \mathrm{Prob}\left(D_1^* = d\right) \\
&= 1 + \mathrm{E}\left(T^*\right) - \mathrm{E}\left(T^*\right) \cdot \mathrm{E}\left(D_1^*\right).
\end{aligned}$$

Folglich gilt $\mathrm{E}\left(T^*\right) = 1/\mathrm{E}\left(D_1^*\right)$. Weil $\mathrm{E}\left(D_k^*\right) \geq 1/2$ für alle k gilt, folgt wie behauptet $\mathrm{E}\left(T^*\right) \leq 2$.

Die Gleichung $\mathrm{E}\left(T^*\right) \cdot \mathrm{E}\left(D_1^*\right) = 1$ ist als Waldsche Identität bekannt (Feller 1971). An dieser Beweisstelle wird auch erkennbar, warum wir den Lauf des (1+1) EA in zwei Teile unterteilen. Andernfalls müssten wir alle Bits des Strings berücksichtigen und nicht nur die der linken Hälfte. Dann ist B_j^* binomialverteilt mit den Parametern $n-1$ und $1/n$, was dann nur noch $\mathrm{E}\left(D_j^*\right) \geq 1/n$ liefert. Das reicht aber nicht für den Beweis von Satz 3.7.

Jetzt müssen wir die erwartete Anzahl erfolgloser Schritte Z_i nach oben abschätzen. Im betrachteten Intervall $[X_{i-1}+1, \ldots, X_i]$ gilt stets, dass der aktuelle String in der linken Hälfte höchstens $i-1$ Einsen enthält. Für eine erfolgreiche Generation genügt es, wenn eine der mindestens $(n/2)-(i-1)$

Nullen mutiert und alle anderen Bits der linken Hälfte nicht mutieren. Die Wahrscheinlichkeit hierfür beträgt

$$\binom{(n/2)-(i-1)}{1}\frac{1}{n}\left(1-\frac{1}{n}\right)^{(n-2)-1} \geq \left(\frac{n}{2}-i+1\right)\frac{1}{e^{1/2}n}.$$

Die erwartete Zeit, bis k erfolgreiche Mutationen eintreten, ist folglich durch $((n/2)-i+1)^{-1}e^{1/2}n \cdot k$ nach oben beschränkt. Damit haben wir

$$\begin{aligned}
\mathrm{E}\left(Y_i+Z_i\right) &= \sum_k \mathrm{Prob}\left(Y_i=k\right) \cdot \mathrm{E}\left(Y_i+Z_i \mid Y_i=k\right)\\
&= \sum_k \mathrm{Prob}\left(Y_i=k\right) \cdot \left(\frac{n}{2}-i+1\right)^{-1} \cdot e^{1/2}n \cdot k\\
&= \mathrm{E}\left(Y_i\right) \cdot \left(\frac{n}{2}-i+1\right)^{-1} \cdot e^{1/2}n\\
&\leq 2 \cdot \left(\frac{n}{2}-i+1\right)^{-1} \cdot e^{1/2}n.
\end{aligned}$$

Insgesamt haben wir also

$$\begin{aligned}
\mathrm{E}\left(T_1\right) = \sum_{1\leq i\leq n/2} \mathrm{E}\left(Y_i+Z_i\right) &\leq 2e^{1/2}n \cdot \sum_{0\leq i<n/2}\left(\frac{n}{2}-i\right)^{-1}\\
&= 2e^{1/2}n \cdot \sum_{1\leq i\leq n/2}\frac{1}{i}\\
&\leq 2e^{1/2}n\ln n = O\left(n\log n\right)
\end{aligned}$$

für die erwartete Länge der ersten Phase. In der zweiten Phase kann man im wesentlichen analoge Überlegungen anstellen. Die Bits in der linken Hälfte haben alle den Wert Eins und behalten diesen auch. Für die Bits der rechten Hälfte müssen wir nun zusätzlich berücksichtigen, dass es eine notwendige Bedingung für eine erfolgreiche Mutation ist, dass kein Bit der linken Hälfte des aktuellen Strings mutiert. Das ändert den Term $(1-1/n)^{(n/2)-1}$ für die Wahrscheinlichkeit einer erfolgreichen Mutation in $(1-1/n)^{n-1}$. Damit erhalten wir also

$$\mathrm{E}\left(T_2\right) \leq 2en\ln n = O\left(n\log n\right)$$

und insgesamt haben wir

$$\mathrm{E}\left(T\right) = \mathrm{E}\left(T_1\right) + \mathrm{E}\left(T_2\right) \leq 2\left(e+e^{1/2}\right)n\ln n = O\left(n\log n\right),$$

wie behauptet. □

Die Funktion f_B ist also ein zweites Beispiel dafür, dass die Wahl $p(n) = 1/n$ für die Mutationswahrscheinlichkeit zu einer effizienten Optimierung führen kann. Vor allem kann man aber an der Funktion f_B leichter als an der Funktion f_1 erkennen, warum eine substanziell größere Mutationswahrscheinlichkeit problematisch sein kann. Die Funktion f_B verlangt in viel stärkerem Maße als die Funktion f_1, dass bestimmte Bits *nicht* mutieren. Bei f_1 reicht es aus, wenn die Anzahl der Bits, die ihren Wert von Null auf Eins ändern, größer ist als die Anzahl der Bits, die ihren Wert von Eins auf Null ändern. Bei f_B hingegen ist nötig, dass alle Bits links von dem Bit, das unter allen mutierenden Bits, die ihren Wert von Null auf Eins ändern, minimalen Index hat, nicht mutieren. Diese Eigenschaft, die impliziert, dass eine Kette führender Einsen erhalten bleibt, macht es leicht, einen Zustand anzugeben, von dem aus startend eine Mutationswahrscheinlichkeit substanziell größer als $1/n$ ungünstig ist.

Satz 3.8. *Es gibt einen Zustand x^*, so dass der $(1+1)$ EA mit Mutationswahrscheinlichkeit $p(n) = \alpha(n)/n$, wobei $\lim_{n\to\infty} \alpha(n) \to \infty$ gilt, wenn er in x^* gestartet wird, im Erwartungswert $\Omega(\alpha(n) n \log n)$ Schritte bis zum Erreichen des globalen Maximums $\mathbf{1}$ der Funktion $f_B \colon \{0,1\}^n \to \mathbb{R}$ braucht.*

Beweis. Wir nehmen ohne Beschränkung der Allgemeinheit an, dass n gerade ist und wählen als Startzustand $x^* := 1^{n/2}0^{n/2}$. Weil $n/2$ Bits den Wert Null haben, muss jedes dieser Bits mindestens einmal seinen Wert ändern. Dazu ist es erforderlich, dass ein solches Bit mutiert in einem Schritt, in dem der neue Bitstring y aufgrund seines Funktionswertes als neuer aktueller String x akzeptiert wird. Die Wahrscheinlichkeit, nach t derartigen Schritten, die wir hier kurz erfolgreich nennen, noch mindestens eines unter diesen $n/2$ Bits zu haben, das seinen Wert noch gar nicht geändert hat, beträgt

$$1 - \left(1 - \left(1 - \frac{\alpha(n)}{n}\right)^t\right)^{n/2}$$

analog zum Beweis von Satz 3.4. Wir setzen $t = ((n/\alpha(n)) - 1) \ln n$ und haben dann

$$1 - \left(1 - \left(1 - \frac{\alpha(n)}{n}\right)^{((n/\alpha(n))-1)\ln n}\right)^{n/2} \geq 1 - e^{-1/2}$$

als untere Schranke für die Wahrscheinlichkeit, dass nach $((n/\alpha(n)) - 1) \ln n$ erfolgreichen Schritten das globale Maximum $\mathbf{1}$ von f_B noch nicht erreicht ist. Wir haben uns schon überlegt, dass es für einen erfolgreichen Schritt bei der

Funktion f_{B} erforderlich ist, dass von den Bits, die zu einer ununterbrochenen Kette führender Einsen gehören, keines mutiert. Weil wir mit $n/2$ führenden Einsen starten, haben wir

$$\left(1 - \frac{\alpha(n)}{n}\right)^{n/2} = e^{-\Omega(\alpha(n))}$$

als untere Schranke für die Wahrscheinlichkeit, dass ein Schritt erfolgreich ist. Das gibt insgesamt

$$e^{\Omega(\alpha(n))} \cdot \left(\frac{n \ln n}{\alpha(n)} - \ln n\right) = \Omega\left(\alpha(n) n \ln n\right)$$

als untere Schranke für die erwartete Anzahl Schritte, die der (1+1) EA mit Mutationswahrscheinlichkeit $p(n) = \alpha(n)/n$ zum Optimum der Funktion f_{B} braucht. $\square$

Wir haben also gute Gründe, $p(n) = \Theta(1/n)$ für eine gute Wahl für die Mutationswahrscheinlichkeit zu halten. Tatsächlich ist $1/n$ wie erwähnt die am häufigsten empfohlene Mutationswahrscheinlichkeit (Mühlenbein 1992; Bäck 1993). Auch Rudolph (1997a) betrachtet bei seinen ausführlichen Untersuchungen des (1+1) EA keine andere Mutationswahrscheinlichkeit. Mitunter findet man sogar die Vermutung, dass $1/n$ eine immer optimale Wahl für die Mutationswahrscheinlichkeit ist. Das ist natürlich eine regelrecht waghalsige Verallgemeinerung, die wir im Kapitel 7 näher untersuchen wollen. Wir werden dort anhand einer geeigneten Beispielfunktion einsehen, dass in manchen Situationen essenziell andere Wahlen für die Mutationswahrscheinlichkeit substanziell besser sein können. Bis dahin wollen wir uns aber an der gängigen Praxis orientieren und den (1+1) EA stets (stillschweigend) mit der Wahl $p(n) = 1/n$ betrachten. Wir werden speziell im nächsten Kapitel, in dem wir die Klasse der linearen Funktionen untersuchen, noch einmal sehen, dass $p(n) = 1/n$ eine vernünftige Wahl für die Mutationswahrscheinlichkeit ist.

4 Lineare Funktionen

Wir haben uns in Kapitel 2 überlegt, dass weitreichender und hilfreicher als die Klassifikation einzelner Funktionen (eigentlich Funktionsfamilien) die Charakterisierung ganzer Funktionenklassen ist. Ganz im Sinne einer solchen deskriptiven Klassifikation betrachten wir hier lineare Funktionen. Sie sind uns schon in Kapitel 2 als $N0$-Funktionen begegnet, wir wollen hier aber trotzdem noch eine formale Definition angeben.

Definition 4.1. *Eine Funktion $f\colon \{0,1\}^n \to \mathbb{R}$ heißt* linear*, wenn es Gewichte $w_0, w_1, \ldots, w_n \in \mathbb{R}$ gibt, so dass*

$$f(x) = w_0 + \sum_{1 \leq i \leq n} w_i \cdot x_i$$

für alle $x \in \{0,1\}^n$ gilt.

Man überlegt sich leicht, dass die Gewichte $w_0, \ldots, w_n$ eindeutig sind. Verschiedene Gewichte gehören notwendig zu verschiedenen linearen Funktionen. Wir haben in Kapitel 3 schon zwei verschiedene lineare Funktionen kennengelernt und ausführlich untersucht. Für die Funktion f_1 haben wir $w_0 = 0$ und $w_i = 1$ für alle anderen Werte von i. Für die Funktion f_{B} haben wir ebenfalls $w_0 = 0$ und $w_i = 2^{n-i}$ für die übrigen Werte von i. In gewisser Weise sind f_1 und f_{B} zwei extrem verschiedene lineare Funktionen, wie wir uns schon überlegt haben. Während bei f_1 alle Bits gleichermaßen wichtig sind bei der Entscheidung, ob eine Mutation erfolgreich ist, ist bei f_{B} jedes Bit alleine wichtiger als alle rechts von ihm stehenden Bits zusammen. Darum ist das Resultat, dass der (1+1) EA auf beiden Funktionen im wesentlichen gleiche erwartete Laufzeit hat, überraschend. Es gibt Anlass zu Spekulationen, dass vielleicht sogar alle linearen Funktionen in Zeit $O(n \log n)$ vom (1+1) EA optimiert werden können (Mühlenbein 1992). Tatsächlich werden wir das hier nachweisen (Droste, Jansen und Wegener 1998d).

Wir hatten schon gesagt, wann eine Funktion f von einer Variablen x_i essenziell abhängt. Ist f linear, so gilt, dass f genau dann von x_i essenziell abhängt, wenn das Gewicht w_i von 0 verschieden ist. Es ist eine typische und übliche Forderung an Funktionen, dass sie von allen Variablen essenziell abhängen. Wir nennen solche Funktionen vollständig nicht-trivial.

In Kapitel 2 hatten wir von einer sinnvollen deskriptiven Klassifikation gefordert, dass die von ihr in einer Klasse zusammengefassten Funktionen vergleichbare Schwierigkeit in Bezug auf ihre Optimierbarkeit haben, wobei sich die Schwierigkeit durchaus auf ein bestimmtes Optimierverfahren beziehen kann. Wir wollen hier zeigen, dass das auf die Klasse der linearen Funktionen in Bezug auf den (1+1) EA zutrifft. Dazu klären wir vorher formal, was wir eigentlich meinen, wenn wir einem Algorithmus auf einer ganzen Klasse von Funktionen eine gewisse Laufzeit zuschreiben.

Definition 4.2. *Ein Optimieralgorithmus A hat auf einer Klasse von Funktionen F eine (erwartete) Laufzeit $\Omega(t)$, wenn für alle Funktionen $f \in F$ gilt, dass A zur Optimierung von f (im erwarteten Fall) $\Omega(t)$ Schritte braucht. A hat auf F eine (erwartete) Laufzeit $O(t)$, wenn für alle Funktionen $f \in F$ gilt, dass A zur Optimierung von f (im erwarteten Fall) $O(t)$ Schritte braucht. Der Algorithmus A hat auf F eine (erwartete) Laufzeit $\Theta(t)$, wenn er (erwartete) Laufzeit $\Omega(t)$ und (erwartete) Laufzeit $O(t)$ auf F hat.*

Wie bei jeder Definition gibt es auch hier Wahlfreiheiten, die auf verschiedene Weisen genutzt werden können. Eine denkbare alternative Definition für $\Omega(t)$ zu einer Funktionenklasse F wäre zu sagen, dass es genügt, wenn es mindestens eine Funktion $f \in F$ gibt, so dass A im erwarteten Fall $\Omega(t)$ Schritte braucht. Die Festlegung, wie wir sie hier getroffen haben, erfordert eine größere Homogenität von Funktionenklassen. Es ist also auf diese Art prinzipiell schwieriger zu zeigen, dass Algorithmus A erwartete Laufzeit $\Theta(t)$ auf F hat, zum Ausgleich dafür ist die Aussagekraft dann größer. Mit unserer Festlegung können wir jetzt jedenfalls leicht das Ergebnis dieses Kapitels formulieren.

Satz 4.3. *Der $(1+1)$ EA mit $p(n) = 1/n$ hat auf der Klasse der linearen Funktionen erwartete Laufzeit $O(n \log n)$. Auf der Klasse der vollständig nicht-trivialen linearen Funktionen hat der $(1+1)$ EA mit $p(n) = 1/n$ erwartete Laufzeit $\Theta(n \log n)$.*

Beweis. Wir beginnen mit der leicht zu beweisenden unteren Schranke für den zweiten Teil der Aussage. Wenn eine lineare Funktion f vollständig nicht-trivial ist, gibt es offenbar genau einen Punkt $x^* \in \{0,1\}^n$, der maximalen Funktionswert hat. Für diesen Punkt x^* gilt $x_i^* = 0$, wenn $w_i < 0$ gilt und $x_i^* = 1$, wenn $w_i > 0$ gilt. Nach zufälliger Initialisierung haben mit Wahrscheinlichkeit mindestens $1/2$ mindestens $\lfloor n/2 \rfloor$ Bits des aktuellen Strings den falschen Wert und müssen wenigstens einmal mutieren. Analog zum Beweis von Satz 3.4 folgt daraus die untere Schranke $\Omega(n \log n)$ für die erwartete Laufzeit.

Für den Beweis der oberen Schranke machen wir ohne Beschränkung der Allgemeinheit einige vereinfachende Annahmen. Offenbar können Bits mit

Gewicht 0 die Optimierung nicht erschweren. Wir können also $w_i \neq 0$ für alle $i \in \{1, \dots, n\}$ voraussetzen. Weil w_0 für den (1+1) EA gar keine Rolle spielt, nehmen wir $w_0 = 0$ an. Es ist auch klar, dass der (1+1) EA Bits mit Wert Null und Eins auf gleiche Weise behandelt. Wir können darum Gewichte $w_i < 0$ durch $-w_i$ ersetzen und dann die Rollen von Nullen und Einsen an diesen Positionen vertauschen. Wir nehmen darum $w_i > 0$ für alle $i \in \{1, \dots, n\}$ an. Mit diesen Annahmen ist jetzt $\mathbf{1}$ der eindeutige Maximalpunkt und ein Bit hat genau dann „den richtigen Wert“, wenn es den Wert Eins hat. Weil alle Positionen im aktuellen String völlig gleichartig behandelt werden, können wir ohne Beschränkung der Allgemeinheit annehmen, dass $w_1 \geq w_2 \geq \cdots \geq w_n$ gilt. Damit haben wir die Bits genau wie bei der Funktion f_{B} von links nach rechts nach sinkender Wichtigkeit sortiert. Schließlich nehmen wir noch an, dass n gerade ist.

Bei der Funktion f_1 haben wir nur die Anzahl der Bits betrachtet, die schon den richtigen Wert haben. Diese Anzahl kann bei f_1 niemals sinken. Bei der Funktion f_{B} trifft das nicht zu. Dort haben wir die linke und rechte Hälfte separat betrachtet. Wesentlich dabei war, dass die linke Hälfte sich nicht mehr ändern kann, wenn erst einmal alle Bits der linken Hälfte den richtigen Wert haben. Das trifft für viele lineare Funktionen, insbesondere auch für f_1, nicht zu. Offenbar sind f_1 und f_{B} in gewissem Sinne zwei extreme lineare Funktionen. Wir wollen Ideen aus den Beweisen zu Satz 3.4 und Satz 3.7 verbinden, um einen Beweis für alle linearen Funktionen zu erhalten.

Sei f jetzt also die zu optimierende lineare Funktion mit

$$f(x) = \sum_{1 \leq i \leq n} w_i x_i$$

für alle $x \in \{0,1\}^n$, wobei $w_1 \geq w_2 \geq \cdots \geq w_n > 0$ gilt. Wir betrachten einen Lauf des (1+1) EA auf f und untersuchen dabei den Werteverlauf einer zweiten linearen Funktion $g\colon \{0,1\}^n \to \mathbb{R}$, mit

$$g(x) = \sum_{1 \leq i \leq n/2} 2x_i + \sum_{n/2 < i \leq n} x_i$$

für alle $x \in \{0,1\}^n$. Offenbar nimmt g genau wie f den eindeutigen Maximalwert für $x = \mathbf{1}$ an, es ist $g(\mathbf{1}) = 3n/2$. Wir nennen eine Generation erfolgreich, wenn $f(y) \geq f(x)$ und $x \neq y$ gilt. In diesen Generationen ist das Kind y von seinem Elter x echt verschieden (es mutierte also wenigstens ein Bit) und auf Grund der Funktionswerte wird y neuer aktueller Punkt und ersetzt den alten Wert von x. Die Differenz $g(y) - g(x)$ kann in solchen Generationen auch negative Werte annehmen. Weil aber $f(y) - f(x)$ nicht negativ ist, gibt es mindestens ein Bit in y mit Wert Eins, das in x den Wert Null hat.

Nehmen wir für den Moment an, dass für jeden aktuellen String $x \neq \mathbf{1}$ die erwartete Anzahl erfolgreicher Generationen, bis erstmalig ein String x' mit $g(x') > g(x)$ aktueller String wird, durch eine Konstante c^* nach oben beschränkt ist (unabhängig von x). Dann können wir leicht die behauptete obere Schranke nachweisen. Sei $g(x) = i < 3n/2$. Dann enthält x mindestens $z_i = \lfloor 3n/4 - i/2 \rfloor$ Bits mit Wert Null. Folglich gibt es mindestens z_i verschiedene Mutationen von genau einem Bit, die zu erfolgreichen Generationen gehören und zu größeren Funktionswerten als $g(x)$ führen. Folglich beträgt die Wahrscheinlichkeit für das Vorkommen einer solchen Mutation mindestens $z_i(1/n)(1-1/n)^{n-1} \geq z_i/(en)$. Also erhalten wir

$$\sum_{0 \leq i < 3n/2} c^* \cdot \frac{en}{z_i} \leq c^* \cdot en \cdot 2 \cdot \sum_{1 \leq j \leq \lceil 3n/4 \rceil} \frac{1}{j} = O\left(n \log n\right)$$

als obere Schranke für die erwartete Laufzeit des (1+1) EA auf f. Wir wollen jetzt also tatsächlich eine geeignete Konstante c^* als obere Schranke nachweisen.

Sei x der aktuelle String, natürlich $x \neq \mathbf{1}$. Sei y der in einer erfolgreichen Generation erzeugte String. Sei $D(x) = g(y) - g(x)$ die uns interessierende Zufallsvariable, welche die Differenz in g beschreibt, die durch diese erfolgreiche Generation bewirkt wird. Ähnlich wie im Beweis zu Satz 3.7 wollen wir eine Zufallsvariable $D^*(x)$ beschreiben, die nur ganzzahlige Werte annimmt, durch 1 nach oben beschränkt ist, für die

$$\mathrm{Prob}\left(D^*(x) \leq r\right) \geq \mathrm{Prob}\left(D(x) \leq r\right)$$

für alle Werte von r gilt und für die wir $\mathrm{E}\left(D^*(x)\right) \geq c^*$ nachweisen können, wobei c^* eine positive Konstante ist, die nicht von x abhängt.

Wir machen die konkrete Definition von D^* davon abhängig, wie groß das Minimum der Indizes der Bits ist, die in y den Wert Eins und in x den Wert Null haben. Sei l dieses Minimum und also auch die Position des am weitesten links im String befindlichen Bits mit dieser Eigenschaft. Weil wir nur erfolgreiche Generationen betrachten, gibt es so ein Bit sicher und l ist wohl definiert. Wir führen eine Fallunterscheidung nach dem Wert von l durch. Sei jetzt also zunächst dieser minimale Index durch $n/2$ nach oben beschränkt, also $l \leq n/2$. Es gibt folglich in der linken Hälfte von x ein Bit mit dem Wert Null, das mutiert und darum in y den Wert Eins hat. Wir definieren $D^*(x)$ als Summe von Beiträgen einzelner Bits, die in x und y

verschiedenen Wert haben. Im Gegensatz zur tatsächlichen Differenz $D(x)$ berücksichtigen wir aber nicht alle mutierten Bits. Das Bit x_l trägt 2 zu $D^*(x)$ bei. Jedes Bit x_i der linken Hälfte ($i \leq n/2$), das seinen Wert von Eins in x auf Null in y ändert, trägt -2 zu $D^*(x)$ bei. Jedes Bit x_i der rechten Hälfte ($i > n/2$), das seinen Wert von Eins in x auf Null in y ändert, trägt -1 zu $D^*(x)$ bei. Falls es gar kein Bit mit negativem Beitrag gibt, ziehen wir noch 1 von $D^*(x)$ ab. Insgesamt haben wir also

$$\begin{aligned} D^*(x) = \min \{1, 2 - 2\,&|\{i \leq n/2 \mid x_i = 1 \wedge y_i = 0\}| \\ &- |\{n/2 < i \leq n \mid x_i = 1 \wedge y_i = 0\}|\} \end{aligned}$$

als Definition von $D^*(x)$ in diesem Fall. Offenbar ist deshalb und weil wir den positiven Beitrag der Bits, die ihren Wert von Null in x auf Eins in y ändern, ignorieren, stets

$$\text{Prob}\,(D^*(x) \leq r) \leq \text{Prob}\,(D(x) \leq r)\,,$$

wie gefordert. Offenbar ist $D^*(x) \leq 1$, außerdem nimmt $D^*(x)$ nur ganzzahlige Werte an. Wir untersuchen jetzt den Erwartungswert von $D^*(x)$.

Es sei j_2 die Anzahl Bits der linken Hälfte (mit Index höchstens $n/2$), die in x den Wert Eins haben und die in einer erfolgreichen Mutation bei gegebenem Minimalindex l mutieren können. Es sei j_1 die Anzahl Bits der rechten Hälfte (mit Index größer als $n/2$), die in x den Wert Eins haben. Es bezeichnen also j_2 und j_1 die Anzahl Bits, die Beitrag -2 bzw. -1 zu $D^*(x)$ liefern können. Unter Vernachlässigung aller Bedingungen ist die erwartete Anzahl mutierender Bits unter den so beschriebenen Bits natürlich $(j_2 + j_1)/n$. Die Bedingung, dass die betrachtete Generation erfolgreich ist, kann die Anzahl dieser Bits, die tatsächlich mutieren, nur verringern. Wir können also

$$\frac{-(2j_2 + j_1)}{n}$$

als Abschätzung für den Beitrag dieser Bits verwenden. Falls von diesen Bits gar keines mutiert, haben wir noch den zusätzlichen Beitrag -1, der sicherstellt, dass stets $D^*(x) \leq 1$ gilt. Wir müssen folglich die bedingte Wahrscheinlichkeit betrachten, dass gar kein Bit seinen Wert von Eins in x nach Null in y ändert. Dazu müssen wir nur die schon beschriebenen $j_2 + j_1$ Bits betrachten. Es bezeichne A das Ereignis, dass kein Bit seinen Wert von Eins nach Null wechselt. Es bezeichne B das Ereignis, dass das mutierte Kind y akzeptiert wird, die Generation also erfolgreich ist. Wir suchen dann eine obere Schranke für $\text{Prob}\,(A \mid B)$. Es ist natürlich

$$\text{Prob}\,(A \mid B) = \frac{\text{Prob}\,(A \cap B)}{\text{Prob}\,(B)}$$

und weil $A \subseteq B$ gilt, haben wir

$$\text{Prob}\,(A \mid B) = \frac{\text{Prob}\,(A)}{\text{Prob}\,(B)}$$

und brauchen eine obere Schranke für Prob (A) sowie eine untere Schranke für Prob (B). Die Wahrscheinlichkeit, dass keines der j_2+j_1 Bits mutiert beträgt offensichtlich genau $(1-1/n)^{j_2+j_1}$, wir kennen also die Wahrscheinlichkeit von A exakt. Damit eine Generation erfolgreich ist, genügt es jedenfalls, dass höchstens eines der j_2+j_1 Bits mutiert, wir haben also

$$\text{Prob}\,(B) \geq \left(1-\frac{1}{n}\right)^{j_2+j_1} + (j_2+j_1)\,\frac{1}{n}\left(1-\frac{1}{n}\right)^{j_2+j_1-1}.$$

Damit erhalten wir

$$\begin{aligned} & \frac{(1-1/n)^{j_2+j_1}}{(1-1/n)^{j_2+j_1}+(j_2+j_1)(1/n)(1-1/n)^{j_2+j_1-1}} \\ \leq\; & \frac{1-1/n}{1-1/n+(j_2+j_1)/n} \leq \frac{1}{1+(j_2+j_1)/n} \end{aligned}$$

als obere Schranke für Prob $(A \mid B)$. Folglich haben wir

$$\text{E}\,(D^*(x)) \geq 2 - \frac{2j_2+j_1}{n} - \frac{1}{1+(j_2+j_1)/n}.$$

Wir untersuchen einige Fälle. Gilt $j_2+j_1 \leq n/4$, so haben wir $2j_2+j_1 \leq n/2$, also

$$\text{E}\,(D^*(x)) \geq 2 - \frac{1}{2} - \frac{1}{1+0/n} = \frac{1}{2}.$$

Gilt andererseits $n/4 < j_2+j_1 \leq n/2$, so haben wir $2j_2+j_1 \leq n$, also

$$\text{E}\,(D^*(x)) > 2 - 1 - \frac{1}{1+1/4} = \frac{1}{5}.$$

Es bleibt jetzt noch der Fall $j_2+j_1 > n/2$. Man sieht leicht ein, dass der Ausdruck für $\text{E}\,(D^*(x))$ minimal wird, wenn j_2 möglichst groß wird. Man beachte, dass $j_2 < n/2$ gemäß Definition gilt: zur ersten Hälfte gehören nur $n/2$ Bits, von denen mindestens eines seinen Wert von Null nach Eins wechselt, also nicht zu den j_2 Bits gehört. Wir setzen (zu pessimistisch) $j_2 = n/2$, haben

$$\text{E}\,(D^*(x)) \geq 1 - \frac{j_1}{n} - \frac{1}{(3/2)+(j_1/n)}$$

und unterscheiden jetzt weiter nach j_1. Ist $j_1 \leq (1-\varepsilon)n/3$ für eine Konstante $\varepsilon > 0$, so haben wir

$$\mathrm{E}\left(D^*(x)\right) \geq 1 - \frac{1-\varepsilon}{3} - \frac{1}{3/2} = \frac{\varepsilon}{3}.$$

Es bleibt jetzt noch der Fall $j_1 > (1-\varepsilon)n/3$, für den wir unsere Abschätzungen noch verbessern müssen. Wir betrachten jetzt alle

$$v := \binom{j_2 + j_1}{2}$$

Paare von Bits, so dass ein Bit zu den j_2 Bits der ersten und das andere zu den j_1 Bits der zweiten Hälfte gehört. Sei $v' \leq v$ die Anzahl solcher Paare, die gleichzeitig ihren Wert von Eins nach Null ändern können, ohne dass die Mutation erfolglos wird. Ist $v' \geq v/2$, so können wir den Term

$$\frac{v}{2} \cdot \frac{1}{n^2} \cdot \left(1 - \frac{1}{n}\right)^{j_2+j_1-2}$$

zur unteren Schranke für $\mathrm{Prob}\,(B)$, als der Wahrscheinlichkeit für eine erfolgreiche Generation, addieren. Dieser Ausdruck kann durch eine positive Konstante nach unten beschränkt werden. Wir erinnern uns, dass wir den Fall $j_1 > (1-\varepsilon)n/3$ untersuchen und

$$\mathrm{E}\left(D^*(x)\right) \geq 1 - \frac{j_1}{n} - \mathrm{Prob}\,(A \mid B)$$

betrachten. Mit j_1 bezeichnen wir die für uns kritischen Bits der zweiten Hälfte, es gilt also $j_1/n \leq 1/2$. Wir haben die untere Schranke für $\mathrm{Prob}\,(B)$ um eine positive Konstante vergrößern können, das gibt uns $\mathrm{Prob}\,(A \mid B) \leq (1/2) - \delta$ und insgesamt

$$\mathrm{E}\left(D^*(x)\right) \geq 1 - \frac{1}{2} - \left(\frac{1}{2} - \delta\right) = \delta$$

für eine Konstante $\delta > 0$.

Betrachten wir nun noch den Fall $v' < v/2$. Für $\mathrm{Prob}\,(A \mid B)$ verwenden wir jetzt wieder die triviale Abschätzung $1/2$, also $j_1 = n/2$. Das gibt zunächst nur $\mathrm{E}\left(D^*(x)\right) \geq 0$, wir können jetzt aber den Term verbessern, der durch die von Eins nach Null mutierenden Bits beigesteuert wird. Bisher haben wir diesen Term mit $-(2j_2 + j_1)/n$ angesetzt. Wir betrachten nun aber den Fall, dass von allen Paaren aus je einem für uns kritischem Bit der ersten und zweiten Hälfte (die wir mit j_2 bzw. j_1 zählen), weniger als die Hälfte in einer erfolgreichen Generation gemeinsam mutieren können. Es gibt also eine Konstante $\alpha > 0$, so dass wenigstens αn^2 solcher Paare nicht gemeinsam mutieren können in einer erfolgreichen Generation. Die Wahrscheinlichkeit, dass unter diesen Paaren genau eines gemeinsam mutiert, kann also auch durch eine Konstante $\alpha' > 0$ nach unten abgeschätzt werden. Das gibt für eine Konstante $\alpha'' > 0$

$$-\frac{2j_2 + j_1}{n} + \alpha'' \geq -\frac{3}{2} + \alpha''$$

als untere Schranke für den Beitrag mutierender Bits zu $\mathrm{E}\left(D^*(x)\right)$ in diesem Fall. Wir haben also $\mathrm{E}\left(D^*(x)\right) \geq \alpha''$ und somit in allen bisher betrachteten Fällen $\mathrm{E}\left(D^*(x)\right) \geq \gamma$ für eine positive Konstante γ.

Wir haben jetzt den Fall, dass in der linken Hälfte wenigstens ein Bit von Null zu Eins mutiert, vollständig abgeschlossen. Nehmen wir also jetzt an, dass alle Bits, die im Elter x den Wert Null und im Kind y den Wert Eins haben, zur rechten Hälfte gehören, der Index l also größer als $n/2$ ist. Wir definieren in diesem Fall $D^*(x)$ viel ähnlicher zur tatsächlichen Differenz $g(y) - g(x)$. Das macht die Analyse schwieriger als im vorher beschriebenen Fall, in dem wir recht grob abgeschätzt haben. Konkret definieren wir in diesem Fall einfach

$$D^*(x) = \min\{1, g(y) - g(x)\}$$

und sehen sofort, dass $D^*(x)$ nur ganzzahlige Werte annimmt und durch 1 nach oben beschränkt ist. Außerdem gilt natürlich $D^*(x) \leq D(x)$, da ja $D(x) = g(y) - g(x)$ ist. Wir betrachten $\mathrm{E}\left(D^*(x)\right)$ und wollen zeigen, dass dieser Erwartungswert durch eine positive Konstante nach unten beschränkt ist. Es bezeichne s die Anzahl Bits, die von Null nach Eins mutieren. Wir approximieren von nun an durch Binomialverteilung gegebenen Wahrscheinlichkeiten durch die Poissonverteilung, was für hinreichend große n keine Schwierigkeiten bereitet: Es ist bekannt, dass die Binomialverteilung mit Parametern n und $p(n)$ unter der Voraussetzung, dass $n \cdot p(n) \in \mathbb{R}$ konstant ist für wachsendes n, gegen die Poissonverteilung mit Parameter $\lambda = n \cdot p(n)$ konvergiert. Hier haben wir binomialverteilte Zufallsvariablen mit Parameter $n/2$ und $1/n$, verwenden folglich $\lambda = 1/2$. Für hinreichend großes n wird der Betrag der Differenz zwischen der binomialverteilten Zufallsvariablen und ihrer poissonverteilten Approximation kleiner als ε, wobei $\varepsilon > 0$ eine beliebig kleine Konstante ist. Wir können also sicher sein, dass für hinreichend große Werte von n der Fehler, den wir durch diese Approximation machen, keine Rolle spielt.

Ist $s \geq 4$, so ist die Lage verhältnismäßig einfach. Wir betrachten die Situation unabhängig von der Bedingung, dass die Generation erfolgreich ist. Das kann den für uns kritischen Beitrag von Bits, die von Eins nach Null mutieren, nur vergrößern. Wir nehmen (ebenfalls pessimistisch) an, dass alle Bits, die nicht von Null nach Eins mutieren, im Elter den Wert Eins haben und vergrößern so potentiell den negativen Beitrag. Die Wahrscheinlichkeit, dass genau j_2 Bits der linken Hälfte (mit Index höchstens $n/2$) und genau j_1 Bits der rechten Hälfte (mit Index größer $n/2$) gleichzeitig mutieren beträgt

$$\left(\frac{1}{j_2!}\left(\frac{1}{2}\right)^{j_2}\frac{1}{\sqrt{e}}\right)\cdot\left(\frac{1}{j_1!}\left(\frac{1}{2}\right)^{j_1}\frac{1}{\sqrt{e}}\right)=\frac{1}{j_1!\cdot j_2!}\cdot\left(\frac{1}{2}\right)^{j_2+j_1}\cdot\frac{1}{e}.$$

Wir haben also

$$\mathrm{E}\left(D^*(x)\right)\geq\sum_{j_2\geq 0, j_1\geq 0}\min\left\{1, 4-2j_2-j_1\right\}\cdot\frac{1}{j_1!j_2!}\left(\frac{1}{2}\right)^{j_2+j_1}\frac{1}{e}$$

und wollen eine positive Konstante als untere Schranke für $\mathrm{E}\left(D^*(x)\right)$ nachweisen. Für $(j_2, j_1)=(0,0)$ haben wir den Beitrag e^{-1}. Für $j_2=0$ und $j_1\leq 4$ ist der Beitrag sicher nicht negativ, das gilt auch für $j_2=1$ und $j_1\leq 2$. Für $j_2=0$ und $j_1\geq 5$ haben wir

$$\sum_{j_1\geq 5}(4-j_1)\cdot\frac{1}{j_1!}\left(\frac{1}{2}\right)^{j_1}\frac{1}{e}\geq-\frac{1}{e}\cdot\frac{1}{5!\cdot 2^4}\cdot\sum_{j_1\geq 1}2^{-j_1}\geq-\frac{1}{1920e}$$

als Beitrag. Für $j_2=1$ und $j_1\geq 3$ ergibt sich

$$\sum_{j_1\geq 3}(2-j_1)\cdot\frac{1}{j_1!}\left(\frac{1}{2}\right)^{j_1}\frac{1}{e}\geq-\frac{1}{e}\cdot\frac{1}{3!\cdot 2^3}\cdot\sum_{j_1\geq 1}2^{-j_1}\geq-\frac{1}{48e}$$

als Beitrag und für $j_2\geq 2$ haben wir den Beitrag

$$\sum_{j_2\geq 2, j_1\geq 0}(4-2j_2-j_1)\frac{1}{j_2!j_1!}\left(\frac{1}{2}\right)^{j_2+j_1}\frac{1}{e}\geq 2\sum_{j_2\geq 2}(3-2j_2)\frac{1}{j_2!}\left(\frac{1}{2}\right)^{j_2}\frac{1}{e}\geq-\frac{1}{2e}$$

also insgesamt

$$\mathrm{E}\left(D^*(x)\right)\geq\frac{1}{e}-\frac{1}{1920e}-\frac{1}{48e}-\frac{1}{2e}\geq\frac{1}{3e}.$$

In den Fällen $s\in\{1,2,3\}$ gehen wir weniger grob vor. Es mutiert kein Bit der linken Hälfte von Null nach Eins. Wir nehmen an, dass die Gewichte

$w_1 \geq w_2 \geq \cdots \geq w_n > 0$ sortiert sind. Weil wir nur erfolgreiche Generationen betrachten, haben wir $j_2 \leq s-1$ oder $j_2 = s$ und $j_1 = 0$. Wir betrachten im folgenden nicht die bedingten Wahrscheinlichkeiten, die *alle* um einen konstanten positiven Faktor größer sind, was die Betrachtungen nicht wesentlich ändert und unsere Schranken auch nur verbessert. Für $s = 1$ haben wir

$$\mathrm{E}\left(D^*(x)\right) \geq \underbrace{\frac{1}{e}}_{(j_2,j_1)=(0,0)} - \underbrace{\frac{1}{2e}}_{(j_2,j_1)=(1,0)} + \sum_{j_1 \geq 0} (1-j_1) \frac{1}{j_1!} \left(\frac{1}{2}\right)^{j_1} \frac{1}{e} \geq \frac{1}{4e}.$$

Für $s = 2$ haben wir zunächst

$$\underbrace{\frac{1}{e}}_{(j_2,j_1)=(0,0)} + \underbrace{0}_{(j_2,j_1)=(1,0)} - \underbrace{\frac{1}{4e}}_{(j_2,j_1)=(2,0)} = \frac{3}{4e}.$$

Negativen Beitrag können noch die Fälle $(1, j_1)$ und $(0, j_1)$ haben, dann ist aber auf jeden Fall auch $(j_2, j_1) = (0, 1)$ erfolgreich. Das gibt dann insgesamt

$$\begin{aligned}
\mathrm{E}\left(D^*(x)\right) &\geq \frac{3}{4e} + \underbrace{\frac{1}{2e}}_{(j_2,j_1)=(0,1)} + \sum_{j_1 \geq 2} (2-j_1) \frac{1}{j_1!} \left(\frac{1}{2}\right)^{j_1} \frac{1}{e} - \sum_{j_1 \geq 0} j_1 \frac{1}{j_1!} \left(\frac{1}{2}\right)^{j_1} \frac{1}{e} \\
&\geq \frac{5}{4e} - \frac{1}{24e} - \frac{11}{12e} = \frac{7}{24e}.
\end{aligned}$$

Für $s = 3$ schließlich können wir die Fälle $j_2 = 0$ und $j_1 \in \{1, 2\}$ ignorieren, weil sie positiven Beitrag haben. Für $(j_2, j_1) = (0, 0)$ haben wir positiven Beitrag e^{-1}, für $(j_2, j_1) = (1, 0)$ positiven Beitrag $e^{-1/2}$. Für $(j_2, j_1) = (3, 0)$ haben wir negativen Beitrag $-e^{-1}/16$. Es bleiben noch die Fälle $j_2 = 0$ und $j_1 \geq 3$ mit Beitrag

$$\sum_{j_1 \geq 3} (3-j_1) \frac{1}{j_1!} \left(\frac{1}{2}\right)^{j_1} \frac{1}{e} \geq -\frac{1}{24e},$$

$j_2 = 1$ und $j_1 \geq 2$ mit Beitrag

$$\sum_{j_1 \geq 2} (1-j_1) \frac{1}{j_1!} \left(\frac{1}{2}\right)^{1+j_1} \frac{1}{e} \geq -\frac{1}{4e},$$

sowie $j_2 = 2$ und $j_1 \geq 0$ mit

$$-\frac{1}{2} \sum_{j_1 \geq 0} (1+j_1) \frac{1}{j_1!} \left(\frac{1}{2}\right)^{2+j_1} \frac{1}{e} \geq -\frac{1}{4e},$$

so dass wir auch hier insgesamt wie in allen Fällen $\mathrm{E}(D^*(x)) \geq d^*$ für eine positive Konstante d^* haben. Wir behaupten, dass die erwartete Anzahl Generationen, bis der (1+1) EA auf der Funktion f ausgehend von einem Punkt $x \in \{0,1\}^n \setminus \{\mathbf{1}\}$ erstmals einen Punkt $x' \in \{0,1\}^n$ mit $g(x') > g(x)$ erreicht, durch eine Konstante c^* nach oben beschränkt ist. Wir nennen diese Anzahl von Generationen $T^*(x)$ und weisen $\mathrm{E}(T^*(x)) \leq 1/d^*$ (also $c^* = 1/d^*$) nach. Dazu betrachten wir einen Random Walk, der durch die Zufallsvariablen $D^*(x)$ definiert ist. Wir starten bei 0 und schätzen die erwartete Anzahl Schritte, bis erstmalig 1 erreicht wird, nach oben ab. Wir führen einen Induktionsbeweis über $g(x)$. Für $g(x) = 0$ ist $x = \mathbf{0}$, der Nullstring, und die Aussage ist trivial. Wir nehmen nun an, dass die Aussage für alle x' mit $g(x') < k$ gilt und betrachten einen String $x \in \{0,1\}^n \setminus \{\mathbf{1}\}$ mit $g(x) = k$, für den $\mathrm{E}(T^*(x))$ maximal ist. Falls $D^*(x) = 1$ ist, sind wir schon nach dem ersten Schritt fertig. Andernfalls haben wir $D^*(x) = d \leq 0$ und gemäß Induktionsannahme brauchen wir im Durchschnitt $-dc^*$ Schritte, bis wir wiederum die 0 erreichen. Folglich gilt

$$\begin{aligned}
\mathrm{E}(T^*(x)) &\leq 1 + \mathrm{Prob}(D^*(x) \neq 1) \cdot \mathrm{E}(T^*(x)) - \sum_{d \leq 0} \mathrm{Prob}(D^*(x) = d)\, dc^* \\
&= 1 + \mathrm{Prob}(D^*(x) \neq 1) \cdot \mathrm{E}(T^*(x)) - \sum_{d \leq 1} \mathrm{Prob}(D^*(x) = d)\, dc^* \\
&\quad\; + c^* \mathrm{Prob}(D^*(x) = 1) \\
&= 1 + (1 - \mathrm{Prob}(D^*(x) = 1)) \cdot \mathrm{E}(T^*(x)) - \mathrm{E}(D^*(x))\, c^* \\
&\quad\; + c^* \mathrm{Prob}(D^*(x) = 1).
\end{aligned}$$

Daraus folgt

$$\mathrm{E}(T^*(x)) \cdot \mathrm{Prob}(D^*(x) = 1) \leq 1 - \mathrm{E}(D^*(x))\, c^* + c^* \mathrm{Prob}(D^*(x) = 1).$$

Wir haben $\mathrm{E}(D^*(x)) \geq d^*$ und $c^* = 1/d^*$, also $1 - \mathrm{E}(D^*(x))\, c^* \leq 0$. Damit ergibt sich schließlich $\mathrm{E}(T^*(x)) \leq c^*$, da $\mathrm{Prob}(D^*(x) = 1) > 0$ gilt. □

Wir haben jetzt gesehen, dass der (1+1) EA mit der verbreiteten Wahl $p(n) = 1/n$ für die Mutationswahrscheinlichkeit eine ganze Funktionenklasse effizient, konkret in Zeit $O(n \log n)$ optimiert. Natürlich können lineare Funktionen deterministisch in Zeit $O(n)$ optimiert werden. Der (1+1) EA setzt aber nicht voraus, dass die Zielfunktion linear ist, er ist auch nicht für solche Funktionen optimiert. Darum darf der zusätzliche Faktor $\log n$ als akzeptabel und der (1+1) EA als für diese Klasse effiziente Heuristik gewertet werden. Die Wichtigkeit dieses Resultats wird dadurch erhöht, dass viele verbreitete Benchmarkfunktionen linear sind (Salomon 1996). Der Erfolg des (1+1) EA auf linearen Funktionen, die ja bitweise optimierbar sind, kann zu der Annahme verführen, dass auch größere Funktionenklassen, die im gewissen Sinne auch bitweise optimierbar sind, ähnlich effizient optimiert werden können (Mühlenbein 1992). Wir erhellen diese Vermutung im nächsten Kapitel, in dem wir uns mit der Menge der unimodalen Funktionen beschäftigen, die unter anderem alle vollständig nicht-trivialen linearen Funktionen enthält.

Vorher wollen wir uns aber noch mit der Wahrscheinlichkeit beschäftigen, mit der die Laufzeit des (1+1) EA bei der Optimierung linearer Funktionen erheblich vom Erwartungswert $\Theta(n \log n)$ abweicht. Eine triviale obere Schranke liefert die Markov-Ungleichung. Es gibt eine Konstante $c > 0$, so dass die erwartete Laufzeit des (1+1) EA auf allen linearen Funktionen durch $cn \ln n$ nach oben beschränkt ist. Dann gilt für jede lineare Funktion f, dass die Wahrscheinlichkeit, dass der (1+1) EA zum Optimieren von f mehr als $t \cdot cn \ln n$ Generationen braucht, durch $1/t$ nach oben beschränkt ist. Dank Markov-Ungleichung können wir also zum Beispiel sagen, dass die erwartete Laufzeit nur mit Wahrscheinlichkeit höchstens $1/n$ $\Omega\left(n^2 \log n\right)$ beträgt. Tatsächlich kann man aber leicht eine deutlich bessere Schranke angeben.

Satz 4.4. *Die Wahrscheinlichkeit, dass der* $(1+1)$ *EA mit* $p(n) = 1/n$ *zur Optimierung einer linearen Funktion* $f\colon \{0,1\}^n \to \mathbb{R}$ $\Omega\left(n^2 \log n\right)$ *Generationen braucht, ist durch* $e^{-\Omega(n)}$ *nach oben beschränkt.*

Beweis. Wir hatten uns schon überlegt, dass es eine Konstante $c > 0$ gibt, so dass die Wahrscheinlichkeit, dass der (1+1) EA mehr als $2cn \ln n$ Schritte bis zum Erreichen eines globalen Maximums von f braucht, durch $1/2$ nach oben beschränkt ist. Wir erinnern uns, dass wir im Beweis von Satz 4.3 keine Aussagen über den initialen Zustand gemacht haben. Wir haben tatsächlich (pessimistisch und völlig unrealistisch) angenommen, dass anfangs gar kein Bit den richtigen Wert hat. Darum können wir jetzt wie folgt argumentieren.

Wenn nach $2cn \ln n$ Schritten noch kein globales Maximum erreicht ist, stimmt im folgenden Sinn die Situation mit der Anfangssituation überein: Alle Annahmen, die wir im Beweis von Satz 4.3 über den Anfangszustand gemacht haben, sind erfüllt. Das ist trivialerweise richtig, weil wir dort gar nichts voraussetzen. Wir können darum hier gedanklich unsere Betrachtungen nach $2cn \ln n$ Schritten abbrechen und den Algorithmus so betrachten, als sei er neu gestartet worden. Durch iterierte Anwendung dieses Arguments erhalten wir, dass wir einen Lauf des (1+1) EA, der T Schritte lang ist, als

$$\left\lfloor \frac{T}{2cn \ln n} \right\rfloor$$

unabhängige Wiederholungen von Läufen mit je $2cn \ln n$ Generationen betrachten. Die Wahrscheinlichkeit, dass kein globales Maximum gefunden wird, ist in jedem Lauf durch $1/2$ nach oben beschränkt. Insgesamt ist folglich die Wahrscheinlichkeit, nach T Generationen kein globales Optimum gefunden zu haben, durch

$$\left(\frac{1}{2}\right)^{\lfloor T/(2cn \ln n) \rfloor}$$

nach oben beschränkt. Mit $T = (2cn^2 \ln n)/\ln 2$ haben wir e^{-n} als Schranke für die Wahrscheinlichkeit. □

Um die Wahrscheinlichkeit, dass das globale Maximum einer vollständig nicht-trivialen linearen Funktion vom (1+1) EA wesentlich schneller als in $\Theta(n \log n)$ Generationen gefunden wird, abzuschätzen, reichen triviale Abschätzungen nicht aus. Unsere Überlegungen aus dem Beweis von Satz 3.4 für die untere Schranke zur Funktion f_1 liefern für jede Konstante $\varepsilon > 0$ als obere Schranke für die Wahrscheinlichkeit, dass der (1+1) EA eine vollständig nicht-triviale lineare Funktion in höchstens $\varepsilon n \ln n$ Schritten optimiert nur $(1/2) \cdot \left(1 - e^{-n^{1-\varepsilon}/2}\right)$. Ist ε eine Konstante mit $0 < \varepsilon < 1$, so konvergiert diese Schranke gegen $1/2$. Tatsächlich kann man aber die Wahrscheinlichkeit, dass der (1+1) EA eine vollständig nicht-trivial lineare Funktion sehr schnell optimiert, wesentlich genauer abschätzen. Wir zeigen, dass die Wahrscheinlichkeit, dass der (1+1) EA mit substanziell weniger als $(1/2)n \ln n$ Generationen auskommt, klein ist.

Satz 4.5. *Für jede Konstante α mit $\alpha < 1$ und jede Konstante ε mit $\varepsilon > 0$ gilt, dass die Wahrscheinlichkeit, dass der $(1+1)$ EA mit Mutationswahrscheinlichkeit $p(n) = 1/n$ das globale Maximum einer vollständig nicht-trivialen linearen Funktion $f\colon \{0,1\}^n \to \mathbb{R}$ in höchstens*

$$\alpha \cdot n \ln n$$

Generationen findet, im Grenzwert für $n \to \infty$ durch ε nach oben beschränkt ist.

Der Beweis beruht wesentlich auf Überlegungen zum so genannten Sammelkartenproblem (Coupon Collector's Problem), wie man sie bei Motwani und Raghavan (1995) findet. Das Sammelkartenproblem kann auf verschiedene Weisen beschrieben werden. Die für unsere Anwendung anschaulichste spricht gar nicht von Sammelkarten, sondern von Bällen und Eimern. Man betrachtet

m Eimer, in die s Bälle geworfen werden. Man wirft die s Bälle unabhängig voneinander (etwa unabhängig nacheinander), jeder Eimer hat bei jedem Ball die gleiche Wahrscheinlichkeit $1/n$, getroffen zu werden. Wir betrachten die Situation nach s geworfenen Bällen und suchen Abschätzungen für die Wahrscheinlichkeit, noch einen gänzlich leeren Eimer zu finden. Man sieht leicht ein (ähnlich zu unseren Überlegungen im Beweis zu Satz 3.4), dass die erwartete Anzahl Bälle, bis alle Eimer gefüllt sind, gerade $m \ln m + O(m)$ beträgt. Tatsächlich kann man diesen Erwartungswert durch scharfe obere und untere Schranken mit hoher Wahrscheinlichkeit „einrahmen", also die Wahrscheinlichkeit schon für verhältnismäßig kleine Abweichungen durch kleine Schranken abschätzen. In diesem Sinn kann man bei dieser Versuchsanordnung von einem beinahe deterministischen Verhalten sprechen. Wir übernehmen das nächste recht technische Lemma von Motwani und Raghavan (1995) (Satz 3.8, Seite 61) ohne Beweis.

Lemma 4.6. *Für jede Konstante $c \in \mathbb{R}$ gilt, dass die Wahrscheinlichkeit, nach $m \ln m + cm$ geworfenen Bällen noch mindestens einen leeren Eimer zu finden, im Grenzwert für $m \to \infty$ genau $1 - e^{-e^{-c}}$ beträgt.*

Weil Lemma 4.6 für jede Konstante $c \in \mathbb{R}$, also insbesondere auch für Konstante $c < 0$ gilt, kann man direkt auch eine Schranke für eine Unterschreitung des Erwartungswerts ableiten.

Folgerung 4.7. *Für jede Konstante $c \in \mathbb{R}$ gilt, dass die Wahrscheinlichkeit, nach $m \ln m - cm$ geworfenen Bällen keinen leeren Eimer mehr zu finden, im Grenzwert für $m \to \infty$ genau $e^{-e^{c}}$ beträgt.*

Nach diesen Vorbereitungen sind wir nun gut gerüstet, um den noch fehlenden Beweis zu Satz 4.5 nachzuliefern.

Beweis zu Satz 4.5. Sei $f\colon \{0,1\}^n \to \mathbb{R}$ eine vollständig nicht-triviale lineare Funktion. Dann hat f ein eindeutiges globales Maximum. Wir verwenden Chernoff-Schranken und sehen, dass mit Wahrscheinlichkeit mindestens $1 - e^{-n\gamma^2/4}$ im initialen Bitstring mindestens $(1-\gamma)n/2$ Bits einen anderen Wert haben als das entsprechende Bit im optimalen Bitstring. Das gilt für jede Konstante γ mit $0 < \gamma < 1$. Offenbar müssen diese $(1-\gamma)n/2$ Bits alle mindestens einmal ihren Wert ändern, also mutieren, wenn der (1+1) EA das globale Maximum erreichen soll. Wir wollen die Anzahl Generationen, die das dauert, mit Hilfe von Folgerung 4.7 abschätzen. Dazu verwenden wir folgende Modellierung.

In T Generationen finden $n \cdot T$ unabhängige Zufallsexperimente statt, in denen jeweils mit Wahrscheinlichkeit $p(n) = 1/n$ ein Bit mutiert. Grob gesprochen identifizieren wir die Bits mit Eimern und das Mutieren eines Bits mit einem Ball, der den entsprechenden Eimer trifft. Diese Modellierung ist aber nicht ganz zutreffend. Wenn in einer Generation M Bits mutieren, so ist klar, dass auch M unterschiedliche Bits mutieren. Beim Sammelkartenproblem hingegen beträgt die Wahrscheinlichkeit, dass bei M geworfenen Bällen M unterschiedliche Eimer getroffen werden

$$\prod_{i=1}^{M-1} \left(1 - \frac{i}{n}\right),$$

ist also echt kleiner als 1, wenn $M > 1$ gilt. Wir erkennen, dass wir beim Sammelkartenproblem eine größere Wahrscheinlichkeit haben, Bälle auf weniger Eimer zu konzentrieren, also eine größere Wahrscheinlichkeit, nach s geworfenen Bällen, einen leeren Eimer zu finden, als nach s Mutationen ein Bit zu finden, das noch gar nicht mutiert ist. Insofern muss man beim Sammelkartenproblem länger darauf warten, alle Eimer zu treffen als beim (1+1) EA, alle Bits zu mutieren. Wir tragen dem Rechnung, indem wir unsere Modellierung etwas verfeinern. Wir betrachten eine einzelne Generation, in der M Bits mutieren. Weil wir das Mutieren eines Bits mit dem Werfen eines Balls identifizieren, braucht über den Fall $M = 0$ nichts gesagt zu werden. Ist $M = 1$, so stimmt die Situation offenbar exakt mit der Situation beim Sammelkartenspiel überein. Jedes Bit mutiert mit Wahrscheinlichkeit $1/n$ genau wie jeder Eimer von einem Ball mit Wahrscheinlichkeit $1/n$ getroffen wird. Unterschiede gibt es nur für $M > 1$. Beim zweiten mutierenden Bit hat jedes Bit Wahrscheinlichkeit $1/(n-1)$, von dieser Mutation betroffen zu sein, beim dritten mutierenden Bit Wahrscheinlichkeit $1/(n-2)$, beim M-ten mutierenden Bit schließlich Wahrscheinlichkeit $1/(n-(M-1))$. Wir können dem Rechnung tragen, indem wir annehmen, dass solche Bälle nicht mehr auf n sondern nur noch auf $n-i$ Eimer geworfen werden. Unsere Strategie ist jetzt die folgende. Wir überlegen uns in einem ersten Schritt, dass in einer Generation mit großer Wahrscheinlichkeit nicht allzu viele Bits gleichzeitig mutieren, M also nicht allzu groß werden kann. Nehmen wir an, dass $m_{\max}$ eine obere Schranke ist. Das garantiert uns, dass $1/(n-(M-1))$ nicht zu groß wird. Dann nehmen wir vereinfachend (und für uns ungünstigerweise) an, dass bei jedem geworfenen Ball jeder Eimer mit Wahrscheinlichkeit $1/(n-m_{\max})$ getroffen wird. Wir bestimmen jetzt zunächst einen geeigneten Wert für $m_{\max}$.

Die Wahrscheinlichkeit, dass in einer Generation mindestens $\ln n$ Bits gleichzeitig mutieren, ist nach oben durch

$$\binom{n}{\ln n}\left(\frac{1}{n}\right)^{\ln n} \leq \frac{1}{(\ln n)!} \leq \frac{e^{\ln n}}{(\ln n)^{\ln n}} = e^{\ln n-(\ln n)\ln\ln n}$$

beschränkt. Die Wahrscheinlichkeit, dass in $n^{O(1)}$ Generationen eine Mutation von mindestens $\ln n$ Bits gleichzeitig stattfindet, ist folglich ebenfalls durch $e^{-\Omega((\ln n)\ln\ln n)}$ nach oben beschränkt.

Wir haben uns schon überlegt, dass wir von den insgesamt n Bits tatsächlich nur $(1-\gamma)n/2$ Bits betrachten. Das sind gerade die Bits, von denen wir (mit hoher Wahrscheinlichkeit) wissen, dass sie ihren Wert mindestens einmal ändern müssen. Um Lemma 4.6 anwenden zu können, muss bei jeder Mutation die Wahrscheinlichkeit, dass eines dieser Bits mutiert, durch $2/((1-\gamma)n)$ nach oben beschränkt sein. Gemäß unseren Überlegungen, die wir bis jetzt angestellt haben, mutiert ein Bit mit durch $1/(n-\ln n)$ nach oben beschränkter Wahrscheinlichkeit in einer Generation. Wenn wir zwei aufeinanderfolgende Generationen betrachten, ist für jedes Bit die Wahrscheinlichkeit, in mindestens einer dieser beiden Generationen zu mutieren, immer noch durch $2/(n-\ln n)$ nach oben beschränkt. Weil für jede Wahl von γ mit $\gamma > 0$ offenbar

$$\frac{2}{n-\ln n} \leq \frac{2}{(1-\gamma)n}$$

gilt, können wir gedanklich je zwei Generationen zu einer zusammenfassen. Wir betrachten also effektiv nur noch $(\alpha/2)\cdot n\ln n$ Generationen.

Wir wollen also effektiv $(\alpha/2)\cdot n\ln n$ Generationen betrachten und haben uns überlegt, dass wir über ein Sammelkartenproblem sprechen können, bei dem s Bälle in

$$(1-\gamma)\cdot\frac{n}{2}-\ln n$$

Eimer geworfen werden. Dabei ist $\alpha < 1$ eine positive Konstante, die aus der Aussage stammt, also nicht wählbar ist und γ eine positive Kostante, die wir uns frei mit $0 < \gamma < 1$ wählen können. Um Lemma 4.6 anwenden zu können, müssen wir sicherstellen, dass in den effektiv $(\alpha/2)\cdot n\ln n$ Generationen nicht mehr als

$$\left((1-\gamma)\cdot\frac{n}{2}-\ln n\right)\cdot\ln\left((1-\gamma)\cdot\frac{n}{2}-\ln n\right)-c\cdot\left((1-\gamma)\cdot\frac{n}{2}-\ln n\right)$$

Mutationen stattfinden. Dabei ist $c > 0$ zunächst einmal eine frei wählbare Konstante. Als Schranke für die Wahrscheinlichkeit erhalten wir in diesem Fall e^{-e^c} durch Anwendung von Folgerung 4.7, wobei wir dann c so wählen müssen, dass die Wahrscheinlichkeit kleiner als die vorgegebene positive Konstante ε wird.

Die Wahrscheinlichkeit, dass „in T Generationen insgesamt mehr als $(1+\beta)T$ Bälle in Eimer treffen“, also mehr als $(1+\beta)T$ Mutationen in nT Generationen effektiv stattfinden, lässt sich durch Anwendung der Chernoff-Schranken nach oben durch

$$\left(\frac{e^{\beta}}{(1+\beta)^{1+\beta}}\right)^{T}$$

beschränken. Das gilt für jede Konstante $\beta > 0$.

Wir sehen also, dass mit großer Wahrscheinlichkeit in effektiv $(\alpha/2) \cdot n \ln n$ Generationen nicht mehr als $(1+\beta)(\alpha/2) \cdot n \ln n$ Mutationen stattfinden. Dabei ist β eine von uns frei wählbare Konstante mit $\beta > 0$. Wir müssen jetzt durch geeignete Wahl der Konstanten sicher stellen, dass

$$\begin{aligned}&(1+\beta) \cdot \frac{\alpha}{2} \cdot n \ln n \\ &\quad \leq \left(\frac{(1-\gamma)n}{2} - \ln n\right) \cdot \ln\left(\frac{(1-\gamma)n}{2} - \ln n\right) - c \cdot \left(\frac{(1-\gamma)n}{2} - \ln n\right)\end{aligned}$$

gilt. Dazu genügt es, dass

$$\begin{aligned}&(1+\beta) \cdot \frac{\alpha}{2} \\ &\quad \leq \left(\frac{1-\gamma}{2} - \frac{\ln n}{n}\right) \cdot \left(1 + \frac{\ln\left(\frac{1-\gamma}{2} - \frac{\ln n}{n}\right)}{\ln n}\right) - c \cdot \left(\frac{1-\gamma}{2 \ln n} - \frac{\ln n}{n \ln n}\right)\end{aligned}$$

gilt. Wir können dabei wie gesagt die Konstanten β mit $\beta > 0$ und γ mit $0 < \gamma < 1$ frei wählen. Wir betrachten nur das asymptotische Verhalten für $n \to \infty$, darum genügt es offenbar sicherzustellen, dass

$$\frac{(1+\beta)\alpha}{2} < \frac{1-\gamma}{2}$$

gilt. Das ist der Fall, wenn

$$\beta < \frac{1-\gamma}{\alpha} - 1$$

gilt. Wir dürfen β frei wählen, es muss aber $\beta > 0$ gelten. Wenn $\gamma < 1 - \alpha$ gilt, kann man für β leicht einen geeigneten Wert bestimmen. Wir können auch γ frei wählen, dabei muss allerdings $0 < \gamma < 1$ gelten. Weil wir $\alpha < 1$ voraussetzen, ist das aber möglich.

Bei geeigneter Wahl von β und γ, etwa $\gamma := (1-\alpha)/2$ und $\beta := (1-\alpha)/(3\alpha)$, haben wir also insgesamt gezeigt, dass mit Wahrscheinlichkeit höchstens

$$e^{-n\gamma^2/4} + e^{-\Omega((\ln n)\ln\ln n)} + \left(\frac{e^{\beta}}{(1+\beta)^{1+\beta}}\right)^{(\alpha/2)n\ln n} + e^{-e^{c}}$$

der (1+1) EA in höchstens $\alpha \cdot n \ln n$ Generationen das globale Maximum von f findet. Für $n \to \infty$ konvergiert dieser Ausdruck gegen $e^{-e^{c}}$, was bei Wahl eines hinreichend großen Wertes für c kleiner wird als ε. $\square$

5 Unimodale Funktionen

Man nennt eine Funktion unimodal, wenn sie nur eine lokale Extremstelle hat. Wir werden gleich formal und exakt definieren, was das für Funktionen $f\colon \{0,1\}^n \to \mathbb{R}$ konkret heißt. Die Betrachtung unimodaler Funktionen ist im wesentlichen durch das Interesse motiviert, eine Art deskriptive Klassifikation zu liefern. Es ist eine bei der Anwendung evolutionärer Algorithmen häufige Erfahrung, dass der evolutionäre Algorithmus nicht in annehmbarer Zeit ein globales Maximum findet, weil er in einem lokalen Maximum „steckenbleibt". Wenn aber nun eine Zielfunktion nur ein einziges lokales Maximum hat, dass dann natürlich auch gleichzeitig das eindeutige globale Maximum ist, so kann ein evolutionärer Algorithmus (ebenso wie eine lokale Suche) nicht in einem suboptimalen lokalen Maximum „steckenbleiben". Man kann darum vermuten, dass unimodale Funktionen einfach zu optimieren sind. Wir werden in diesem Kapitel einen Beweis vorbereiten, dass unimodale Funktionen tatsächlich einfacher sind als andere Funktionen. Wenn wir uns im folgenden Kapitel mit schwierigsten Funktionen beschäftigen, werden wir dann konkret zeigen können, dass solche schwierigste Funktionen nicht unimodal sein können. Das Hauptergebnis dieses Kapitels wird aber sein, dass diese „Einfachheit" unimodaler Funktionen doch sehr begrenzt ist. Wir werden ganz konkret eine unimodale Funktion untersuchen, für deren Optimierung der (1+1) EA im erwarteten Fall exponentiell lange braucht.

Definition 5.1. *Eine Funktion $f\colon \{0,1\}^n \to \mathbb{R}$ heißt* unimodal*, wenn sie genau ein lokales Maximum hat. Ein Punkt $x' \in \{0,1\}^n$ heißt* lokales Maximum*, wenn*

$$\forall x \in \{0,1\}^n : H(x, x') \leq 1 \Rightarrow f(x) \leq f(x')$$

gilt.

Nach Definition 5.1 ist die Funktion $f_{\mathrm{N},a}$, die genau bei a den globalen Maximalwert 1 und überall sonst den Wert 0 hat, *nicht* unimodal. Jeder Punkt $x \in \{0,1\}^n$, der mindestens Hammingabstand 2 von a hat, hat nur Nachbarn mit Hammingabstand 1, die den gleichen Funktionswert haben. Folglich ist jeder dieser Punkte ein lokales Maximum und $f_{\mathrm{N},a}$ hat insgesamt $2^n - n$ lokale Maxima. Das entspricht in gewisser Weise unserer intuitiven Vorstellung von unimodal, die wir bei reellwertigen Funktionen gewonnen haben. Man kann solche Funktionen mit Gradientenverfahren optimieren: in jedem nichtoptimalen Punkt zeigt der Gradient in Richtung einer lokalen Verbesserung. Man findet also immer, wenn man das globale Optimum noch nicht erreicht hat, relativ leicht einen Hinweis auf einen besseren Punkt. Das ist bei Funktionen, die gemäß Definition 5.1 unimodal sind, genau so, wie das nächste Lemma zeigt.

Lemma 5.2. *Sei $f\colon \{0,1\}^n \to \mathbb{R}$ eine unimodale Funktion. Für alle $x \in \{0,1\}^n$ gilt, dass x ein globales Maximum ist oder es einen Punkt $y \in \{0,1\}^n$ mit $H(x,y) = 1$ gibt, für den $f(y) > f(x)$ gilt.*

Beweis. Der Beweis folgt direkt aus der Definition. Sei $x \in \{0,1\}^n$. Wenn es einen Punkt $y \in \{0,1\}^n$ mit $H(x,y) = 1$ gibt, so dass $f(y) > f(x)$ gilt, ist x offenbar kein globales Maximum und die Aussage ist richtig. Andernfalls ist x gemäß Definition 5.1 ein lokales Maximum. Weil f nach Voraussetzung unimodal ist, ist x das einzig lokale Maximum und folglich global maximal. $\square$

Man kann eine unimodale Funktion also optimieren, indem man in einem beliebigen Punkt startet, alle n Nachbarn mit Hammingabstand 1 betrachtet und einen mit größerem Funktionswert als neuen aktuellen Punkt wählt. Wenn es keinen solchen Nachbarn gibt, ist gemäß Lemma 5.2 der aktuelle Punkt global maximal. Wir sehen, dass eine lokale Suche immer zum Erfolg führt und mit Zeit $O(n \cdot 2^n)$ auskommt: jede Verbesserung kommt mit der Untersuchung von $O(n)$ Nachbarn aus, mehr als $2^n - 1$ Verbesserungen kann es nicht geben. Wie beschrieben ändert sich von einem aktuellen Punkt zum nächsten immer genau 1 Bit. In diesem Sinne sind unimodale Funktionen bitweise optimierbar. Das macht auch dem (1+1) EA eine Optimierung in gewissem Sinne leicht. Er braucht im Durchschnitt nicht länger als eine lokale Suche, wie der nächste Satz zeigt. Er gibt eine einfach zu beweisende, aber nützliche allgemeine obere Schranke für die erwartete Laufzeit des (1+1) EA auf unimodalen Funktionen an. Man findet diese Aussage bei Rudolph (1997a) als Bemerkung ohne Beweis.

Satz 5.3. *Die erwartete Laufzeit des $(1+1)$ EA mit $p(n) = 1/n$ auf einer unimodalen Funktion $f\colon \{0,1\}^n \to \mathbb{R}$ mit k unterschiedlichen Funktionswerten (also $|\{f(x) \mid x \in \{0,1\}^n\}| = k$) ist $O(kn)$.*

Beweis. Wir setzen $|\{f(x) \mid x \in \{0,1\}^n\}| = k$ voraus, sei etwa

$$\{f(x) \mid x \in \{0,1\}^n\} = \{v_1, v_2, \dots, v_k\},$$

sei dabei $v_1 > v_2 > \cdots > v_k$. Sei x der aktuelle String. Solange x nicht global maximal ist, gibt es einen Nachbarn x' mit Hammingabstand 1, so dass $f(x') > f(x)$ gilt. Die Wahrscheinlichkeit für eine direkte Mutation von x nach x' beträgt $(1/n)(1-1/n)^{n-1} \geq 1/(en)$. Die erwartete Zeit für eine echte Verbesserung des Funktionswertes ist folglich durch en nach oben beschränkt. Weil es nur k verschiedene Funktionswerte gibt, kann es nur $k-1$ Verbesserungen geben. Wir benutzen also die triviale Partitionierung $P_i = \{x \in \{0,1\}^n \mid f(x) = v_i\}$ und wissen, dass es nur k verschiedene Ranggruppen gibt. Außerdem haben wir $1/(en)$ als untere Schranke für alle Übergangswahrscheinlichkeiten. Damit haben wir also $en(k-1) = O(kn)$ als obere Schranke für die erwartete Laufzeit. □

Wir sehen direkt, dass die im Satz 5.3 angegebene Schranke nicht für alle Funktionen scharf ist. So erhalten wir für die lineare Funktion f_1 die Schranke $O(n^2)$ und für die lineare Funktion f_{B} gar die Schranke $O(n2^n)$ anstatt $O(n \log n)$. Man kann aber dennoch auch nützliche Folgerungen ziehen.

Folgerung 5.4. *Die erwartete Laufzeit des* $(1+1)$ *EA mit* $p(n) = 1/n$ *ist* $O(n2^n)$ *für jede unimodale Funktion* $f\colon \{0,1\}^n \to \mathbb{R}$.

Es lassen sich auch leicht brauchbare obere Schranken für einige Funktionen deduzieren. Betrachten wir zunächst die Funktion f_{R}, die wir in Kapitel 2 als Beispiel kennengelernt haben, bei dem Fitness Distance Correlation keine brauchbare Klassifizierung liefert.

Folgerung 5.5. *Die erwartete Laufzeit des* $(1+1)$ *EA mit* $p(n) = 1/n$ *auf der Funktion* $f_R\colon \{0,1\}^n \to \mathbb{R}$ *ist* $O(n^2)$.

Beweis. Offensichtlich nimmt die Funktion f_{R} nur $2n$ verschiedene Funktionswerte an. Es genügt also zu zeigen, dass f_{R} unimodal ist.

Sei $x \in \{0,1\}^n$ und bezeichne i die Anzahl der Einsen in x, also $i = \|x\|_1$. Ist $i = n$, so ist $x = \mathbf{1}$ das globale Maximum und natürlich auch ein lokales Maximum. Sei nun also $i < n$. Ist $x = 1^i 0^{n-i}$, so hat $1^{i+1}0^{n-i-1}$ Hammingabstand 1 von x und einen um 1 größeren Funktionswert. Andernfalls ist $i > 0$ und x hat mindestens einen Nachbarn mit Hammingabstand 1, der genau $i-1$ Einsen enthält und einen um 1 größeren Funktionswert hat.

Wir sehen, dass für alle $x \neq \mathbf{1}$ gilt, dass ein Nachbar mit Hammingabstand 1 existiert, der echt größeren Funktionswert hat. Folglich ist $\mathbf{1}$ das einzige lokale Maximum und f_{R} unimodal. □

Wir haben uns schon überlegt, dass unimodale Funktionen genau wie lineare Funktionen durch das Mutieren einzelner Bits optimiert werden können.

Mühlenbein (1992) äußert darum die Vermutung, dass der (1+1) EA auch alle unimodalen Funktionen in erwarteter Zeit $O(n \log n)$ optimiert. Schon Rudolph (1997a) bezweifelt das und präsentiert eine Funktion f_{L}, für die er durch umfangreiche Versuche nahelegt, dass die erwartete Laufzeit in der Größenordnung von n^2 liegt. Wir verifizieren hier diese Vermutung (Droste, Jansen und Wegener 1998a).

Definition 5.6. *Die Funktion $f_L\colon \{0,1\}^n \to \mathbb{R}$ ist definiert durch*

$$f_L(x) := \sum_{1 \le i \le n} \prod_{1 \le j \le i} x_j$$

für alle $x \in \{0,1\}^n$.

Die Funktion f_{L}, die häufig auch LeadingOnes genannt wird, weist dem Punkt x als Funktionswert die Anzahl führender Einsen in x zu. Daraus folgt, dass der (1+1) EA genau wie bei der linearen Funktion f_{B} eine ununterbrochene Kette führender Einsen stets erhalten wird. Anders als bei f_{B} spielen aber alle Bits rechts vom am weitesten links befindlichen Bit mit dem Wert Null für den Funktionswert gar keine Rolle. Es gibt also zu jedem Zeitpunkt, zu dem das globale Maximum $\mathbf{1}$ noch nicht erreicht ist, genau ein Bit, das mutieren muss, damit der Funktionswert wächst. Das ist die zentrale Eigenschaft, die dafür verantwortlich ist, dass die erwartete Laufzeit $\Theta(n^2)$ beträgt.

Satz 5.7. *Die erwartete Laufzeit des $(1+1)$ EA mit $p(n) = 1/n$ auf der Funktion $f_L\colon \{0,1\}^n \to \mathbb{R}$ beträgt $\Theta(n^2)$. Außerdem gibt es zwei positive Konstante $c_1 < c_2$, so dass die Wahrscheinlichkeit, dass der $(1+1)$ EA weniger als $c_1 n^2$ oder mehr als $c_2 n^2$ Generationen braucht, $e^{-\Omega(n)}$ ist.*

Beweis. Einen Beweis für die obere Schranke $O(n^2)$ findet man bei Rudolph (1997a). Noch etwas leichter kann man die obere Schranke auch aus Satz 5.3 folgern: die Funktion f_{L} ist offensichtlich unimodal und hat genau $n+1$ verschiedene Funktionswerte. Wir wollen aber nachweisen, dass die Wahrscheinlichkeit für eine Abweichung exponentiell klein ist. Darum müssen wir den Prozess etwas genauer untersuchen.

Solange das globale Maximum $\mathbf{1}$ noch nicht erreicht ist, beträgt die Wahrscheinlichkeit für eine Erhöhung der Anzahl führender Einsen (und also auch des Funktionswertes) mindestens $(1/n)(1-1/n)^{n-1} \ge 1/(en)$. Die Wahrscheinlichkeit, dass es in $2en^2$ Generationen nicht wenigstens n Generationen gibt, in denen der Funktionswert echt größer wird, lässt sich durch Anwendung der Chernoff-Schranken durch

$$e^{-\left(2en^2/en\right)(1/2)^2/2} = e^{-n/4} = e^{-\Omega(n)}$$

nach oben beschränken. Wir haben also die obere Schranke mit $c_2 = 2e$ gezeigt; die Konstante kann offensichtlich unter Beibehaltung der Schranke $e^{-\Omega(n)}$ für die Wahrscheinlichkeit auch auf $e + \varepsilon$ für jede Konstante $\varepsilon > 0$ gesenkt werden.

Für die untere Schranke gehen wir zweischrittig vor. Zunächst machen wir uns klar, dass alle Bits, die einen größeren Index haben als das Bit, das unter allen Bits mit Wert Null minimalen Index hat, rein zufällig sind. Zur Klarheit formulieren wir diese Aussage noch einmal anders. Sei der Funktionswert des aktuellen Strings x genau i, also $f_{\mathrm{L}}(x) = i$. Dann gilt $x_1 = x_2 = \cdots = x_i = 1$ und $x_{i+1} = 0$. Wir behaupten, dass der String $x_{i+2}x_{i+3}\cdots x_n$ zufällig gleichverteilt aus $\{0,1\}^{n-(i+1)}$ ist. Dass das direkt nach der Initialisierung gilt, ist klar. Wenn es für den aktuellen String x gilt, so auch für den Nachfolger: Der neue String y wird genau dann akzeptiert, wenn $y_1 = x_1$, $y_2 = x_2$, $\ldots$, $y_i = x_i$ gilt. Die Bits mit Indizes größer als i spielen dabei keine Rolle. Weil die zufällige Mutation eines zufälligen Strings natürlich einen zufälligen String ergibt, gilt die Behauptung.

Im zweiten Schritt zeigen wir jetzt, dass mit Wahrscheinlichkeit $1 - e^{-\Omega(n)}$ $n^2/6$ Generationen nicht ausreichen, um das globale Maximum $\mathbf{1}$ zu erreichen. Wir weisen also $c_2 = 1/6$ nach. Wir nennen eine Generation essenziell, wenn $f_{\mathrm{L}}(y) > f_{\mathrm{L}}(x)$ gilt. Damit eine Generation essenziell ist, muss auf jeden Fall das $(f_{\mathrm{L}}(x) + 1)$-te Bit mutieren. Die Wahrscheinlichkeit dafür beträgt $p(n) = 1/n$. Die erwartete Anzahl essenzieller Generationen in $n^2/6$ Generationen ist folglich durch $(1/n) \cdot n^2/6 = n/6$ nach oben beschränkt. Anwendung der Chernoff-Schranken ergibt, dass die Wahrscheinlichkeit, dass von $n^2/6$ Generationen mindestens $n/3$ essenziell sind, nach oben durch $(e/4)^{n/6} = e^{-\Omega(n)}$ beschränkt ist. In einer essenziellen Generation nimmt der Funktionswert um mindestens 1 zu, weil jedenfalls das Bit, das unter allen Bits in x mit Wert Null minimalen Index hat, zu der Kette führender Einsen hinzukommt. Er kann sich auch noch weiter erhöhen. Der Funktionswert nimmt genau um $1 + k$ zu, wenn k die Anzahl zufälliger Bits bezeichnet, die direkt auf das $(f_{\mathrm{L}}(x) + 1)$-te Bit folgen und den Wert Eins haben. Die Wahrscheinlichkeit, an der i-ten Stelle ein Bit mit Wert Eins zu haben beträgt genau $1/2$ für alle i mit $f_{\mathrm{L}}(x) + 1 < i \leq n$. Die erwartete Anzahl solcher „zusätzlicher Einsen“ beträgt darum genau 1. In $n/3$ essenziellen Generationen ist folglich die erwartete Anzahl solcher zusätzlicher Einsen durch

$n/3$ nach oben beschränkt. Anwendung der Chernoff-Schranken ergibt, dass die Wahrscheinlichkeit, in $n/3$ essenziellen Generationen mindestens $2n/3$ zusätzliche Einsen zu haben, nach oben durch $(e/4)^{n/3} = e^{-\Omega(n)}$ beschränkt ist. Die Wahrscheinlichkeit, nach $n^2/6$ Generationen das globale Maximum $\mathbf{1}$ erreicht zu haben, ist folglich durch $(e/4)^{n/6}+(e/4)^{n/3} = e^{-\Omega(n)}$ nach oben beschränkt. Schließlich müsste es sonst in den $n^2/6$ Generationen mehr als $n/3$ essenzielle Generationen gegeben haben oder in höchstens $n/3$ essenziellen Generationen müssten mindestens $2n/3$ zusätzliche Einsen hinzugekommen sein. □

Die Funktion f_{L} ist also eine unimodale Funktion, für die im Durchschnitt $cn \ln n$ Generationen ($c \in \mathbb{R}$ konstant) dem (1+1) EA mit $p(n) = 1/n$ zur Optimierung nicht ausreichen. Außerdem ist sie eine Beispielfunktion, bei der die einfache Schranke aus Satz 5.3 sogar scharf ist. Offensichtlich ist die Klasse der unimodalen Funktionen damit aber noch nicht in befriedigendem Maße behandelt. Die erwartete Laufzeit $\Theta(n^2)$ ist sowohl verhältnismäßig effizient als auch von der erwarteten Laufzeit $O(n \log n)$ für lineare Funktionen nicht allzu weit entfernt. Außerdem lässt die obere Schranke aus Satz 5.3 zu, dass es unimodale Funktionen gibt, für die der (1+1) EA mit $p(n) = 1/n$ im erwarteten Fall sogar eine exponentielle Anzahl von Generationen zur Optimierung braucht. Dazu ist es allerdings nötig, dass es „Pfade“ zum Optimum gibt, die exponentielle Länge haben. Unter einem Pfad verstehen wir eine Folge von Punkten $v_1, v_2, \ldots, v_l \in \{0,1\}^n$ mit Hammingabstand 1 zwischen Nachbarpunkten ($H(v_i, v_{i+1}) = 1$) und strikt wachsenden Funktionswerten ($f(v_i) < f(v_{i+1})$). Als erste haben Horn, Goldberg und Deb (1994) einen derartigen langen Pfad konstruiert. Es ist unmittelbar klar, dass eine lokale Suche, die am Pfadanfang beginnt, tatsächlich exponentiell lange zum Erreichen des Pfadendes, also des globalen Maximums, braucht. Für den (1+1) EA mit $p(n) = 1/n$ ist das allerdings nicht unmittelbar klar. Zwar mutiert im Durchschnitt in jeder Generation genau 1 Bit, aber es gibt ja auch Mutationen von mehreren Bits gleichzeitig. Deshalb ist es möglich, dass der (1+1) EA Abkürzungen auf dem Pfad findet. Weil es mit Wahrscheinlichkeit ungefähr $1/(ek!)$ (Approximation der Binomialverteilung durch die Poissonverteilung) Mutationen gibt, bei denen k Bits gleichzeitig mutieren, ist es sogar recht wahrscheinlich, dass Abkürzungen, die nur simultanes Mutieren einiger weniger Bits erfordern, tatsächlich gefunden werden. Horn, Goldberg und Deb (1994) sind dieser Problematik empirisch nachgegangen und waren überzeugt, in ihren Experimenten für den (1+1) EA mit $p(n) = 1/n$ exponentielle Laufzeiten zu beobachten. Tatsächlich konnte aber Rudolph (1997b) nachweisen, dass im Durchschnitt $O(n^3)$ Generationen ausreichen. Er formalisiert und untersucht daraufhin eine allgemeinere Variante langer

Pfade (Rudolph 1997a), die Horn, Goldberg und Deb (1994) informal beschrieben haben. Die Idee dieser Variante ist, die Anzahl von Bits, die für eine Abkürzung gleichzeitig mutieren müssen, nach unten durch einen Parameter k zu beschränken. Allerdings nimmt mit wachsender unterer Schranke k die Pfadlänge deutlich ab. Rudolph (1997a) beweist zwei unterschiedliche obere Schranken für die erwartete Zeit, die der (1+1) EA vom Pfadanfang zum Pfadende braucht. Es gibt Wahlen von k, für die beide obere Schranken exponentiell groß sind. Man kann also mit gewisser Berechtigung sagen, dass lange k-Pfade gute Kandidaten sind, wenn man unimodale Funktionen sucht, für die der (1+1) EA mit $p(n) = 1/n$ im Durchschnitt zur Optimierung exponentiell lange braucht. Wir führen darum jetzt hier formal lange k-Pfade ein und machen uns ihre wesentlichen Eigenschaften klar.

Definition 5.8. *Sei $n \geq 1$ eine natürliche Zahl. Wir definieren lange k-Pfade der Dimension n, P_k^n, für alle natürlichen Zahlen $k > 1$ mit $(n-1)/k \in \mathbb{N}$. Der* lange k-Pfad der Dimension r *ist dabei eine Folge paarweise verschiedener Bitstrings aus $\{0,1\}^r$. Der lange k-Pfad der Dimension 1 ist $P_k^1 := (0,1)$. Den langen k-Pfad der Dimension n definieren wir rekursiv basierend auf dem langen k-Pfad der Dimension $n-k$. Sei $P_k^{n-k} = (v_1, \ldots, v_l)$ dieser lange k-Pfad der Dimension $n-k$. Dann entsteht der lange k-Pfad der Dimension n durch Konkatenation der Folgen S_0, B_k^n und S_1 in dieser Reihenfolge. Dabei ist $S_0 := (0^k v_1, 0^k v_2, \ldots, 0^k v_l)$, $B_k^n := (0^{k-1}1 v_l, 0^{k-2}1^2 v_l, \ldots, 01^{k-1} v_l)$ und $S_1 := (1^k v_l, 1^k v_{l-1}, \ldots, 1^k v_1)$. Die Punkte in B_k^n bilden eine Art Brücke zwischen S_0 und S_1, darum heißen die Punkte aus B_k^n* Brückenpunkte. *Der lange k-Pfad der Dimension n besteht aus $|P_k^n|$ verschiedenen Punkten. Wir bezeichnen die Anzahl Punkte $|P_k^n|$ in P_k^n als Länge, der i-te Punkte heißt p_i. Den Punkt p_{i+j} ($j \in \mathbb{N}$) nennen wir* j-ten Nachfolger *von p_i.*

Die rekursive Definition langer k-Pfade erlaubt es, ihre wesentlichen Eigenschaften leicht zu bemerken und zu beweisen. Alle Eigenschaften, die wir im folgenden beschreiben, sind schon von Rudolph (1997a) festgestellt und nachgewiesen worden. Wir geben auch hier kurze Beweise an, einerseits um die Vollständigkeit der Darstellung zu gewährleisten, andererseits weil durch den Umgang mit langen k-Pfaden in den Beweisen eine gewisse Vertrautheit entsteht, die für unsere nachfolgenden Überlegungen hilfreich ist.

Lemma 5.9. *Der lange k-Pfad der Dimension n hat Länge*

$$|P_k^n| = (k+1)2^{(n-1)/k} - k + 1.$$

Alle Pfadpunkte sind verschieden.

Beweis. Für Dimension 1 haben wir $P_k^1 = (0,1)$, die beiden Pfadpunkte sind offensichtlich verschieden, für die Pfadlänge haben wir $|P_k^1| = 2$, was mit

$$(k+1) \cdot 2^{(1-1)/k} - k + 1 = k + 1 - k + 1 = 2$$

übereinstimmt. Nehmen wir an, dass die Aussage für alle Dimensionen echt kleiner als n gilt. Offensichtlich unterscheiden sich dann alle Punkte in S_0, B_k^n und S_1, da alle Punkte in S_0 genau k führende Nullen haben, alle Punkte in S_1 genau k führende Einsen und alle Punkte in B_k^n eine von k und 0 verschiedene Anzahl führender Einsen. Dass die Punkte innerhalb von S_0 und S_1 paarweise verschieden sind, folgt aus der Induktionsannahme. Für die Pfadlänge haben wir

$$\begin{aligned} |P_k^n| &= 2\left|P_k^{n-l}\right| + (k-1) \\ &= 2(k+1)2^{(n-k-1)/k)} - 2k + 2 + k - 1 \\ &= (k+1)2^{(n-1)/k} - k + 1 \end{aligned}$$

wie behauptet. □

Bis jetzt haben wir von langen k-Pfaden der Dimension n gesprochen, ohne nachzuweisen, dass die zentrale Pfadeigenschaft, dass benachbarte Punkte Hammingabstand 1 haben, erfüllt ist. Wir holen das jetzt nach und zeigen gleichzeitig, dass Abkürzungen auf dem Pfad das simultane Mutieren von mindestens k Bits erfordern.

Lemma 5.10. *Seien n und k so gegeben, dass der lange k-Pfad der Dimension n wohldefiniert ist. Für alle $i \in \mathbb{N}$ mit $0 < i < k$ gilt das folgende. Wenn $x \in P_k^n$ mindestens i verschiedene Nachfolger auf dem Pfad P_k^n hat, so hat der i-te Nachfolger genau Hammingabstand i von x und alle anderen Punkte auf dem Pfad P_k^n, die Nachfolger von x sind, haben von i verschiedene Hammingabstände von x.*

Beweis. Die Aussage gilt offensichtlich für $n = 1$ und alle Werte von k. Wir nehmen jetzt an, dass sie für P_k^{n-k} gilt. Wir erinnern uns an die Konstruktion von P_k^n aus S_0, B_k^n und S_1, wobei $|P_k^n| = (k+1)2^{(n-1)/k} - k + 1$, $|S_0| = |S_1| = (k-1)2^{(n-k-1)/k} - k + 1$ und $|B_k^n| = k - 1$ gilt. Sei $x \in P_k^n$ der p-te Punkte auf dem Pfad. Sei i mit $0 < i < k$ so gegeben, dass x mindestens i verschiedene Nachfolger auf dem Pfad P_k^n hat. Es gibt fünf Fälle, die wir nach der Lage von x und seinem i-ten Nachfolger unterscheiden.

1. Fall: $p < |S_0|$ und $p + i \leq |S_0|$
Es liegen also sowohl x als auch sein i-ter Nachfolger in S_0, also essenziell

in demselben langen k-Pfad der Dimension $n-k$. Die Aussage über die Punkte in S_0 folgt aus der Induktionsannahme. Die Punkte in S_1 haben alle Hammingabstand mindestens $k > i$ von allen Punkten in S_0, da sie sich in den ersten k Bits unterscheiden. Die Punkte in B_k^n haben alle größere Hamminabstände als i, da sie gleichen Hammingabstand haben wie der letzte Punkt von S_0 zuzüglich dem Unterschied in den ersten k Bits, der zwischen 1 und $k-1$ liegt.

2. Fall: $p > |S_0| + |B_k^n|$
Es liegen also sowohl x als auch sein i-ter Nachfolger in S_1. Die Aussage folgt analog zum ersten Fall aus der Induktionsannahme.

3. Fall: $p \leq |S_0|$ und $p + i \leq |S_0| + |B_k^n|$
Es liegt also x in S_0, der i-te Nachfolger gehört zu den Brückenpunkten. Nach Induktionsannahme hat x vom letzten Pfadpunkt in S_0 Hammingabstand $|S_0| - p$, was zwischen 0 und $k-1$ liegt, weil $0 < i < k$ gilt. Nach Definition von B_k^n hat der j-te Punkt in B_k^n Hammingabstand j vom letzten Punkt in S_0. Weil sich alle Punkte in S_0 in den ersten k Bits gleichen und alle Punkte in B_k^n sich nur in den ersten k Bits unterscheiden können, addieren sich die Hammingabstände. Alle Punkte in S_1 haben Hammingabstand mindestens k, weil sich alle Punkte in S_0 von allen Punkte in S_1 mindestens in den ersten k Bits unterscheiden.

4. Fall: $|S_0| < p \leq |S_0| + |B_k^n|$ und $p + i > |S_0| + |B_k^n|$
Dieser Fall stimmt essenziell mit dem dritten Fall überein. Der i-te Nachfolger von x übernimmt die Rolle von x, liegt allerdings in S_1 statt S_0. Der Punkt x übernimmt die Rolle des i-ten Nachfolger und gehört zu den Brückenpunkten.

5. Fall: $|S_0| < p < |S_0| + |B_k^n|$ und $p + I \leq |S_0| + |B_k^n|$
Es gehören also sowohl x als auch sein i-ter Nachfolger zu den Brückenpunkten. Folglich sind die beiden Strings auf den hinteren $n-k$ Bits gleich, wir betrachten also nur die vorderen k Bits, die von der Form $0^r 1^{k-r}$ sind. Offensichtlich hat der j-te Nachfolger von x in B_k^n genau Hammingabstand j von x. Alle Punkte in S_1 haben echt größeren Hammingabstand. □

Aus Lemma 5.10 folgt direkt, dass für jeden Punkt $x \in P_k^n$ gilt, dass alle Punkte, die wenigstens k Positionen weiter hinten auf dem langen k-Pfad als x liegen, Hammingabstand mindestens k von x haben. Um also auf P_k^n eine Abkürzung zu nehmen, die einen mindestens k Positionen weiterbringt, müssen mindestens k Bits gleichzeitig mutieren. Für alle i mit $0 < i < k$ gilt sogar, dass einen das simultane Mutieren von i Bits auf dem Pfad bestenfalls um i Positionen weiterbringt.

Wir kennen jetzt lange k-Pfade und ihre wesentlichen Eigenschaften. Wir sind aber (anders als Horn, Goldberg und Deb (1994)) nicht an der erwarteten Anzahl von Generationen interessiert, die der (1+1) EA mit $p(n) = 1/n$ vom Pfadanfang bis zum Pfadende braucht. Wir betrachten den (1+1) EA immer nach zufälliger Initialisierung. Wir müssen also zunächst den langen k-Pfad der Dimension n, der ja nur $(k+1)2^{(n-1)/k} - k + 1$ der 2^n Punkte belegt, in eine Funktion $f_{\mathrm{P},k}$ einbetten. Dabei muss die Einbettung so erfolgen, dass einerseits die entstehende Funktion $f_{\mathrm{P},k}$ unimodal ist und andererseits der (1+1) EA mit genügend großer Wahrscheinlichkeit als ersten Pfadpunkt einen nicht zu weit hinten liegenden Punkt erreicht. Wir lösen diese Aufgabe etwas anders als Rudolph (1997a). Unsere abweichende Definition hat keine Auswirkungen auf die zwei oberen Schranken (Rudolph 1997a), erleichtert aber merklich den Nachweis der unteren Schranke.

Definition 5.11. *Seien n und k so gegeben, dass der lange k-Pfad der Dimension n wohldefiniert ist. Die* lange k-Pfad Funktion der Dimension n *$f_{P,k}\colon \{0,1\}^n \to \mathbb{R}$ ist definiert durch*

$$f_{P,k}(x) := \begin{cases} n^2 + l & \text{falls } x \text{ der } l\text{-te Punkte auf } P_k^n \text{ ist,} \\ n^2 - n \sum\limits_{1 \le i \le k} x_i - \sum\limits_{k < i \le n} x_i & \text{falls } x \notin P_k^n \text{ gilt.} \end{cases}$$

für alle $x \in \{0,1\}^n$.

Wir bemerken zunächst, dass $f_{\mathrm{P},k}$ unimodal ist. Für jeden Punkt $x \notin P_k^n$ genügt es offenbar, die Anzahl der Einsen in x um 1 zu senken, um den Funktionswert zu erhöhen. Dieses Vorgehen endet spätestens im Nullstring $\mathbf{0}$, wobei $\mathbf{0} \in P_k^n$ der erste Pfadpunkt ist. Jeder Punkt auf dem Pfad außer dem letzten Punkt hat genau einen Nachbarn mit Hammingabstand 1, der auch auf dem Pfad liegt und echt größeren Funktionswert hat. Der letzte Pfadpunkt ist das einzige lokale Maximum, also auch eindeutiges globales Maximum.

In der Funktion $f_{\mathrm{P},k}$ haben wir auch ein erstes Beispiel für eine Funktionsfamilie, die nicht für alle Werte von n definiert ist. Wählen wir etwa k fest, so gibt es offenbar nicht für alle Wahlen von n einen langen k-Pfad der Dimension n. Das stört uns in unserer asymptotischen Betrachtungsweise allerdings nicht, weil die Funktion $f_{\mathrm{P},k}$ für jede feste Wahl von k für unendlich viele Werte von n definiert ist. Wählen wir später k in Abhängigkeit von n, so müssen wir natürlich darauf achten, dass für unendliche viele Paarungen von n und k die Funktion $f_{\mathrm{P},k}$ definiert ist.

Wir betrachten nun die erwartete Laufzeit des (1+1) EA mit $p(n) = 1/n$ auf der Funktion $f_{\mathrm{P},k}$ und weisen für eine geeignete Wahl von k nach, dass die erwartete Laufzeit exponentiell ist. Dazu beginnen wir mit allgemeinen oberen Schranken und sehen, dass man leicht zwei verschiedene obere Schranken nachweisen kann.

Satz 5.12 (Rudolph (1997a)). *Seien n und k so gegeben, dass der lange k-Pfad der Dimension n wohldefiniert ist. Die erwartete Anzahl Generationen, die der $(1+1)$ EA mit $p(n) = 1/n$ zum Optimieren der Funktion $f_{P,k}\colon \{0,1\}^n \to \mathbb{R}$ braucht, ist durch $O\left(n^{k+1}/k\right)$ und $O\left(n\,|P_k^n|\right)$ nach oben beschränkt.*

Beweis. Wir teilen einen Lauf des (1+1) EA auf $f_{\mathrm{P},k}$ in zwei Phasen. Die erste Phase hat Länge T_1, beginnt mit der Initialisierung und endet, wenn der (1+1) EA einen Punkt auf dem Pfad erreicht. Die zweite Phase beginnt mit Ende der ersten Phase und endet mit Erreichen des globalen Maximum von $f_{\mathrm{P},k}$. Sie hat eine Länge von T_2. Offenbar beträgt die erwartete Laufzeit des (1+1) EA genau $\mathrm{E}\left(T_1\right) + \mathrm{E}\left(T_2\right)$.

Nach der Initialisierung gehört entweder der aktulle String x zum Pfad, in diesem Fall gilt $T_1 = 0$, oder x gehört nicht zum Pfad, dann betrachten wir die Funktion $f_{\mathrm{P},k}$ nur auf den Punkten, die nicht zum Pfad P_k^n gehören. Eingeschränkt auf diese Punkte gleicht die Funktion $f_{\mathrm{P},k}$ einer linearen Funktion mit Gewichten $w_0 = 0$, $w_1 = \cdots = w_k = -n$, $w_{k+1} = \cdots = w_n = -1$. Wir folgern daraus und mit Satz 4.3, dass $\mathrm{E}\left(T_1\right) = O\left(n \log n\right)$ gilt.

Um $\mathrm{E}\left(T_2\right)$ nach oben abzuschätzen, gehen wir wie folgt vor. Es sei $P_k^n = (v_1, v_2, \ldots, v_l)$ mit $l = |P_k^n|$. Wir teilen P_k^n in r aufeinander folgende Teilpfade $U_1, U_2, \ldots, U_r$ mit $U_r = \{v_l\}$. Wegen der entsprechenden Definition der Pfadfunktion (Definition 5.11) ist sofort klar, dass die U_i auch Ranggruppen im Sinne von Definition 3.3 sind. Wir bestimmen eine untere Schranke q für die Übergangswahrscheinlichkeit q_i, von einem Punkt aus U_i zu irgendeinem Punkt innerhalb von $U_{i+1} \cup U_{i+2} \cup \cdots \cup U_r$ zu wechseln für alle $i \in \{1, 2, \ldots, r-1\}$. Dann ist die erwartete Zeit, bis der (1+1) EA wenigstens zum nächsten Teilpfad kommt, durch $1/q_i \leq 1/q$ nach oben beschränkt. Weil $r-1$ Wechsel zum jeweils mindestens nächsten Teilpfad sicher ausreichen, haben wir $\mathrm{E}\left(T_2\right) \leq (r-1)/q$.

Für die erste untere Schranke wählen wir $r = l$ und $U_i = \{v_i\}$. Weil zwei auf dem Pfad benachbarte Punkte immer Hammingabstand 1 haben, ist $q \geq (1/n)(1-1/n)^{n-1} \geq 1/(en)$ eine untere Schranke. Damit haben wir $\mathrm{E}\left(T_2\right) = O\left(n\,|P_k^n|\right)$ gezeigt. Es ist $n\,|P_k^n| = \omega(n \log n)$ für alle möglichen Wahlen von n und k. Das gibt insgesamt die obere Schranke $O\left(n\,|P_k^n|\right)$.

Für die zweite obere Schranke wählen wir $r = 2+(n-1)/k$. Wir erinnern uns, dass P_k^n aus zwei "Kopien" von P_k^{n-k} und Brückenpunkten gebildet wird. Der Teilpfad U_1 besteht aus der ersten "Kopie" S_0 und den Brückenpunkten B_k^n. Wir wenden dieses Vorgehen wiederum auf die zweite "Kopie" S_1 an und definieren durch Iteration $U_2, \ldots, U_r$. Im letzten Schritt haben wir einen Pfad der Länge 2 und U_r enthält nur noch das globale Maximum von $f_{\mathrm{P},k}$. Aus der Definition des langen k-Pfades (Definition 5.8) folgt direkt, dass es zu jedem Punkt aus S_0 und B_k^n mindestens einen Punkt in S_1 gibt, der Hammingabstand höchstens k hat. Folglich haben wir $q \geq (1/n)^k(1-1/n)^{n-k} \geq 1/(en^k)$, woraus $\mathrm{E}\left(T_2\right) = O\left(n^{k+1}/k\right)$ folgt. Es ist $n^{k+1}/k = \omega(n \log n)$ für alle möglichen Werte von n und k. Also haben wir insgesamt $O\left(n^{k+1}/k\right)$ als obere Schranke. $\square$

Je nach Wahl von k ist die erwartete Laufzeit polynomiell in n. Das gilt zum einen für $k = O(1)$, zum anderen für $k = \Omega(n/\log n)$. Andererseits haben wir für $k = \sqrt{n-1}$ die oberen Schranken $O\left(n^{\sqrt{n-1}+1/2}\right)$ und $O\left(n^{3/2}2^{\sqrt{n}}\right)$, also $O\left(n^{3/2}2^{\sqrt{n}}\right)$ als kleinere, aber immer noch exponentielle obere Schranke. Wir werden jetzt nachweisen, dass die erwartete Laufzeit des (1+1) EA auf der Funktion $f_{\mathrm{P},\sqrt{n-1}}$ in der Tat exponentiell ist. Der Beweis der oberen Schranke war verhältnismäßig einfach. Darum ist es vielleicht etwas verblüffend, dass wir nachweisen können, dass für $k = \sqrt{n-1}$ die obere Schranke tatsächlich asymptotisch exakt ist (Droste, Jansen und Wegener 1998b).

Satz 5.13. *Sei $n \in \mathbb{N}$ so gegeben, dass $\sqrt{n-1} \in \mathbb{N}$ gilt. Dann ist der lange $\sqrt{n-1}$-Pfad der Dimension n wohldefiniert und die erwartete Laufzeit des $(1+1)$ EA mit $p(n) = 1/n$ auf der Funktion $f_{P,\sqrt{n-1}}\colon \{0,1\}^n \to \mathbb{R}$ beträgt $\Theta\left(n^{3/2}2^{\sqrt{n}}\right)$.*

Beweis. Wenn $n \in \mathbb{N}$ und $\sqrt{n-1} \in \mathbb{N}$ gilt, so haben wir $(n-1)/\sqrt{n-1} = \sqrt{n-1} \in \mathbb{N}$ und der lange $\sqrt{n-1}$-Pfad der Dimension n ist wohldefiniert. Für die erwartete Laufzeit des (1+1) EA mit $p(n) = 1/n$ liefert Satz 5.12 die obere Schranke $O\left(n^{3/2}2^{\sqrt{n}}\right)$, wie wir uns schon überlegt haben. Wir müssen also nur die korrespondierende untere Schranke nachweisen. Dazu teilen wir einen Lauf des (1+1) EA auf $f_{\mathrm{P},\sqrt{n-1}}$ wie im Beweis zu Satz 5.12 in zwei Phasen mit Länge T_1 und T_2. Für die erste Phase genügt uns die triviale untere Schranke $\mathrm{E}\left(T_1\right) \geq 0$. Wir überlegen uns aber, dass mit Wahrscheinlichkeit mindestens $1/4$ der erste Pfadpunkt, den der (1+1) EA erreicht, zu S_0 oder B_k^n gehört. Alle Punkte in S_0 beginnen mit dem Teilstring 0^k, alle Punkte in S_1 mit dem Teilstring 1^k. Nach der zufälligen Initialisierung haben wir mit Wahrscheinlichkeit mindestens $1/2$ unter den ersten k Bits des initialen

Strings mindestens $k/2$ Nullen. So lange der aktuelle String des (1+1) EA x nicht auf dem Pfad liegt, kann ein Kind y nur dann akzeptiert werden, wenn $y \in P_k^n$ gilt oder y unter den ersten k Bits mindestens so viele mit Wert Null hat wie x. Daraus folgt, dass die Wahrscheinlichkeit, den i-ten Punkt in S_0 zu erreichen, mindestens genauso groß ist wie die Wahrscheinlichkeit, den $(l_1 - i)$-ten Punkt in S_1 zu erreichen, wobei $l_1 = |S_1|$ gilt. Daraus folgt die Behauptung.

Wir suchen jetzt eine untere Schranke für $\mathrm{E}(T_2)$. Dabei gehen wir davon aus, dass am Anfang der zweiten Phase der aktuelle String x zu S_0 oder B_k^n gehört. Sei v_j ein beliebiger Pfadpunkt in P_k^n. Nur die Punkte $v_j, v_{j+1}, \ldots, v_l$ haben mindestens so großen Funktionswert unter $f_{\mathrm{P},\sqrt{n-1}}$ wie v_j; wir verwenden dabei $l = |P_k^n|$. Wenn der aktuelle String $x = v_j$ und das Kind $y = v_m$ mit $m \geq j$ sind, so sagen wir, dass der (1+1) EA einen Qualitätsgewinn $m - j$ macht. Wir behaupten, dass man den erwarteten Qualitätsgewinn unabhängig von j (für ausreichend große Werte von n) durch $2/n$ nach oben abschätzen kann. Dazu benutzen wir die Pfadeigenschaften, wie wir sie in Lemma 5.10 beschrieben haben. Für alle r mit $0 < r < k$ gilt, dass die Wahrscheinlichkeit, dass der (1+1) EA Qualitätsgewinn r macht, durch

$$\left(\frac{1}{n}\right)^r \left(1 - \frac{1}{n}\right)^{n-r} \leq \frac{1}{n^r}$$

nach oben beschränkt ist. Um einen Qualitätsgewinn von k oder mehr zu machen, müssen mindestens $k = \sqrt{n-1}$ Bits gleichzeitig mutieren. Die Wahrscheinlichkeit dafür ist durch

$$\binom{n}{k} \left(\frac{1}{n}\right)^k \leq \frac{n^k}{k!} \left(\frac{1}{n}\right)^k = \frac{1}{k!}$$

nach oben beschränkt. Der mögliche Qualitätsgewinn in diesem Fall ist natürlich durch $|P_k^n|$ nach oben beschränkt. Insgesamt können wir also den erwarteten Qualitätsgewinn durch

$$\sum_{1 \leq r < k} \frac{r}{n^r} + \frac{1}{k!} \cdot |P_k^n|$$

nach oben abschätzen. Wir benutzen die Stirlingsche Formel und haben

$$k! > \left(\frac{k}{e}\right)^k = 2^{\Omega(k \log k)},$$

mit $k = \sqrt{n-1}$ also $k! = 2^{\Omega(\sqrt{n} \log n)}$ und damit einerseits

$$\frac{1}{k!} \cdot |P_k^n| = 2^{-\Omega(\sqrt{n} \log n)} \cdot O\left(\sqrt{n} \cdot 2^{\sqrt{n}}\right) = 2^{-\Omega(\sqrt{n} \log n)}.$$

Andererseits ist

$$\sum_{1\leq r<k}\frac{r}{n^r}=\sum_{1\leq r<k}\sum_{r\leq j<k}\frac{1}{n^j}<\sum_{1\leq r<k}\left(\sum_{0\leq j<\infty}\frac{1}{n^j}-\sum_{0\leq j<r}\frac{1}{n^j}\right)$$
$$=\sum_{1\leq r<k}\left(\frac{1}{1-1/n}-\frac{1-(1/n)^r}{1-1/n}\right)<\frac{1}{1-1/n}\sum_{1\leq r<\infty}\frac{1}{n^r}=\frac{1}{n-1}+\frac{1}{(n-1)^2},$$

also haben wir insgesamt

$$\frac{1}{n-1}+\frac{1}{(n-1)^2}+2^{-\Omega\left(\sqrt{n}\log n\right)}$$

als obere Schranke für den erwarteten Qualitätsgewinn, was für hinreichend große n durch $2/n$ nach oben abgeschätzt werden kann.

Wir nehmen an, dass am Anfang der zweiten Phase der aktuelle Zustand x zu S_0 oder B_k^n gehört. Dann ist bis zum Erreichen des globalen Optimums ein Qualitätsgewinn von mindestens

$$|S_1|=\left|P_k^{n-k}\right|\geq k2^{(n-k-1)/k}\geq c\sqrt{n}2^{\sqrt{n}}$$

erforderlich für eine positive Konstante c. Der erwartete Qualitätsgewinn in $(1/4)cn^{3/2}2^{\sqrt{n}}$ Generationen ist durch $(1/2)c\sqrt{n}2^{\sqrt{n}}$ nach oben beschränkt. Anwendung der Markov-Ungleichung ergibt, dass die Wahrscheinlichkeit, dass das Optimum nach $(1/4)cn^{3/2}2^{\sqrt{n}}$ Generationen erreicht ist, durch $1/2$ nach oben beschränkt ist. Das gibt insgesamt

$$\frac{1}{4}\cdot\frac{1}{2}\cdot\frac{1}{4}cn^{3/2}2^{\sqrt{n}}=\Omega\left(n^{3/2}2^{\sqrt{n}}\right)$$

als untere Schranke für die erwartete Laufzeit. □

Wir wollen die Erkenntnisse, die wir in diesem Kapitel gewonnen haben, unter der Perspektive „Klassifikation“ , die wir in Kapitel 2 eingenommen haben, noch einmal rekapitulieren. Die empirisch gewonnene Erfahrung, dass das Verweilen in lokalen Optima evolutionären Algorithmen einer erfolgreichen und effizienten Optimierung im Wege steht, gibt Anlass zur Vermutung, dass unimodale Funktionen tatsächlich leicht zu optimieren sind. Wir haben hier gesehen, dass die Klasse der unimodalen Funktionen keine hilfreiche deskriptive Klassifikation ist: sie enthält sowohl vollständig nicht-triviale lineare Funktionen, die der (1+1) EA in erwarteter Zeit $\Theta(n\log n)$ optimiert, als auch sehr schwierige Funktionen wie $f_{\mathrm{P},\sqrt{n-1}}$, bei denen der (1+1) EA im Durchschnitt exponentiell lange zur Optimierung braucht. Andererseits haben wir eingesehen, dass der (1+1) EA mit $p(n) = 1/n$ jede unimodale Funktion in erwarteter Zeit $O(n2^n)$ optimiert. Im nächsten Kapitel werden wir sehen, dass das tatsächlich substanziell schneller ist als bei schwierigsten Funktionen, von denen wir uns jetzt zwei näher ansehen wollen.

6 Einfache Worst Case Beispiele

In diesem Kapitel beschäftigen wir uns mit schwierigsten Funktionen. Ganz im Sinne unseres bisherigen Vorgehens sehen wir die Schwierigkeit einer Funktion durch die erwartete Anzahl Generationen die der (1+1) EA mit $p(n) = 1/n$ zur Optimierung braucht, bestimmt. Wir werden uns zunächst klar machen, dass diese erwartete Laufzeit anhand eines einfachen Arguments nach oben beschränkt werden kann. Dann werden wir uns zwei sehr unterschiedliche Beispielfunktionen ansehen, die beide diese maximale erwartete Laufzeit tatsächlich asymptotisch erfordern (Droste, Jansen und Wegener 1998a). Wir werden aber gerade an diesen beiden Funktionen, die unter dem Aspekt der erwarteten Laufzeit von asymptotisch gleicher Schwierigkeit sind, erkennen, dass speziell in diesem Bereich eine differenziertere Betrachtung hilfreich und dem Gegenstand angemessener ist.

Satz 6.1. *Die erwartete Anzahl Generationen, bis der* (1+1) *EA mit* $p(n) = 1/n$ *ein globales Maximum einer Funktion* $f\colon \{0,1\}^n \to \mathbb{R}$ *erreicht, ist durch* n^n *nach oben beschränkt.*

Beweis. Für alle $x, y \in \{0,1\}^n$ gilt $H(x,y) \leq n$. Folglich gilt für alle $x \in \{0,1\}^n$, dass die Wahrscheinlichkeit, ausgehend von x durch Mutation mit Mutationswahrscheinlichkeit $p(n) = 1/n$ ein Kind y zu erzeugen, das unter f global maximal ist, durch $(1/n)^n$ nach unten beschränkt ist. Folglich ist die erwartete Anzahl Generationen, bis ein solches Kind y erzeugt wird, durch n^n nach oben beschränkt. □

Der Beweis von Satz 6.1 macht deutlich, dass die obere Schranke n^n extrem pessimistisch ist. Selbst wenn eine Funktion $f\colon \{0,1\}^n \to \mathbb{R}$ nur ein einziges globales Maximum hat, gibt es nur einen einzigen Punkt, der Hammingabstand n von diesem Maximum hat. Alle anderen Punkte haben höchstens Hammingabstand $n-1$. Wir verdeutlichen dieses Argument und betrachten eine Funktion $g\colon \{0,1\}^n \to \mathbb{R}$, für die mindestens zwei verschiedene Punkte $m_1, m_2 \in \{0,1\}^n$ global maximal sind. Dann ist die erwartete Laufzeit des (1+1) EA mit $p(n) = 1/n$ auf g durch $2n^{n-1}$ nach oben beschränkt: Für alle $x \in \{0,1\}^n$ gilt $\min\{H(x,m_1), H(x,m_2)\} \leq n-1$. Die Wahrscheinlichkeit ausgehend von x ein Kind y zu erzeugen, das entweder mit m_1 oder m_2 identisch ist (also unter g global maximal ist), ist folglich durch $(1/n)^{n-1}(1-1/n) \geq (1/2)(1/n)^{n-1}$ nach unten beschränkt. Darauf folgt wie im Beweis von Satz 6.1 die obere Schranke $2n^{n-1}$. Unter dieser Perspektive ist man eher geneigt anzunehmen, dass es keine Funktion gibt, für die der (1+1) EA asymptotisch tatsächlich $\Theta(n^n)$ Schritte benötigt. Aber die

Intuition täuscht. Wir werden sogar zwei ganz unterschiedliche Funktionen kennenlernen, für die der (1+1) EA im Durchschnitt $\Theta(n^n)$ Generationen zum Erreichen des Optimums braucht. Klar ist, dass solche Funktionen eine eindeutige Maximalstelle haben müssen und dass das bitweise Komplement dieser Maximalstelle — der einzige Punkt mit Hammingabstand n zu dieser Maximalstelle — mit Wahrscheinlichkeit $\Omega(1)$ erreicht werden muss.

Definition 6.2. *Die Funktion* $f_T\colon \{0,1\}^n \to \mathbb{R}$ *ist definiert durch*

$$f_T(x) := \left(\sum_{1\leq i\leq n} x_i\right) + (n+1)\left(\prod_{1\leq i\leq n} (1-x_i)\right)$$

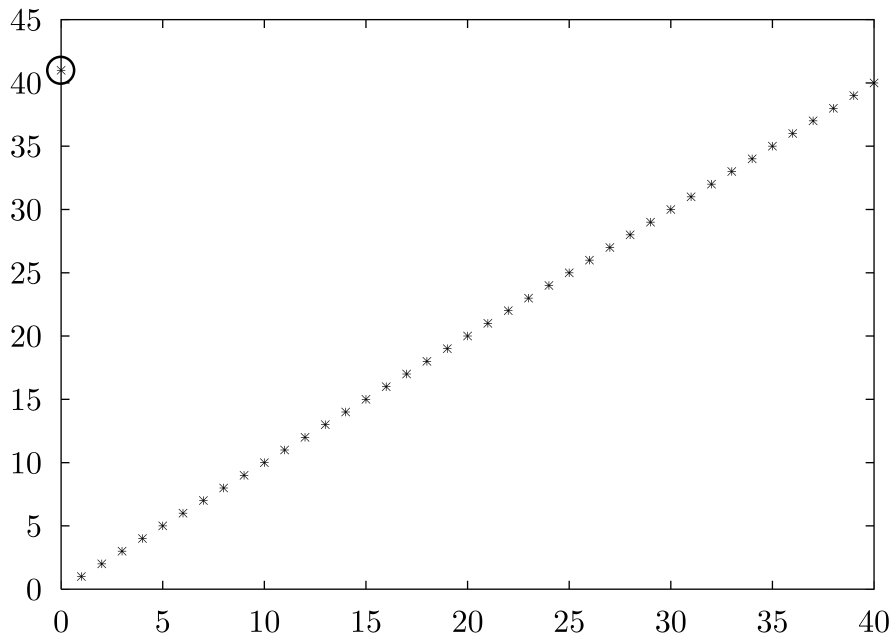

Abbildung 2: Die Funktion $f_{\mathrm{T}}\colon \{0,1\}^{40} \to \mathbb{R}$.

Die Funktion f_{T}, die wir für $n = 40$ in Abbildung 2 dargestellt finden, wird mitunter auch Trap genannt, was den fallenartigen Charakter der Funktion gut beschreibt. Offenbar stimmt f_{T} auf allen Punkten $x \in \{0,1\}^n \setminus \{\mathbf{0}\}$ mit der linearen Funktion f_1, die einem String x als Funktionswert die Anzahl Einsen in x zuweist, überein. Nur für den Nullstring $\mathbf{0}$ wird abweichend von f_1 der Funktionswert $n+1$ angenommen, die Funktion f_{T} hat hier also ihr globales Maximum. Wir wissen aus Satz 3.4, dass der (1+1) EA auf der Funktion f_1 in Zeit $O(n \log n)$ mit großer Wahrscheinlichkeit den Einsstring

1 erreicht. Solange der (1+1) EA nicht zufällig direkt mit $x = \mathbf{0}$ initialisiert oder ausgehend vom aktuellen String x durch Mutation $y = \mathbf{0}$ erzeugt, verhält sich der Algorithmus auf f_{T} wie auf f_1. Die Funktion f_{T} gibt also Suchalgorithmen nur „Hinweise", die in Richtung des lokalen Optimums **1** und weg vom einzigen globalen Optimum **0** führen. Solche Funktionen werden gelegentlich auch irreführend (deceptive) genannt. Wir dürfen also annehmen, dass das lokale Optimum **1** vom (1+1) EA mit $p(n) = 1/n$ rasch gefunden wird. Dann haben aber alle Strings außer dem globalen Optimum **0** echt kleineren Funktionswert, so dass nur noch eine Mutation von allen n Bits gleichzeitig akzeptiert wird. Wir vermuten darum, dass die erwartete Laufzeit des (1+1) EA mit $p(n) = 1/n$ auf der Funktion f_{T} $\Omega(n^n)$ beträgt. Dass diese Vermutung zutrifft, zeigt der nächste Satz.

Satz 6.3. *Die erwartete Anzahl Generationen, bis der* (1+1) *EA mit* $p(n) = 1/n$ *das globale Optimum der Funktion* $f_T\colon \{0,1\}^n \to \mathbb{R}$ *erreicht, beträgt* $\Theta(n^n)$. *Die Wahrscheinlichkeit, dass der* $(1+1)$ *EA mit* $p(n) = 1/n$ *das globale Optimum innerhalb der ersten* $n^{O(1)}$ *Generationen erreicht, ist durch* $e^{-\Omega(n)}$ *nach oben beschränkt.*

Beweis. Die Wahrscheinlichkeit, dass nach zufälliger Initialisierung der aktuelle String x weniger als $n/4$ Einsen enthält, ist durch $e^{-n/16} = e^{-\Omega(n)}$ nach oben beschränkt. Das globale Maximum **0** kann nur durch eine simultane Mutation aller Bits, die den Wert Eins haben, erreicht werden. Wenn der aktuelle String wenigstens $n/4$ Einsen enthält, müssen also mindestens $n/4$ Bits gleichzeitig mutieren. Die Wahrscheinlichkeit für das Erreichen des globalen Optimums in einem Schritt ist folglich durch $(1/n)^{n/4}$ nach oben beschränkt. Die Wahrscheinlichkeit, dass das in $n^{O(1)}$ Schritten geschieht, ist durch $n^{O(1)} \cdot (1/n)^{n/4} = e^{-\Omega(n \log n)}$ nach oben beschränkt. Wir wissen aus dem Beweis zu Satz 3.4, dass die Wahrscheinlichkeit, dass der (1+1) EA das globale Optimum **1** der linearen Funktion f_1 nicht in $4en \ln n$ Schritten findet, nach oben durch $1/2$ beschränkt ist. Solange das globale Maximum **0** nicht erreicht ist, verhält sich der (1+1) EA auf der Funktion f_{T} wie auf f_1. Folglich ist die Wahrscheinlichkeit, dass das lokale Optimum **1** von f_{T} nicht in $O(n^2 \log n)$ Schritten erreicht wird, durch $e^{-\Omega(n)}$ nach oben beschränkt. Wir sehen also, dass mit Wahrscheinlichkeit $1 - e^{-\Omega(n)}$ das lokale Optimum **1** erreicht wird. Dann wartet man auf eine Mutation aller n Bits gleichzeitig. Die Wahrscheinlichkeit für eine solche Mutation beträgt $(1/n)^n$. Folglich ist die erwartete Laufzeit des (1+1) EA auf der Funktion f_{T} nach unten durch

$$\left(1 - e^{-\Omega(n)}\right) \cdot n^n = \Omega(n^n)$$

beschränkt. Die obere Schranke folgt aus Satz 6.1. Die Wahrscheinlichkeit, dass die abschließende simultane Mutation aller n Bits in $n^{O(1)}$ Schritten passiert, ist nach oben durch $e^{-\Omega(n \log n)}$ beschränkt. Damit haben wir bewiesen, dass insgesamt mit Wahrscheinlichkeit $1 - e^{-\Omega(n)}$ zum Erreichen des Optimums $n^{O(1)}$ Schritte nicht ausreichen. □

Zur Optimierung der Funktion f_{T} braucht der (1+1) EA mit $p(n) = 1/n$ also mit überwältigender Wahrscheinlichkeit $\Theta(n^n)$ Generationen. Wir wissen außerdem, dass die durchschnittliche Laufzeit durch n^n nach oben beschränkt ist.

Als zweites Beispiel betrachten wir nun eine Funktion, bei der die erwartete Laufzeit des (1+1) EA mit $p(n) = 1/n$ ebenfalls $\Theta(n^n)$ beträgt. Allerdings werden wir sehen, dass hier nicht mit überwältigender Wahrscheinlichkeit tatsächlich so viele Schritte benötigt werden. Im Gegenteil, man hat eine gute Chance, dass der (1+1) EA die Funktion effizient optimiert. Um einen wichtigen strukturellen Unterschied zwischen den beiden Funktionen erkennen zu können, kommen wir nochmal auf die Polynomdarstellung einer Funktion $f\colon \{0,1\}^n \to \mathbb{R}$ zurück, die wir in Kapitel 2 schon angesprochen hatten.

Jede Funktion $f\colon \{0,1\}^n \to \mathbb{R}$ kann als multivariates Polynom

$$f(x) = \sum_{I \subseteq \{1,2,\ldots,n\}} c_f(I) \cdot \prod_{i \in I} x_i$$

dargestellt werden mit Koeffizienten $c_f(I) \in \mathbb{R}$. Man überlegt sich leicht, dass die Darstellung eindeutig ist. Unter dem Grad einer Funktion verstehen wir den Grad des Polynoms in dieser eindeutigen Polynomdarstellung, wobei der Grad $\deg(f)$ durch

$$\deg(f) := \max \{|I| \mid (I \subseteq \{1,2,\ldots,n\}) \wedge c_f(I) \neq 0\}$$

definiert ist. In diesem Sinne haben also alle konstanten Funktionen Grad 0, alle nicht-konstanten linearen Funktionen Grad 1. Ähnlich wie bei NK-Funktionen könnte man jetzt auf die Idee kommen, dass der Grad einer Funktion eine geeignete Basis für eine deskriptive Klassifikation sein könnte. Und natürlich ist die Idee nicht sonderlich gut: schon die Optimierung von Polynomen mit Grad 2 ist NP-hart. Unklar ist aber zum einen, wie eine vom Grad 2, die für den (1+1) EA schlecht zu optimieren ist, konkret aussieht. Zum anderen ist offen, wie schwierig die Optimierung von Funktionen vom Grad 2 für den (1+1) EA konkret sein kann. Zwar folgt aus der sehr verbreiteten Vermutung, dass NP≠RP gilt, dass der (1+1) EA nicht mit vernünftiger

Erfolgswahrscheinlichkeit eine schwierige Funktion vom Grad 2 in Polynomialzeit optimieren kann. Es ist aber unklar, wie groß die erwartete Laufzeit des (1+1) EA für eine Funktion vom Grad 2 tatsächlich werden kann. Es ist zum Beispiel grundsätzlich denkbar, dass es für Funktionen vom Grad 2 eine allgemeine obere Schranke für die erwartete Laufzeit des (1+1) EA gibt, die kleiner ist als n^n, etwa $O(2^n)$. Das nächste Beispiel beantwortet beide Fragen. Es zeigt, dass es sogar Funktionen vom Grad 2 gibt, die in dieser Beziehung zu den schwierigsten Funktionen gehören können. Wir geben also explizit eine solche Funktion an.

Definition 6.4. *Die Funktion* $f_Q\colon \{0,1\}^n \to \mathbb{R}$ *ist definiert durch*

$$f_Q(x) := \left(\sum_{1 \le i \le n} x_i - \left(\frac{n}{2} + \frac{1}{3} \right) \right)^2$$

für alle $x \in \{0,1\}^n$.

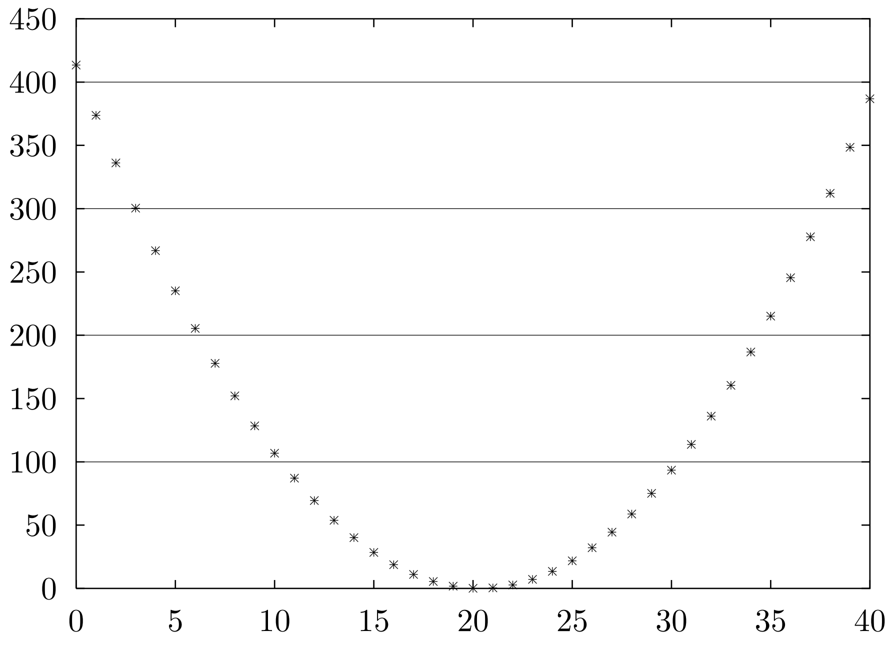

Abbildung 3: Die Funktion $f_{\mathrm{Q}}\colon \{0,1\}^{40} \to \mathbb{R}$.

Die Funktion f_{Q}, die für $n = 40$ in Abbildung 3 dargestellt ist, wird gelegentlich auch Distance genannt. Offenbar ist f_{Q} eine symmetrische Funktion, die leicht als Quadrat einer linearen Funktion beschrieben werden kann.

Dass sie bis auf die geringfügige aber wichtige Verschiebung dem Quadrat der Funktion f_1 gleicht, erklärt die Bezeichung f_{Q}. Offenbar hat also f_{Q} in der Polynomdarstellung nur Grad 2. Die Funktion f_{T} hingegen hat Grad n. Man erkennt, dass f_{Q} ein eindeutiges globales Maximum im Nullstring $\mathbf{0}$ hat, der Einsstring $\mathbf{1}$ ist das einzig andere lokale Maximum und nur wenig schlechter. Die „Idee der Funktion“ ist grob gesprochen die folgende. Mit zufälliger Initialisierung startet man irgendwo „in der Mitte“, also bei etwa $n/2$ Einsen. Dann kann man sich verbessern, indem man entweder die Anzahl der Einsen zunehmen oder abnehmen lässt. In beiden „Richtungen“ nimmt der Funktionswert streng monoton zu. Hat man sich einmal für eine „Seite“, also für streng monotone Verbesserung des Funktionswertes durch Zunahme oder Abnahme der Anzahl der Einsen „entschieden“, wird es zunehmend schwieriger, die Seite zu wechseln. Anfangs sind beide Wahlen etwa gleich wahrscheinlich, darum findet man mit Wahrscheinlichkeit etwa 1/2 rasch das globale Optimum $\mathbf{0}$, aber eben auch mit Wahrscheinlichkeit etwa 1/2 das lokale Optimum $\mathbf{1}$. Von da aus kann das globale Optimum $\mathbf{0}$ nur durch simultanes Mutieren aller n Bits gefunden werden, was im Durchschnitt n^n Generationen dauert. Wir konkretisieren diese Überlegungen im nächsten Satz.

Satz 6.5. *Die erwartete Anzahl Generationen, bis der* $(1+1)$ *EA mit* $p(n) = 1/n$ *das globale Optimum der Funktion* $f_Q\colon \{0,1\}^n \to \mathbb{R}$ *erreicht, beträgt* $\Theta(n^n)$. *Für jede Konstante* $\varepsilon > 0$ *gilt: Mit Wahrscheinlichkeit mindestens* $(1/2) - o(1)$ *braucht der* $(1+1)$ *EA mit* $p(n) = 1/n$ $\Theta(n^n)$ *Generationen, um das globale Optimum von* f_Q *zu finden. Mit Wahrscheinlichkeit mindestens* $(1/2) - \varepsilon$ *findet der* $(1+1)$ *EA mit* $p(n) = 1/n$ *das globale Optimum von* f_Q *in* $O(n \log n)$ *Generationen.*

Beweis. Wir beginnen mit einer genauen Untersuchung des aktuellen Strings x direkt nach der Initialisierung. Die Anzahl der Einsen in x ist binomialverteilt mit Parametern n und $1/2$. Folglich beträgt die erwartete Anzahl Einsen $n/2$, die Wahrscheinlichkeit, genau k Einsen in diesem ersten String zu haben, beträgt

$$\binom{n}{k}\left(\frac{1}{2}\right)^n.$$

Wir nehmen ohne Beschränkung der Allgemeinheit an, dass n gerade ist. Dann wird diese Wahrscheinlichkeit für $k = n/2$ maximal und wir sehen mit Hilfe der Stirlingschen Formel, dass

$$\binom{n}{n/2}2^{-n} = \frac{n!}{2^n \cdot ((n/2)!)^2} \leq \frac{\sqrt{3\pi n} \cdot n^n}{2^n \cdot (\pi n) \cdot (n/2)^n} = O\left(\frac{1}{\sqrt{n}}\right)$$

gilt. Folglich ist sowohl die Wahrscheinlichkeit, mit mehr als $(n/2) + n^{1/4}$ Einsen im aktuellen String nach der Initialiserung zu starten als auch die Wahrscheinlichkeit, dass dieser String weniger als $(n/2) - n^{1/4}$ Einsen enthält, nach unten beschränkt durch

$$\frac{1}{2} - \frac{n^{1/4}}{\sqrt{n}} = \frac{1}{2} - o(1).$$

Wenn die Anzahl der Einsen im aktuellen String wenigstens um $n^{1/4}$ von $n/2$ verschieden ist, dann kann ein Wechsel auf „die andere Seite“ nur noch stattfinden, wenn mindestens $2n^{1/4}$ Bits gleichzeitig mutieren. Die Wahrscheinlichkeit für eine solche Mutation ist durch

$$\left(\frac{1}{n}\right)^{2n^{1/4}} = e^{-\Omega\left(n^{1/4}\log n\right)}$$

nach oben beschränkt. Die Wahrscheinlichkeit, dass solch eine Mutation in n^2 Generationen nicht vorkommt, ist folglich durch $n^2 \cdot e^{-\Omega(n^{1/4}\log n)} = e^{-\Omega(n^{1/4}\log n)}$ nach oben beschränkt.

Wir betrachten die Situation, in der die Anzahl Einsen im aktuellen String schon bei der Initialisierung um mindestens $n^{1/4}$ von $n/2$ abweicht und mindestens n^2 Schritte keine Mutation von mindestens $2n^{1/4}$ Bits gleichzeitig vorkommt. Dann bleibt der aktuelle String x in der Hälfte des Suchraums, zu der die initiale String gehört. In diesem Fall verhält sich die Funktion f_{Q} im wesentlichen wie die lineare Funktion f_1. Folglich ist die erwartete Anzahl Generationen, bis das lokale Maximum der Suchraumhälfte gefunden wird, durch $O\left(n \log n\right)$ nach oben beschränkt.

Wenn der initiale String mindestens $(n/2)+n^{1/4}$ Einsen enthält, ist die Wahrscheinlichkeit, nach n^2 Generationen noch nicht das lokale Optimum $\mathbf{1}$ gefunden zu haben, durch $O\left((\log n)/n\right)$ nach oben beschränkt. Daraus folgt, dass die erwartete Laufzeit durch

$$\left(1 - \left(\left(\frac{1}{2} + o(1)\right) + O\left(\frac{\log n}{n}\right)\right)\right) \cdot n^n = \Omega\left(n^n\right)$$

nach unten beschränkt ist und mit Wahrscheinlichkeit $(1/2) - o(1)$ auch n^n Generationen zur Optimierung benötigt werden.

Wenn der initiale String höchstens $(n/2) - n^{1/4}$ Einsen enthält, können wir aus dem Beweis zu Satz 3.4 folgern, dass die Wahrscheinlichkeit, nach

$$\frac{2}{\varepsilon} \cdot 2en \ln n$$

Generationen noch nicht das globale Optimum $\mathbf{0}$ gefunden zu haben, durch

$$1 - \left(\left(\frac{1}{2} + o(1) \right) + e^{-\Omega\left(n^{1/4} \log n\right)} + \frac{\varepsilon}{2} \right) \leq \frac{1}{2} - \varepsilon$$

für hinreichend große n nach unten beschränkt ist. □

Die unterschiedliche Wahrscheinlichkeit, mit der dem (1+1) EA auch eine polynomielle Anzahl von Generationen ausreicht, um ein globales Optimum zu finden, ist auch von praktischer Bedeutung, wie wir uns schon in der Einleitung (Kapitel 1) überlegt hatten. Es ist absolut hoffnungslos, die Funktion f_{T} für nicht sehr kleine n mit dem (1+1) EA optimieren zu wollen. Man kann nicht erwarten, in akzeptabler Zeit das globale Optimum zu finden, auch wenn man den Algorithmus oft neu startet. Bei der Funktion f_{Q} ist das anders. Man hat bei jedem Lauf gute Chancen, das globale Optimum zu finden. Die Wahrscheinlichkeit, es nach beispielsweise zehn Neustarts nicht wenigstens einmal innerhalb Zeit $O(n \log n)$ gefunden zu haben, ist für nicht zu kleine Werte von n deutlich kleiner als 1%. Wir haben also in f_{Q} ein konkretes Beispiel für eine Funktion, bei der die erwartete Laufzeit ein irreführendes Schwierigkeitsmaß ist. Offenbar ist f_{Q} trotz exponentiell großer erwarteter Laufzeit praktisch gut optimierbar. Wenn man in die Analyse die Möglichkeit von Neustarts, also unabhängigen Wiederholungen, die in der Praxis üblich sind, mit einbezieht, wird deutlich, dass die Funktion f_{Q} in erwarteter Zeit $O(n \log n)$ optimiert werden kann. Das setzt allerdings voraus, dass eine Schranke $cn \ln n$ mit einem nicht zu kleinem Wert für die Konstante $c > 0$ für die Anzahl Generationen, nach denen ein Abbruch und anschließender Neustart erfolgt, verwendet wird. Die Verwendung kleinerer Schranken verringert drastisch die Wahrscheinlichkeit, dass in einem Lauf auch nur ein lokales Optimum gefunden wird. Wird die Schranke substanziell zu klein gewählt, wächst dadurch die erwartete Laufzeit. Wählt man andererseits eine Schranke, die zu groß ist, also eine Schranke $\omega(n \log n)$, so wächst die erwartete Laufzeit sofort auf die Größe dieser Schranke. In der Praxis ist also bei nicht zu kleinen Werten von n eher nicht damit zu rechnen, dass f_{Q} in erwarteter Zeit $O(n \log n)$ optimiert wird. Trotzdem ist klar, dass f_{Q} zu den praktisch gut optimierbaren Funktionen gehört, während das bei f_{T} nicht der Fall ist. Wir stellen fest, dass es zum Nachweis der Schwierigkeit einer Funktion nicht ausreicht, untere Schranken für die erwartete Laufzeit anzugeben. Es ist auch erforderlich abzuschätzen, mit welcher Wahrscheinlichkeit in einem Durchlauf eine akzeptable Anzahl von Generationen zur Optimierung ausreicht.

7 Variationen der Mutation

Wir haben bis jetzt den (1+1) EA auf einer Anzahl verschiedener Zielfunktionen und Zielfunktionsklassen untersucht, wobei wir uns fast ausschließlich auf die Wahl $p(n) = 1/n$ für die Mutationswahrscheinlichkeit beschränkt haben. Wir haben an den verschiedenen Beispielen einerseits strukturelles Wissen über den (1+1) EA erworben, von dem wir hoffen, dass es allgemeine Gültigkeit und eine gewisse Verallgemeinerbarkeit besitzt, die bei der Untersuchung komplexerer evolutionärer Algorithmen, die neben Mutation und Selektion noch andere Operatoren im Laufe der evolutionären Suche einsetzen, hilfreich ist. Andererseits haben wir auch verschiedene Methoden und Techniken kennengelernt, um untere und obere Schranken für die erwartete Laufzeit des (1+1) EA auf einer Zielfunktion mit bestimmten Eigenschaften nachzuweisen. Wir werden im folgenden sehen, dass sich die Anwendbarkeit dieser Techniken nicht auf den (1+1) EA mit $p(n) = 1/n$ beschränkt, sondern auch bei der Analyse von evolutionären Algorithmen, die als Varianten des (1+1) EA beschrieben werden können, gegeben ist. Wir haben mit Ende des vorangegangenen Kapitels die Untersuchung des (1+1) EA keineswegs beendet. Wir werden den (1+1) EA mit Mutationswahrscheinlichkeit $p(n) = 1/n$ auch in diesem und den folgenden Kapiteln weiterhin untersuchen und einige Resultate diskutieren, die wir bis jetzt noch nicht hergeleitet haben. Dabei dient uns dieser einfache evolutionäre Algorithmus, der uns schon recht vertraut geworden ist, als Referenz und Vergleichsmaßstab.

Es gibt verschiedene Möglichkeiten, den (1+1) EA zu modifizieren und zu erweitern, um so zu allgemeineren Kenntnissen über evolutionäre Algorithmen zu kommen. Die beiden vom (1+1) EA eingesetzten Suchoperatoren sind Mutation und Selektion. Wir werden uns in diesem Kapitel mit Varianten der Mutation beschäftigen, also den bisher gesteckten Rahmen der bitweisen Mutation mit fester Mutationswahrscheinlichkeit $p(n) = 1/n$ verlassen. Im Kapitel 8 betrachten wir die Selektion, die beim (1+1) EA einfach deterministisch im wesentlichen auf Basis der Funktionswerte durchgeführt wird. Für evolutionäre Algorithmen wesentliche Suchoperatoren und Konzepte, die der (1+1) EA gar nicht verwendet, sind eine echte Population, die aus mehr als nur einem einzigen Individuum besteht und der Suchoperator Crossover, der aus mehr als nur einem Elter ein Kind erzeugt. Wir werden beide Konzepte zusammen in Kapitel 9 ansehen.

Ein wichtiger Aspekt, den wir bisher völlig unberücksichtigt gelassen haben, ist die Verwendung von Zufallszahlen in evolutionären Algorithmen. Viele unserer Resultate aus den vorangegangenen Kapiteln beruhen wesentlich darauf, dass in jeder Generation n voneinander unabhängige Zufallsexperimente durchgeführt werden, für jedes Bit des aktuellen Strings x ein Experiment, in dem entschieden wird, ob das Bit im Kind y den gleichen Wert wie im Elter x hat oder den invertierten Wert. Genauso wesentlich ist es im allgemeinen für unsere Ergebnisse, dass alle $(T+1) \cdot n$ Zufallsexperimente, die insgesamt in T Generationen durchgeführt werden, vollständig unabhängig sind. In praktischen Anwendungen, in Implementierungen randomisierter Algorithmen, stehen in aller Regel keine "echten Zufallszahlen" zur Verfügung. Wir wollen uns hier nicht auf die eher philosophische Diskussion, was echte Zufallszahlen sind und wie man sie prinzipiell erzeugen kann, einlassen. Wir stellen aber fest, dass in der Praxis in aller Regel an Stelle echter Zufallszahlen so genannte Pseudozufallszahlen verwendet werden, die durch einen deterministischen Algorithmus erzeugt werden. Wir werden uns hier ebenfalls nicht mit dem Problem der Erzeugung von Pseudozufallszahlen und den zahlreichen verschiedenen Tests, denen man sie unterziehen kann, um ihre Qualität zu beurteilen, beschäftigen. Es ist aber nicht von der Hand zu weisen, dass die Erzeugung guter Pseudozufallszahlen rechenaufwendig ist und die Laufzeit so einfacher evolutionärer Algorithmen, wie es beispielsweise der (1+1) EA ist, bei Optimierung einfach auszuwertender Zielfunktionen ganz klar von der Zeit dominiert wird, die für die Erzeugung der Pseudozufallszahlen erforderlich ist. Darum ist es gut nachzuvollziehen und nicht verwunderlich, dass in der Praxis der Wunsch entsteht, das erforderliche Maß an Zufälligkeit, konkret also die Anzahl notwendiger Zufallsexperimente, erheblich zu reduzieren. Ein einfacher Weg dorthin ist die Modifikation der Mutation. Bei der bitweisen Mutation mit Mutationswahrscheinlichkeit $p(n) = 1/n$, wie wir sie in Algorithmus 3.1 beschrieben haben, sind n unabhängige Zufallsexperimente erforderlich, um festzustellen, wieviele Bits und welche Bits mutieren. Dabei beträgt die erwartete Anzahl mutierender Bits $n \cdot p(n) = n \cdot (1/n) = 1$. Es liegt darum nahe, eine einfachere Mutation zu verwenden, die nur noch auf einem einzigen an Stelle von n unabhängigen Zufallsexperimenten beruht. In jeder Mutation wählt man genau ein Bit aus, dessen Wert geändert wird. Die Wahl dieses Bits geschieht zufällig gleichverteilt. Wir halten den Algorithmus der Klarheit halber formal fest, verzichten allerdings darauf, ihm einen eigenen Namen zu geben.

Algorithmus 7.1.

1. *Wähle* $x \in \{0,1\}^n$ *zufällig gleichverteilt.*
2. $y := x$
   Wähle $i \in \{1, 2, \ldots, n\}$ *zufällig gleichverteilt.*
   Ersetze das i*-te Bit in* y *durch sein Komplement.*
3. *Falls* $f(y) \geq f(x)$ *gilt, setze* $x := y$
4. *Weiter bei* 2.

Weil Algorithmus 7.1 in jeder Generation das tut, was der (1+1) EA mit Mutationswahrscheinlichkeit $p(n) = 1/n$ im erwarteten Fall tut, nämlich genau ein gleichverteilt zufällig gewähltes Bit mutieren, kann man die Hoffnung haben, dass die beiden Algorithmen sich im wesentlichen gleich verhalten. Wir machen uns zunächst klar, dass das im allgemeinen nicht der Fall sein kann. Dazu erinnern wir uns zunächst daran, dass die erwartete Anzahl Generationen, bis der (1+1) EA mit $p(n) = 1/n$ ein globales Optimum einer Funktion $f\colon \{0,1\}^n \to \mathbb{R}$ erreicht, unabhängig von f auf jeden Fall durch n^n nach oben beschränkt ist (Satz 6.1). Wir können einen Lauf der Länge T als

$$\left\lfloor \frac{T}{2n^n} \right\rfloor$$

unabhängige Wiederholungen von Läufen der Länge $2n^n$ betrachten. Darum ist die Wahrscheinlichkeit, dass nach T Generationen, der aktuelle Zustand x des (1+1) EA bei Wahl der Mutationswahrscheinlichkeit $p(n) = 1/n$ unter f nicht global maximal ist, durch $2^{-T/(2n^n)}$ nach oben beschränkt.

Satz 7.2. *Sei $f\colon \{0,1\}^n \to \mathbb{R}$ eine Funktion, so dass es $m_1, m_2 \in \{0,1\}^n$ gibt mit folgenden Eigenschaften:*

1. $f(m_1) = \max\{f(x) \mid x \in \{0,1\}^n\}$
2. $f(m_1) > f(m_2)$
3. $\forall x \in \{0,1\}^n : H(x, m_2) = 1 \Rightarrow f(x) < f(m_2)$

Dann ist für alle $T \in \mathbb{N}$, die Wahrscheinlichkeit, dass für den aktuellen Zustand x von Algorithmus 7.1 nach T Generationen noch nicht $f(x) = f(m_1)$ gilt, durch 2^{-n} nach unten beschränkt.

Beweis. Die Funktion f ist offenbar nicht unimodal, sie besitzt mindestens zwei lokale Optima m_1 und m_2, von denen nur m_1, nicht aber m_2 auch global maximal ist. Mit Wahrscheinlichkeit 2^{-n} gilt nach der zufälligen Initialisierung in Zeile 1 von Algorithmus 7.1 $x = m_2$. Dann gilt zu jedem Zeitpunkt $x = m_2$. Andernfalls muss in einer Mutation in Zeile 2 ein Kind y erzeugt werden, für das $f(y) \geq f(x) = f(m_2)$ gilt. Es können aber nur Kinder y erzeugt werden, für die $H(x, y) = H(m_2, y) = 1$ gilt. Für alle Strings mit y mit $H(y, m_2) = 1$ gilt nach Voraussetzung $f(y) < f(m_2)$. Also wird m_2 nie verlassen. □

Wir bemerken zunächst, dass zwar alle Funktionen, welche die Bedingungen in Satz 7.2 erfüllen, nicht unimodal sind. Es ist aber nicht so, dass auch umgekehrt alle Funktionen, die nicht unimodal sind, auch notwendig die Bedingungen von Satz 7.2 erfüllen. Ein triviales Beispiel sind konstante Funktionen, die gemäß Definition 5.1 2^n lokale Maxima haben. Natürlich findet auch Algorithmus 7.1 direkt bei der Initialisierung ein globales Maximum einer konstanten Funktion. Wir werden gleich sehen, dass es aber auch nicht-triviale Funktionen gibt, die nicht unimodal sind, aber dennoch von Algorithmus 7.1 effizient optimiert werden.

Wenn wir von unimodalen Funktionen absehen, ist die Frage nach der erwarteten Laufzeit von Algorithmus 7.1 im allgemeinen nicht sinnvoll. Es gibt für die in Satz 7.2 charakterisierten Funktionen für jede Laufzeitschranke T eine nicht-verschwindende Wahrscheinlichkeit von $\Omega(2^{-n})$, dass die Laufzeit größer als T ist. Dieser wesentliche Unterschied zwischen Algorithmus 7.1 und dem (1+1) EA mit Mutationswahrscheinlichkeit $p(n) = 1/n$ (Algorithmus 3.1) bedeutet aber nicht, dass die beiden Algorithmen auf allen Funktionen völlig verschiedenes Verhalten zeigen. Wir untersuchen im folgenden einige unimodale und auch einige nicht-unimodale Funktionen und werden sehen, dass in gewissen Bereichen deutliche Ähnlichkeit im Verhalten beider Algorithmen zu erkennen ist.

Satz 7.3. *Algorithmus 7.1 hat auf der Klasse der linearen Funktionen erwartete Laufzeit $O(n \log n)$. Algorithmus 7.1 hat auf der Klasse der vollständig nicht-trivialen linearen Funktionen erwartete Laufzeit $\Theta(n \log n)$.*

Beweis. Wir beginnen mit der oberen Schranke und ignorieren Bits, die Gewicht $w_i = 0$ haben. Solche Bits können beliebigen Wert haben und das Verhalten von Algorithmus 7.1 nicht beeinflussen. Sie spielen darum für den Nachweis oberer Schranken keine Rolle. Für den Beweis der oberen Schranke nehmen wir darum im folgenden ohne Beschränkung der Allgemeinheit an, dass $w_i \neq 0$ für alle $i \in \{1, 2, \ldots, n\}$ gilt. Weil in jeder Generation genau ein Bit mutiert (also insbesondere niemals mehrere Bits gleichzeitig mutieren), können wir leicht folgende Invarianz feststellen. Hat ein Bit an Position i am aktuellen String x den gleichen Wert wie im eindeutigen globalen Maximum, so ändert sich dieser Wert in nachfolgenden Generationen nicht. Wir sagen, dass Bits, die den gleichen Wert haben wie im globalen Maximum, den korrekten Wert haben. Wenn genau i Bits im aktuellen String schon ihren korrekten Wert haben, so beträgt die Wahrscheinlichkeit, dass eines der übrigen $n - i$ Bits zur Mutation ausgewählt wird $(n - i)/n$. In diesem Fall erhöht sich die Anzahl der Bits mit korrektem Wert um 1. Anfangs ist die Anzahl korrekt gesetzter Bits durch 0 nach unten beschränkt, insgesamt

n Erhöhungen der Anzahl korrekt gesetzter Bits reichen sicher aus, um das globale Maximum zu erreichen. Wir haben darum

$$\sum_{0 \leq i < n} \frac{n}{n-i} = n \sum_{1 \leq i \leq n} \frac{1}{i} \leq n(1 + \ln n) = O\left(n \log n\right)$$

als obere Schranke für die erwartete Laufzeit.

Die untere Schranke für vollständig nicht-triviale lineare Funktionen ergibt sich im Grunde analog. Mit Wahrscheinlichkeit mindestens $1/2$ haben nach zufälliger Initialisierung mindestens $\lfloor n/2 \rfloor$ Bits nicht den korrekten Wert. Die erwartete Zeit, bis ein weiteres Bits den korrekten Wert hat, beträgt $(n-i)/n$, wenn i die Anzahl korrekt gesetzter Bits im aktuellen Zustand x bezeichnet. Weil in jeder Generation genau 1 Bit mutiert, sind auch tatsächlich so viele Erhöhungen der Anzahl Bits mit korrektem Wert nötig, wie anfangs Bits mit nicht korrekt gesetztem Wert vorhanden sind. Also haben wir

$$\frac{1}{2} \sum_{n/2 < i < n} \frac{n}{n-i} = \frac{n}{2} \sum_{1 \leq i < n/2} \frac{1}{i} > \frac{n}{2} \ln\left(\frac{n}{2} - 1\right) = \Omega(n \log n)$$

als untere Schranke in diesem Fall. □

Wir sehen, dass sich sowohl für die Klasse der linearen Funktionen als auch für die Klasse der vollständig nicht-trivialen Funktionen für Algorithmus 7.1 asymptotisch die gleiche Laufzeit ergibt wie für den (1+1) EA mit $p(n) = 1/n$. Dabei ist festzustellen, dass lineare Funktionen, die nicht vollständig nicht-trivial sind (für die also nicht alle Gewichte w_i (mit $i > 0$) von Null verschieden sind), nicht zur Klasse der unimodalen Funktionen gehören. Es gibt jedoch trotz dieser augenscheinlichen Gleichheit zwei bemerkenswerte Unterschiede zwischen den beiden Algorithmen in dieser Beziehung. Zum einen ist der Beweis von Satz 7.3 substanziell einfacher als der Beweis von Satz 4.3. Im wesentlichen reduziert sich der Beweis für alle linearen Funktionen auf den Beweis für die extrem einfach zu analysierende lineare Funktion f_1 (mit $f_1(x) = \|x\|_1$), weil die wesentliche Eigenschaft, dass die Anzahl korrekt gesetzter Bits nicht abnehmen kann, unter Algorithmus 7.1 für alle linearen Funktionen gilt. Für den (1+1) EA ist das nicht so. Eine Mutation von 01^{n-1} zu 10^{n-1} ist bei Optimierung der linearen Funktion f_{B} (mit $f_{\mathrm{B}}(x) = \sum 2^{n-i} x_i$) zwar sehr unwahrscheinlich, aber sie ist möglich. Der zweite bemerkenswerte Unterschied besteht darin, dass die obere Laufzeitschranke, die wir für Algorithmus 7.1 nachgewiesen haben, um den Faktor e kleiner ist, als für den (1+1) EA mit $p(n) = 1/n$ auf der Funktion f_1. Dieser Unterschied der oberen Schranken entspricht tatsächlich der Realität. Er erklärt sich aus dem Umstand, dass der (1+1) EA mit Wahrscheinlichkeit

$$(1 - p(n))^n = \left(1 - \frac{1}{n}\right)^n \approx \frac{1}{e}$$

gar kein Bit mutiert, also effektiv nichts tut. Das erhöht die erwartete Laufzeit um den Faktor e. In praktischen Implementierungen wird man natürlich, zumindest wenn die Auswertung der Zielfunktion aufwendig ist, wenigstens eine erneute Auswertung der Zielfunktion verhindern. Es ändert aber nichts an dem Umstand, dass eine Durchführung von n Zufallsexperimenten auch in diesem offenbar nutzlosen Fall im allgemeinen erforderlich ist.

Dass Beweise für Algorithmus 7.1 leichter zu führen sind als für den (1+1) EA, wird sich auch in den folgenden Beispielen zeigen. Wir betrachten die Funktionen f_{L}, die einem String $x \in \{0,1\}^n$ als Funktionswert die Anzahl führender Einsen zuweist, sowie die Funktion f_{R}, die wir in Kapitel 2 als Gegenbeispiel zur Klassifikation mittels Fitness Distance Correlation kennengelernt haben. Wir hatten uns schon überlegt, dass beide Funktionen unimodal sind.

Satz 7.4. *Die erwartete Laufzeit von Algorithmus 7.1 auf der Funktion* $f_L \colon \{0,1\}^n \to \mathbb{R}$ *ist* $\Theta(n^2)$.

Beweis. Genau wie im Beweis von Satz 5.7 überlegt man sich leicht, dass im aktuellen Zustand x alle Bits mit größerem Index als das Bit, das unter allen Bits mit Wert Null minimalen Index hat, rein zufällig sind. Für die obere Schranke verwendet man, dass n Mutationen, bei denen der Funktionswert wächst, zum Erreichen des globalen Optimums $\mathbf{1}$ ausreichen. Für die untere Schranke verwendet man, dass mit Wahrscheinlichkeit mindestens $1/2$ mindestens $\lfloor n/2 \rfloor$ Mutationen, bei denen der Funktionswert wächst, nötig sind. Die Wahrscheinlichkeit, den Funktionswert zu erhöhen, beträgt stets genau $1/n$, wenn nicht schon das globale Maximum erreicht ist. Das gibt n^2 als obere Schranke für die erwartete Laufzeit und $n^2/4$ als untere Schranke, zusammen also $\Theta(n^2)$ wie behauptet. □

Satz 7.5. *Die erwartete Laufzeit von Algorithmus 7.1 auf der Funktion* $f_R \colon \{0,1\}^n \to \mathbb{R}$ *ist* $\Theta(n^2)$.

Beweis. Für die obere Schranke überlegen wir uns kurz, dass die Funktion f_{R} sich wie eine lineare Funktion mit Gewichten $w_0 = 0$, $w_1 = w_2 = \cdots = w_n = -1$ verhält, solange der aktuelle String nicht von der Form $1^i 0^{n-i}$ ist. Wir folgern aus Satz 7.3, dass die erwartete Zeit bis zum Erreichen des

Nullstrings $\mathbf{0}$ durch $O(n \log n)$ nach oben beschränkt ist. Dann werden nur noch Strings der Form $1^i 0^{n-i}$ als neue aktuelle Zustände x akzeptiert, wobei der Funktionswert mit einer wachsenden Anzahl führender Einsen zunimmt. Wir schließen aus Satz 7.4, dass das globale Maximum $\mathbf{1}$ im Durchschnitt Nach $O(n^2)$ Schritten erreicht wird, so dass wir insgesamt $O(n^2)$ als obere Schranke für die erwartete Laufzeit haben.

Nach zufälliger Initialisierung gilt mit Wahrscheinlichkeit 1/4, dass der aktuelle Zustand x mit zwei Nullen beginnt, also $x_1 = x_2 = 0$ gilt. So lange der String nicht von der Form $1^i 0^{n-i}$ ist, kann ein Kind y nur dann als neuer aktueller Zustand x akzeptiert werden, wenn die Anzahl von Einsen in y echt kleiner ist als in x. Folglich wird als erster String der Form $1^i 0^{n-i}$ der Nullstring $\mathbf{0}$ erreicht. Wir folgern analog zum Beweis von Satz 7.4, dass die erwartete Anzahl Generationen bis zum Erreichen des globalen Optimums $\mathbf{1}$ dann n^2 beträgt, so dass wir $n^2/4 = \Omega(n^2)$ als untere Schranke haben. □

Als weiteres Beispiel für unimodale Funktionen hatten wir in Kapitel 5 lange k-Pfadfunktionen $f_{\mathrm{P},k}$ kennengelernt. Wir erinnern uns, dass Rudolph (1997a) zwei verschiedene obere Schranken für die erwartete Laufzeit des (1+1) EA mit $p(n) = 1/n$ auf $f_{\mathrm{P},k}$ bewiesen hat (Satz 5.12). Dabei hängt es von der Größe von k im Vergleich zu n ab, welche der beiden oberen Schranken die kleinere ist. Wir haben für eine spezielle Wahl von k, nämlich für $k = \sqrt{n-1}$, im Beweis zu Satz 5.13 recht aufwendig und mühevoll nachgewiesen, dass die kleinere untere Schranke asymptotisch scharf ist. Wir werden hier sehen, dass es für Algorithmus 7.1 substanziell leichter ist, untere Schranken zu beweisen.

Die zwei unterschiedlichen oberen Schranken für die erwartete Laufzeit des (1+1) EA auf $f_{\mathrm{P},k}$ beruhen auf zwei unterschiedlichen Ideen, wie der (1+1) EA sich verhält, wenn er erst einmal den langen k-Pfad P_k^n gefunden hat. Es besteht die Möglichkeit, dass er dem Pfad folgt und praktisch ausschließlich Mutationen erfolgreich sind, bei denen jeweils ein einzelnes Bit mutiert. Das führt zur oberen Schranke $O(n\,|P_k^n|)$. Andererseits kann der Algorithmus Abkürzungen auf dem Pfad beschreiten. Dazu ist es aber erforderlich, dass k Bits simultan mutieren. Diese Überlegung führt zur oberen Schranke $O\left(n^{k+1}/k\right)$. Algorithmus 7.1 kann natürlich keine Abkürzungen benutzen: in jeder Generation wird genau ein Bit zur Mutation ausgewählt. Es ist darum nicht verwunderlich, dass die auf Mutationen einzelner Bits beruhende obere Schranke für Algorithmus 7.1 als asymptotisch exakte Schranke nachgewiesen werden kann.

Satz 7.6. *Seien n und k so gegeben, dass der lange k-Pfad der Dimension n wohldefiniert ist. Die erwartete Laufzeit von Algorithmus 7.1 auf der Funktion $f_{P,k}\colon \{0,1\}^n \to \mathbb{R}$ ist $\Theta(n\,|P_k^n|)$.*

Beweis. Für die obere Schranke überlegen wir uns analog zum Beweis von Satz 7.5, dass im Durchschnitt nach $O\,(n \log n)$ Generationen jedenfalls der Pfadanfang erreicht ist. Dann sind noch $|P_k^n|$ Mutationen erforderlich, die den Funktionswert vergrößern. Dabei beträgt jeweils die Wahrscheinlichkeit für eine solche Mutation genau $1/n$. Daraus folgt insgesamt die obere Schranke $O\,(n \log n + n\,|P_k^n|) = O\,(n\,|P_k^n|)$.

Für die untere Schranke ist es hilfreich, sich die rekursive Definition des langen k-Pfades der Dimension n basierend auf den zwei „Pfadkopien" S_0 und S_1 sowie den Brückenpunkten B_k^n wieder ins Gedächtnis zu rufen (Definition 5.8). Mit Wahrscheinlichkeit $1/4$ beginnt der aktuelle String x nach der zufälligen Initialisierung mit zwei Bits, die beide den Wert Null haben, also $x_1 = x_2 = 0$. Wenn x ein Punkt des Pfades ist, so liegt x wegen dieser Nullbits sicher nicht in S_1. Gehört x nicht zum Pfad, so schätzen wir die Anzahl Schritte bis zum ersten Erreichen eines Pfadpunktes nach unten mit 0 ab, kümmern uns also gar nicht um diese Phase. Wir interessieren uns aber dafür, welcher Punkt als erster Pfadpunkt erreicht wird. Es mutiert in jeder Generation genau ein Bit. Folglich kann von x aus nicht in einer Generation ein Punkt aus S_1 erreicht werden. Wird nach einer Mutation das Kind y akzeptiert, ersetzt also y sein Elter, so ist y entweder Pfadpunkt oder hat wiederum zwei erste Bits mit Wert Null, also $y_1 = y_2 = 0$ in diesem Fall. Wir sehen, dass als erster Pfadpunkt kein Punkt aus S_1 erreicht wird. Wir haben also bis jetzt insgesamt bewiesen, dass mindestens mit Wahrscheinlichkeit $1/4$ ein Punkt aus $S_0 \cup B_k^n$ erster Punkt des Pfades ist, der besucht wird. Das liefert $|S_1|/4 = \Omega\,(|S_1|)$ als untere Schranke für die erwartete Laufzeit. Gemäß Konstruktion ist $|S_0| = |S_1| \geq |B_k^n|$, also $|S_1| \geq |P_k^n|/3$. Das liefert insgesamt $|P_k^n|/12 = \Omega\,(|P_k^n|)$ als untere Schranke für die erwartete Länge des Teilpfades, die zurückzulegen ist. Das ist gleich der Anzahl von Mutationen, die zu Pfadnachfolgern führen. Die Wahrscheinlichkeit für eine solche Mutation ist auf dem Pfad stets genau $1/n$, wiraus insgesamt die untere Schranke $\Omega\,(n\,|P_{k^n}|)$ für die erwartete Laufzeit folgt. $\square$

Es ist klar, dass bei Algorithmus 7.1 eine Untersuchung der Wahrscheinlichkeit, mit der innerhalb einer gewissen Anzahl Generationen T ein globales Optimum einer Funktion f gefunden wird, mindestens ebenso sinnvoll ist wie für den (1+1) EA. Es folgt beispielsweise direkt aus dem Beweis zu Satz 6.5

zusammen mit Satz 7.3, dass Algorithmus 7.1 zum einen das globale Optimum der Funktion f_{Q} (Definition 6.4) mit Wahrscheinlichkeit mindestens $(1/2) - o(1)$ in $O(n \log n)$ Generationen findet, zum anderen aber auch mit Wahrscheinlichkeit mindestens $(1/2) - o(1)$ das globale Maximum der Funktion f_{Q} überhaupt nicht findet. Man kann offensichtlich auch leicht Funktionsfamilien $f_k\colon \{0,1\}^n \to \mathbb{R}$ finden, bei denen sich die Erfolgswahrscheinlichkeit von Algorithmus 7.1 anhand des Parameters k einstellen lässt. Betrachten wir etwa kurz die Funktionsfamilie $f_k\colon \{0,1\}^n \to \mathbb{R}$, die wir durch

$$f_k(x) := \begin{cases} \|x\|_1 & \text{falls } \|x\|_1 \neq k \\ -1 & \text{sonst} \end{cases}$$

für alle $x \in \{0,1\}^n$ definieren. Der Parameter $k \in \{0, 1, \ldots, n-1\}$ stellt die Schwierigkeit der Funktion ein. Wir betrachten zunächst kurz den (1+1) EA mit $p(n) = 1/n$ auf f_k. Wir können einen Lauf des (1+1) EA gedanklich in drei Phasen unterteilen. Grob gesprochen wird in der ersten Phase die „Sprungstelle“, also ein String mit genau $k-1$ Bits mit Wert Eins erreicht, in der zweiten Phase wird diese Sprungstelle überwunden, in der dritten Phase wird schließlich das eindeutige globale Optimum $\mathbf{1}$ erreicht. Formal definieren wir drei Phasen, deren Längen wir mit T_1, T_2 bzw. T_3 bezeichnen. Die erste Phase beginnt mit der zufälligen Initialisierung und endet, wenn für den aktuellen String x des (1+1) EA erstmals $f_k(x) \geq k-1$ gilt. Die zweite Phase beginnt direkt mit Ende der ersten Phase und endet, wenn für den aktuellen String x des (1+1) EA erstmals $f_k(x) > k-1$ gilt. Die dritte Phase schließlich beginnt direkt mit Ende der ersten Phase und endet, wenn erstmalig $x = \mathbf{1}$ gilt.

In der ersten und dritten Phase gleicht die Funktion f_k „aus Sicht des Algorithmus“ der linearen Funktion f_1. Darum gilt $\mathrm{E}(T_1) + \mathrm{E}(T_3) = O(n \log n)$ für alle Werte von k. Um die Gesamtlaufzeit $\mathrm{E}(T_1 + T_2 + T_3)$ nach oben abzuschätzen, unterscheiden wir zwei Fälle.

1. Fall: $k \leq (n/2) - \sqrt{2n \ln n}$

Wir wollen zeigen, dass in diesem Fall des Überwinden der Sprungstelle die Laufzeit nicht wesentlich bestimmt. Es ist klar, dass $\mathrm{E}(T_2) = O(n^2)$ für jeden Wert von k gilt, weil es zur Überwindung der Sprungstelle sicher genügt, dass genau zwei Bits mit Wert Null gleichzeitig mutieren, was Wahrscheinlichkeit $\Omega(1/n^2)$ hat. Die Wahrscheinlichkeit, dass der aktuelle String x direkt nach der zufälligen Initialisierung höchstens $k-1$ Einsen enthält, die Sprungstelle also überhaupt überwunden werden muss, lässt sich durch Anwendung der

Chernoff-Schranken nach oben durch

$$\begin{aligned}
\operatorname{Prob}\left(\|x\|_1 < k\right) &\leq \operatorname{Prob}\left(\|x\|_1 < \frac{n}{2} - \sqrt{2n\ln n}\right)\\
&= \operatorname{Prob}\left(\|x\|_1 < \frac{n}{2}\left(1 - 2\sqrt{\frac{2\ln n}{n}}\right)\right)\\
&< e^{-(n/2)\left(2\sqrt{2(\ln n)/n}\right)^2/2} = e^{-2\ln n} = \frac{1}{n^2}
\end{aligned}$$

beschränken, so dass die Laufzeit in diesem Fall durch

$$O\left(n\log n\right) + \left(1/n^2\right)\cdot O\left(n^2\right) = O\left(n\log n\right)$$

nach oben beschränkt ist.

2. Fall: $k > (n/2) - \sqrt{2n\ln n}$

Um die Sprungstelle zu überwinden, genügt es, genau 2 von den $n-(k-1)$ Bits mit Wert Null gleichzeitig zu mutieren. Die Wahrscheinlichkeit für dieses Ereignis beträgt $\Omega\left((n-k)^2/n^2\right)$, das liefert uns $\mathrm{E}\left(T_2\right) = O\left((n/(n-k))^2\right)$ und folglich

$$O\left(n\log n + \left(\frac{n}{n-k}\right)^2\right)$$

als Schranke für die erwartete Laufzeit des (1+1) EA in diesem Fall.

Wir haben also insgesamt für $k \leq n-\sqrt{n/\log n}$ erwartete Laufzeit $O\left(n\log n\right)$ und erwartete Laufzeit $O\left((n/(n-k))^2\right)$ sonst nachgewiesen.

Wir wollen jetzt aber lieber das Verhalten von Algorithmus 7.1 auf der Funktion f_k untersuchen, um das es uns ja eigentlich geht. Es gibt grundsätzlich drei verschiedene Möglichkeiten. Entweder gilt nach der Initialisierung für den aktuellen String $\|x\|_1 < k$, $\|x\|_1 = k$ oder $\|x\|_1 > k$. Wenn $\|x\|_1 > k$ gilt, bleibt diese Eigenschaft erhalten, das globale Maximum $\mathbf{1}$ wird dann im erwarteten Fall innerhalb von $O\left(n\log n\right)$ Schritten erreicht. Gilt andererseits $\|x\|_1 < k$, so bleibt auch diese Eigenschaft erhalten. Die Anzahl der Einsen im aktuellen String x wird im Durchschnitt in $O\left(n\log n\right)$ Generationen bis $k-1$ zunehmen. Eine weitere Zunahme ist dann nicht mehr möglich, weil die Anzahl Einsen sich in jeder Generation nur um 1 verändern kann und $f_k(y) = -1$ für alle y mit $\|y\|_1 = k$ gilt. Im Fall $\|x\|_1 = k$ für das initiale x wird in der allerersten Mutation ein Kind y erzeugt, das entweder eine um 1 größere oder eine um 1 kleine Anzahl Einsen hat als das Elter x. In beiden Fällen gilt $f_k(y) > f_k(x)$, so dass y auf jeden Fall neuer aktueller String x

wird. Danach haben wir entweder $\|x\|_1 < k$ oder $\|x\|_1 > k$ und wir können argumentieren wie vorhin. Wir haben also gezeigt, dass mit Wahrscheinlichkeit

$$\left(\sum_{k<i\leq n}\binom{n}{i}2^{-n}\right)+\binom{n}{k}\cdot 2^{-n}\cdot\frac{n-k}{n}$$

Algorithmus 7.1 das globale Optimum der Funktion f_k findet und dafür im erwarteten Fall $O(n \log n)$ Generationen braucht. Andererseits sehen wir, dass mit Wahrscheinlichkeit

$$\left(\sum_{0\leq i<k}\binom{n}{i}2^{-n}\right)+\binom{n}{k}\cdot 2^{-n}\cdot\frac{k}{n}$$

Algorithmus 7.1 das globale Optimum der Funktion f_k überhaupt nicht findet. Die Erfolgswahrscheinlichkeit lässt sich also wie behauptet durch Wahl des Parameters k einstellen.

Wir beenden hiermit unsere Untersuchungen von Algorithmus 7.1, dessen Einführung rein praktisch motiviert war. Wir werden auf die darin verwendete Mutation allerdings in Kapitel 8 noch einmal zurückkommen. Jetzt wenden wir uns anderen Variationen der bitweisen Mutation zu, die auch von praktischem Interesse sind, bei denen unsere Überlegungen aber zunächst in starkem Maße theoretisch motiviert sind. Wir werden in einem ersten Schritt noch beim (1+1) EA bleiben, wie wir ihn als Algorithmus 3.1 definiert haben. Wir wenden uns jetzt aber der Wahl der Mutationswahrscheinlichkeit $p(n)$ zu, untersuchen also explizit andere Wahlen für die Mutationswahrscheinlichkeit als die übliche und am häufigsten empfohlene $p(n) = 1/n$.

Wir haben in Kapitel 3 gesehen, dass es gute Gründe gibt, die Festlegung $p(n) = 1/n$ für eine gute Wahl zu halten. Wir haben sogar gesehen, dass $p(n) = \Theta(1/n)$ für lineare Funktionen optimal ist (Satz 3.5 und Satz 3.8). Mitunter findet man die Vermutung, dass die Mutationswahrscheinlichkeit $p(n) = 1/n$ in allen Fällen optimal sein könnte. Wir haben allerdings nur für lineare Funktionen gezeigt, dass die Wahl $p(n) = 1/n$ günstig ist. Gerade für lineare Funktionen ist auch intuitiv klar, dass diese Wahl für die Mutationswahrscheinlichkeit geeignet ist. Schließlich können lineare Funktionen optimiert werden, indem jeweils nur ein Bit mutiert. Bei der Wahl von $p(n) = 1/n$ beträgt die erwartete Anzahl mutierender Bits in jeder Generation $p(n) \cdot n = (1/n) \cdot n = 1$. Also besteht gute Übereinstimmung zwischen der Funktionsklasse und der Wahl der Mutationswahrscheinlichkeit. Es ist darum gut denkbar, dass für andere Funktionen andere Wahlen für die Mutationswahrscheinlichkeit $p(n)$ günstiger sind.

Es ist leicht zu erkennen, dass wir auch hier schon Beispiele besprochen haben, bei denen die Wahl $p(n) = 1/n$ bei weitem nicht optimal ist. Wir verwenden dabei als Maß für die Güte einer Mutationswahrscheinlichkeit wie gewohnt die erwartete Laufzeit des (1+1) EA bei dieser Wahl. Wir haben für die Funktionen f_{T} und f_{Q} in Kapitel 6 nachgewiesen, dass die erwartete Laufzeit des (1+1) EA mit $p(n) = 1/n$ auf diesen Funktionen $\Theta(n^n)$ beträgt. Man sieht praktisch sofort, dass man das leicht unterbieten kann.

Satz 7.7. *Für jede Funktion $f\colon \{0,1\}^n \to \mathbb{R}$ ist die erwartete Laufzeit des $(1+1)$ EA mit $p(n) = 1/2$ durch $O(2^n)$ nach oben beschränkt. Für jede Funktion $f\colon \{0,1\}^n \to \mathbb{R}$, die ein eindeutiges globales Maximum hat, beträgt die erwartete Laufzeit des $(1+1)$ EA mit $p(n) = 1/2$ genau 2^n.*

Beweis. Bei Mutationswahrscheinlichkeit $p(n) = 1/2$ wird bei jeder Mutation für jedes Bit unabhängig zufällig mit Wahrscheinlichkeit $1/2$ entschieden, ob es seinen Wert beibehält oder wechselt. Das bedeutet, dass bei jeder Mutation das Kind y unabhängig von x zufällig gleichverteilt aus $\{0,1\}^n$ gewählt wird. Damit gilt für jeden Punkt $z \in \{0,1\}^n$, dass in jeder Generation mit Wahrscheinlichkeit $(1/2)^n$ genau z erzeugt wird, also $y = z$ gilt. Sei $z \in \{0,1\}^n$ so gewählt, dass $f(z) = \max\{f(x) \mid x \in \{0,1\}^n\}$ gilt. Dann beträgt die erwartete Anzahl Generationen, bis erstmals der aktuelle Zustand z ist, genau 2^n. Daraus folgt die obere Schranke $O(2^n)$ für alle Funktionen. Hat f nur genau ein globales Maximum, so ist z dieses Maximum und es folgt direkt 2^n als erwartete Laufzeit. □

Die Wahl $p(n) = 1/2$ senkt also die erwartete Laufzeit für die Funktionen f_{T} und f_{Q} von $\Theta(n^n)$ auf 2^n. Das ist eine Verbesserung um einen Faktor $\Theta((n/2)^n)$, was zunächst beeindruckend klingt. Für f_{T} kann man unter Umständen tatsächlich von einer Verbesserung sprechen. Für die Funktion f_{Q} ist es aber fraglich, ob $p(n) = 1/2$ tatsächlich eine bessere Wahl ist. Wir hatten gesehen, dass der (1+1) EA mit $p(n) = 1/n$ die Funktion f_{Q} mit Wahrscheinlichkeit mindestens $(1/2) - \varepsilon$ in Zeit $O(n \log n)$ optimiert, wobei ε eine beliebige positive Konstante ist. Die Wahrscheinlichkeit, dass der (1+1) EA mit $p(n) = 1/2$ die Funktion f_{Q} in Zeit $n^{O(1)}$ optimiert, ist offenbar durch $n^{O(1)} \cdot 2^{-n} = 2^{-\Omega(n)}$ nach oben beschränkt. Während die Funktion f_{Q}, wie wir uns in Kapitel 6 ausführlich überlegt haben, mit der Wahl $1/n$ für die Mutationswahrscheinlichkeit gut optimierbar ist, ist das für $p(n) = 1/2$ nicht mehr gegeben. Aus praktischer Sicht ist klar, dass beide Funktionen mit der Wahl $p(n) = 1/2$ nicht optimierbar sind. Exponentielle erwartete Rechenzeiten deuten für die Praxis an, dass die Funktion nicht in akzeptabler Zeit optimiert wird. Wenn dann noch mit Wahrscheinlichkeit sehr nahe bei 1 auch noch exponentielle Zeit benötigt wird (was bei f_{T} der Fall ist, nicht aber bei f_{Q}), so ist die Funktion in der Praxis so nicht optimierbar. Man kann darum aus praktischer Sicht zunächst weiterhin vermuten, dass die Wahl $p(n) = 1/n$ „immer optimal“ ist: ein „Gegenbeispiel“, bei dem die erwartete Rechenzeit in der Größenordnung von exponentiell auf „kleiner, aber immer noch exponentiell“ sinkt, hat keine praktische Relevanz. Es gilt also, entweder die Vermutung, dass $p(n) = 1/n$ eine für die Praxis immer vernünftige Wahl ist, zu verifizieren, oder ein Gegenbeispiel zu finden, das folgende Eigenschaften hat. Wählt man $1/n$ für die Mutationswahrscheinlichkeit des (1+1) EA, so ist die erwartete Rechenzeit des (1+1) EA nicht polynomiell nach oben beschränkt. Außerdem werden mit Wahrscheinlichkeit nahe bei 1 mehr als polynomiell viele Generationen zum Auffinden eines globalen Optimums benötigt. Für eine andere Wahl der Mutationswahrscheinlichkeit sinkt die erwartete Rechenzeit hingegen auf einen polynomiellen Wert. In der Tat werden wir jetzt ein solches Gegenbeispiel angeben und nachweisen, dass es die erforderlichen Eigenschaften hat, die es zu einem auch praktisch relevanten Gegenbeispiel für die Vermutung macht, dass $p(n) = 1/n$ eine immer gute Wahl ist.

Der Einfachheit halber definieren wir die Funktion, die uns als Gegenbeispiel dient, nur für Werte von n, die Zweierpotenzen sind. Man kann durch geeignete Rundungen die Definition auf jeden nicht zu kleinen Wert von n ausdehnen. Wir verzichten auf diese Möglichkeit, die zu einer komplizierteren Notation führt, ohne substanziellen Gewinn zu bringen. Die Funktion, die wir f_{M} nennen, und die gelegentlich auch PATHTOJUMP genannt wird (Jansen und Wegener 2000b), ist nur für $n \geq 32$ definiert. Diese untere Schranke für die Dimension n garantiert, dass $n/4 > \log n$ gilt, was für die Definition wesentlich ist.

Definition 7.8. *Sei $n = 2^k > 16$ mit $k \in \mathbb{N}$. Wir definieren zunächst eine Partition des Suchraums $\{0,1\}^n$ in fünf disjunkte Mengen.*

$$
\begin{aligned}
P_1 &:= \{x \in \{0,1\}^n \mid n/4 < \|x\|_1 < 3n/4\} \\
P_2 &:= \{x \in \{0,1\}^n \mid \|x\|_1 = n/4\} \\
P_3 &:= \left\{x \in \{0,1\}^n \mid \exists i \in \{0,1,\dots,(n/4)-1\} \colon x = 1^i 0^{n-i}\right\} \\
P_4 &:= \left\{x \in \{0,1\}^n \mid (\|x\|_1 = \log n) \wedge \left(\sum_{1 \leq i \leq 2\log n} x_i = 0\right)\right\} \\
P_0 &:= \{0,1\}^n \setminus (P_1 \cup P_2 \cup P_3 \cup P_4)
\end{aligned}
$$

Die Funktion $f_M\colon \{0,1\}^n \to \mathbb{R}$ *ist definiert durch*

$$f_M(x) := \begin{cases} n - \|x\|_1 & \textit{falls } x \in P_1, \\ (3/4)n + \sum\limits_{1 \leq i \leq n/4} x_i & \textit{falls } x \in P_2, \\ 2n - i & \textit{falls } x \in P_3 \textit{ und } x = 1^i 0^{n-i}, \\ 2n+1 & \textit{falls } x \in P_4, \\ \min\{\|x\|_1, n - \|x\|_1\} & \textit{falls } x \in P_0, \end{cases}$$

für alle $x \in \{0,1\}^n$*.*

Wir verwenden hier wieder die Einteilung des Suchraums $\{0,1\}^n$ in Partitionen und betrachten hier direkt P_0, P_1, P_2, P_3, P_4. Wir machen uns zunächst klar, dass

$$P_0 <_{f_{\mathrm{M}}} P_1 <_{f_{\mathrm{M}}} P_2 <_{f_{\mathrm{M}}} P_3 <_{f_{\mathrm{M}}} < P_4$$

gilt. Für alle $x \in P_0$ haben wir

$$f_{\mathrm{M}}(x) = \min\{\|x\|_1, n - \|x\|_1\} < \frac{n}{4},$$

weil alle Strings mit mehr als $n/4$ Einsen und weniger als $3n/4$ Einsen zu P_1 und alle Strings mit genau $n/4$ Einsen zu P_2 gehören. Damit ist auch klar, dass für alle $x \in P_1$

$$f_{\mathrm{M}}(x) = n - \|x\|_1 \in \left\{\frac{n}{4} + 1, \frac{n}{4} + 2, \ldots, \frac{3n}{4} - 1\right\}$$

und folglich $P_0 <_{f_{\mathrm{M}}} P_1$ gilt. Die Menge P_2 enthält genau alle Strings mit genau $n/4$ Einsen, als Funktionswerte haben wir für alle $x \in P_2$ darum

$$f_{\mathrm{M}}(x) = \frac{3n}{4} + \sum_{1 \leq i \leq n/4} x_i \in \left\{\frac{3n}{4}, \frac{3n}{4} + 1, \ldots, n\right\}.$$

Daraus folgt $P_1 <_{f_{\mathrm{M}}} P_2$. Die Menge P_3 enthält alle Strings der Form $1^i 0^{n-i}$, wobei $0 \leq i < n/4$ gilt. Folglich haben wir

$$f_{\mathrm{M}}(x) = f_{\mathrm{M}}\left(1^i 0^{n-i}\right) = 2n - i \in \left\{\frac{7n}{4} + 1, \frac{7n}{4} + 2, \ldots, 2n\right\}$$

für alle $x \in P_3$, was $P_2 <_{f_{\mathrm{M}}} P_3$ impliziert. Die Menge P_4 schließlich enthält alle Strings mit genau $\log n$ Einsen, für die zusätzlich gilt, dass auf den ersten $2 \log n$ Positionen kein Bit mit Wert Eins vorkommt. Für die Funktionswerte der Strings $x \in P_4$ haben wir einfach $f_{\mathrm{M}}(x) = 2n + 1$. Offensichtlich gilt also $P_3 <_{f_{\mathrm{M}}} P_4$. Weil wir $n \geq 32$ voraussetzen, haben wir $n/4 > \log n$, so dass P_2 und P_4 tatsächlich disjunkt sind. Dass alle anderen P_i paarweise disjunkt sind, ist offensichtlich.

Der Definition der Funktion f_{M} liegt folgender Gedanke zu Grunde. Die einzigen lokalen Maxima findet man in den Mengen P_3 und P_4. Alle $x \in P_4$ sind außerdem global maximal. Für alle anderen Punkte findet man mit Mutationswahrscheinlichkeit $1/n$ rasch einen Nachbarn, also einen Punkt mit Hammingabstand 1, der echt größeren Funktionswert hat. Auf diese Weise erreicht man leicht und rasch den einzigen Punkt in P_3, der lokal maximal ist, nämlich $\mathbf{0}$. Der Hammingabstand von $\mathbf{0}$ zu irgendeinem global maximalen Punkt, beträgt genau $\log n$. Wenn n nicht zu klein ist, haben auch alle Punkte aus $P_1 \cup P_2 \cup P_3$ wenigstens Hammingabstand $\log n$ zu jedem Punkt aus P_4. Das Überwinden eines so großen Hammingabstandes ist aber mit der Mutationswahrscheinlichkeit $1/n$ nicht einfach. An dieser Stelle ist eine substanziell größere Mutationswahrscheinlichkeit günstiger. Andererseits gehört der aktuelle String x des (1+1) EA nach zufälliger Initialisierung mit überwältigender Wahrscheinlichkeit zu P_1. Um erst einmal in die Nähe der globalen Maxima zu kommen, ist es dann aber vermutlich eher günstig, eine Mutationswahrscheinlichkeit in der Größenordnung $\Theta(1/n)$ zu wählen. Wir stützen diese Vermutung auf dem Umstand, dass die Funktion f_{M} sich eingeschränkt auf die Punkte aus P_1 wie eine lineare Funktion verhält. Es ist also durchaus nicht klar, ob eine substanziell andere Mutationswahrscheinlichkeit tatsächlich zu einer schnelleren Optimierung von f_{M} führt. Wir werden jetzt in einem ersten Schritt nachweisen, dass die Wahl $p(n) = 1/n$ mit großer Wahrscheinlichkeit nicht zu einer Optimierung in einer polynomiellen Anzahl von Generationen führt. Tatsächlich zeigen wir das nicht nur für die Wahl $p(n) = 1/n$ sondern für alle Mutationswahrscheinlichkeiten $p(n)$, die substanziell von $(\log n)/n$ verschieden sind.

Satz 7.9. *Sei $n = 2^k > 32$ mit $k \in \mathbb{N}$. Sei $\alpha\colon \mathbb{N} \to \mathbb{R}$ definiert durch*

$$\alpha(n) := p(n) \cdot \frac{n}{\log n}.$$

Falls $\alpha(n) \to 0$ für $n \to \infty$ oder $\alpha(n) \to \infty$ für $n \to \infty$ gilt, konvergiert die Wahrscheinlichkeit, dass der (1+1) EA mit Mutationswahrscheinlichkeit $p(n)$ für die Optimierung der Funktion $f_M\colon \{0,1\}^n \to \mathbb{R}$ eine superpolynomielle Anzahl von Generationen braucht, gegen 1.

Beweis. Die Wahrscheinlichkeit, dass der aktuelle String x direkt nach der Initialisierung zur Menge P_1 gehört, lässt sich durch Anwendung der Chernoff-

Schranken durch

$$1 - 2 \cdot \left(\frac{e^{1/2}}{(3/2)^{3/2}}\right)^{n/2} = 1 - e^{-\Omega(n)}$$

nach unten beschränken. Wir nehmen im folgenden an, dass der aktuelle String unmittelbar nach der Initialisierung zu $P_1 \cup P_2 \cup P_3$ gehört. Weil

$$P_0 <_{f_{\mathrm{M}}} P_1 <_{f_{\mathrm{M}}} P_2 <_{f_{\mathrm{M}}} P_3 <_{f_{\mathrm{M}}} P_4$$

gilt, wird dann niemals ein String aus P_0 erreicht. Nach Voraussetzung haben wir

$$\alpha(n) = p(n) \cdot \frac{n}{\log n}$$

und damit auch

$$p(n) = \frac{\log n}{n} \cdot \alpha(n).$$

Wir unterscheiden zwei Fälle.

1. Fall: $\alpha(n) \to \infty$ für $n \to \infty$, also $p(n) = \omega((\log n)/n)$
Wir betrachten nur die allerletzte Generation, in der erstmals durch Mutation mit Mutationswahrscheinlichkeit $p(n)$ ein Kind $y \in P_4$ erzeugt wird. Weil der aktuelle Zustand x zu diesem Zeitpunkt sicher nicht zu P_0 gehört, gilt $\|x\|_1 < 3n/4$. Wir betrachten $n/4$ Bits in x, die alle den Wert Null haben. Von diesen $n/4$ Bits mit dem Wert Null dürfen in dieser letzten Mutation höchstens $\log n$ viele Bits mutieren, andernfalls gilt $\|y\|_1 > \log n$, also $y \notin P_4$. Die erwartete Anzahl mutierender Bits unter diesen $n/4$ Bits beträgt

$$p(n) \cdot \frac{n}{4} = \frac{1}{4}\alpha(n) \log n,$$

was um den Faktor $\alpha(n)/4$ größer ist als $\log n$. Wir wenden die Chernoff-Schranken an und sehen, dass die Wahrscheinlichkeit, dass in einer Generation höchstens $\log n$ Bits von $n/4$ Bits mutieren nach oben durch

$$e^{-\left((1-4/\alpha(n))^2 \alpha(n) \log n\right)/8} = e^{-\Omega(\alpha(n) \log n)} = n^{-\Omega(\alpha(n))}$$

beschränkt ist für $\alpha(n) > 4$. Folglich ist die Wahrscheinlichkeit, in $n^{O(1)}$ Generationen ein globales Optimum zu finden nach oben durch $n^{O(1)} \cdot n^{-\Omega(\alpha(n))} = n^{-\Omega(\alpha(n)}$ beschränkt, was gegen 0 konvergiert. Es gibt sogar eine Konstante $c > 0$, so dass die Wahrscheinlichkeit, in $n^{c\alpha(n)}$ Generationen ein globales

Optimum zu finden, durch $n^{-c\alpha(n)}$ nach oben beschränkt ist. Folglich ist in diesem Fall auch die erwartete Laufzeit des (1+1) EA mit $p(n)$ auf der Funktion f_{M} durch $n^{\alpha(n)}$ nach unten beschränkt.

2. Fall: $\alpha(n) \to 0$ für $n \to \infty$, also $p(n) = o((\log n)/n)$
Wir betrachten wiederum nur die allerletzte Generation, in der erstmals ein Kind $y \in P_4$ durch Mutation mit Mutationswahrscheinlichkeit $p(n)$ erzeugt wird. Gemäß Voraussetzung gehört zu diesem Zeitpunkt der aktuelle String x zu $P_1 \cup P_2 \cup P_3$. Der Hammingabstand von x zu jedem Punkt $x' \in P_4$ ist durch $\log n$ nach unten beschränkt, wie wir uns schon überlegt haben. Wir haben hier (anders als in der Definition 7.8) $n \geq 64$ vorausgesetzt, was $n/4 > 2\log n$ impliziert. Alle Punkte in P_1 enthalten wenigstens $P_1 \cup P_2$ enthalten wenigstens $n/4$ Einsen, alle Punkte in P_4 enthalten genau $\log n$ Einsen. Der Hammingabstand ist also durch $(n/4) - \log n > \log n$ nach unten beschränkt. Alle Punkte in P_3 sind von der Form $0^i 1^{n-i}$. Gilt $i \geq \log n$, so enthält der Punkt aus P_3 auf den ersten $2\log n$ Bits mindestens $\log n$ Bits mit dem Wert Eins, während jeder Punkt aus P_4 gar kein Bit mit dem Wert Eins unter den ersten $2\log n$ Bits hat. Sei nun also $i < \log n$. Der String $1^i 0^{n-i} \in P_3$ hat unter den hinteren $n - 2\log n$ Bits kein Bit mit dem Wert Eins. Jeder String aus P_4 hat unter den hinteren $n - 2\log n$ Bits genau $\log n$ Bits mit dem Wert 1, so dass der Hammingabstand in diesem Fall durch $\log n$ nach unten beschränkt ist. Wir bemerken, dass die Voraussetzung $z_1 = z_2 = \cdots = z_{2\log n} = 0$ für alle $z \in P_4$ hier essenziell für die Argumentation ist.

Weil der Hammingabstand zwischen x und dem zu erzeugenden Kind y durch $\log n$ nach unten beschränkt ist, müssen wenigstens $\log n$ Bits gleichzeitig mutieren. Die erwartete Anzahl gleichzeitig mutierender Bits beträgt

$$p(n) \cdot n = \alpha(n) \log n,$$

was um den Faktor $\alpha(n)$ kleiner als ist als $\log n$. Wir wenden die Chernoff-Schranken an und sehen, dass die Wahrscheinlichkeit, dass in einer Generation mindestens $\log n$ Bits gleichzeitig mutieren, durch

$$\left(\frac{e^{(1/\alpha(n))-1}}{(1/\alpha(n))^{1/\alpha(n)}}\right)^{\alpha(n)\log n} = \left(e^{1-\alpha(n)} \cdot \alpha(n)\right)^{\log n} = \alpha(n)^{\Omega(\log n)} = n^{\Omega(\log \alpha(n))}$$

nach unten beschränkt ist. Wir erinnern uns daran, dass wir in diesem Fall $\alpha(n) \to 0$ für $n \to \infty$ voraussetzen. Wir haben also $\log \alpha(n) \to -\infty$ für $n \to \infty$. Wir folgern analog zum ersten Fall, dass es eine Konstante c mit $0 < c < \infty$ gibt, so dass mit Wahrscheinlichkeit $n^{c \log \alpha(n)}$ nach $n^{-c \log \alpha(n)}$ der (1+1) EA mit $p(n)$ noch kein globales Maximum der Funktion f_{M} erreicht hat und die erwartete Laufzeit durch $n^{-\Omega(\log \alpha(n))}$ nach unten beschränkt ist. $\square$

Wir haben jetzt gesehen, dass die Funktion f_{M} nicht leicht zu optimieren ist. Ist die Mutationswahrscheinlichkeit zu klein ($p(n) = o((\log n)/n)$, so dauert die erforderliche Mutation von mindestens $\log n$ Bits gleichzeitig zu lange. Man erreicht also dann mehr oder minder schnell das lokale Optimum $\mathbf{0}$, tut sich aber schwer mit dem abschließenden Sprung über $\log n$ Bits zu einem globalen Maximum. Ist andererseits die Mutationswahrscheinlichkeit zu groß ($p(n) = \omega((\log n)/n)$), so mutieren in jedem Schritt zu viele Bits mit, was schon die Annäherung an ein globales Maximum erschwert. Offen ist bis jetzt nur der Fall $p(n) = \Theta((\log n)/n)$, für den wir im folgenden nachweisen werden, dass er tatsächlich zu polynomiellen erwarteten Laufzeiten führt. Vorher wollen wir noch einige grundsätzliche Überlegungen anstellen, um die Beweisführung zu motivieren.

Wir vermuten, dass bei einem typischen Lauf des (1+1) EA auf f_{M} mit vernünftiger (vor allem nicht zu großer) Mutationswahrscheinlichkeit $p(n)$ das lokale Optimum $\mathbf{0}$ gefunden wird und von da aus ein direkter Sprung zu einem der globalen Maxima stattfindet. Dann ist also ein Hammingabstand von genau $\log n$ zu überwinden. Die Wahrscheinlichkeit für eine solche Mutation ist maximal für $p(n) = (\log n)/n$ wie wir uns später in etwas anderem Zusammenhang einmal genau überlegen werden (Beweis zu Satz 7.22). Es ist aber nicht unbedingt optimal, $p(n) = (\log n)/n$ zu wählen. Für die asymptotische erwartete Laufzeit spielt nicht nur die erwartete Anzahl Generationen, für die $x = \mathbf{0}$ gilt, eine Rolle. Es kommt auch darauf an, das globale Maximum $\mathbf{0}$ erst einmal zu erreichen. Wenn wir einmal grob vereinfachend annehmen, dass vor allem Mutationen einzelner Bits dafür hilfreich sind, haben wir mit

$$\frac{\log n}{n}\left(1 - \frac{\log n}{n}\right)^{n-1} \approx \frac{\log n}{n} e^{-\log n} = n^{-O(1)}$$

zwar noch eine polynomiell nach unten beschränkte Wahrscheinlichkeit für eine solche Mutation. Wir erkennen an dem Ausdruck $e^{-\log n}$, dass es zu einer etwas übersichtlicheren Notation beitragen kann, von $(\log n)/n$ zu $(\ln n)/n$ überzugehen. Es ist klar, dass Mutationswahrscheinlichkeiten kleiner als $(\ln n)/n$ zu größeren Wahrscheinlichkeit führen. Wählen wir $p(n) = (c \ln n)/n$ für eine (kleine) Konstante $c > 0$, so bleibt die Wahrscheinlichkeit für eine Mutation von $\log n$ Bits gleichzeitig polynomiell nach unten beschränkt, während die Wahrscheinlichkeit für solche Mutationen einzelner Bits größer wird. Wir haben also einen gewissen Trade Off. Wir werden versuchen, durch Wahl einer passenden Konstante $c > 0$ eine möglichst kleine obere Schranke für die erwartete Rechenzeit nachzuweisen.

Satz 7.10. *Sei* $n = 2^k > 32$ *mit* $k \in \mathbb{N}$. *Sei* $c \in \mathbb{R}$ *eine Konstante mit* $c > 0$. *Sei* $p(n) := (c \ln n)/n$. *Die erwartete Laufzeit des* $(1+1)$ *EA mit* $p(n)$ *auf der Funktion* $f_M\colon \{0,1\}^n \to \mathbb{R}$ *ist durch* $O\left(n^{2+c} \ln^{-1} n + n^{c - \log c - \log \ln 2}\right)$ *nach oben beschränkt. Für* $c = 1/(4 \ln 2) < 0{,}361$ *ist die erwartete Laufzeit durch* $O\left(n^{2{,}361}/\ln n\right)$ *nach oben beschränkt.*

Beweis. Im Beweis von Satz 7.9 haben wir ausgenutzt, dass die Partitionen $P_0, P_1, \ldots, P_4$ gleichzeitig Ranggruppen sind. Wir werden hier eine noch deutlich feinere Unterteilung in Ranggruppen vornehmen, wie sie für den Nachweis oberer Schranken für die erwartete Laufzeit des (1+1) EA generell nützlich ist. Wir definieren die Ranggruppen

$$Q_i := \{x \in \{0,1\}^n \mid f_{\mathrm{M}}(x) = i\}$$

für alle $i \in \{0, 1, \ldots, 2n+1\}$. Als wir uns weiter oben überlegt haben, dass die Partitionen $P_0, P_1, \ldots, P_4$ tatsächlich Ranggruppen sind, haben wir dabei implizit gezeigt, dass

$$\bigcup_{0 \leq i \leq 2n+1} Q_i = \{0,1\}^n$$

gilt. Offenbar gilt $Q_i <_{f_{\mathrm{M}}} Q_{i+1}$ für alle i mit $Q_i, Q_{i+1} \neq \emptyset$. Wir ignorieren im folgenden die leeren Mengen Q_i und bezeichnen zu einer Ranggruppe Q_i die Ranggruppe Q_j mit $j = \min\{k > i \mid Q_k \neq \emptyset\}$ als Nachfolger von Q_i. Offensichtlich hat jede nicht-leere Ranggruppe Q_i mit $i < 2n+1$ einen eindeutigen Nachfolger in diesem Sinne. Wir bemerken, dass wir diese im Grunde triviale Einteilung des Suchraums $\{0,1\}^n$ in Ranggruppen unter anderem schon im Beweis zu Satz 3.4 und im Beweis zu Satz 5.7 benutzt haben.

Wir werden jetzt für jede nicht-leere Ranggruppe Q_i mit $0 \leq i < 2n+1$ eine untere Schranke für die Wahrscheinlichkeit q_i angeben, in einer Generation von Q_i in eine Ranggruppe Q_j mit $j > i$ zu kommen. Wir haben dann $1/q_i$ als obere Schranke für die erwartete Zeit, um von Q_i in eine höhere Ranggruppe zu kommen und schließlich insgesamt

$$\sum_{0 \leq i \leq 2n} \frac{1}{q_i}$$

als obere Schranke für die Laufzeit des (1+1) EA auf f_{M}. Wir werden jeweils viele Ranggruppen Q_i zusammenfassen und gemeinsam behandeln, wobei wir uns im wesentlichen an der Partitionierung des Suchraums $\{0,1\}^n$ in die Mengen $P_0, \ldots, P_4$ orientieren, wie wir sie in Definition 7.8 vorgenommen haben.

1. Fall: $i \in \{0, 1, \ldots, (3/4)n - 1\}$
Wir betrachten also einen String x mit $0 \leq f_{\mathrm{M}}(x) < (3/4)n$, folglich gilt $x \in P_0 \cup P_1$. Wir nehmen zunächst an, dass $\|x\|_1 > n/4$ gilt. Dann gilt $f_{\mathrm{M}}(x) = n - \|x\|_1$. Offenbar haben wir

$$q_i \geq \binom{\|x\|_1}{1} \frac{c \ln n}{n} \left(1 - \frac{c \ln n}{n}\right)^{n-1} \geq \frac{(n-i)c \ln n}{n^{1+c}} = \Omega\left(\frac{\ln n}{n^c}\right),$$

weil die Situation im wesentlichen der bei der linearen Funktion f_1 entspricht (vergleiche den Beweis zu Satz 3.4, hier allerdings mit anderer Mutationswahrscheinlichkeit). Wenn $\|x\|_1 < n/4$ gilt, haben wir $x \in P_0$ und es ist $f_{\mathrm{M}}(x) = \|x\|_1$. In diesem Fall sind die Rollen von Nullen und Einsen vertauscht; die Argumente bleiben unter Berücksichtigung dieses Umstandes aber alle richtig. Wir haben also insgesamt

$$\sum_{0 \leq i < (3/4)n} \frac{1}{q_i} < n \cdot O\left(\frac{n^c}{\ln n}\right) = O\left(\frac{n^{1+c}}{\ln n}\right)$$

als obere Schranke für diesen Teil.

2. Fall: $i \in \{(3/4)n, (3/4)n + 1, \ldots, n - 1\}$
Wir betrachten also einen String x mit $(3/4)n \leq f_{\mathrm{M}}(x) < n$, also gilt $x \in P_2$, $\|x\|_1 = n/4$ und unter dem vorderen Viertel der Bits von x (also in $x_1 x_2 \cdots x_{n/4}$) haben genau $i - (3/4)n$ Bits den Wert Eins. Außerdem gibt es $n - i$ Bits unter den hinteren drei Viertel Bits in x (also in $x_{(n/4)+1} x_{(n/4)+2} \cdots x_n$) mit dem Wert Eins. Um von Q_i nach Q_{i+1} zu kommen, genügt es, genau je eines der

$$\frac{n}{4} - \left(i - \frac{3n}{4}\right) = n - i$$

der vorderen Bits mit Wert Null und eines der $n - i$ hinteren Bits mit Wert Eins gleichzeitig zu mutieren. Die Wahrscheinlichkeit dafür beträgt

$$(n-i)^2 \left(\frac{c \ln n}{n}\right)^2 \left(1 - \frac{c \ln n}{n}\right)^{n-2},$$

wir haben also

$$q_i = \Omega\left((n-i)^2 \frac{\ln^2 n}{n^2} e^{-c \ln n}\right) = \Omega\left((n-i)^2 \frac{\ln^2 n}{n^{2+c}}\right)$$

als untere Schranke für q_i in diesem Fall. Als obere Schranke haben wir folglich

$$\begin{aligned}\sum_{(3/4)n\le i<n}\frac{1}{q_i} &= \sum_{(3/4)n\le i<n} O\left(\frac{1}{(n-i)^2}\frac{n^{2+c}}{\ln^2 n}\right)\\ &= \sum_{1\le i<n/4} O\left(\frac{n^{2+c}}{i^2\ln^2 n}\right) = O\left(\frac{n^{2+c}}{\ln^2 n}\right)\end{aligned}$$

für diese Phase.

3. Fall: $i\in\{n,n+1,\ldots,2n-1\}$
Wir betrachten also einen String x mit $n\le f_{\mathrm{M}}(x)<2n$, also gilt $x=1^j0^{n-j}$ mit $j\in\{1,2,\ldots,n/4\}$. Wir bemerken, dass $Q_{n+1}=Q_{n+2}=\cdots=Q_{(7/4)n}=\emptyset$ gilt. Um von Q_i zur nachfolgenden Ranggruppe zu kommen genügt es, unter den Bits mit Wert Eins genau das mit maximalem Index zu mutieren. Die Wahrscheinlichkeit dafür beträgt

$$\frac{c\ln n}{n}\left(1-\frac{c\ln n}{n}\right)^{n-1},$$

wir haben also

$$q_i=\Omega\left(\frac{\ln n}{n}e^{-c\ln n}\right)=\Omega\left(\frac{\ln n}{n^{1+c}}\right)$$

in diesem Fall. Daraus folgt

$$\sum_{n\le i<2n}\frac{1}{q_i}=\sum_{n\le i<2n} O\left(\frac{n^{1+c}}{\ln n}\right)=O\left(\frac{n^{2+c}}{\ln n}\right)$$

für diese Phase.

4. Fall: $i=2n$
Wir betrachten also einen String x mit $f_{\mathrm{M}}(x)=2n$, folglich gilt $x=\mathbf{0}$. Um von Q_i nach Q_{i+1} zu kommen, müssen genau $\log n$ der hinteren $n-2\log n$ Bits gleichzeitig mutieren. Wir haben also

$$\begin{aligned}q_i &= \binom{n-2\log n}{\log n}\left(\frac{c\ln n}{n}\right)^{\log n}\left(1-\frac{c\ln n}{n}\right)^{n-\log n}\\ &\ge \left(\frac{n-2\log n}{\log n}\right)^{\log n}\left(\frac{c\ln n}{n}\right)^{\log n}\left(1-\frac{c\ln n}{n}\right)^{n}\left(1-\frac{c\ln n}{n}\right)^{-\log n}\\ &= \left(1-\frac{2\log n}{n}\right)^{\log n}\left(\frac{c\ln n}{\log n}\right)^{\log n}\Omega\left(\frac{1}{n^c}\right)\cdot\Omega\left(1\right)\\ &= \Omega(1)\cdot(c\ln 2)^{\log n}\cdot\Omega\left(\frac{1}{n^c}\right)\cdot\Omega(1)=\Omega\left(n^{\log c+\log\ln 2-c}\right)\end{aligned}$$

in diesem Fall. Das ergibt

$$\frac{1}{q_i} = O\left(n^{c-\log c-\log\ln 2}\right)$$

als obere Schranke für diese letzte Phase.

Insgesamt haben wir also

$$\begin{aligned}\sum_{0\leq i\leq 2n}\frac{1}{q_i} &= O\left(\frac{n^{1+c}}{\ln n}\right)+O\left(\frac{n^{2+c}}{\ln^2 n}\right)+O\left(\frac{n^{2+c}}{\ln n}\right)+O\left(n^{c-\log c-\log\ln 2}\right)\\ &= O\left(n^{2+c}\ln^{-1} n+n^{c-\log c-\log\ln 2}\right)\end{aligned}$$

für die erwartete Laufzeit. Um einen möglichst günstigen Wert für c zu bestimmen, setzen wir $2+c=c-\log c-\log\ln 2$, also $c=(4\ln 2)^{-1}\leq 0{,}361$ und erhalten $O\left(n^{2{,}361}/\ln n\right)$ als obere Schranke. □

Die Funktion f_{M} ist in gewissem Sinne das erste nicht-triviale Beispiel für eine Funktion, bei der die Wahl $p(n)=1/n$ für die Mutationswahrscheinlichkeit des (1+1) EA bei weitem nicht optimal ist. Wir haben zwar auch für die Funktionen f_{Q} und f_{T} nachgewiesen, dass die erwartete Laufzeit für eine andere Wahl erheblich kleiner wird als mit $p(n)=1/n$. In beiden Fällen blieb die erwartete Laufzeit aber exponentiell. Bei f_{M} ist das anders. Während für alle Wahlen der Mutationswahrscheinlichkeit, die von $(\log n)/n$ substanziell verschieden sind, fast sicher eine superpolynomielle Anzahl von Generationen zur Optimierung benötigt wird, haben wir für $p(n)=(0{,}361\ln n)/n$ eine erwartete Laufzeit von $O\left(n^{2{,}361}/\ln n\right)$. Wir können uns sogar relativ leicht überlegen, dass es sehr unwahrscheinlich ist, dass der (1+1) EA mit passend eingestellter Mutationswahrscheinlichkeit wesentlich länger zur Optimierung von f_{M} braucht.

Satz 7.11. *Sei $n=2^k>32$ mit $k\in\mathbb{N}$. Sei $c\in\mathbb{R}$ eine Konstante mit $c>0$. Sei $p(n):=(c\ln n)/n$. Die Wahrscheinlichkeit, dass der $(1+1)$ EA mit Mutationswahrscheinlichkeit $p(n)$ nicht innerhalb von*

$$O\left(\frac{n^{3+c}}{\ln n}+n^{1+c-\log c-\log\ln 2}\right)$$

Generationen ein globales Maximum der Funktion $f_M\colon\{0,1\}^n\to\mathbb{R}$ erreicht, ist durch $e^{-\Omega(n)}$ nach oben beschränkt. Für $c=1/(4\ln 2)\leq 0{,}361$ ist die Wahrscheinlichkeit, dass innerhalb von $O\left(n^{3{,}361}/\ln n\right)$ Generationen ein globales Maximum von f_M erreicht wird, durch $1-e^{-\Omega(n)}$ nach unten beschränkt.

Beweis. Der Beweis kann im wesentlichen wie der Beweis zu Satz 7.10 geführt werden. Wenn ein Ereignis mit Wahrscheinlichkeit q eintritt, wobei $0 < q < 1$ gilt, ist die Wahrscheinlichkeit, dass dieses Ereignis in $\lceil n/q \rceil$ unabhängigen Versuchen gar nicht eintritt, durch

$$(1-q)^{\lceil n/q \rceil} \leq (1-q)^{n/q} = e^{-\Omega(n)}$$

nach oben beschränkt. Wir sehen, dass wir im Beweis zu Satz 7.10 weniger als $2n$ Übergänge von einer Ranggruppe in eine andere betrachten. Wenn wir im Vergleich zum Beweis von Satz 7.10 jeweils die betrachtete Anzahl Generationen um einen Faktor n erhöhen, ist die Wahrscheinlichkeit, irgendwann länger auf einen Übergang warten zu müssen, durch

$$2ne^{-\Omega(n)} = e^{-\Omega(n)}$$

nach oben beschränkt. Daraus folgen die Behauptungen. □

Natürlich ist es schön, ein wenig mehr über „die Wahrheit des (1+1) EA“ zu wissen: es gibt keine feste Wahl der Mutationswahrscheinlichkeit $p(n)$, die im Vergleich zu anderen Mutationswahrscheinlichkeiten immer zu akzeptablen Laufzeiten führt. Andererseits offenbart sich durch Satz 7.9 und Satz 7.11 ein neues, unangenehmes, weil schwieriges Problem: Wie findet man zu einer Zielfunktion $f\colon \{0,1\}^n \to \mathbb{R}$ eine angemessene Mutationswahrscheinlichkeit $p(n)$ für den (1+1) EA? Dieses Problem ist offensichtlich verwandt mit dem Klassifikationsproblem, von dessen Schwierigkeit wir uns in Kapitel 2 ausführlich überzeugt haben. Ohne strukturelles Wissen über f kann man nicht erwarten, in Polynomialzeit eine angemessene Mutationswahrscheinlichkeit bestimmen zu können. Die Bestimmung einer „passenden“ Mutationswahrscheinlichkeit erfordert nämlich unter anderem, die lineare Funktion f_1, für die $p(n) = \Theta(1/n)$ optimal ist, von der stark irreführenden Funktion f_{T} zu unterscheiden, für die $p(n) = 1/2$ erheblich günstiger ist. Die Funktionen f_1 und f_{T} unterscheiden sich aber nur in einem einzigen Punkt, wir haben tatsächlich $f_{\mathrm{T}}(x) = f_1(x)$ für alle $x \in \{0,1\}^n \setminus \{\mathbf{0}\}$. Offensichtlich kann vernünftigerweise nicht angenommen werden, dass dieser Unterschied alleine mit Hilfe von Orakelanfragen in Polynomialzeit gefunden wird. Wir werden das Problem der Wahl einer angemessenen Mutationswahrscheinlichkeit, das wir anhand der Funktion f_{M} als auch praktisch relevant motiviert haben, umgehen, indem wir einen anderen Algorithmus betrachten, der nicht das Einstellen einer Mutationswahrscheinlichkeit erfordert. Um unseren Ansatz richtig einordnen zu können, müssen wir etwas weiter ausholen.

Es ist unter Anwendern evolutionärer Algorithmen seit langem bekannt, dass es ein schwieriges Problem ist, für evolutionäre Algorithmen eine geeignete

Parametrisierung zu finden. Wir haben das formal in praktisch relevanter Weise durch Untersuchung der Funktion f_{M} nachgewiesen. Tatsächlich kann man sich aber auch überlegen, dass es für manche Funktionen vielleicht gar keine geeignete Parametereinstellung gibt, die zu einer effizienten Optimierung führt, obwohl die Funktion grundsätzlich von evolutionären Algorithmen optimiert werden könnte. Wir konkretisieren diesen Punkt: Genauso wie klar ist, dass unterschiedliche Funktionen unterschiedliche Parametrisierungen erfordern, kann man sich vorstellen, dass es Funktionen gibt, bei denen in unterschiedlichen Regionen des Suchraums $\{0,1\}^n$ unterschiedliche Parametrisierungen erforderlich sind, damit ein evolutionärer Algorithmus effizient ein globales Maximum der Funktion finden kann. Tatsächlich könnte man für den (1+1) EA eine Funktion konstruieren, die ganz ähnlich wie die Funktion f_{M} aufgebaut ist: es gibt einen relativ großen Bereich des Suchraums, in dem eine Mutationswahrscheinlichkeit $p(n) = 1/n$ gut geeignet ist, um den Funktionswert rasch zu erhöhen. Am „Ende" dieser Region steht dann ein Sprung über $\Theta(\log n)$ Bits, der mit einer Mutationswahrscheinlichkeit in der Größenordnung von $(\log n)/n$ effizient durchgeführt werden kann. Damit die Zeit, die man mit einer Mutationswahrscheinlichkeit $p(n) = \Theta((\log n)/n)$ im „Anfangsteil" länger braucht, stärker ins Gewicht fällt, sollte man diesen Anfangsteil länger gestalten. Wir haben in Kapitel 5 lange k-Pfade kennengelernt und haben so grundsätzlich die nötigen Werkzeuge kennengelernt, um eine solche Funktion zu konstruieren. Wir werden später auf diese Idee in etwas abgewandelter Form noch einmal zurückkommen, allerdings unter anderen Vorzeichen und mit anderer Zielsetzung. Für den Moment mag es uns reichen, dass es berechtigte Zweifel daran gibt, ob es immer sinnvoll ist, die Parametrisierung evolutionärer Algorithmen in einem Lauf konstant zu halten. Tatsächlich gibt es viele (vor allen Dingen praktische) Ansätze, die sich mit nicht-statischer Parametrisierung evolutionärer Algorithmen beschäftigen. Eine hilfreiche Systematisierung dieses lebendigen und bunten Teilgebiets der Erforschung und Entwicklung evolutionärer Algorithmen bietet Bäck (1998).

Dort wird zwischen drei verschiedenen Typen nicht-statischer Parametrisierung unterschieden, die von verschiedener Kompliziertheit und Mächtigkeit sind. Die einfachste Form ist die so genannte dynamische Parametersteuerung, bei der die Parameter nach einem festen Schema variiert werden, das nur von der Anzahl Generationen abhängen kann. Bekanntestes Beispiel ist vermutlich Simulated Annealing (Kirkpatrick, Gelatt und Vecchi 1983). Dort wird eine randomisierte lokale Suche durchgeführt, wobei auch Verschlechterungen mit einer gewissen Wahrscheinlichkeit akzeptiert werden, wobei diese Wahrscheinlichkeit im Laufe der Zeit immer kleiner wird. Die Idee ist, physikalische Systeme beim Abkühlen nachzuahmen, die dabei bekanntlich energieminimale Zustände annehmen, wenn das Abkühlen hinreichend langsam geschieht. Der Parameter, der beim Simulated Annealing die Wahrscheinlichkeit, mit der auch Verschlechterungen akzeptiert werden, steuert, wird darum oft auch Temperatur genannt.

Die nächste Stufe bildet die so genannte adaptive Parametersteuerung, bei der das Steuerungsschema alle bisher betrachteten Punkte im Suchraum und deren Funktionswerte berücksichtigen kann. Ein bekanntes Beispiel für eine Steuerungsregel diesen Typs ist Rechenbergs so genannte Ein-Fünftel-Regel. Man beobachtet einen evolutionären Algorithmus über eine bestimmte (nicht zu große) Anzahl Generationen und bestimmt die Anzahl von Generationen innerhalb dieses Intervalls, in denen Verbesserungen der Funktionswerte der Population erzielt werden konnten. Ist der Anteil dieser erfolgreichen Generationen am Beobachtungsintervall größer als $1/5$, so wird der Parameter, der die Stärke der Mutationen steuert, vergrößert. Andernfalls wird er verkleinert. Man beachte, dass diese Steuerungsregel für Evolutionsstrategien entwickelt wurde und von reellwertigen Suchräumen ausgeht. Dort bedeutet Mutation das Addieren einer kleinen, (üblicherweise) normalverteilten Zufallsgröße. Insofern kann man leicht sinnvoll von der Mutationsstärke sprechen, was im Suchraum $\{0, 1\}^n$ nicht so einfach ist. Die Grundidee, die diesem Vorgehen zugrunde liegt, ist die folgende: Ist die Erfolgsrate sehr groß, so ist es anscheinend sehr einfach, bessere Individuen zu erzeugen. Dann dürfte der Abstand von einem (lokalen) Maximum noch sehr groß sein und es ist günstig, größere „Schritte“ zu machen. Ist die Erfolgsrate hingegen klein, so nimmt man an, einem (lokalen) Maximum schon nahe zu sein und will durch kleine Schrittweiten eine genauere Annäherung ermöglichen. Es liegt nahe zu vermuten, dass dieses Vorgehen dazu führen kann, dass der evolutionäre Algorithmus in lokalen Maxima verweilt und nicht zu einem globalen Maximum findet (Rudolph 1999).

Beim dritten Typus nicht-statischer Parametersteuerung spricht man von selbst-adaptiver Parametersteuerung. Hierbei werden die Parameter vom evolutionären Algorithmus selbst durch Anwendung der üblichen Suchoperatoren Mutation, Crossover und Selektion entwickelt. Dieses Vorgehen ist in Evolutionsstrategien üblich (Bäck 1996).

Wir schlagen hier eine Variante des (1+1) EA vor, die wir dynamischer (1+1) EA nennen und die ein sehr einfaches, festes, nur zeitabhängiges Schema zur Steuerung der Mutationswahrscheinlichkeit verwendet. Wir untersuchen also ein Beispiel von dynamischer Parametersteuerung. Unsere Idee ist dabei ganz einfach: Weil wir nicht wissen, welcher Wert für die Mutationswahrscheinlichkeit bei der aktuellen Zielfunktion nützlich ist, bzw. welcher Wert in der Region des Suchraums, zu der der aktuelle String x gehört, hilfreich ist, probieren wir verschiedene Werte für die Mutationswahrscheinlichkeit aus. Weil, wie schon erwähnt, in verschiedenen Stadien der Optimierung verschiedene Werte sinnvoll sein können, beschränken wir das Ausprobieren verschiedener Mutationswahrscheinlichkeiten nicht auf den Anfang der Optimierung, sondern dehnen es auf die gesamte Laufzeit aus. Wir wollen möglichst alle sinnvollen Werte für die Mutationswahrscheinlichkeit ausprobieren und bestimmen darum zunächst ein sinnvolles Intervall. Wir haben uns schon überlegt, dass $p(n) = 1/2$ zu einem rein zufälligen Durchsuchen des Suchraums $\{0,1\}^n$ führt. Damit ist die Idee der Lokalität, also der Vorstellung, dass man typischerweise überdurchschnittlich gute Punkte im Suchraum in der Nähe überdurchschnittlich guter Punkte findet, schon praktisch aufgegeben. Wir legen darum $1/2$ als obere Schranke für die Mutationswahrscheinlichkeit fest (wie schon bei der Definition des (1+1) EA als Algorithmus 3.1). Wir wollen natürlich nicht, dass der (1+1) EA viele unnütze Versuche unternimmt. Weil schon die in vielen Fällen sinnvolle Mutationswahrscheinlichkeit $1/n$ impliziert, dass im Durchschnitt bei einer Mutation nur noch ein einziges Bit mutiert, legen wir $1/n$ als untere Schranke für die Wahl von $p(n)$ fest. Aus dem Intervall $[1/n; 1/2]$ wollen wir nun einerseits möglichst viele Mutationswahrscheinlichkeiten ausprobieren. Andererseits wollen wir natürlich nicht allzu viele Versuche mit völlig unpassenden Mutationswahrscheinlichkeiten verschwenden. Wir haben uns für lineare Funktionen überlegt, dass die Mutationswahrscheinlichkeit $p(n) = \Theta(1/n)$ optimal ist. Wenn wir nur in jedem i-ten Schritt eine solche Mutationswahrscheinlichkeit einsetzen, müssen wir damit rechnen, dass die erwartete Laufzeit des dynamischen (1+1) EA auf linearen Funktionen bis auf $\Theta(in \log n)$ steigt. Dabei wollen wir i natürlich möglichst klein halten. Wir suchen hier einen Kompromiss und legen fest, die Mutationswahrscheinlichkeit in jeder Generation zu verdoppeln. Das impliziert dann einen zusätzlichen Faktor $\log n$ für das Ausprobieren nicht passender Mutationswahrscheinlichkeiten, was in vielen praktischen Anwendungen akzeptabel sein dürfte. Der Klarheit halber definieren wir jetzt erst formal den dynamischen (1+1) EA (Droste, Jansen und Wegener 2000a; Jansen und Wegener 2000b).

Algorithmus 7.12. Dynamischer (1+1) EA

1. $p(n) := 1/n$
2. *Wähle* $x \in \{0,1\}^n$ *zufällig gleichverteilt.*
3. $y := x$
   Für alle $i \in \{1, 2, \ldots, n\}$
      Mit Wahrscheinlichkeit $p(n)$*:*
         Ersetze das i*-te Bit in* y *durch sein Komplement.*
4. *Falls* $f(y) \geq f(x)$ *gilt, setze* $x := y$
5. $p(n) := 2p(n)$*; Falls* $p(n) > 1/2$*, setze* $p(n) := 1/n$*.*
6. *Weiter bei* 3.

Wir werden uns jetzt zunächst überlegen, dass der dynamische (1+1) EA wie wir ihn als Algorithmus 7.12 definiert haben die Funktion f_{M}, anhand der wir nachgewiesen haben, dass die Mutationswahrscheinlichkeit $p(n) = 1/n$ in praktisch relevanter Weise nicht immer optimal ist, effizient optimiert. Wir werden sehen, dass der dynamische (1+1) EA mit der Funktion f_{M} in gewisser Weise besser fertig wird als der (1+1) EA mit fester Mutationswahrscheinlichkeit $p(n)$.

Satz 7.13. *Sei* $n = 2^k > 32$ *mit* $k \in \mathbb{N}$*. Die erwartete Anzahl Generationen, bis Algorithmus 7.12 erstmalig ein globales Optimum der Funktion* $f_M \colon \{0,1\}^n \to \mathbb{R}$ *erreicht, ist durch* $O\left(n^2 \log n\right)$ *nach oben beschränkt.*

Beweis. Wir führen den Beweis im wesentlichen analog zum Beweis von Satz 7.10, wir benutzen insbesondere die gleiche Partitionierung. Wir weisen genau wie dort für alle i mit $0 \leq i \leq 2n$ und $Q_i \neq \emptyset$ eine hinreichend große untere Schranke für die Wahrscheinlichkeit q_i, in einer Generation von Q_i nach $\bigcup\limits_{j>i} Q_j$ zu wechseln, nach. Dabei betrachten wir hier stets nur Generationen mit einer „passenden“ Mutationswahrscheinlichkeit. Das gibt dann bei der Summierung der erwarteten Wartezeiten zur Gesamtlaufzeit einen zusätzlichen Faktor $\log n$, weil jede Mutationswahrscheinlichkeit

$$p(n) \in \left\{\frac{1}{n}, \frac{2}{n}, \ldots, \frac{2^{\lfloor \log n \rfloor - 1}}{n}\right\}$$

in jedem $\lfloor \log n \rfloor$-ten Schritt angenommen wird. Wir gehen jetzt ganz genau wie im Beweis zu Satz 7.10 die vier verschiedenen Fälle durch, besprechen hier aber nur noch Dinge, die im Vergleich zu dem Beweis dort anders sind.

1. Fall: $i \in \{0, 1, \ldots, (3/4)n - 1\}$
Wir betrachten die Generationen, in denen $p(n) = 1/n$ gilt. Dann haben wir

$$q_i \geq \binom{n/4}{1} \frac{1}{n} \left(1 - \frac{1}{n}\right)^{n-1} = \Omega(1)$$

also

$$\sum_{0 \leq i < 3n/4} \frac{1}{q_i} = O(n)$$

für diesen Teil des Suchraums.

2. Fall: $i \in \{(3/4)n, (3/4)n + 1, \ldots, n - 1\}$
Wir betrachten wieder genau die Generationen, in denen $p(n) = 1/n$ gilt und haben damit

$$q_i \geq \left(\frac{n-i}{n}\right) \left(1 - \frac{1}{n}\right)^{n-2} = \Omega\left(\left(\frac{n-i}{n}\right)^2\right),$$

was

$$\sum_{3n/4 \leq i < n} \frac{1}{q_i} = O\left(n^2\right) \sum_{1 \leq i \leq n/4} \frac{1}{i^2} = O\left(n^2\right)$$

für diesen Teil des Suchraums ergibt.

3. Fall: $i \in \{n, n + 1, \ldots, 2n - 1\}$
Auch hier betrachten wir ausschließlich Generationen, in denen $p(n) = 1/n$ gilt, was hier

$$q_i \geq \frac{1}{n} \left(1 - \frac{1}{n}\right)^{n-1} = \Omega\left(\frac{1}{n}\right)$$

liefert. Damit haben wir

$$\sum_{n \leq i < 2n} \frac{1}{q_i} = O\left(n^2\right)$$

als obere Schranke für die erwartete Anzahl Generationen in diesem Bereich des Suchraums.

4. Fall: $i = 2n$
Bis jetzt war zur Erreichen der Ranggruppe entweder das Mutieren irgend eines Bits von $\Theta(n)$ Bits, das Mutieren irgend eines Bitpaares von $(n - i)^2$

Bitpaaren oder das Mutieren eines einzigen bestimmten Bits ausreichend. In all diesen Fällen haben wir gesehen, dass die feste Wahl $p(n) = 1/n$ ausreicht, um die obere Schranke $O(n^2)$ für die erwartete Anzahl Generationen, die der (1+1) EA in diesen Teilen des Suchraums verbringt, zu garantieren. Wir wissen aber aus Satz 7.9, dass die statische Wahl $p(n) = 1/n$ nicht zu einer insgesamt polynomiell beschränkten erwarteten Laufzeit führt. Wie wir uns schon überlegt haben, brauchen wir genau für das Verlassen von Q_{2n} eine Mutationswahrscheinlichkeit $p(n) = \Theta((\log n)/n)$. Wenn $p(n) = c(\ln n)/n$ gilt mit einer positiven Konstanten $c \in \mathbb{R}$, dann haben wir wie im Beweis zu Satz 7.10

$$q_{2n} = \binom{n - 2\log n}{\log n} \left(\frac{c \ln n}{n}\right)^{\log n} \left(1 - \frac{c \ln n}{n}\right)^{n - \log n} = \Omega\left(n^{\log c + \log \ln 2 - c}\right)$$

als untere Schranke für die Wahrscheinlichkeit, ein globales Maximum von Q_{2n} aus in einer Generation zu erreichen. Wir hatten uns überlegt, dass die Wahl $c = (4 \ln 2)^{-1}$ eine gute Wahl ist. Beim (1+1) EA kann man geleitet von dieser Analyse die Mutationswahrscheinlichkeit $p(n) = (\ln n)/((4 \ln 2)n)$ *wählen*. Bei der dynamischen Variante ist die Wahl einer bestimmten Mutationswahrscheinlichkeit nicht vorgesehen, der Algorithmus benutzt die Mutationswahrscheinlichkeit $2^i/n$, wobei i so gewählt ist, dass stets $1/n \leq p(n) \leq 1/2$ gilt. Wir sehen, dass die erwartete Zeit für den Übergang von Q_{2n} zu einem globalen Maximum vom konkreten Wert von n abhängt. Weil die Mutationswahrscheinlichkeit in jedem Schritt genau verdoppelt wird, ist aber gesichert, dass irgendeine Mutationswahrscheinlichkeit aus dem Intervall $[(c \ln n)/n; (2c \ln n)/n]$ gewählt wird (c eine Konstante mit $c \geq 1$). Man sieht leicht, dass $c = 1$ die günstigste Wahl ist, dann hat man

$$\frac{1}{q_{2n}} = O\left(n^{1 - \log \ln 2}\right) = O\left(n^{1,53}\right)$$

als erwartete Anzahl Generationen für diesen abschließenden Sprung in ein globales Maximum.

Weil wir insgesamt $O(n^2)$ als obere Schranke für die Anzahl Generationen mit geeigneter Mutationswahrscheinlichkeit haben, folgt daraus $O(n^2 \log n)$ als obere Schranke für die erwartete Laufzeit insgesamt. □

Wir haben gesehen, dass die erwartete Laufzeit von Algorithmus 7.12 von der Dimension des Suchraums n abhängen kann, weil die konkrete Größe von n die Mutationswahrscheinlichkeiten, die ausprobiert werden, bestimmt.

Bei der Funktion f_{M} beobachten wir dieses Phänomen beim abschließenden Sprung in ein globales Maximum. Es hat hier allerdings keine Auswirkungen auf die erwartete Laufzeit insgesamt, weil diese von der Zeit dominiert wird, die der Algorithmus auf dem Weg zur „Sprungstelle“ **1** verbringt. Wir können uns aber durchaus vorstellen, dass es Funktionen gibt, bei denen die erwartete Laufzeit von Algorithmus 7.12 tatsächlich stark von der Dimension des Suchraums n abhängt.

Analog zu Satz 7.11 kann man auch hier eine Aussage darüber machen, nach wievielen Schritten der dynamische (1+1) EA ein globales Optimum der Funktion f_{M} mit überwältigend großer Wahrscheinlichkeit erreicht hat. Wir halten das Resultat und einen skizzenhaften Beweis fest.

Satz 7.14. *Sei $n = 2^k > 32$ mit $k \in \mathbb{N}$. Die Wahrscheinlichkeit, dass der dynamische $(1+1)$ EA innerhalb der ersten $O\left(n^3 \log n\right)$ Generationen ein globales Maximum der Funktion $f_M\colon \{0,1\}^n \to \mathbb{R}$ findet, ist durch $1 - e^{-\Omega(n)}$ nach unten beschränkt.*

Beweis. Analog zum Beweis von Satz 7.11 erhöhen wir im Vergleich zum Beweis von Satz 7.13 in jedem der vier Fälle die Anzahl Generationen für diese Phase um den Faktor n. Das gibt eine obere Schranke von $e^{-\Omega(n)}$ innerhalb der gegebenen Anzahl Generationen die entsprechende Phase nicht erfolgreich zu beenden. Daraus folgt die Behauptung. □

Dass der dynamische (1+1) EA die Funktion f_{M} effizient optimiert und die Wahl der Mutationswahrscheinlichkeit überflüssig macht, ist natürlich schön. Insbesondere ist positiv hervorzuheben, dass wir im Beweis zu Satz 7.10 gesehen haben, dass die Wahl einer geeigneten Konstante c für die Mutationswahrscheinlichkeit $p(n) = (c \ln n)/n$ nicht einfach ist und eine Analyse der Situation voraussetzt. Das widerspricht in gewisser Weise ja bereits dem „Geist“ evolutionärer Algorithmen, die ja ohne besondere Kenntnis und Analyse der Zielfunktion zu akzeptablen Ergebnissen bei der Optimierung führen sollen. Es ist aber natürlich nicht so, dass überzeugende Effizienz für eine Zielfunktion ein insgesamt überzeugendes Argument für einen bestimmten evolutionären Algorithmus ist. Wir überzeugen uns darum jetzt durch Untersuchung einiger Zielfunktionen davon, dass der dynamische (1+1) EA insgesamt eine vernünftige Variante des statischen (1+1) EA ist. Wir beginnen diese Überlegungen mit ganz grundsätzlichen Betrachtungen zu schwierigsten Funktionen, wie wir sie in Kapitel 6 kennengelernt haben. Das hat den Vorteil, dass obere Schranken für die erwartete Laufzeit auf solchen Funktionen allgemeine obere Schranken für alle Funktionen sind.

Satz 7.15. *Für jede Funktion* $f\colon \{0,1\}^n \to \mathbb{R}$ *gilt, dass die erwartete Laufzeit des dynamischen* $(1+1)$ *EA auf* f *durch* $4^n \log n$ *nach oben beschränkt ist.*

Beweis. Der dynamische (1+1) EA verwendet insgesamt $\lfloor \log n \rfloor$ verschiedene Mutationswahrscheinlichkeiten, die alle von der Form $p(n) = 2^i/n$ sind und für die stets $1/n \leq p(n) \leq 1/2$ gilt. Die maximale Mutationswahrscheinlichkeit beträgt folglich $2^{\lfloor \log n \rfloor - 1}/n$ und wir haben $1/4 \leq 2^{\lfloor \log n \rfloor - 1} \leq 1/2$. Der Hammingabstand zwischen dem aktuellen String x des dynamischen (1+1) EA und einem globalen Maximum der Funktion f ist stets durch n nach oben beschränkt. Folglich gilt in jeder Generation, dass mit Wahrscheinlichkeit mindestens $p(n)^n$ ein globales Maximum erreicht wird. Weil in jeder $\lfloor \log n \rfloor$-ten Generation die maximale Mutationswahrscheinlichkeit $2^{\lfloor \log n \rfloor - 1}/n \geq 1/4$ angenommen wird, haben wir $4^n \lfloor \log n \rfloor$ als obere Schranke für die erwartete Laufzeit. □

Wir wissen aus Kapitel 6, dass es Funktionen gibt, zu deren Optimierung der (1+1) EA mit Mutationswahrscheinlichkeit $p(n) = 1/n$ im Durchschnitt $\Theta(n^n)$ Schritte braucht. Die Funktionen f_{T} und f_{Q} sind zwei Beispiele dafür. Auf solchen Funktionen ist der dynamische (1+1) EA erheblich schneller. Wir sehen auch hier, dass die obere Schranke für die erwartete Laufzeit erheblich vom konkreten Wert von n abhängen kann: ist n eine Zweierpotenz, so wird die Mutationswahrscheinlichkeit $2^{\lfloor \log n \rfloor - 1}/n = 2^{\log n}/2 = 1/2$ angenommen und wir haben sogar $2^n \log n$ als obere Schranke. Wir sehen, dass das Worst Case Verhalten des dynamischen (1+1) EA wesentlich günstiger ist, als das des (1+1) EA mit fester Wahl der Mutationswahrscheinlichkeit. Es ist aber weder zutreffend, daraus jetzt einen generellen Vorteil des dynamischen (1+1) EA gegenüber der statischen Variante abzuleiten, noch sinnvoll, den dynamischen (1+1) EA so abzuwandeln, dass die Mutationswahrscheinlichkeit 1/2 auf jeden Fall angenommen wird. Das senkt zwar wie besprochen die Worst Case Laufzeit auf $2^n \log n$, aber Unterschiede in diesem Bereich sind ohne praktische Bedeutung. Zum einen ist eine rein zufällige Suche (also ein (1+1) EA mit fester Mutationswahrscheinlichkeit $p(n) = 1/2$) immer noch um einen Faktor $\log n$ schneller als der so veränderte (1+1) EA. Zum anderen — und das ist das entscheidende Argument — sind Funktionen, die nur in einer exponentiellen Anzahl von Generationen optimiert werden können, eben praktisch durch diese Algorithmen gar nicht optimierbar. Ob dann konkret die erwartete Laufzeit eher 2^n oder eher n^n beträgt, mag theoretisch interessant sein, praktisch läuft es aber auf dasselbe hinaus. Weil unsere theoretischen Untersuchungen evolutionärer Algorithmen von praktischen Interessen geleitet sind, wollen wir also praktisch relevante Argumente für eine Verwendung des dynamischen (1+1) EA suchen. Ein guter Ansatzpunkt dazu ist zum Beispiel die lineare Funktion f_1, die in gewisser Weise die einfachste vollständig nicht-triviale Zielfunktion und darum einer theoretischen Analyse besonders gut zugänglich ist.

Satz 7.16. *Die erwartete Laufzeit des dynamischen* (1+1) *EA auf der Funktion* $f_1\colon \{0,1\}^n \to \mathbb{R}$ *ist nach oben durch* $O\left(n \log^2 n\right)$ *beschränkt.*

Beweis. Wir definieren eine Partition des Suchraums in die Mengen

$$Q_i := \{x \in \{0,1\}^n \mid \|x\|_1 = i\}$$

für $i \in \mathbb{N}$ mit $0 \leq i \leq n$. Wir bemerken, dass $f_1(x) = i$ für alle $x \in Q_i$ gilt. Wir zerlegen einen Lauf des dynamischen (1+1) EA so in Subphasen, dass eine Subphase stets Länge $\lfloor \log n \rfloor$ hat und in jeder Subphase alle $\lfloor \log n \rfloor$ überhaupt vorkommenden verschiedenen Mutationswahrscheinlichkeiten jeweils genau einmal angenommen werden. Wenn wir eine untere Schranke für die Wahrscheinlichkeit q_i kennen, innerhalb einer derartigen Subphase von Q_i nach $\bigcup\limits_{j>i} Q_j$ zu wechseln, haben wir

$$(\log n) \left(\sum_{0 \leq i \leq n} \frac{1}{q_i} \right)$$

als obere Schranke für die erwartete Laufzeit.

Um von Q_i mindestens nach Q_{i+1} zu kommen, genügt es offenbar, genau eines der $n-i$ Bits mit Wert Null zu mutieren. In jeder Subphase gibt es eine Generation mit $p(n) = 1/n$, wir haben also

$$q_i \geq \binom{n-i}{1} \frac{1}{n} \left(1 - \frac{1}{n}\right)^{n-1} = \Omega\left(\frac{n-i}{n}\right)$$

als untere Schranke. Damit haben

$$(\log n) \left(\sum_{0 \leq i < n} \frac{n}{n-i} \right) = (\log n) \cdot O\left(n \log n\right) = O\left(n \log^2 n\right)$$

als obere Schranke für die erwartete Laufzeit. □

Wir sehen, dass das Ausprobieren von $\lfloor \log n \rfloor$ verschiedenen Mutationswahrscheinlichkeiten in unserer Analyse der erwarteten Laufzeit bei der Funktion

f_1 nicht mehr als einen Faktor $\log n$ ausmacht. Ein zweites Beispiel, das wir untersuchen wollen, ist die Funktion f_{L}, die wir im Kapitel 5 als einfache unimodale Funktion kennengelernt haben. Der Funktionswert $f_{\text{L}}(x)$ ist gleich der Anzahl führender Einsen in x.

Satz 7.17. *Die erwartete Laufzeit des dynamischen* $(1+1)$ *EA auf der Funktion* $f_L\colon \{0,1\}^n \to \mathbb{R}$ *beträgt* $\Theta\left(n^2 \log n\right)$. *Außerdem gibt es zwei positive Konstante* $c_1 < c_2$, *so dass die Wahrscheinlichkeit, dass der dynamische* $(1+1)$ *EA zum Optimieren der Funktion* $f_L\colon \{0,1\}^n \to \mathbb{R}$ *weniger als* $c_1 n^2 \log n$ *Generationen oder mehr als* $c_2 n^2 \log n$ *Generationen braucht, durch* $e^{-\Omega(n)}$ *nach oben beschränkt ist.*

Beweis. Der Beweis der oberen Schranke folgt auf die gleiche Weise dem Beweis zu Satz 5.7 wie der Beweis zu Satz 7.16 dem Beweis zu Satz 3.4 folgt. Wir partitionieren den Suchraum in die Mengen

$$Q_i := \{x \in \{0,1\}^n \mid f_{\text{L}}(x) = i\}$$

für $i \in \mathbb{N}$ mit $0 \leq i \leq n$ und betrachten wieder Subphasen der Länge $\lfloor \log n \rfloor$. Um von Q_i mindestens nach Q_{i+1} zu kommen, genügt es, die Anzahl führender Einsen um mindestens 1 zu erhöhen. Das geschieht, indem man genau das Bit mutiert, das unter allen Bits mit Wert Null minimalen Index hat. Weil wir in jeder Subphase eine Generation haben, in der $p(n) = 1/n$ gilt, liefert das

$$q_i \geq \frac{1}{n}\left(1 - \frac{1}{n}\right)^{n-1} = \Omega\left(\frac{1}{n}\right)$$

als untere Schranke, so dass wir insgesamt

$$(\log n)\left(\sum_{0 \leq i < n} \frac{1}{q_i}\right) = (\log n) \cdot O\left(n^2\right) = O\left(n^2 \log n\right)$$

als obere Schranke für die erwartete Laufzeit haben. Analog zum Beweis von Satz 5.7 ist hier die Wahrscheinlichkeit, in $2en^2 \log n$ Generationen nicht mindestens n Generationen zu haben, die den Funktionswert erhöhen, durch $e^{-\Omega(n)}$ beschränkt.

Die untere Schranke ist nicht ganz so einfach zu beweisen. Für die obere Schranke betrachten wir im wesentlichen nur die Generationen, in denen $p(n) = 1/n$ gilt und ignorieren alle anderen. Das ist zulässig, weil wegen der strikten deterministischen Selektion in Zeile 4 von Algorithmus 7.12 die Anzahl führender Einsen im aktuellen String x niemals abnehmen kann. Das gilt dann natürlich insbesondere auch für Generationen mit anderen Mutationswahrscheinlichkeiten als $1/n$. Will man hingegen eine untere Schranke für die erwartete Laufzeit nachweisen, so kann man nicht einfach manche Generationen ignorieren. Bei jeder der $\lfloor \log n \rfloor$ verschiedenen Mutationswahrscheinlichkeiten, die tatsächlich angenommen werden, gibt es eine positive Wahrscheinlichkeit, die Anzahl führender Einsen zu vergrößern. Darum müssen auch solche Generationen betrachtet und berücksichtigt werden.

Wir haben uns im Beweis zu Satz 5.7 bei der Untersuchung des (1+1) EA mit $p(n) = 1/n$ auf der Funktion f_{L} überlegt, dass alle Bits mit größerem Index als das Bit, das unter allen Bits mit Wert Null minimalen Index hat, rein zufällige Werte haben. Diese Überlegungen waren unabhängig von der Mutationswahrscheinlichkeit, so dass sie auch bei Einsatz des dynamischen (1+1) EA gelten. Für die untere Schranke wollen wir den Lauf des dynamischen (1+1) EA auf f_{L} erst dann zu betrachten beginnen, wenn schon wenigstens $n/2$ Einsen im aktuellen String x vorhanden sind. Wir betrachten x zum ersten Zeitpunkt, zu dem das zutrifft. Weil stets gilt, dass alle Bits „rechts des linkesten Bits mit Wert Null“ rein zufällig sind, ist die Wahrscheinlichkeit, dass $f_{\mathrm{L}}(x) > (1+\delta)n/2$ zu diesem Zeitpunkt gilt, für jede Konstante $\delta > 0$ durch $e^{-\Omega(n)}$ beschränkt. Wir werden später $\delta > 0$ für unsere Zwecke hinreichend klein wählen. Die Anzahl führender Einsen wächst um mindestens 1, wenn das Bit an Position $(f_{\mathrm{L}}(x)+1)$ mutiert und die ersten $f_{\mathrm{L}}(x)$ Bits alle nicht mutieren. Wenn die aktuelle Mutationswahrscheinlichkeit $p(n)$ beträgt, so ist die Wahrscheinlichkeit für eine Erhöhung des Funktionswertes durch

$$(1-p(n))^{f_{\mathrm{L}}(x)} \cdot p(n) \leq (1-p(n))^{n/2} \cdot p(n)$$

nach oben beschränkt. Wir betrachten nun wieder eine Subphase der Länge $\lfloor \log n \rfloor$. Die Wahrscheinlichkeit, dass in einer solchen Subphase mindestens einmal die Anzahl führender Einsen größer wird, ist durch

$$\begin{aligned}\sum_{0 \leq i \leq \lfloor \log n \rfloor - 1} \frac{2^i}{n} \cdot \left(1 - \frac{2^i}{n}\right)^{n/2} &= \frac{1}{n} \sum_{0 \leq i \leq \lfloor \log n \rfloor - 1} 2^i \left(1 - \frac{2^i}{n}\right)^{\left(n/2^i\right) \cdot 2^{i-1}} \\ &\leq \frac{1}{n} \sum_{0 \leq i < \infty} 2^i \cdot e^{-2^{i-1}} \leq \frac{\alpha}{n}\end{aligned}$$

für eine Konstante $\alpha > 0$ nach oben beschränkt. Wir bemerken, dass die Wahrscheinlichkeit, die Anzahl führender Einsen in einer Generation mit Mutationswahrscheinlichkeit $p(n) = 1/n$ zu erhöhen, durch $\Omega(1/n)$ nach unten beschränkt ist. Bei der Funktion f_{L} ist das Ausprobieren größerer Mutationswahrscheinlichkeiten also nicht hilfreich, da größere Mutationswahrscheinlichkeiten nicht substanziell zu einer Vergrößerung der Wahrscheinlichkeit für eine Erhöhung des Funktionswertes beitragen.

Wir beginnen unsere Betrachtungen mit höchstens $(1+\delta)n/2$ vielen führenden Einsen in x, wobei wir $\delta > 0$ frei wählen können. Die Wahrscheinlichkeit, dass bei einer Vergrößerung des Funktionswertes die Anzahl führender Einsen um k wächst, ist durch $(1/2)^{k-1}$ nach oben beschränkt. Folglich ist die Wahrscheinlichkeit, in βn Generationen, in denen der Funktionswert größer wird, insgesamt mehr als $2\beta n$ führende Einsen hinzu zu gewinnen, durch $2^{-\beta n}$ nach oben beschränkt. Wir brauchen also mit Wahrscheinlichkeit $1 - e^{-\Omega(n)}$ mindestens $((1-\delta)/4)n$ viele Generationen, in denen die Anzahl führender Einsen zunimmt. Wir ändern nun den stochastischen Prozess, um die Anzahl von Generationen in einer Subphase, in denen die Anzahl führender Einsen erhöht wird, kontrollieren zu können. Dazu führen wir nach jeder Generation, in der die Anzahl führender Einsen zugenommen hat, $\lfloor \log n \rfloor$ Generationen ein, in denen keine Vergrößerung des Funktionswertes zugelassen wird. Dieser veränderte Prozess hat im Vergleich zum ursprünglichen Lauf des dynamischen (1+1) EA auf f_{L} höchstens $n \lfloor \log n \rfloor$ Generationen mehr, da nach höchstens n-maliger Zunahme der Anzahl führender Einsen mit Sicherheit das globale Maximum $\mathbf{1}$ erreicht ist. Die Wahrscheinlichkeit, den Funktionswert in einer Subphase zu erhöhen, ist auch im veränderten Prozess durch α/n nach oben beschränkt. Die Wahrscheinlichkeit, innerhalb von $(1-\delta)/(8\alpha)n^2$ Subphasen mehr als $((1-\delta)/4)n$ viele Generationen zu haben, in denen die Anzahl führender Einsen größer wird, ist exponentiell klein. Mit Wahrscheinlichkeit $1 - e^{-\Omega(n)}$ sind also

$$\frac{1-\delta}{8\alpha} n^2 \lfloor \log n \rfloor - n \lfloor \log n \rfloor = \Omega\left(n^2 \log n\right)$$

Generationen zum Erreichen des globalen Maximums der Funktion f_{L} nicht ausreichend. Wir wählen $\delta > 0$ hinreichend klein und haben damit sowohl die Aussage über die mit Wahrscheinlichkeit $1 - e^{-\Omega(n)}$ mindestens benötigte Anzahl Generationen als auch die untere Schranke für die erwartete Anzahl Generationen gezeigt. □

Wir sehen, dass das Ausprobieren von $\lfloor \log n \rfloor$ verschiedenen Mutationswahrscheinlichkeiten in der erwarteten Laufzeit einen zusätzlichen Faktor $\log n$ einbringt, der bei der festen Wahl $p(n) = 1/n$ eingespart wird. Als nächstes betrachten wir lineare Funktionen, für die wir im folgenden nachweisen werden, dass die erwartete Laufzeit des dynamischen (1+1) EA durch $O\left(n^2 \log n\right)$ beschränkt ist. Um die Aussage und ihren Beweis richtig einordnen zu können, weisen wir vorher noch nach, dass die erwartete Laufzeit des (1+1) EA nach oben durch $O(n^2)$ beschränkt ist. Natürlich folgt diese Aussage trivial aus der allgemeinen Schranke $O(n \log n)$ aus Satz 4.3. Wir verwenden hier jedoch eine erheblich einfacherer Beweismethode, die allerdings auch zu einer deutlich schwächeren Aussage führt.

Satz 7.18. *Der* $(1+1)$ *EA mit* $p(n) = 1/n$ *hat auf der Klasse der linearen Funktionen erwartete Laufzeit* $O(n^2)$.

Beweis. Sei $f\colon \{0,1\}^n \to \mathbb{R}$ eine lineare Funktion mit

$$f(x) = w_0 + \sum_{1 \le i \le n} w_i x_i.$$

Wir nehmen ohne Beschränkung der Allgemeinheit an, dass $w_0 = 0$ und $w_1 \ge w_2 \ge \cdots \ge w_n > 0$ gilt. Damit ist der Einsstring $\mathbf{1}$ das eindeutige globale Maximum von f. Wir definieren eine Partition des Suchraums durch

$$Q_i := \left\{ x \in \{0,1\}^n \mid \sum_{1 \le j < i} w_j \le f(x) < \sum_{1 \le j \le i} w_j \right\}$$

für alle $i \in \{1, 2, \ldots, n\}$. Außerdem sei

$$Q_{n+1} = \{0,1\}^n \setminus \bigcup_{1 \le j \le n} Q_j = \{\mathbf{1}\}.$$

Für alle $x \in Q_i$ gilt, dass unter den i Bits mit kleinstem Index mindestens eines ist, das den Wert Null hat. Um von Q_i mindestens nach Q_{i+1} zu kommen, genügt es offenbar, genau dieses Bit zu mutieren. Also haben wir

$$q_i \ge \frac{1}{n}\left(1 - \frac{1}{n}\right)^{n-1} = \Omega\left(\frac{1}{n}\right)$$

als untere Schranke für alle i und somit insgesamt

$$\sum_{1 \le i \le n} \frac{1}{q_i} = O\left(n^2\right)$$

als obere Schranke für die erwartete Laufzeit. □

Basierend auf dem Beweis zu Satz 7.18 kann man nun leicht die obere Schranke $O(n^2 \log n)$ für den dynamischen (1+1) EA nachweisen. Dazu geht man analog zum Beweis von Satz 7.16 vor.

Satz 7.19. *Die erwartete Laufzeit des dynamischen* $(1+1)$ *EA ist auf der Klasse der linearen Funktion durch* $O\left(n^2 \log n\right)$ *nach oben beschränkt.*

Beweis. Wir übernehmen die Partitionierung des Suchraums aus dem Beweis zu Satz 7.18 und betrachten Subphasen der Länge $\lfloor \log n \rfloor$ wie im Beweis zu Satz 7.16. Dann gilt analog zum Beweis von Satz 7.18

$$q_i = \Omega\left(\frac{1}{n}\right)$$

für alle i und wir haben insgesamt

$$(\log n)\left(\sum_{1 \leq i \leq n} \frac{1}{q_i}\right) = (\log n) \cdot O\left(n^2\right) = O\left(n^2 \log n\right)$$

als obere Schranke für die erwartete Laufzeit. □

Im Vergleich zur trivialen Schranke $O\left(n^2\right)$ für lineare Funktionen trägt also das Ausprobieren von $\lfloor \log n \rfloor$ verschiedenen Mutationswahrscheinlichkeiten auch hier höchstens einen Faktor $\log n$ zur erwarteten Laufzeit bei. Im Vergleich zur asymptotisch exakten Schranke $O\left(n \log n\right)$ ist der Unterschied natürlich größer. Wir vermuten allerdings, dass die erwartete Laufzeit des dynamischen (1+1) EA auf der Klasse der linearen Funktionen tatsächlich $O\left(n \log^2 n\right)$ beträgt. Ob diese Vermutung zutrifft und wie das gegebenenfalls bewiesen werden kann, ist ein offenes Problem.

Die Gleichförmigkeit, mit welcher der Term $\log n$ als zusätzlicher Faktor in der erwarteten Laufzeit beim dynamischen (1+1) EA im Vergleich zur statischen Variante auftaucht, bietet Anlass zur Vermutung, dass die erwartete Laufzeit des dynamischen (1+1) EA immer höchstens um den Faktor $\log n$ größer ist. Allerdings haben wir bisher hauptsächlich Funktionen betrachtet, bei der wir in der statischen Variante die Mutationswahrscheinlichkeit $p(n) = 1/n$ verwenden. Es gibt jedoch prinzipiell keinen Grund, warum die Situation bei anderen Mutationswahrscheinlichkeiten substanziell anders sein sollte. Wir haben uns schon überlegt, dass die erwartete Laufzeit des dynamischen (1+1) EA vom konkreten Wert von n abhängen kann, weil dieser bestimmt, welche Werte für die Mutationswahrscheinlichkeiten tatsächlich ausprobiert werden. Wir lassen diesen Umstand noch in unsere Überlegungen miteinfließen und formulieren basierend auf diesen Überlegungen die Vermutung etwas genauer: Hat der (1+1) EA mit Mutationswahrscheinlichkeit $p(n) = 2^i/n$ mit $i \in \{0, 1, \ldots, \lfloor \log n \rfloor - 1\}$ auf einer Funktion $f\colon \{0,1\}^n \to \mathbb{R}$

erwartete Laufzeit $O(t(n))$, so ist die erwartete Laufzeit des dynamischen (1+1) EA nach oben durch $O(t(n)\log n)$ beschränkt.

Diese Vermutung enthält offensichtlich implizit die Annahme, dass man nur Schritte mit der „richtigen“ Mutationswahrscheinlichkeit zu betrachten braucht, also mit der Mutationswahrscheinlichkeit, die als feste Wahl in der statischen Variante benutzt wird. Man nimmt damit implizit an, dass Generationen mit anderen Mutationswahrscheinlichkeiten nicht stören können. Es ist aber doch denkbar, dass eine Funktion in gewissen Bereichen des Suchraums „Fallen“ enthält, also Punkte mit verhältnismäßig großem Funktionswert und großem Hammingabstand zu besseren Punkten, insbesondere zu globalen Maxima. Solche Punkte bezeichnen wir als Fallen, da sie, wenn sie einmal betreten sind, nur verhältnismäßig schwer wieder verlassen werden können. Wenn man sich eine Funktion vorstellt, die Fallen derart enthält, dass sie mit einer bestimmten festen Mutationswahrscheinlichkeit mit großer Wahrscheinlichkeit nicht gefunden werden, mit anderen Mutationswahrscheinlichkeiten aber mit deutlich größerer Wahrscheinlichkeit gefunden werden, wird deutlich, dass diese implizite Annahme falsch sein könnte. Das Ausprobieren verschiedener Mutationswahrscheinlichkeiten, wie der dynamische (1+1) EA es praktiziert, könnte bei der Optimierung solcher Funktionen sehr hinderlich sein. Wir werden jetzt eine solche Funktion konkret konstruieren und dann formal nachweisen, dass der dynamische (1+1) EA in der Tat mit großer Wahrscheinlichkeit zur Optimierung dieser Funktion erheblich länger braucht als der (1+1) EA mit passender fester Mutationswahrscheinlichkeit. Das widerlegt die oben formulierte Vermutung.

Wir nennen die Funktion, die uns zur Falsifizierung der genannten Vermutung dient, f_{F}. Sie ist in gewisser Weise ähnlich konstruiert wie die Funktion f_{M}. Es gibt einen zunächst sehr breiten, dann schmaler werdenden Pfad zum globalen Optimum. Den bei f_{M} nötigen „Sprung“ zum Optimum kreieren wir hier nicht, wir machen den letzten Pfadpunkt zum globalen Optimum. In einigem Abstand vom Pfad schaffen wir aber eine recht große Menge von Punkten, die alle zweitbesten Funktionswert haben. Das ist unsere Falle, wie wir sie oben beschrieben haben. Als Abstand von dieser Falle zum Pfad wählen wir $\log n$, weil wir schon gesehen haben, dass das ein Abstand ist, der mit Mutationswahrscheinlichkeit $1/n$ mit großer Wahrscheinlichkeit nicht in einer polynomiellen Anzahl von Generationen überwunden wird, der aber andererseits mit Mutationswahrscheinlichkeit $\Theta((\log n)/n)$ noch mit großer Wahrscheinlichkeit in Polynomialzeit überwunden werden kann. Etwas problematisch ist die Länge des Pfades. Bei der Funktion f_{M} ist die Pfadlänge $\Theta(n)$, das genügt uns hier nicht. Wir brauchen eine deutlich längere, wenn auch polynomielle Länge. Um zu einer verhältnismäßig einfachen und gut strukturierten Beschreibung zu kommen, verwenden wir die langen k-Pfade aus Kapitel 5. Wir definieren lange k-Pfade auf einer nur logarithmischen Anzahl von Bits, was uns einerseits auf strukturierte Art und Weise den größten Anteil des Suchraums unangetastet lässt, so dass wir hier die anderen genannten Eigenschaften der Funktion f_{F} realisieren können, andererseits liefert es aber einen langen k-Pfad, der natürlich in Wahrheit nicht sehr lang ist, also polynomielle Länge hat.

Definition 7.20. *Sei $n = 2^k > 2^{20}$ mit $k \in \mathbb{N}$. Setze $j := 3k^2 + 1$. Damit ist $(j-1)/k = 3k \in \mathbb{N}$ und der lange k-Pfad der Dimension j P_k^j ist wohldefiniert. Wir bezeichnen den i-ten Punkt von P_k^j als p_i. Wir definieren zunächst eine Partition des Suchraums $\{0,1\}^n$ in sieben disjunkte Mengen $P_0, P_1, \ldots, P_6$.*

$$
\begin{aligned}
P_1 &:= \{x \in \{0,1\}^n \mid 7n/16 < \|x\|_1 \leq 9n/16\} \\
P_2 &:= \{x \in \{0,1\}^n \mid \|x\|_1 = 7n/16\} \\
P_3 &:= \left\{x \in \{0,1\}^n \mid \left(\sqrt{n} < \|x\|_1 < 7n/16\right) \wedge \left(\sum_{j<i\leq j+\sqrt{n}} x_i = \sqrt{n}\right)\right\} \\
P_4 &:= \left\{x \in \{0,1\}^n \mid \exists i \in \left\{1,2,\ldots,\sqrt{n}\right\} : x = 0^j 1^i 0^{n-i-j}\right\} \\
P_5 &:= \left\{x \in \{0,1\}^n \mid \left(x_1 x_2 \cdots x_j \in P_k^j\right) \wedge \left(\sum_{j<i\leq n} x_i = 0\right)\right\} \\
P_6 &:= \Bigg\{x \in \{0,1\}^n \mid \left(x_1 x_2 \cdots x_j \in P_k^j\right) \wedge \left(\sum_{j<i\leq j+k} x_i = 0\right) \\
&\qquad \wedge \left(\sum_{j<i\leq n} x_i = k\right)\Bigg\} \\
P_0 &:= \{0,1\}^n \setminus (P_1 \cup P_2 \cup P_3 \cup P_4 \cup P_5 \cup P_6)
\end{aligned}
$$

Basierend hierauf definieren wir die Funktion $f_F \colon \{0,1\}^n \to \mathbb{R}$ durch

$$
f_F(x) := \begin{cases}
n - \|x\|_1 & \text{falls } x \in P_1, \\
n - \|x\|_1 + \sum\limits_{j<i\leq j+\sqrt{n}} x_i & \text{falls } x \in P_2, \\
2n - \|x\|_1 & \text{falls } x \in P_3, \\
4n - i & \text{falls } (x \in P_4) \wedge (x = 0^j 1^i 0^{n-i-j}), \\
4n + 2i & \text{falls } (x \in P_5) \wedge (x_1 \cdots x_j = p_i \in P_k^j), \\
4n + 2\left|P_k^j\right| - 1 & \text{falls } x \in P_6, \\
\min\{\|x\|_1, n - \|x\|_1\}/3 & \text{falls } x \in P_0.
\end{cases}
$$

für alle $x \in \{0,1\}^n$.

Wir machen uns zunächst klar, dass

$$P_0 <_{f_\mathrm{F}} P_1 <_{f_\mathrm{F}} P_2 <_{f_\mathrm{F}} P_3 <_{f_\mathrm{F}} P_4 <_{f_\mathrm{F}} P_5 \setminus \{x_\mathrm{opt}\} <_{f_\mathrm{F}} P_6 <_{f_\mathrm{F}} \{x_\mathrm{opt}\}$$

gilt. Dabei bezeichnet x_opt das eindeutige globale Maximum von f_F, das zu P_5 gehört, auf den ersten j Bits der letzte Pfadpunkt von P_k^j ist und auf den anderen $n-j$ Bits nur Bits mit dem Wert Null hat. Das folgt einfach aus dem Umstand, dass Strings in den verschiedenen Teilmengen nur Funktionswerte in bestimmten Intervallen annehmen, im einzelnen haben wir

$$\begin{aligned}
\forall x \in P_0 \quad &: \quad 0 \leq f_\mathrm{F}(x) \leq \frac{1}{3}n \\
\forall x \in P_1 \quad &: \quad \frac{7}{16}n \leq f_\mathrm{F}(x) < \frac{9}{16}n \\
\forall x \in P_2 \quad &: \quad \frac{9}{16}n \leq f_\mathrm{F}(x) \leq \frac{9}{16}n + \sqrt{n} \\
\forall x \in P_3 \quad &: \quad \frac{25}{16}n < f_\mathrm{F}(x) < 2n - \sqrt{n} \\
\forall x \in P_4 \quad &: \quad 4n - \sqrt{n} \leq f_\mathrm{F}(x) < 4n - 1 \\
\forall x \in P_5 \setminus \{x_\mathrm{opt}\} \quad &: \quad 4n + 2 \leq f_\mathrm{F}(x) \leq 4n + 2\left(\left|P_k^j\right| - 1\right) \\
\forall x \in P_6 \quad &: \quad f_\mathrm{F}(x) = 4n + 2\left|P_k^j\right| - 1 \\
& \quad\;\; f_\mathrm{F}(x_\mathrm{opt}) = 4n + 2\left|P_k^j\right|
\end{aligned}$$

für die verschiedenen Teilmengen P_i.

Jetzt zeigen wir, dass der dynamische (1+1) EA mit großer Wahrscheinlichkeit substanziell länger zur Optimierung der Funktion f_F braucht als der (1+1) EA mit fester Mutationswahrscheinlichkeit $p(n) = 1/n$. Wir formulieren das Resultat als Aussage über die Wahrscheinlichkeit, mit der die Optimierung innerhalb einer gewissen Anzahl von Generationen gelingt. Wir haben uns schon in der Einleitung (Kapitel 1) überlegt, dass das in gewissem Sinn ein stärkerer und eher praxisrelavanter Aussagetyp ist als eine Aussage über erwartete Laufzeiten. Das Resultat ist offenbar inhärent zweiteilig. Zum einen besteht es aus einer oberen Schranke für die Zeit, der der (1+1) EA mit $p(n) = 1/n$ zur Optimierung von f_F braucht, zum anderen aus einer unteren Schranke für den dynamischen (1+1) EA. Wir beginnen mit der strukturell einfacher zu beweisenden oberen Schranke.

Satz 7.21. *Sei $n = 2^k > 2^{20}$ mit $k \in \mathbb{N}$. Der $(1+1)$ EA mit Mutationswahrscheinlichkeit $p(n) = 1/n$ findet das globale Optimum der Funktion $f_F\colon \{0,1\}^n \to \mathbb{R}$ innerhalb der ersten $O\left(n^4 \log^2 n \log\log n\right)$ Generationen mit Wahrscheinlichkeit $1 - e^{-\Omega(\log n \log\log n)}$.*

Beweis. Wir beginnen mit der Betrachtung des (1+1) EA mit $p(n) = 1/n$ und dem Beweis der oberen Schranke. Wir teilen einen Lauf in Phasen ein und betrachten jeweils die Wahrscheinlichkeit, eine Phase nicht wie vorgesehen abzuschließen. Ein solches Ereignis bezeichnen wir als Fehlschlag. Wir summieren schließlich alle Fehlschlagswahrscheinlichkeiten.

Die Wahrscheinlichkeit, dass der aktuelle String x direkt nach der Initialisierung nicht zu P_1 gehört, ist nach oben durch

$$\begin{aligned}\mathrm{Prob}\,(x \notin P_1) &\leq \mathrm{Prob}\left(\|x\|_1 > \frac{9}{16}n\right) + \mathrm{Prob}\left(\|x\|_1 < \frac{7}{16}n\right)\\ &= 2\mathrm{Prob}\left(\|x\|_1 < \left(1-\frac{1}{8}\right)\frac{n}{2}\right)\\ &< e^{-n/256} = e^{-\Omega(n)}\end{aligned}$$

beschränkt (Chernoff-Schranken). Wenn der (1+1) EA in P_1 startet, erreicht er nie einen Punkt in P_0. Eine wichtige strukturelle Eigenschaft der Funktion f_{F} ist die Existenz der „Falle“ P_6, wie wir uns schon überlegt haben. Wir betrachten es als Fehlschlag, wenn der (1+1) EA irgendwann in diese Falle gerät. Für alle $x \in P_1 \cup P_2 \cup P_3 \cup P_4 \cup P_5$ und alle $y \in P_6$ gilt, dass der Hammingabstand von x und y durch

$$\sqrt{n} - (j+k) = 2^{k/2} - 3k^2 - k - 1$$

nach unten beschränkt ist. Weil wir außerdem $k > 20$ voraussetzen, ist $\sqrt{n} - (j+k)$ durch k nach unten beschränkt. Es gilt $k = \log n$. Die Wahrscheinlichkeit, in einer Generation mindestens $\log n$ Bits gleichzeitig zu mutieren, ist durch

$$\begin{aligned}&\binom{n}{\log n}\left(\frac{1}{n}\right)^{\log n}\left(1-\frac{1}{n}\right)^{n-\log n}\\ \leq\ & \frac{n^{\log n}}{(\log n)!}\cdot\frac{1}{n^{\log n}} \leq \frac{e^{\log n}}{\sqrt{3\pi\log n}(\log n)^{\log n}}\\ \leq\ & \frac{1}{(\log n)^{\log n}} = e^{-(\log n)\ln\log n} = e^{-\Omega(\log n\log\log n)}\end{aligned}$$

nach oben beschränkt. Die Wahrscheinlichkeit, dass sich eine solche Mutation in einer polynomiellen Anzahl von Generationen ereignet, ist nach oben durch

$$n^{O(1)} \cdot e^{-\Omega(\log n\log\log n)} = e^{-\Omega(\log n\log\log n)}$$

beschränkt. Ähnlich wir im Beweis zu Satz 7.10 definieren wir jetzt die trivialen Ranggruppen

$$Q_i := \{x \in \{0,1\}^n \mid f_{\mathrm{F}}(x) = i\}$$

für $0 \leq i \leq 4n + 2\left|P_k^j\right|$ und geben jeweils eine untere Schranke für die Wahrscheinlichkeit q_i, in einer Generation von Q_i nach $\bigcup_{j>i} Q_j$ zu wechseln, an. Weil wir mit großer Wahrscheinlichkeit annehmen können, dass der (1+1) EA in P_1 startet und darum P_0 nie erreicht, geben wir untere Schranken nur für $x \in \{0,1\}^n \setminus P_0$, also $f_{\mathrm{F}}(x) \geq 7n/16$ an.

1. Fall: $i \in \{7n/16, (7n/16)+1, \ldots, (9n/16)-1\}$
Wir haben $x \in P_1$, außerdem ist $\|x\|_1 = n - i$. Um von Q_i nach Q_{i+1} zu gelangen, genügt es, die Anzahl der Bits mit Wert Eins in x um 1 zu senken. Das ist möglich durch die Mutation eines einzelnen Bits mit Wert Eins, also haben wir

$$q_i \geq \binom{n-i}{1}\frac{1}{n}\left(1-\frac{1}{n}\right)^{n-1} \geq \frac{7}{16e} = \Omega(1)$$

als untere Schranke für q_i. Die Wahrscheinlichkeit, in $O(\log n \log\log n)$ Generationen, nicht von Q_i mindestens nach Q_{i+1} zu wechseln, ist folglich durch $e^{-\Omega(\log n \log\log n)}$ nach oben beschränkt. Wir haben also insgesamt, dass mit Wahrscheinlichkeit $e^{-\Omega(\log n \log\log n)}$ nach den ersten $O(n \log n \log\log n)$ Generationen, der aktuelle String mindestens zu P_2 gehört.

2. Fall: $i \in \{9n/16, (9n/16)+1, \ldots, (9n/16)+\sqrt{n}-1\}$
Wir haben $x \in P_2$, $\|x\|_1 = 7n/16$. Man gelangt von Q_i nach Q_{i+1}, wenn man die Anzahl der Einsen in x unverändert lässt und die Anzahl der Bits mit Index zwischen $j+1$ und $j+\sqrt{n}$, die Wert Eins haben, um 1 erhöht. Wir bezeichnen die Anzahl Bits mit diesen Indizes, die schon Wert Eins haben, mit $l(i)$. Es ist $l(i) = i - 9n/16$. Wir haben folglich

$$q_i \geq \binom{\sqrt{n}-l(i)}{1}\binom{(7n/16)-l(i)}{1}\left(\frac{1}{n}\right)^2\left(1-\frac{1}{n}\right)^{n-2} = \Omega\left(\frac{\sqrt{n}-l(i)}{n}\right)$$

als untere Schranke für q_i in diesem Fall. Folglich ist die Wahrscheinlichkeit, nicht innerhalb von

$$O\left(\frac{n}{\sqrt{n}-l(i)} \cdot \log n \log\log n\right)$$

Generationen von Q_i nach Q_{i+1} zu gelangen, durch $e^{-\Omega(\log n \log\log n)}$ nach oben beschränkt. Wir haben also insgesamt für diese Phase, dass mit Wahrscheinlichkeit $e^{-\Omega(\log n \log\log n)}$ innerhalb von

$$\begin{aligned}&\sum_{0\le l<\sqrt{n}} \frac{n}{\sqrt{n}-l}\cdot \log n \log\log n\\ = \ & n\log n\log\log n \sum_{1\le l\le\sqrt{n}} \frac{1}{l} = O\left(n\log^2\log\log n\right)\end{aligned}$$

mindestens ein String mit Funktionswert $(9n/16)+\sqrt{n}$ erreicht ist.

3. Fall: $i \in \{(9n/16)+\sqrt{n},\ldots,2n-\sqrt{n}-1\}$
Es ist entweder $x\in P_2$ und x hat unter allen Strings in P_2 maximalen Funktionswert oder es gilt $x\in P_3$. In beiden Fällen haben alle Bits in x, deren Index zwischen $j+1$ und $j+\sqrt{n}$ liegt, Wert Eins, und der Funktionswert steigt um 1, wenn die Anzahl dieser Bits mit Wert Eins unverändert bleibt und die Anzahl von Bits mit Wert Eins insgesamt um 1 kleiner wird. Es genügt also, genau eines von $\|x\|_1-\sqrt{n}$ Bits mit Wert Eins zu mutieren, was uns

$$q_i \ge \binom{\|x\|_1-\sqrt{n}}{1}\frac{1}{n}\left(1-\frac{1}{n}\right)^{n-1} = \Omega\left(\frac{\|x\|_1-\sqrt{n}}{n}\right)$$

als untere Schranke für q_i liefert. Für jeden Wert von i, steht die Anzahl von Einsen in x fest, sie liegt zwischen $7n/16$ und $\sqrt{n}+1$. Die Wahrscheinlichkeit, nicht innerhalb von

$$O\left(\frac{n}{\|x\|_1-\sqrt{n}}\cdot\log n\log\log n\right)$$

Generationen die nächste nicht-leere Ranggruppe zu erreichen, ist durch $e^{-\Omega(\log n\log\log n)}$ nach oben beschränkt. Damit haben wir insgesamt, dass die Wahrscheinlichkeit, dass innerhalb von

$$\sum_{\sqrt{n}<l\le 7n/16}\frac{n}{l-\sqrt{n}}\cdot\log n\log\log n = O\left(n\log^2 n\log\log n\right)$$

nicht mindestens ein String aus P_4 erreicht wird, durch $e^{-\Omega(\log n\log\log n)}$ nach oben beschränkt ist.

4. Fall: $i\in\{4n-\sqrt{n},4n-\sqrt{n}+1,\ldots,4n-1\}$
Wir haben $x\in P_4$, es ist tatsächlich $x=0^j1^{4n-i}0^{i-3n-j}$. Um von Q_i mindestens nach Q_{i+1} zu kommen genügt es, genau das Bit zu mutieren, das unter allen Bits mit Wert Eins maximalen Index hat. Wir haben also

$$q_i\ge\frac{1}{n}\left(1-\frac{1}{n}\right)^{n-1}=\Omega\left(\frac{1}{n}\right)$$

und die Wahrscheinlichkeit, nach $O\left(n^2 \log n \log\log n\right)$ Generationen nicht mindestens einen String aus P_5 erreicht zu haben, ist durch $e^{-\Omega(\log n \log\log n)}$ nach oben beschränkt.

5. Fall: $i \in \left\{4n+2, 4n+4, \ldots, 4n+2\left|P_k^j\right|-2, 4n+2\left|P_k^j\right|\right\}$
Der aktuelle String x gehört zu den Pfadpunkten, jedenfalls in dem Sinne, dass $x_1x_2\cdots x_j \in P_k^j$ gilt. Es genügt also genau wie im vierten Fall, genau ein passend gewähltes Bit zu mutieren, um eine Ranggruppe weiter zu kommen. Für jede Ranggruppe gilt, dass die Wahrscheinlichkeit, nicht innerhalb von $O\left(n \log n \log\log n\right)$ Generationen mindestens in die nächste Ranggruppe zu kommen, durch $e^{-\Omega(\log n \log\log n)}$ nach oben beschränkt. Die Anzahl zu durchquerender Ranggruppen ist $\left|P_k^j\right|$ und gemäß Lemma 5.9 haben wir

$$\begin{aligned}\left|P_k^j\right| &= (k+1)2^{(j-1)/k} - k + 1 = (k+1)2^{3k} - k + 1 \\ &= (1+\log n)n^3 - \log n + 1 = O\left(n^3 \log n\right).\end{aligned}$$

Die Wahrscheinlichkeit, nach $O\left(n^4 \log^2 n \log\log n\right)$ Schritten, nicht das globale Maximum erreicht zu haben, ist folglich insgesamt durch $e^{-\Omega(\log n \log\log n)}$ nach oben beschränkt wie behauptet. □

Wir sehen also, dass der (1+1) EA mit Mutationswahrscheinlichkeit $p(n) = 1/n$ die Funktion f_{F} fast sicher in Polynomialzeit optimiert. Dieses Resultat kontrastieren wir jetzt mit einer unteren Schranke für den dynamischen (1+1) EA. Wir zeigen konkret, dass eine polynomielle Anzahl von Schritten für die Optimierung von f_{F} fast sicher nicht ausreicht.

Satz 7.22. *Sei $n = 2^k > 2^{20}$ mit $k \in \mathbb{N}$. Der dynamische $(1+1)$ EA findet das globale Optimum der Funktion $f_F\colon \{0,1\}^n \to \mathbb{R}$ mit Wahrscheinlichkeit $1 - e^{-\Omega\left(\log^2 n\right)}$ nicht innerhalb der ersten $n^{O(1)}$ Generationen.*

Beweis. Wie bereits erläutert besteht die Beweisidee für die untere Schranke darin zu zeigen, dass der dynamische (1+1) EA mit großer Wahrscheinlichkeit die „Falle“ P_6 betritt und dann lange braucht, um sie wieder zu verlassen. Nehmen wir zunächst an, dass der dynamische (1+1) EA irgendeinen Punkt $x \in P_6$ erreicht. Dann gilt $f(x) > f(y)$ für alle $y \in \{0,1\}^n \setminus \{x_{\mathrm{opt}}\}$, wobei x_{opt} das eindeutige globale Optimum von f_{F} bezeichnet. Offensichtlich gilt für alle $x \in P_6$, dass der Hammingabstand zwischen x und x_{opt} durch $\log n$ nach unten beschränkt ist. Um von x aus das globale Maximum x_{opt} zu erreichen, müssen genau alle die Bits gleichzeitig mutieren, die in x und x_{opt} unterschiedlichen Wert haben. Die Wahrscheinlichkeit dafür ist durch

$$\max\left\{\left(\frac{2^i}{n}\right)^{\log n}\left(1-\frac{2^i}{n}\right)^{n-\log n} \mid i \in \{0,1,\ldots,\lfloor\log n\rfloor-1\}\right\}$$

nach oben beschränkt. Um das Maximum zu bestimmen, betrachten wir die Ableitung der Funktion

$$\left(\frac{2^i}{n}\right)^{\log n}\left(1-\frac{2^i}{n}\right)^{n-\log n}$$

nach 2^i und setzen diese gleich 0, also

$$\frac{\log n}{n}\cdot\left(2^i\right)^{(\log n)-1}\cdot\left(1-\frac{2^i}{n}\right)^{n-\log n} = \frac{n-\log n}{n^{(\log n)-1}}\cdot\left(2^i\right)^{\log n}\cdot\left(1-\frac{2^i}{n}\right)^{n-(\log n)-1},$$

was $i=\log\log n$ als einzige Lösung ergibt. Man überzeugt sich leicht, dass

$$\max\left\{\left(\frac{2^i}{n}\right)^{\log n}\left(1-\frac{2^i}{n}\right)^{n-\log n} \mid i\in\{0,1,\dots,\lfloor\log n\rfloor-1\}\right\} = \left(\frac{\log n}{n}\right)^{\log n}\left(1-\frac{\log n}{n}\right)^{n-\log n}$$

gilt und wir haben

$$\left(\frac{\log n}{n}\right)^{\log n}\left(1-\frac{\log n}{n}\right)^{n-\log n} \leq e^{(\log n)((\ln\log n)-\ln n)} = e^{-\Omega\left(\log^2 n\right)}$$

als obere Schranke für die Wahrscheinlichkeit, die Menge P_6 in einer Generation zu verlassen. Die Wahrscheinlichkeit, die Menge P_6, die „Falle“, in $n^{O(1)}$ Generationen zu verlassen, ist folglich durch

$$n^{O(1)}e^{-\Omega\left(\log^2 n\right)} = e^{-\Omega\left(\log^2 n\right)}$$

nach oben beschränkt. Wir zeigen jetzt, dass der dynamische (1+1) EA die Menge P_6 mit großer Wahrscheinlichkeit betritt.

Wie wir uns schon im Beweis zu Satz 7.21 überlegt haben, gehört der aktuelle String x unmittelbar nach der Initialisierung mit Wahrscheinlichkeit $1-e^{-\Omega(n)}$ zu P_1. Gehört der aktuelle String x erst einmal zu P_1, wird P_0 nicht mehr betreten. Aus dem Beweis zu Satz 7.21, insbesondere aus der Betrachtung der ersten vier Fälle, geht ebenfalls hervor, dass der dynamische (1+1) EA mit Wahrscheinlichkeit $1-e^{-\Omega\left(\log^2 n\right)}$ innerhalb der ersten $O\left(n^2\log^2 n\right)$ Generationen einen String in P_5 erreicht, wenn nicht vorher schon P_6 erreicht wird. Weil wir zeigen wollen, dass P_6 mit großer Wahrscheinlichkeit erreicht wird, können wir diese hier für uns günstige Möglichkeit ignorieren. Für alle $x \in P_1 \cup P_2 \cup P_3$ und alle $y \in P_5$ gilt, dass der Hammingabstand zwischen x und y mindestens $\sqrt{n} - \left(3\log^2 n\right) - 1 > \sqrt{n}/2$ beträgt. Es ist $|P_5| = \left|P_k^j\right| = O\left(n^3 \log n\right)$. Folglich ist die Wahrscheinlichkeit, in einer Generation von $P_1 \cup P_2 \cup P_3$ nach P_5 zu kommen, durch

$$O\left(n^3 \log n\right) \cdot \max\left\{ \left(\frac{2^i}{n}\right)^{\sqrt{n}/2} \left(1 - \frac{2^i}{n}\right)^{n-\sqrt{n}/2} \mid i \in \{0, 1, \ldots, \lfloor \log n \rfloor - 1\} \right\}$$

beschränkt. Man überlegt sich analog zu den entsprechenden Betrachtungen im Beweis zu Satz 7.21, dass das Maximum für $2^i = \sqrt{n}/2$ angenommen wird. Folglich haben wir, dass die Wahrscheinlichkeit, dass der dynamische (1+1) EA in einer Generation P_5 von $P_1 \cup P_2 \cup P_3$ aus erreicht, nach oben durch

$$O\left(n^3 \log n\right) \left(\frac{1}{2\sqrt{n}}\right)^{\sqrt{n}/2} = e^{-\Omega\left(\sqrt{n} \log n\right)}$$

beschränkt ist. Wir haben also bis jetzt, dass mit Wahrscheinlichkeit $1 - e^{-\Omega\left(\log^2 n\right)}$ ein String aus P_4 erreicht wird, wobei wir (zu unseren Ungunsten) die Möglichkeit ignorieren, vorher P_6 zu erreichen.

Wir betrachten also die Situation, dass für den aktuellen String x des dynamischen (1+1) EA $x \in P_4$ gilt, also $x = 0^j 1^i 0^{n-i-j}$ mit $i \in \{1, 2, \ldots, \sqrt{n}\}$. Wir erinnern uns an die Struktur des langen k-Pfades P_k^j, mit der wir uns in Kapitel 5 ausführlich vertraut gemacht haben. Wir wissen, dass P_k^j aus zwei „Kopien“ von P_k^{j-k} und einer „Brücke“ zusammengesetzt ist. Wir bezeichnen die hinten in P_k^j liegende „Kopie“ von P_k^{j-k} im folgenden etwas ungenau als hintere Pfadhälfte. Für jedes $x \in P_4$ und jeden Punkt y aus der hinteren Pfadhälfte gilt, dass der Hammingabstand zwischen x und y um $k = \log n$ größer ist als der Hammingabstand zwischen x und dem korrespondieren Punkt der vorderen Pfadhälfte, der sich von y genau den in ersten k Bits unterscheidet. Wir wollen zeigen, dass der dynamische (1+1) EA mit großer Wahrscheinlichkeit als ersten Punkt auf dem Pfad keinen Punkt aus der hinteren Pfadhälfte betritt. Dazu unterscheiden wir zwei Fälle nach der aktuellen Mutationswahrscheinlichkeit $p(n)$ in der betrachteten Generation. Betrachten wir zunächst den Fall $p(n) \geq \left(\log^2 n\right)/n$. Eine notwendige Bedingung für eine Mutation von $x \in P_4$ aus zu einem Kind y mit $f_{\mathrm{F}}(x) < f_{\mathrm{F}}(y)$ und $y \notin P_6$ ist, das alle Bits mit Index größer als j nicht mutieren. Die

Wahrscheinlichkeit dafür ist durch

$$\left(1-\frac{\log^2 n}{n}\right)^{n-j} \leq \left(1-\frac{\log^2 n}{n}\right)^{\left(\left(n/\log^2 n\right)-1\right)\cdot\left(\log^2 n\right)/2} = e^{-\Omega\left(\log^2 n\right)}$$

nach oben beschränkt. Die Wahrscheinlichkeit, dass sich eine solche Mutation innerhalb von $n^{O(1)}$ Generationen ereignet, ist durch

$$n^{O(1)}e^{-\Omega\left(\log^2 n\right)} = e^{-\Omega\left(\log^2 n\right)}$$

nach oben beschränkt, wir können uns also auf Generationen mit Mutationswahrscheinlichkeit $p(n) < \left(\log^2 n\right)/n$ beschränken. Wir wollen zeigen, dass in Generationen, in denen die Mutationswahrscheinlichkeit $p(n)$ so nach oben beschränkt ist, mit großer Wahrscheinlichkeit der erste Punkt in P_5 kein Punkt der hinteren Pfadhälfte ist. Nehmen wir an, dass die Wahrscheinlichkeit, in einer Generation vom aktuellen Zustand $x \in P_4$ zu einem Punkt y aus der ersten Pfadhälfte zu kommen, q beträgt. Dann gibt es einen zu y korrespondierenden Punkt in der hinteren Pfadhälfte, der mit Wahrscheinlichkeit höchstens

$$q\left(\frac{\log^2 n}{n}\right)^{\log n}$$

erreicht wird. Die Wahrscheinlichkeit, dass ein Ereignis mit Wahrscheinlichkeit

$$q\left(\frac{\log^2 n}{n}\right)^{\log n}$$

vor einem Ereignis mit Wahrscheinlichkeit q eintritt, beträgt

$$\sum_{i\geq 0}\left(\left(1-q-q\left(\frac{\log^2 n}{n}\right)^{\log n}\right)^i q\left(\frac{\log^2 n}{n}\right)^{\log n}\right) = \frac{q\left(\frac{\log^2 n}{n}\right)^{\log n}}{q+q\left(\frac{\log^2 n}{n}\right)^{\log n}}$$

$$= \frac{\left(\frac{\log^2 n}{n}\right)^{\log n}}{1+\left(\frac{\log^2 n}{n}\right)^{\log n}} \leq \left(\frac{\log^2 n}{n}\right)^{\log n} = e^{-\Omega\left(\log^2 n\right)}.$$

Folglich erreicht der dynamische (1+1) EA von P_4 aus mit Wahrscheinlichkeit $1-e^{-\Omega\left(\log^2 n\right)}$ als ersten Punkt in P_5 einen Punkt, der nicht zur hinteren

Pfadhälfte gehört. Die Restlänge des Pfades, die noch zurückzulegen ist, ist durch

$$\left|P_k^{j-k}\right| = (k+1)2^{(j-k-1)/k} - k + 1 = (k+1)2^{3k-1} - k + 1 = \Theta\left(n^3 \log n\right)$$

nach unten beschränkt. Analog zum Beweis der unteren Schranke für den (1+1) EA mit $p(n) = 1/n$ auf der langen Pfadfunktion (Satz 5.13) schätzen wir die Verweildauer auf dem Pfad ab, indem wir den erwarteten Fortschritt in einer Generation nach oben abschätzen. Um einen Fortschritt von mehr als $k = \log n$ mit einer Generation zu erzielen, müssen mindestens k Bits gleichzeitig mutieren. Als wir die Wahrscheinlichkeit, P_6 in einer Generation zu verlassen, abgeschätzt haben, haben wir uns überlegt, dass die Wahrscheinlichkeit, dass der dynamische (1+1) EA in einer Generation einen String mit Hammingabstand $\log n$ erreicht, nach oben durch $e^{-\Omega\left(\log^2 n\right)}$ beschränkt ist. Es gibt $O\left(n^3 \log n\right)$ Punkte auf dem restlichen Pfad, folglich ist der erwartete Fortschritt durch Mutationen von mindestens $\log n$ Bits gleichzeitig durch

$$O\left(n^3 \log n\right) e^{-\Omega\left(\log^2 n\right)} = e^{-\Omega\left(\log^2 n\right)}$$

nach oben beschränkt. In den anderen Generationen ist der Fortschritt je Generation allein durch die Pfadeigenschaft durch $\log n$ nach oben beschränkt. Uns genügt hier diese triviale Abschätzung. Wir haben also jetzt insgesamt, dass der dynamische (1+1) EA mit Wahrscheinlichkeit $1 - e^{-\Omega\left(\log^2 n\right)}$ mindestens $\Omega\left(n^3\right)$ Generationen auf dem Pfad verbringt, es sei denn, er wechselt zu einem $y \in P_6$. Die Wahrscheinlichkeit für ein solches Ereignis während $x \in P_5$ gilt, wollen wir jetzt nach unten abschätzen.

Für alle $x \in P_5 \setminus \{x_{\text{opt}}\}$ gilt, dass es genau

$$\binom{n-j-k}{k} = \binom{n - 3\log^2 n - \log n - 1}{\log n} > \left(\frac{n}{2\log n}\right)^{\log n}$$

Strings in P_6 gibt, die alle Hammingabstand genau $\log n$ von x haben und echt größeren Funktionswert. Die Wahrscheinlichkeit, in einer Generation mit Mutationswahrscheinlichkeit $p(n) = (\log n)/n$ zu einem solchen String zu wechseln, ist folglich durch

$$\begin{aligned}
& \left(\frac{n}{2\log n}\right)^{\log n} \cdot \left(\frac{\log n}{n}\right)^{\log n} \left(1 - \frac{\log n}{n}\right)^{n - \log n} \\
= \; & \frac{1}{n} \cdot \left(1 - \frac{\log n}{n}\right)^{((n/\log n)-1)\log n} \\
\geq \; & \frac{1}{n}\left(\frac{1}{e}\right)^{\log n} = n^{-1-1/\ln 2} > n^{-2{,}45}
\end{aligned}$$

nach unten beschränkt. Wir haben uns überlegt, dass die Verweildauer auf dem Pfad mit Wahrscheinlichkeit $1-e^{-\Omega\left(\log^2 n\right)}$ durch $\Omega\left(n^3\right)$ nach unten beschränkt ist. Folglich gibt es mit dieser Wahrscheinlichkeit auch $\Omega\left(n^3/\log n\right)$ Generationen mit Mutationswahrscheinlichkeit $p(n)=(\log n)/n$. Die Wahrscheinlichkeit, dass sich in $\Omega\left(n^3/\log n\right)$ unabhängigen Versuchen mit Erfolgswahrscheinlichkeit $\Omega\left(n^{-2,45}\right)$ gar kein Erfolg einstellt, ist nach oben durch

$$\left(1-\frac{1}{n^{2,45}}\right)^{\Omega\left(n^3/\log n\right)}=\left(1-\frac{1}{n^{2,45}}\right)^{n^{2,45}\Omega\left(n^{0,55}/\log n\right)}=e^{-\Omega\left(\sqrt{n}\right)}$$

beschränkt. Folglich haben wir jetzt insgesamt, dass der dynamische (1+1) EA mit Wahrscheinlichkeit $1-e^{-\Omega\left(\log^2 n\right)}$ in die „Falle“ P_6 gerät und diese in $n^{O(1)}$ Generationen nicht wieder verlässt. □

Wir erkennen jetzt, dass unser Bemühen, mit dem dynamischen (1+1) EA einen im Vergleich zum (1+1) EA mit fester Mutationswahrscheinlichkeit robusteren Algorithmus zu entwerfen, in gewissem Sinn gescheitert ist. Man kann die beiden Algorithmen als unvergleichbar bezeichnen: bei einigen Funktionen findet der dynamische (1+1) EA schneller ein globales Maximum, bei der Funktion f_{F} ist es gerade umgekehrt. Natürlich legt schon das NFL-Theorem (Satz 2.1) nahe, dass es solche Beispiele geben kann. Allerdings kann man die Existenz solcher Beispiele nicht direkt aus dem NFL-Theorem folgern. Auch in anderer Hinsicht war die Mühe, die wir uns mit der Funktion f_{F} gemacht haben, nicht umsonst. Wir haben zum einen den Schritt von der abstrakten Aussage, dass es ein solches Beispiel gibt, zu der konkreten Angabe einer Beispielfunktion gemacht. Wir haben durch dieses Beispiel erkannt, dass es sogar einfach auszuwertende Beispiele mit kompakter Beschreibung gibt. Und wir haben durch die intensive Untersuchung der Funktion f_{F} hoffentlich verstanden, was eine Zielfunktion für den dynamischen (1+1) EA schwierig machen kann. Wir schließen hiermit unsere Überlegungen zu anderen Mutationen als der bitweisen Mutation mit Mutationswahrscheinlichkeit $p(n)=1/n$ beim (1+1) EA ab. Im nächsten Kapitel beschäftigen wir uns mit alternativen Selektionsmechanismen. Dabei werden wir auch noch einmal den Aspekt der dynamischen Parametersteuerung aufgreifen.

8 Variationen der Selektion

Ein wichtiger Punkt, der stets genannt wird, wenn es darum geht, die Vorzüge evolutionärer Algorithmen aufzuzeigen, ist Robustheit. Es bleibt meist undefiniert, welche Kriterien erfüllt sein müssen, wenn ein Algorithmus das Prädikat „robust“ verdient. Eines der Kriterien ist aber ohne Zweifel, dass kleine Unterschiede in der Implementierung keinen großen Einfluss auf das Verhalten des Algorithmus haben. Wir wollen unter diesem Aspekt eine Variante des (1+1) EA ansehen, die sich allein in einem unscheinbaren Implementierungsdetail unterscheidet.

Algorithmus 8.1.

1. *Wähle* $p(n) \in (0; 1/2]$.
2. *Wähle* $x \in \{0,1\}^n$ *zufällig gleichverteilt.*
3. $y := x$
   Für alle $i \in \{1, 2, \dots, n\}$
      Mit Wahrscheinlichkeit $p(n)$*:*
         Ersetze das i*-te Bit in* y *durch sein Komplement.*
4. *Falls* $f(y) > f(x)$ *gilt, setze* $x := y$
5. *Weiter bei* 3.

Auf den ersten Blick, könnte man Algorithmus 8.1 mit dem (1+1) EA verwechseln, wie wir ihn als Algorithmus 3.1 definiert haben. Lediglich bei der Selektion in Zeile 4 gibt es einen kleinen Unterschied: Beim (1+1) EA ersetzt das Kind y das Elter x, wenn $f(y) \geq f(x)$ gilt, bei Algorithmus 8.1 hingegen ist dafür erforderlich, dass $f(y) > f(x)$ gilt. Einen Unterschied zwischen den beiden einfachen evolutionären Algorithmen kann es also nur dann geben, wenn $f(y) = f(x)$ gilt. Während der (1+1) EA im Falle der Gleichheit der Funktionswerte „innovationsfreudig“ ist, entscheidet Algorithmus 8.1 in diesen Fällen „konservativ“. Es ist zunächst nicht klar, ob eine der Varianten Vorzüge hat oder ob man nicht völlig beliebig entscheiden kann. Die übliche Implementierung ist die, welche wir im (1+1) EA realisiert finden. Bei Gleichheit der Funktionswerte wird zum Kind gewechselt. Dem liegt folgende Idee zu Grunde. Die zu optimierende Funktion könnten mehr oder minder große so genannte „Plateaus“ besitzen, das sind Mengen benachbarter Punkte (also Punkte mit kleinem Hammingabstand), die alle gleichen Funktionswert haben. Der (1+1) EA vollzieht auf solchen Plateaus eine zufällige Irrfahrt, während Algorithmus 8.1 im ersten erreichten Punkt des Plateaus verharrt, bis durch Mutation von diesem Punkt aus ein Punkt im Suchraum gefunden wird, der echt größeren Funktionswert hat. Man nimmt an, dass eine Irrfahrt das Verlassen des Plateaus erleichtert. Wir untersuchen diesen Aspekt an einem extremen Beispiel, an der Funktion $f_{\mathrm{N},\mathbf{1}}$, die wir in Kapitel 2 eingeführt haben. Die Funktion $f_{\mathrm{N},\mathbf{1}}$ besteht in unserem Sprachgebrauch hier aus nur einem einzigen großen „Plateau“, alle Punkte haben Funktionswert 0, lediglich für $\mathbf{1}$ gilt $f_{\mathrm{N},\mathbf{1}}(\mathbf{1}) = 1$. Wir betrachten die erwartete Laufzeit von Algorithmus 8.1 und vergleichen sie mit der erwarteten Laufzeit des (1+1) EA mit $p(n) = 1/n$ (Droste, Jansen und Wegener 1998a). Es ist klar, dass jeder Algorithmus zum Optimieren einer Funktion $f_{\mathrm{N},a}$ (bei unbekanntem a) $\Omega(2^n)$ Schritte benötigt und die erwartete Laufzeit des (1+1) EA mit $p(n) = 1/n$ genau 2^n ist.

Satz 8.2. *Die erwartete Anzahl Generationen, bis Algorithmus 8.1 mit Mutationswahrscheinlichkeit $p(n) = 1/n$ das globale Maximum der Funktion $f_{N,\mathbf{1}}\colon \{0,1\}^n \to \mathbb{R}$ findet, beträgt $\Theta\left((n/2)^n\right)$.*

Beweis. Weil Algorithmus 8.1 nur bei echten Verbesserungen den aktuellen Zustand x wechselt und bei der Funktion $f_{\mathrm{N},\mathbf{1}}$ nur der Wechseln von $x \in \{0,1\}^n \setminus \{\mathbf{1}\}$ zu $\mathbf{1}$ eine Verbesserung ist, kann die erwartete Laufzeit leicht genau bestimmt werden. Die initiale String enthält mit Wahrscheinlichkeit $\binom{n}{i}/2^n$ genau i Bits mit Wert Null. Dann müssen genau diese i Bits gleichzeitig mutieren, um zu einer Änderung von x zu führen. Gleichzeitig ist damit das globale Maximum von $f_{\mathrm{N},\mathbf{1}}$ erreicht. Mit Mutationswahrscheinlichkeit $1/n$ ergibt das

$$
\begin{aligned}
& \sum_{1\le i\le n} \binom{n}{i} 2^{-n} \cdot \left(\left(\frac{1}{n}\right)^i \left(1-\frac{1}{n}\right)^{n-i} \right)^{-1} \\
= {} & 2^{-n} \sum_{1\le i\le n} \binom{n}{i} n^i \left(\frac{n}{n-1}\right)^{n-i} \\
= {} & 2^{-n} \left(\left(\sum_{0\le i\le n} \binom{n}{i} n^i \left(\frac{n}{n-1}\right)^{n-i} \right) - \left(\frac{n}{n-1}\right)^n \right) \\
= {} & 2^{-n} \left(\left(n + \frac{n}{n-1}\right)^n - \left(\frac{n}{n-1}\right)^n \right) \\
= {} & 2^{-n} \left(n^n - 1\right) \left(1 + \frac{1}{n-1}\right)^n = \Theta\left(\frac{n}{2}\right)^n
\end{aligned}
$$

für die erwartete Laufzeit. □

Der (1+1) EA verhält sich bei der Optimierung der Funktion $f_{\mathrm{N},\mathbf{1}}$ erheblich anders. Er startet wie Algorithmus 8.1 zufällig gleichverteilt, macht dann aber jedes Kind y zum neuen aktuellen String x, bis erstmals $x = \mathbf{1}$ gilt. Es wird also in gewissem Sinne eine rein zufällige Irrfahrt in $\{0,1\}^n$ durchgeführt. Das ist der Effekt auf Plateaus, den wir vorhin besprochen hatten. Und tatsächlich zeigt der nächste Satz, den wir ohne Beweis von Garnier, Kallel und Schoenauer (1999) übernehmen, dass dieses Verhalten sehr hilfreich sein kann.

Satz 8.3 (Garnier, Kallel und Schoenauer (1999)). *Die erwartete Anzahl Generationen, bis der* (1+1) *EA mit Mutationswahrscheinlichkeit* $p(n) = 1/n$ *das globale Maximum der Funktion* $f_{N,\mathbf{1}}\colon \{0,1\}^n \to \mathbb{R}$ *findet, beträgt* $\Theta(2^n)$.

Natürlich kann man sich auch fragen, ob es Beispiele gibt, bei denen Algorithmus 8.1 im Durchschnitt schneller ein globales Maximum findet als der (1+1) EA. Menke (1998) hat sich mit dieser Frage auseinandergesetzt und für die Mutationswahrscheinlichkeit $p(n) = 1/n$ eine konkrete Funktion $f\colon \{0,1\}^4 \to \mathbb{R}$ angegeben, bei der die erwartete Laufzeit des (1+1) EA größer als 188 ist, während die erwartete Laufzeit von Algorithmus 8.1 durch 183 nach oben beschränkt ist. Die Frage, ob es auch Familien von Beispielfunktionen gibt, bei denen Algorithmus 8.1 den (1+1) EA auch in der Größenordnung der erwarteten Laufzeit schlägt für $n \to \infty$, ist offen. Außerdem ist den hier betrachteten Beispielen gemeinsam, dass beide Algorithmen exponentielle erwartete Laufzeit haben, wir also auch „beim schnelleren Algorithmus“ im Durchschnitt exponentiell lange auf einen Optimierungserfolg warten müssen. In diesem Sinn sind die präsentierten Beispiele zumindest aus praktischer Sicht sehr unbefriedigend. Allerdings hat Wegener (Jansen und Wegener 2000a) kürzlich zwei Funktionen vorgestellt, die exemplifizieren, dass beide Varianten vorteilhaft sein können. Diese Beispielfunktionen sind in diesem Sinne auch näher an einer praxisrelevanten Beantwortung der Frage. Bei jeweils einer Funktion findet die eine Variante des (1+1) EA mit gegen 1 konvergierender Wahrscheinlichkeit in einer polynomiell beschränkten Anzahl von Generationen ein globales Maximum, während die andere Variante mit gegen 1 konvergierender Wahrscheinlichkeit nicht in $n^{O(1)}$ Generationen erfolgreich ist. Diese Aussagen gelten in dieser Deutlichkeit aber nicht für die erwartete Laufzeit, insbesondere kann man mit Hilfe von Neustarts beide Funktionen mit beiden Varianten in erwarteter polynomieller Zeit optimieren.

Der zweite Algorithmus, den wir uns ansehen werden, kann auch als Variante des (1+1) EA betrachtet werden. Er verwendet aber einen deutlich komplizierteren Selektionsmechanismus im Vergleich zu der einfachen, deterministischen Entscheidung, wie sie der (1+1) EA oder auch Algorithmus 8.1

verwenden. Um die Analyse zu vereinfachen, verwenden wir einen vereinfachten Mutationsoperator, den wir uns schon in Kapitel 7 angesehen hatten. Wir mutieren in jeder Generation genau ein Bit in x, das wir zufällig gleichverteilt auswählen. Wir haben bei der Analyse von Algorithmus 7.1 gelernt, dass das bei manchen Funktionen einen großen Unterschied ausmacht, bei einigen Funktionen aber auch die erwartete Laufzeit im wesentlichen unangetastet lässt. Es gilt jedoch stets, dass die Analyse durch die kleinere Anzahl Möglichkeiten in jedem Schritt einfacher wird. Wir werden jetzt zunächst den Algorithmus formal definieren und dann noch etwas dazu sagen, wir er einzuordnen ist.

Algorithmus 8.4. *Sei* $\alpha\colon \mathbb{N} \rightarrow [1;\infty)$ *eine beliebige Funktion, die wir Selektionsstrategie nennen.*

1. *Wähle* $x \in \{0,1\}^n$ *zufällig gleichverteilt. Setze* $t := 1$.
2. $y := x$
   Wähle $i \in \{1, 2, \ldots, n\}$ *zufällig gleichverteilt.*
   Ersetze das i*-te Bit in* y *durch sein Komplement.*
3. *Mit Wahrscheinlichkeit* $\min\left\{1, \alpha(t)^{f(y)-f(x)}\right\}$
   Setze $x := y$.
4. *Setze* $t := t + 1$. *Weiter bei* 2.

Algorithmus 8.4 ist in dieser oder nur unwesentlich anderer Form seit vielen Jahren bekannt. Wenn die Selektionsstrategie α konstant ist, wird der Algorithmus üblicherweise Metropolis-Algorithmus genannt (Metropolis, Rosenbluth, Rosenbluth, Teller und Teller 1953). Ist α hingegen eine nicht-konstante Funktion, so spricht man im allgemeinen von Simulated Annealing (Kirkpatrick, Gelatt und Vecchi 1983). Wir wollen am Beispiel von Algorithmus 8.4 untersuchen, ob dynamische Parametersteuerung im Vergleich zu statischer Parametrisierung substanziell vorteilhaft sein kann. Wir suchen also nach einem Beispiel, bei dem Algorithmus 8.4 mit geeigneter, nicht-konstanter Selektionsstrategie α um Größenordnungen schneller zum Ziel kommt, als das mit jeder konstanten Wahl für α möglich ist. Diese Frage ist in schärferer Form von Jerrum und Sinclair (1997) gestellt worden: Gibt es ein natürliches Problem, so dass Simulated Annealing mit einer geeigneten, natürlichen Selektionsstrategie den Metropolis-Algorithmus für jede feste Wahl von α schlägt? Der wesentliche Unterschied zwischen dieser Frage und unserer bescheideneren Fragestellung liegt vor allem in der Forderung, dass das Beispiel ein „natürliches Problem“ sein soll. Wir wollen hier nicht diskutieren, welche Qualitäten ein Beispiel haben muss, um als natürliches Problem gelten zu können. Klar ist, dass die Beispiele, die wir in diesem Kapitel diskutieren, nicht natürlich sind. Ansätze zu einer Lösung dieser noch immer offenen Frage findet man etwa bei Jerrum und Sorkin (1998). Das wesentliche Resultat, das wir uns im Rest dieses Kapitels erarbeiten werden, ist, dass es tatsächlich Beispiele gibt, die einen deutlichen Vorsprung von Simulated Annealing mit passender Selektionsstrategie gegenüber einem optimal eingestellten Metropolis-Algorithmus nachweisen. Wir werden zwei Beispiele diskutieren, bei denen wesentlich verschiedene Selektionsmechanismen zum Einsatz kommen. Beiden Beispielen ist gemeinsam, dass die Beispielfunktionen verhältnismäßig leicht zu beschreiben und gut strukturiert sind. Außerdem sind die zugehörigen formalen Nachweise leicht zu führen. Sorkin (1991) gibt ebenfalls eine (künstliche) Beispielfunktion an, für die er nachweist, dass Simulated Annealing dem Metropolis-Algorithmus überlegen ist. Sein Beweis ist technisch sehr anspruchsvoll: zentral wird die Theorie schnell mischender Markovketten (Sinclair 1993) verwendet, mit der wir uns hier nicht auseinandersetzen wollen.

Die Grundidee von Simulated Annealing ist der Physik entliehen. Es ist bekannt, dass beim Abkühlen von Metallen das Material eine Molekülstruktur annimmt, die ein möglichst geringes Energieniveau aufweist. Wenn die Temperatur langsam genug abgesenkt wird, wird ein energieminimaler Zustand erreicht. Bei niedriger Temperatur ist das Material erstarrt, eine Änderung des Zustandes ist dann nicht mehr möglich. Man überträgt diesen Prozess in einem Algorithmus und verwendet ihn zur Optimierung, hier Minimierung. Man wechselt in einer Umgebung zufällig den Zustand, wobei Zustände, die geringes Energieniveau haben, bevorzugt werden. Wenn die Temperatur, die ein globales Attribut des Systems ist, noch hoch ist, kann auch zu Zuständen mit höherem Energieniveau gewechselt werden. Das ermöglicht es, lokale Optima zu verlassen. Mit sinkender Temperatur wird das aber zunehmend unwahrscheinlich, bis das System schließlich „erstarrt“ und der Prozess endet.

Es ist leicht, dieses Prinzip auch für die Maximierung anzuwenden. In der Formulierung von Algorithmus 8.4 entspricht das Energieniveau dem Funktionswert, wobei wir größere Funktionswerte bevorzugen. Wenn das durch Mutation erzeugte Kind y nicht schlechter ist, als das Elter x, gilt $\alpha(t)^{f(y)-f(x)} \geq 1$ und das Kind ersetzt das Elter mit Wahrscheinlichkeit 1. Gilt andernfalls $f(y) < f(x)$, so wird das Kind dennoch mit positiver Wahrscheinlichkeit, konkret mit Wahrscheinlichkeit $\alpha(t)^{f(y)-f(x)}$, das Elter ersetzen. Je größer $\alpha(t)$ ist, desto geringer die Wahrscheinlichkeit, auch bei einem schlechteren Funktionswert y zu übernehmen. Es entspricht also $1/\alpha(t)$ in etwa der Temperatur. Aus dem physikalischen Vorbild ist somit motiviert, für α eine monoton wachsende Funktion zu wählen. Wie schon in der Einleitung (Kapitel 1) in Bezug auf evolutionäre Algorithmen erläutert, lassen wir uns aber von solchen „Begründungen“, die aus Analogien zum natürlichen Vorbild folgen, nicht binden oder einschränken. Aus algorithmischer Sicht gibt es keinen Grund, für α nicht beliebige Funktionen zu untersuchen. In der Tat werden wir einerseits eine (in einem großen Anfangsbereich) monoton wachsende Funktion α und andererseits aber auch eine monoton fallende Funktion α untersuchen.

Bevor wir uns konkreten Funktionen zuwenden, an denen wir exemplarisch sehen wollen, wann eine geeignete dynamische Parametrisierung einer konstanten Funktion α überlegen ist, wollen wir uns zunächst die Analysemethode erarbeiten, mit der wir unsere Ergebnisse erzielen. Die Analyse evolutionärer Algorithmen wird in der Regel durch den Einsatz dynamischer Parametrisierung wesentlich erschwert im Vergleich zu evolutionären Algorithmen, die statisch parametrisiert werden. Im vorangegangenen Kapitel bei der Analyse des dynamischen (1+1) EA sind wir dieser zusätzlichen Schwierigkeit im wesentlichen aus dem Weg gegangen, indem wir uns fast ausschließlich mit konstanten Mutationswahrscheinlichkeiten beschäftigt haben. Für obere Schranken haben wir uns jeweils eine „passende" Mutationswahrscheinlichkeit ausgesucht und überprüft, dass in anderen Generationen kein „Schaden" verursacht werden kann. Das führte dort stets zu einem Faktor $\log n$ für das Warten auf die gewünschte Mutationswahrscheinlichkeit. Unser Vorgehen für die untere Schranke im Beweis zu Satz 7.22 war ähnlich organisiert.

Auch hier werden wir das Problem, die dynamische Parametrisierung in unserer Analyse zu berücksichtigen, umgehen. Wir werden Algorithmus 8.4 analysieren unter der Annahme, dass α konstant ist. Wir beschäftigen uns also primär mit dem Metropolis-Algorithmus. Es wird sich später herausstellen, dass die für diesen einfachen Fall gewonnenen Erkenntnisse ausreichen, um den Fall einer nicht konstanten Selektionsstrategie in den Griff zu bekommen. Wir werden dann wesentlich ausnutzen, dass in bestimmten Phasen des Laufs jeweils bekannte untere und obere Schranken für $\alpha(t)$ verwendet werden können.

Beiden Beispielfunktionen, die wir nachher betrachten werden, ist gemeinsam, dass sie symmetrisch sind, also der Funktionswert jeweils nur von der Anzahl der Einsen in der Eingabe abhängt, und im Einsstring $\mathbf{1}$ ein eindeutiges globales Maximum haben. Wir setzen im folgenden voraus, dass die zu optimierende Funktion f diese Eigenschaften hat: sie ist symmetrisch und es gilt $f(\mathbf{1}) = \max\{f(x) \mid x \in \{0,1\}^n\}$ sowie $f(x) < f(\mathbf{1})$ für alle $x \in \{0,1\}^n \setminus \{\mathbf{1}\}$. Unter diesen recht allgemeinen Randbedingungen werden wir jetzt einige hilfreiche Dinge über die erwartete Laufzeit von Algorithmus 8.4 für konstante α herleiten. Wir beginnen mit einigen Definitionen.

Definition 8.5. *Sei $f\colon \{0,1\}^n \to \mathbb{R}$ eine symmetrische Funktion mit eindeutigem globalem Maximum bei $\mathbf{1}$. Sei $\alpha\colon \mathbb{N} \to [1;\infty)$ konstant mit $\alpha(t) = \alpha$ für alle $t \in \mathbb{N}$. Die zufällige Anzahl Generationen, die Algorithmus 8.4 mit konstanter Selektionsstrategie α zum Erreichen von $\mathbf{1}$ braucht, bezeichnen wir mit T. Die erwartete Laufzeit ist $E(T)$. Die zufällige Anzahl Generationen bis zum ersten Erreichen von $\mathbf{1}$ unter der Bedingung, dass der aktuelle String x direkt nach der Initialisierung genau i Einsen enthält, also $\|x\|_1 = i$ gilt, heißt T_i. Die erwartete Laufzeit unter der Bedingung, mit i Einsen zu starten, ist $E(T_i)$. Die zufällige Anzahl Schritte, bis startend in einem Zustand x mit genau i Bits mit Wert Eins erstmals ein Zustand y mit genau $i+1$ Bits mit Wert Eins erreicht wird, bezeichnen wir mit T_i^+. Die erwartete Zeit, bis zur Erhöhung der Anzahl der Einsen von i auf $i+1$ ist dann $E\left(T_i^+\right)$. Die Wahrscheinlichkeit, ausgehend von einem Elter x mit $\|x\|_1 = i$ ein Kind y mit $\|y\|_1 = i-1$ zu erzeugen, bezeichnen wir mit p_i^-. Die Wahrscheinlichkeit, ausgehend von einem Elter x mit $\|x\|_1 = i$ ein Kind y mit $\|y\|_1 = i+1$ zu erzeugen, bezeichnen wir mit p_i^+.*

Nach dieser Regelung unseres Sprachgebrauchs machen wir jetzt einige nützliche Hilfsaussagen. Zunächst liegt auf der Hand, dass sich $\mathrm{E}(T)$ leicht als gewichtete Summe der $\mathrm{E}(T_i)$ ausdrücken lässt, wobei $\mathrm{E}(T_i)$ mit der Wahrscheinlichkeit gewichtet wird, bei zufälliger Initialisierung einen String mit genau i Bits mit Wert Eins zu bekommen. Wir haben also

$$\mathrm{E}(T) = \sum_{0 \le i \le n} \binom{n}{i} 2^{-n} \mathrm{E}(T_i)$$

und brauchen uns nur noch um $\mathrm{E}(T_i)$ zu kümmern. Weil wir bei jeder Mutation genau ein Bit mutieren, kann man $\mathbf{1}$ von einem String mit i Einsen nur erreichen, indem man „alle Zwischenschritte macht“, also mindestens je einmal einen String mit $i+1$, $i+2$, $\dots$, $n-1$ Einsen besucht. Wir haben folglich

$$T_i = \sum_{i \le j < n} T_j^+$$

und werden uns im folgenden um eine Abschätzung von $\mathrm{E}\left(T_j^+\right)$ bemühen.

Lemma 8.6. *Seien f, T_i^+, p_i^-, p_i^+ wie in Definition 8.5 gegeben. Für alle $i \in \{1, 2, \dots, n-1\}$ gilt, dass die erwartete Zeit zum Erhöhen der Anzahl Einsen um 1 durch*

$$E\left(T_i^+\right) = \frac{1}{p_i^+} + \frac{p_i^-}{p_i^+} E\left(T_{i-1}^+\right)$$

gegeben ist.

Beweis. Der Beweis ist einfach und folgt direkt aus der Beschreibung von Algorithmus 8.4 und den Voraussetzungen aus Definition 8.5. Es ist offensichtlich

$$\mathrm{E}\left(T_i^+\right) = p_i^+ + p_i^- \left(1 + \mathrm{E}\left(T_{i-1}\right)^+ + \mathrm{E}\left(T_i^+\right)\right) + \left(1 - p_i^+ - p_i^-\right)\left(1 + \mathrm{E}\left(T_i^+\right)\right),$$

weil die Anzahl der Einsen entweder direkt um 1 größer wird (mit Wahrscheinlichkeit p_i^+), sich zunächst um 1 verkleinert und dann erst wieder auf i und danach auf $i+1$ steigen muss (mit Wahrscheinlichkeit p_i^-) oder unverändert bleibt (mit Wahrscheinlichkeit $1 - p_i^+ - p_i^-$). Durch einfache Äquivalenzumformung erhält man

$$\begin{aligned} p_i^+ \mathrm{E}\left(T_i^+\right) &= 1 + p_i^- \mathrm{E}\left(T_{i-1}^+\right) \\ \Leftrightarrow \mathrm{E}\left(T_i^+\right) &= \frac{1}{p_i^+} + \frac{p_i^-}{p_i^+} \mathrm{E}\left(T_{i-1}^+\right) \end{aligned}$$

wie behauptet. □

Man kann Lemma 8.6 leicht verallgemeinern, indem man den entsprechenden Ausdruck für $\mathrm{E}\left(T_{i-1}^+\right)$ einsetzt und so $\mathrm{E}\left(T_i^+\right)$ in Abhängigkeit von $\mathrm{E}\left(T_{i-2}^+\right)$ ausdrückt. Fortgesetztes Einsetzen führt dann zum folgenden Lemma.

Lemma 8.7. *Seien f, T_i^+, p_i^-, p_i^+ wie in Definition 8.5 gegeben. Für alle $i \in \{1, 2, \ldots, n-1\}$ und alle $j \in \{1, 2, \ldots, i\}$ gilt, dass die erwartete Zeit zum Erhöhen der Anzahl Einsen um 1 durch*

$$E\left(T_i^+\right) = \left(\sum_{0 \leq k < j} \frac{\prod\limits_{0 \leq l < k} p_{i-l}^-}{\prod\limits_{0 \leq l \leq k} p_{i-l}^+}\right) + \frac{\prod\limits_{0 \leq l < j} p_{i-l}^-}{\prod\limits_{0 \leq l < j} p_{i-l}^+} \cdot E\left(T_{i-j}^+\right)$$

gegeben ist.

Beweis. Der Beweis folgt mit Hilfe von Lemma 8.6 durch vollständige Induktion über j. Für $j = 1$ ergibt sich gerade Lemma 8.6. Sei die Aussage richtig für j. Dann haben wir für $j+1$

$$\begin{aligned}
\mathrm{E}\left(T_i^+\right) &= \left(\sum_{0\le k<j} \frac{\prod\limits_{0\le l<k} p_{i-l}^-}{\prod\limits_{0\le l\le k} p_{i-l}^+}\right) + \frac{\prod\limits_{0\le l<j} p_{i-l}^-}{\prod\limits_{0\le l<j} p_{i-l}^+} \cdot \mathrm{E}\left(T_{i-j}^+\right) \\
&= \left(\sum_{0\le k<j} \frac{\prod\limits_{0\le l<k} p_{i-l}^-}{\prod\limits_{0\le l\le k} p_{i-l}^+}\right) + \frac{\prod\limits_{0\le l<j} p_{i-l}^-}{\prod\limits_{0\le l<j} p_{i-l}^+} \cdot \left(\frac{1}{p_{i-j}^+} + \frac{p_{i-j}^-}{p_{i-j}^+} \cdot \mathrm{E}\left(T_{i-j-1}^+\right)\right) \\
&= \left(\sum_{0\le k<j} \frac{\prod\limits_{0\le l<k} p_{i-l}^-}{\prod\limits_{0\le l\le k} p_{i-l}^+}\right) + \frac{\prod\limits_{0\le l<j} p_{i-l}^-}{\prod\limits_{0\le l\le j} p_{i-l}^+} + \frac{\prod\limits_{0\le l\le j} p_{i-l}^-}{\prod\limits_{0\le l\le j} p_{i-k}^+} \cdot \mathrm{E}\left(T_{i-j-1}^+\right) \\
&= \left(\sum_{0\le k\le j} \frac{\prod\limits_{0\le l<k} p_{i-l}^-}{\prod\limits_{0\le l\le k} p_{i-l}^+}\right) + \frac{\prod\limits_{0\le l\le j} p_{i-l}^-}{\prod\limits_{0\le l\le j} p_{i-l}^+} \cdot \mathrm{E}\left(T_{i-(j+1)}^+\right)
\end{aligned}$$

und die Aussage gilt für alle j. □

Der Fall $x = \mathbf{0}$ ist ein Sonderfall. Wenn der aktuelle String nur Bits mit dem Wert Null enthält, ist $p_0^- = 0$ und wir haben einfach

$$\mathrm{E}\left(T_0^+\right) = \frac{1}{p_0^+},$$

womit wir dann aus Lemma 8.7 eine einfache Folgerung ziehen können.

Folgerung 8.8. *Seien f, T_i^+, p_i^-, p_i^+ wie in Definition 8.5 gegeben. Für alle $i \in \{1, 2, \ldots, n-1\}$ und alle $j \in \{1, 2, \ldots, i\}$ gilt, dass die erwartete Zeit zum Erhöhen der Anzahl Einsen um 1 durch*

$$\begin{aligned}
E\left(T_i^+\right) = \left(\sum_{0\le k<i} \frac{\prod\limits_{0\le l<k} p_{i-l}^-}{\prod\limits_{0\le l\le k} p_{i-l}^+}\right) + \frac{\prod\limits_{0\le l<i} p_{i-l}^-}{\prod\limits_{0\le l<i} p_{i-l}^+} \cdot \frac{1}{p_0^+} \\
= \sum_{0\le k\le i} \frac{\prod\limits_{0\le l<k} p_{i-l}^-}{\prod\limits_{0\le l\le k} p_{i-l}^+} = \sum_{0\le k\le i} \frac{1}{p_k^+} \cdot \prod_{k<l\le i} \frac{p_l^-}{p_l^+}.
\end{aligned}$$

gegeben ist.

Mit diesem Handwerkszeug versehen machen wir uns jetzt an die Untersuchung der erwarteten Laufzeit zweier konkreter Funktionen. Wir beginnen mit der Vorstellung der Funktion f_{H}.

Definition 8.9. *Die Funktion $f_H\colon \{0,1\}^n \to \mathbb{R}$ ist gegeben durch*

$$f_H(x) := \begin{cases} \|x\|_1 & \textit{falls } \|x\|_1 \leq n-3, \\ 3n^2+n & \textit{falls } \|x\|_1 = n-2, \\ 3n^2 & \textit{falls } \|x\|_1 = n-1, \\ 3n^2+n+1 & \textit{falls } \|x\|_1 = n \end{cases}$$

für alle $x \in \{0,1\}^n$.

Die Funktion f_{H} ist symmetrisch, sie kann darum übersichtlich in einer Abbildung dargestellt werden, wie wir das für symmetrische Funktionen immer tun. Man erkennt leicht in Abbildung 4 die für uns wesentlichen Eigenschaften der Funktion.

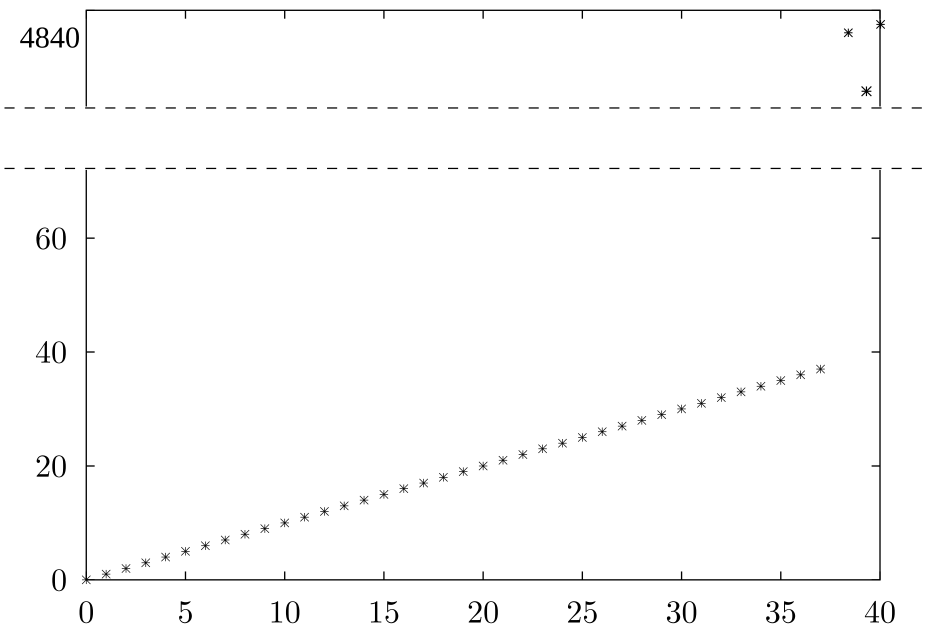

Abbildung 4: Die Funktion $f_{\mathrm{H}}\colon \{0,1\}^{40} \to \mathbb{R}$.

Die Funktion ist fast überall identisch mit der einfachen linearen Funktion f_1, es gilt $f_{\mathrm{H}}(x) = f_1(x)$ für alle x mit $\|x\|_1 \leq n-3$. Dann gibt es einen großen „Sprung“ in den Funktionswerten, die Funktionswerte für Strings x mit $\|x\|_1 > n-3$ sind um etwa $3n^2$ größer als die Funktionswerte aller anderen Strings. Weil Algorithmus 8.4 (anders als der (1+1) EA, der nur die Ordnung der Funktionswerte berücksichtigt) sensitiv auf die absolute Größe der Funktionswerte reagiert, ist diese große Differenz von Bedeutung. Die Idee ist, dass mit großer Wahrscheinlichkeit nicht mehr zu einem String mit weniger als $n-2$ Einsen gewechselt wird, wenn erst einmal ein String mit mindestens $n-2$ Einsen gefunden wird. Wir bemerken, dass die Funktion bis hierhin auch streng monoton wachsend ist. Die Strings x mit $\|x\|_1 = n-1$ spielen eine besondere Rolle. Sie haben echt kleineren Funktionswert als alle ihre Nachbarn. Um das globale Optimum $\mathbf{1}$ zu erreichen, ist es aber erforderlich, dass Algorithmus 8.4 mindestens einmal einen String x mit $\|x\|_1 = n-1$ zum aktuellen String macht. Das ist der einzige Schritt, in dem eine Verschlechterung der Funktionswerte hingenommen werden muss, um einem lokalen Optimum (Punkten x mit $\|x\|_1 = n-2$) zu entkommen. Unsere Vorstellung ist, dass man den f_1-ähnlichen Teil des Suchraums mit nicht zu kleinem α leicht überwinden kann. Danach sollte α allerdings recht klein sein, um die nötige Verschlechterung rasch durchführen zu können. Wir werden im folgenden zeigen, dass es keinen angemessenen konstanten Wert für α „irgendwo in der Mitte" gibt, der insgesamt zu einer überzeugenden Performanz führt, während es recht einfach ist, eine nicht-konstante Funktion α anzugeben, so dass die Optimierung der Funktion f_{H} für Algorithmus 8.4 einfach wird. Es ist allerdings hilfreich, sich vorher zu überlegen, wie sich Algorithmus 8.4 mit konstantem α beim Optimieren der Funktion f_1 verhält in Abhängigkeit von α. Wir benutzen den Sprachgebrauch aus Definition 8.5.

Lemma 8.10. *Die erwartete Laufzeit von Algorithmus 8.4 bei Initialisierung in einem String x mit $\|x\|_1 = i$ und konstanter Selektionsstrategie α auf der Funktion $f_1\colon \{0,1\}^n \to \mathbb{R}$ beträgt*

$$E(T_i) = \sum_{i \leq j < n} E\left(T_j^+\right) = \sum_{i \leq j < n} \left(\frac{1}{\alpha^j \binom{n-1}{j}} \sum_{0 \leq k \leq j} \binom{n}{k} \alpha^k \right).$$

Es ist

$$\alpha \left(\left(1 + \frac{1}{\alpha}\right)^n - 1 \right) \leq E(T_i) \leq \left(1 + \frac{1}{\alpha}\right)^{n-1} \cdot n \left(\ln(n-i) + 1\right).$$

Beweis. Wir bestimmen zunächst die Wahrscheinlichkeiten, in einer Generation ausgehend von einem Elter x mit $\|x\|_1 = i$ ein Kind y mit echt weniger bzw. mehr Bits mit Wert Eins zu erzeugen. Es ist offensichtlich

$$p_i^+ = \frac{n-i}{n}$$

und

$$p_i^- = \frac{i}{n} \cdot \alpha^{(i-1)-i} = \frac{i}{\alpha n}$$

für alle $i \in \{0, 1, \ldots, n-1\}$. Wir wenden Folgerung 8.8 an und haben damit

$$\begin{aligned}
\mathrm{E}\left(T_j^+\right) &= \sum_{0 \leq k \leq j} \frac{n}{n-k} \prod_{k<l\leq j} \frac{l}{\alpha(n-l)} \\
&= \sum_{0 \leq k \leq j} \frac{n}{n-k} \cdot \frac{j! \cdot (n-j-1)!}{\alpha^{j-k} \cdot k! \cdot (n-k-1)!} \\
&= \frac{j! \cdot (n-j-1)!}{\alpha^j \cdot (n-1)!} \sum_{0 \leq k \leq j} \frac{n!}{n-k} \cdot \frac{\alpha^k}{k! \cdot (n-k-1)!} \\
&= \frac{1}{\alpha^j \binom{n-1}{j}} \sum_{0 \leq k \leq j} \binom{n}{k} \alpha^k,
\end{aligned}$$

woraus sich durch einfaches Einsetzen die Behauptung für $\mathrm{E}(T_i)$ ergibt. Für die obere Schranke schauen wir uns noch einmal $\mathrm{E}\left(T_j^+\right)$ an und sehen, dass wir

$$\begin{aligned}
\mathrm{E}\left(T_j^+\right) &= \frac{1}{\alpha^j \binom{n-1}{j}} \sum_{0 \leq k \leq j} \binom{n}{k} \alpha^k = \frac{j!(n-j-1)!}{\alpha^j (n-1)!} \sum_{0 \leq k \leq j} \frac{n! \alpha^k}{k!(n-k)!} \\
&= \frac{j!(n-j-1)!}{\alpha^j (n-1)!} \sum_{0 \leq k \leq j} \frac{n! \alpha^{j-k}}{(j-k)!(n-j+k)!} \\
&= \frac{n}{n-j} \sum_{0 \leq k \leq j} \alpha^{-k} \frac{\binom{j}{k}}{\binom{n-j+k}{n-j}} \\
&\leq \frac{n}{n-j} \sum_{0 \leq k \leq j} \binom{k}{k} \left(\frac{1}{\alpha}\right)^k = \frac{n}{n-j} \left(1 + \frac{1}{\alpha}\right)^j
\end{aligned}$$

haben. Wir setzen dies wiederum einfach ein und erhalten

$$\begin{aligned}
\mathrm{E}(T_i) &= \sum_{i \leq j < n} \mathrm{E}\left(T_j^+\right) \leq \sum_{i \leq j < n} \frac{n}{n-j} \left(1 + \frac{1}{\alpha}\right)^j \\
&= n \left(1 + \frac{1}{\alpha}\right)^n \cdot \sum_{1 \leq j \leq n-i} \frac{1}{j} \left(1 + \frac{1}{\alpha}\right)^{-j} \\
&\leq \left(1 + \frac{1}{\alpha}\right)^{n-1} \cdot n \left(\ln(n-i) + 1\right)
\end{aligned}$$

als obere Schranke für $\mathrm{E}(T_i)$. Für die untere Schranke gehen wir etwas ungenauer vor und schätzen $\mathrm{E}(T_i)$ nach unten ab durch die erwartete Zeit, die allein für den Schritt von $n-1$ Einsen in x zum globalen Maximum benötigt wird. Wir haben also

$$\begin{aligned} \mathrm{E}\left(T_i\right) &\geq \mathrm{E}\left(T_{n-1}^{+}\right) = \frac{1}{\alpha^{n-1}\binom{n-1}{n-1}} \sum_{0\leq k<n} \binom{n}{k}\alpha^k \\ &= \left(\left(\sum_{0\leq k\leq n} \binom{n}{k}\alpha^k\right) - \alpha^n\right) \cdot \alpha^{-(n-1)} \\ &= \frac{(1+\alpha)^n - \alpha^n}{\alpha^{n-1}} = \alpha\left(\left(1+\frac{1}{\alpha}\right)^n - 1\right) \end{aligned}$$

als untere Schranke. □

Das Ergebnis ist natürlich nicht überraschend. Die einfache unimodale Funktion f_1 optimiert man mit Algorithmus 8.4 am besten, wenn die Wahrscheinlichkeit, auch Schritte zu Kindern mit kleineren Funktionswerten zu machen, sehr klein, also α sehr groß ist. Konkret haben wir mit $\alpha(n) = \Omega(n)$, dass die erwartete Laufzeit von Algorithmus 8.4 durch $O\left(n \log n\right)$ beschränkt ist. Lemma 8.10 ist auch weniger für sich alleine gesehen interessant, wir benutzen es als Hilfsaussage für den nächsten Satz.

Satz 8.11. *Die erwartete Laufzeit von Algorithmus 8.4 mit konstanter Selektionsstrategie α auf der Funktion $f_H\colon \{0,1\}^n \to \mathbb{R}$ ist durch*

$$\Omega\left(n\cdot\alpha^n + \frac{(1+1/\alpha)^{n-3}}{n^2}\right)$$

beschränkt. Das ist $\Omega\left(\left(\left(1+\sqrt{5}\right)/2\right)^n/n^2\right)$ für alle Werte von $\alpha \in [1;\infty)$.

Beweis. Für alle x mit $\|x\|_1 < n-2$ ist $f_{\mathrm{H}}(x) = f_1(x)$. Folglich haben p_i^- und p_i^+ die gleichen Werte für f_{H} und f_1, wenn $i < n-2$ gilt. Also gilt für alle $j < n-2$

$$\mathrm{E}\left(T_j^{+}\right) = \frac{1}{\alpha^j\binom{n-1}{j}} \sum_{0\leq k\leq j} \binom{n}{k}\alpha^k$$

bei der Funktion f_{H} analog zum Beweis von Lemma 8.10. Wir nehmen jetzt ohne Beschränkung der Allgemeinheit an, dass n gerade ist. Andernfalls runden wir geeignet und erhalten im wesentlichen gleiche Ergebnisse. Mit Wahrscheinlichkeit mindestens $1/2$ enthält der aktuelle String direkt nach der Initialisierung höchstens $n/2$ Bits mit Wert Eins. Wir orientieren uns am Beweis von Lemma 8.10 und sehen, dass

$$\begin{aligned}
\mathrm{E}\left(T_{n/2}\right) &\geq \frac{1}{2} \sum_{n/2 \leq j < n-2} \mathrm{E}\left(T_j^+\right) \geq \frac{\mathrm{E}\left(T_{n-3}^+\right)}{2} = \frac{1}{2\alpha^{n-3}\binom{n-1}{n-3}} \sum_{0 \leq j < n-2} \binom{n}{j} \alpha^j \\
&\geq \frac{1}{\alpha^{n-3} n^2} \sum_{0 \leq j \leq n-3} \binom{n-3}{j} \alpha^j = \frac{(1+1/\alpha)^{n-3}}{n^2}
\end{aligned}$$

gilt. Wir betrachten jetzt außerdem noch den Schritt von einem String mit $n-2$ Einsen zu einem String mit $n-1$ Einsen. Es ist

$$p_{n-2}^+ = \frac{2}{n} \cdot \alpha^{3n^2 - \left(3n^2 + n\right)} = \frac{2}{n \cdot \alpha^n}.$$

Offensichtlich ist ein solcher Schritt erforderlich mit Wahrscheinlichkeit mindestens $1 - (n+1)/2^n$, also haben wir

$$\mathrm{E}(T) = \Omega\left(n \cdot \alpha^n\right)$$

allein durch diesen Schritt. Das ergibt dann insgesamt

$$\mathrm{E}(T) = \Omega\left(n \cdot \alpha^n + \frac{(1+1/\alpha)^{n-3}}{n^2}\right),$$

wie behauptet. Wir lösen nach $\alpha = 1 + 1/\alpha$ auf und erhalten $\alpha = 1 + \alpha = \left(1+\sqrt{5}\right)/2$, also $\Omega\left(\left(\left(1+\sqrt{5}\right)/2\right)^n / n^2\right)$ wie behauptet. $\square$

Wir hatten uns schon überlegt, dass in dem Moment, in dem für den aktuellen String x von Algorithmus 8.4 $\|x\|_1 = n-2$ gilt, es günstig wäre, α klein zu haben. Wenn man einen Algorithmus mit adaptiver Parametersteuerung hat, kann man Bezug nehmen auf den aktuellen Zustand des Algorithmus und die Parameter dann passend einstellen. Wir nehmen von einem solchen Ansatz hier Abstand, weil er zu weit von unseren eigentlichen Interessen abgeht. Im Gegensatz zum üblichen Vorgehen im Bereich effizienter Algorithmen wollen wir ja nicht einen möglichst effizienten Algorithmus zu einem gegebenen Problem entwickeln. Unser Ziel ist es stets, einen gegebenen Algorithmus auf unterschiedlichen Problemen, also Zielfunktionen, zu analysieren. Ein genaues Einstellen des Algorithmus auf die zu optimierende Funktion wäre also „gemogelt“ und nicht in unserem Sinne.

Algorithmus 8.4 verwendet dynamische Parametersteuerung. Die Selektionsstrategie kann also nur von der Anzahl Generationen abhängen, nicht vom aktuellen String. Damit wollen wir „Mogeleien“ verhindern. Es ist aber andererseits selbstverständlich schon so, dass eine Selektionsstrategie für die Zielfunktion „passend“ sein muss. Nicht jede Selektionsstrategie führt zu einer effizienten Optimierung. Wir werden jetzt für eine relativ große Klasse von Selektionsstrategien zeigen, dass bei Einsatz einer solchen Selektionsstrategie die Funktion f_{H} mit großer Wahrscheinlichkeit in Polynomialzeit optimiert werden kann.

Satz 8.12. *Sei $s(n)$ ein Polynom mit $s(n) \geq 8en^3 \ln n$. Die erwartete Laufzeit von Algorithmus 8.4 mit Selektionsstrategie*

$$\alpha(t) := \max\left\{1 + \frac{1}{n}, \frac{s(n)}{t}\right\}$$

auf der Funktion $f_H\colon \{0,1\}^n \to \mathbb{R}$ beträgt $O\left(s(n)\right)$. Die Wahrscheinlichkeit, dass nicht innerhalb der ersten $O\left(s(n)\right)$ Generationen das globale Maximum erreicht wird, ist durch $O\left(e^{-n}\right)$ nach oben beschränkt.

Beweis. Die Grundidee des Beweises ist, den Lauf von Algorithmus 8.4 in verschiedene Phasen bestimmter Länge einzuteilen. In jeder Phase gibt es dann eine feste untere und eine feste obere Schranke für $\alpha(t)$. Wir verwenden die Ergebnisse über Algorithmus 8.4 mit konstanter Selektionsstrategie α, die wir hergeleitet haben, und setzen je nach benötigter Abschätzung für α die obere oder untere Schranke ein.

Die erste Phase umfasst genau die ersten $s(n)/n$ Generationen. Wir haben

$$\alpha(t) \geq \frac{s(n)}{s(n)/n} = n$$

in dieser Phase. Weil wir $s(n) \geq 8en^3 \ln n$ voraussetzen, haben wir, dass die Länge der ersten Phase durch $8en^2 \ln n$ nach unten beschränkt ist. Aus dem Beweis von Lemma 8.10 wissen wir, dass die erwartete Anzahl Generationen, bis Algorithmus 8.4 einen String erreicht, der mindestens $n-2$ Bits mit Wert Eins enthält, durch

$$\left(1 + \frac{1}{n}\right)^{n-1} \cdot n\left((\ln n) + 1\right) \leq 2en \ln n$$

nach oben beschränkt. Anwendung der Markov-Ungleichung ergibt, dass mit Wahrscheinlichkeit mindestens $1/2$ innerhalb von höchstens $4en \ln n$ Generationen ein solcher String x erreicht wird. Unsere Überlegungen im Beweis von Lemma 8.10 sind unabhängig vom Startzustand. Folglich können wir die erste Phase gedanklich in Subphasen der Länge $4en\ln n$ zerlegen. Wegen der unteren Schranke für $s(n)$ ist klar, dass wir wenigstens $2n$ solche Subphasen haben. Solange in diesen Subphasen nicht ein String mit mindestens $n-2$ Einsen erreicht wird, können wir die Subphasen als unabhängige Wiederholungen eines Zufallsexperiments mit Erfolgswahrscheinlichkeit 1/2 auffassen. Folglich wird innerhalb der ersten Phase ein String mit $n-2$ Einsen mit Wahrscheinlichkeit mindestens $1-2^{-2n}$ erreicht. Wenn ein solcher String einmal erreicht ist, ist es unwahrscheinlich, dass wieder ein String mit weniger Einsen erreicht wird. Die Wahrscheinlichkeit, in einer Generation zu einem solchen String zu wechseln ist in der ersten Phase (in der wir $\alpha(t) \geq n$ haben) nach oben durch

$$\frac{n-2}{n} \cdot \alpha^{(n-3)-\left(3n^2+n\right)} = n^{-3n^2+O(1)}$$

beschränkt. Die Wahrscheinlichkeit, innerhalb der ersten Phase einen solchen Schritt zu machen, ist also durch

$$s(n) \cdot n^{-3n^2+O(1)} = n^{-3n^2+O(1)}$$

nach oben beschränkt und wir haben insgesamt, dass mit Wahrscheinlichkeit $1-4^{-n}$ am Ende der ersten Phase für den aktuellen String x jedenfalls $\|x\|_1 \geq n-2$ gilt.

Die zweite Phase beginnt im Anschluss und endet, wenn entweder das globale Maximum von f_{H} gefunden ist oder insgesamt $s(n)+4n^3$ Generationen vergangen sind. Wir nehmen zu unseren Ungunsten an, dass am Anfang der zweiten Phase der aktuelle String genau $n-2$ Bits mit Wert Eins enthält. Die Wahrscheinlichkeit, innerhalb der zweiten Phase einen String mit weniger als $n-2$ Einsen zu erreichen, ist nach oben durch

$$\left(s(n)+4n^3\right) \cdot \frac{n-2}{n} \cdot \alpha^{(n-3)-\left(3n^2+n\right)} \leq \left(s(n)+4n^3\right) \cdot \alpha^{-3n^2}$$

beschränkt. Wir haben während des ganzen Laufs

$$\alpha(t) \geq 1+\frac{1}{n},$$

also wird mit Wahrscheinlichkeit mindestens

$$\left(s(n)+4n^3\right) \cdot \left(1+\frac{1}{n}\right)^{3n^2} = e^{O(\ln n)} \cdot e^{-\Omega(3/2n)} = O\left(e^{-n}\right)$$

innerhalb der zweiten Phase nicht wieder ein String mit weniger als $n-2$ Einsen erreicht.

Falls das globale Maximum nicht vorher erreicht wird, gibt es in der zweiten Phase $4n^3$ Generationen, in denen $\alpha(t) = 1+1/n$ gilt. Die Wahrscheinlichkeit, in zwei aufeinanderfolgenden Generationen in diesem Teil der zweiten Phase das globale Maximum von einem String mit genau $n-2$ Bits mit Wert Eins aus zu erreichen, ist nach unten durch

$$\left(\frac{2}{n} \cdot \alpha(t)^{3n^2 - \left(3n^2+n\right)}\right) \cdot \frac{1}{n} = \frac{2}{n^2} \cdot \left(1 + \frac{1}{n}\right)^n \geq \frac{2}{en^2}$$

beschränkt. Wir betrachten die $4n^3$ Generationen als $2n^3$ derartige „Doppelgenerationen“ und haben, dass mit Wahrscheinlichkeit mindestens

$$\left(1 - \frac{2}{en^3}\right)^{2n^3} \leq e^{-n}$$

innerhalb der zweiten Phase das globale Maximum gefunden wird.

Wir wissen jetzt also insgesamt, dass mit Wahrscheinlichkeit $1 - O\left(e^{-n}\right)$ innerhalb der ersten $s(n) + 4n^3$ Generationen das globale Maximum von f_{H} gefunden wird. Weil $s(n) = \Omega\left(n^3 \log n\right)$ ist, gilt folglich, dass innerhalb von $O\left(s(n)\right)$ Generationen mit Wahrscheinlichkeit $1 - O\left(e^{-n}\right)$ das globale Maximum erreicht wird.

Für die Aussage über die erwartete Laufzeit müssen wir uns jetzt noch überlegen, was passiert, falls das globale Maximum bis zum Ende der zweiten Phase noch nicht gefunden wurde. Das ist mit Wahrscheinlichkeit $O\left(e^{-n}\right)$ der Fall. Wir betrachten nur die Generationen im Anschluss an die zweite Phase, in denen $\alpha(t) = 1 + 1/n$ konstant ist. Für die Übergangswahrscheinlichkeit haben wir in diesen Generationen

$$p_i^+ = \frac{n-i}{n}, p_i^- = \frac{i}{(1+1/n)n}$$

für alle $i \in \{0, 1, \ldots, n-3\}$ und einerseits

$$p_{n-2}^+ = \frac{2}{(1+1/n)^n n}, p_{n-2}^- = \frac{n-2}{(1+1/n)^{3n^2+3}n}$$

sowie andererseits

$$p_{n-1}^+ = 1/n, p_{n-1}^- = \frac{n-1}{(1+1/n)^n n}$$

in den übrigen Fällen. Für alle $i \in \{0, 1, \ldots, n-1\}$ gilt

$$\mathrm{E}\left(T_i^+\right) = \sum_{0 \leq j \leq i} \frac{1}{p_j^+} \prod_{j < k \leq i} \frac{p_j^-}{p_j^+}.$$

Wir bemerken, dass $\mathrm{E}\left(T_i^+\right)$ streng monoton mit wachsenden Werten von p_j^+ und fallenden Werten p_j^-/p_j^+ wächst. Wir vergleichen jetzt $\mathrm{E}\left(T_i^+\right)$ mit einer rein zufälligen Irrfahrt, also mit Algorithmus 8.4, wenn man $\alpha = 1$ konstant wählt.

Ist $i \in \{0, 1, \ldots, n-3\}$, so ist die Wahrscheinlichkeit, von einem Elter mit i Einsen ein Kind mit $i+1$ Einsen zu erzeugen für unsere Situation und die rein zufällige Irrfahrt gleich. Die Wahrscheinlichkeit, von einem solchen Elter mit i Einsen ein Kind mit $i-1$ Einsen zu erzeugen, ist in unserer Situation im Verhältnis zu einer rein zufälligen Irrfahrt um den Faktor $1 + 1/n$ größer.

Ist $i = n-2$, so haben wir in unserer Situation

$$\frac{p_{n-2}^-}{p_{n-2}^+} = \frac{n-2}{(1+1/n)^{3n^2+3}n} \cdot \frac{(1+1/n)^n n}{2} = \frac{n-2}{2(1+1/n)^{3n^2-n+3}}$$

was für $n \to \infty$ schnell gegen 0 konvergiert. Im Falle der rein zufälligen Irrfahrt haben wir

$$\frac{n-2}{n} \cdot \frac{n}{2} = \frac{n-2}{2}.$$

Folglich ist $\mathrm{E}\left(T_i^+\right)$ für alle $i \in \{0, 1, \ldots, n-2\}$ nach oben durch den entsprechenden Erwartungswert bei der rein zufälligen Irrfahrt beschränkt. Das gilt ebenfalls für $i = n-1$, da die Wahrscheinlichkeit, die Anzahl Einsen von $n-1$ auf n zu erhöhen in unserer Situation mit der rein zufälligen Irrfahrt übereinstimmt und die Wahrscheinlichkeit, die Anzahl Einsen von $n-1$ auf $n-2$ zu senken bei der rein zufälligen Irrfahrt sogar viel größer ist als in unserer Situation. Folglich erhalten wir eine obere Schranke für die erwartete Anzahl zusätzlich benötigter Generationen, wenn wir eine rein zufällige Irrfahrt betrachten. Für diesen Fall, also für $\alpha = 1$, haben wir

$$\begin{aligned}
\mathrm{E}(T_i) &\leq \mathrm{E}(T_0) = \sum_{0 \leq j < n} \frac{1}{\binom{n-1}{j}} \sum_{0 \leq k \leq j} \binom{n}{k} \\
&< 2^n \sum_{0 \leq j < n} \frac{1}{\binom{n-1}{j}} = 2^n \left(2 + \sum_{1 \leq j \leq n-2} \frac{1}{\binom{n-1}{j}}\right) \\
&\leq 2^n \left(2 + \sum_{1 \leq j \leq n-2} \frac{1}{n-1}\right) < 2^n \cdot 3
\end{aligned}$$

als obere Schranke für die erwartete Anzahl Generationen bis zum Erreichen des Optimums unabhängig vom Startpunkt. Das gibt dann insgesamt

$$O\left(s(n)\right) + 3 \cdot 2^n \cdot O\left(e^{-n}\right) = O\left(s(n)\right)$$

als obere Schranke für die erwartete Laufzeit. $\square$

Damit haben wir also in der Funktion f_{H} ein erstes Beispiel, das zeigt, dass Simulated Annealing wesentlich schneller sein kann als ein optimal eingestellter Metropolis-Algorithmus. Wir haben gezeigt, dass selbst ein optimal eingestellter Metropolis-Algorithmus im Durchschnitt exponentiell lange zur Optimierung von f_{H} braucht, während Simulated Annealing mit passend gewählter „Abkühlstrategie" mit Wahrscheinlichkeit exponentiell nahe bei 1 und auch im Durchschnitt mit $O(n^3 \log n)$ Generationen auskommt. Kritisch anzumerken ist allerdings die gewählte Selektionsstrategie $\alpha(t)$. Offensichtlich ist $\alpha(t)$ monoton fallend, die Wahrscheinlichkeit, auch Verschlechterungen zu akzeptieren, nimmt also während des Laufs nur zu. Das entspricht „übersetzt" ins physikalische Vorbild einer monoton wachsenden Temperatur, was der Grundidee von Simulated Annealing gerade entgegengesetzt ist. Man sieht leicht ein (analog zum Beweis von Satz 8.11), dass diese Eigenschaft von α aber für den Erfolg auf f_{H} wesentlich ist. Jede monoton fallende Selektionsstrategie kann nicht besser sein als der Metropolis-Algorithmus. Damit ist f_{H} ein Beispiel, das eine stark nicht-natürliche Selektionsstrategie erfordert. Das motiviert die Betrachtung eines zweiten Beispiels, bei dem wir tatsächlich mit „Abkühlen" zum Erfolg kommen.

Wir wollen uns zunächst überlegen, welche Eigenschaften eine Zielfunktion zu einem geeigneten Kandidaten bei der Suche nach einer Funktion machen, für die eine Selektionsstrategie, die ein „Abkühlen" realisiert, erfolgversprechend ist. Wenn man anfangs eine hohe Temperatur des Systems vorgibt, also Verschlechterungen mit großer Wahrscheinlichkeit akzeptiert, sollte es für den Bereich des Suchraums, in dem Algorithmus 8.4 sich anfangs vermutlich aufhält, günstig sein, sich zunächst in Richtung kleinerer Funktionswerte zu bewegen. Wenn dann mit der Zeit Regionen mit größeren Funktionswerten erreicht sind, sollte es zunehmend günstiger werden, nur noch Hinweisen auf große Funktionswerte zu folgen, also Verschlechterungen nur noch mit kleiner, schließlich praktisch mit verschwindender Wahrscheinlichkeit zu folgen. Wir werden gleich eine Funktion vorstellen, welche diese Eigenschaften in Extremform realisiert. Vorher wollen wir noch kurz diskutieren, was davon zu halten ist, wenn nach längerer Zeit Verschlechterungen nur noch mit „verschwindender Wahrscheinlichkeit" akzeptiert werden.

Nehmen wir an, dass wir eine Zielfunktion f betrachten, die nicht unimodal ist. Dann gibt es mindestens ein lokales Maximum $x^* \in \{0,1\}^n$. Sei dieses lokale Maximum echt schlechter als ein globales Maximum, sei also $f(x^*) < \max\{f(x) \mid x \in \{0,1\}^n\}$. Andernfalls ist f zwar nicht unimodal, kann aber mit gewissem Recht als schwach unimodal bezeichnet werden: Gilt $f(x') = \max\{f(x) \mid x \in \{0,1\}^n\}$ für alle lokalen Maxima $x' \in \{0,1\}^n$, so genügt es offenbar zur Optimierung, irgendein beliebiges lokales Maximum zu erreichen und das befürchtete „Steckenbleiben“ in lokalen Maxima spielt keine Rolle. Sei f also weder unimodal noch schwach unimodal. Für jede natürliche Zahl t gibt es dann eine positive Wahrscheinlichkeit, dass Algorithmus 8.4 nach t Generationen als aktuellen Zustand x gerade dieses lokale Maximum x^*, das nicht global maximal ist, hat. Wenn die Wahrscheinlichkeit, Verschlechterungen zu akzeptieren, zu diesem Zeitpunkt sehr klein ist, ist die erwartete Zeit, bis x^* verlassen wird, sehr groß. Wenn wir zulassen, dass $\alpha(t)$ für $t \to \infty$ unbeschränkt wächst, dann gibt es eine positive Wahrscheinlichkeit, niemals ein globales Optimum zu erreichen. In diesem Fall ist dann die erwartete Laufzeit nicht definiert. Das ist in der Praxis unproblematisch, weil man ohnehin nicht beliebig lange auf einen Erfolg eines evolutionären Algorithmus wartet. Wenn wir uns aber aus einer eher theoretischen Sichtweise für erwartete Laufzeiten interessieren, sollten wir diese Möglichkeit ausschließen. Es gibt verschiedene sinnvolle Strategien, mit diesem Problem umzugehen. Man könnte eine obere Schranke für α festlegen oder α nach einer gewissen Anzahl von Generationen (etwa 2^n) auf 1 setzen und dann eine rein zufällige Suche beginnen. Diesen Ansatz werden wir hier verfolgen. Die Idee hinter dieser zweiten Möglichkeit ist, dass es sinnvoll ist, den Suchraum einfach zufällig zu durchmustern (oder auch aufzuzählen in einer beliebigen Reihenfolge), wenn man schon so lange gebraucht hat, dass man mit einer reinen Aufzählung schon sicher ein globales Maximum gefunden hätte. In diesen Fällen sind die Algorithmen offenbar sowohl aus praktischer als auch aus theoretischer Sicht unbrauchbar: aus praktischer Sicht braucht man eine Optimierung mit nicht zu kleiner Wahrscheinlichkeit in einer angemessen kurzen Anzahl von Generationen. Wir nehmen grob an, dass eine polynomielle Anzahl von Generationen effizient genug ist. Aus theoretischer Sicht ist es eine Mindestanforderung an jeden Suchalgorithmus, eine vollständige Aufzählung des Suchraums zu schlagen.

Die Funktion, die uns als Beispiel dient, dass auch ein tatsächlich „abkühlendes“ Simulated Annealing substanziell besser sein kann als ein optimal eingestellter Metropolis-Algorithmus, nennen wir f_K. Wie bereits besprochen verwirklicht sie in Extremform unsere Vorstellungen davon, wann ein Abkühlprozess sinnvoll sein kann.

Definition 8.13. *Sei $n \in \mathbb{N}$ gerade. Die Funktion $f_K\colon \{0,1\}^n \to \mathbb{R}$ ist definiert durch*

$$f_K(x) := \begin{cases} n/2 - \|x\|_1 & \text{falls } \|x\|_1 \leq n/2, \\ (7n^2 \ln n) - n/2 + \|x\|_1 & \text{falls } \|x\|_1 > n/2 \end{cases}$$

für alle $x \in \{0,1\}^n$.

Weil die Funktion f_K offensichtlich symmetrisch ist, können wir einen guten Eindruck von ihr durch eine Abbildung vermitteln (Abbildung 5).

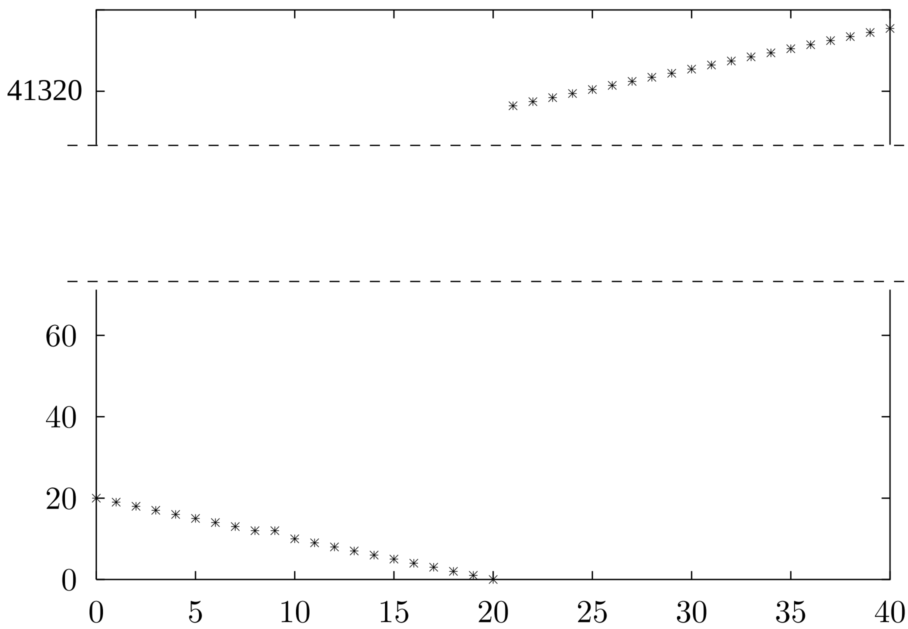

Abbildung 5: Die Funktion $f_K \colon \{0,1\}^{40} \to \mathbb{R}$.

Die Funktion teilt den Suchraum $\{0,1\}^n$ grob gesprochen in zwei klar verschiedene „Hälften". Für alle $x \in \{0,1\}^n$ mit $\|x\|_1 < n/2$ ist im wesentlichen die Funktion $-f_1$ zu optimieren. Für alle $x \in \{0,1\}^n$ mit $\|x\|_1 > n/2$ entspricht die Zielfunktion hingegen im wesentlichen f_1. Die Strings $x \in \{0,1\}^n$ mit $\|x\|_1 = n/2$ haben eine gewisse Ausnahmestellung: Alle Nachbarn mit Hammingabstand 1 haben größeren Funktionswert. In unseren Überlegungen spielen diese Strings keine wesentliche Rolle, darum ignorieren wir sie für den Moment und konzentrieren und auf die beiden beschriebenen „Hälften" des Suchraums. Zwischen den beiden Hälften gibt es eine große Differenz in den Funktionswerten, wir haben konkret für x' mit $\|x'\|_1 = (n/2) + 1$ und x'' mit $\|x''\|_1 = n/2$ eine Differenz von $f_K(x') - f_K(x'') = (7n^2 \ln n) - n + 1$. Nach zufälliger Initialisierung ist die Wahrscheinlichkeit, in der linken Hälfte des Suchraums (mit $\|x\|_1 < n/2$) zu sein, ebenso groß wie die Wahrscheinlichkeit, in der rechten Hälfte des Suchraums (mit $\|x\|_1 > n/2$) zu sein. In beiden Hälften des Suchraums ist aber jeweils eine gänzlich andere Strategie hilfreich. In der rechten Hälfte des Suchraums sollten Verschlechterungen möglichst vermieden werden, um rasch das globale Maximum **1** zu erreichen. Wir wissen aus Lemma 8.10, dass die erwartete Laufzeit bei fester Wahl von

α auf f_1 durch $O\left((1+1/\alpha)^n n \log n\right)$ beschränkt ist. Man sollte also etwa $\alpha = \Omega(n)$ haben. In der linken Hälfte des Suchraums führt jede Tendenz, Verbesserungen zu bevorzugen dazu, dass eher das lokale Maximum **0** erreicht wird, was ein Erreichen des globalen Maximums **1** erschwert. Es liegt also gewissermaßen auf der Hand, dass es keine passende feste Wahl für α gibt. Das weisen wir jetzt auch formal nach.

Satz 8.14. *Sei $n \in \mathbb{N}$ gerade. Die erwartete Laufzeit von Algorithmus 8.4 mit konstanter Selektionsstrategie α ist durch*

$$\Omega\left(\left(\sqrt{\frac{\alpha}{4}}\right)^n + \left(1+\frac{1}{\alpha}\right)^{n-4}\right)$$

nach unten beschränkt. Das ist $\Omega\left(1{,}179^n\right)$ für alle $\alpha \in [1;\infty)$.

Beweis. Wir beginnen wiederum damit, die Übergangswahrscheinlichkeiten p_i^+ und p_i^- in Abhängigkeit von α zu bestimmen. Dann können wir unsere Hilfsaussagen anwenden.

Für $j \in \{0,1,\ldots,(n-2)-1\}$, also in der linken Hälfte des Suchraums, haben wir

$$p_j^+ = \frac{n-j}{\alpha n}, p_j^- = \frac{j}{n}.$$

Wir wenden Folgerung 8.8 an und sehen, dass

$$\begin{aligned}
\mathrm{E}\left(T_i^+\right) &= \sum_{0\le k\le i} \frac{1}{p_k^+} \prod_{k<l\le i} \frac{p_l^-}{p_l^+} = \sum_{0\le k\le i} \frac{\alpha n}{n-k} \prod_{k<l\le i} \frac{l}{n} \cdot \frac{\alpha n}{n-l} \\
&= \sum_{0\le k\le l} \alpha^{i-k+1} \frac{n}{n-k} \frac{i!\cdot(n-i-1)!}{k!\cdot(n-k-1)!} = \sum_{0\le k\le i} \alpha^{i-k+1} \frac{\binom{n}{k}}{\binom{n-1}{i}}
\end{aligned}$$

für $i \in \{0,1,\ldots,(n/2)-1\}$ gilt. Wir betrachten speziell $\mathrm{E}\left(T_i^+\right)$ für $i = (n/2)-1$ und haben

$$\begin{aligned}
\mathrm{E}\left(T_{(n/2)-1}^+\right) &= \sum_{0\le k\le (n-2)-1} \alpha^{(n/2)-k} \frac{\binom{n}{k}}{\binom{n-1}{(n/2)-1}} \\
&\ge \alpha^{n/2} \frac{1}{\binom{n-1}{(n/2)-1}} \ge \frac{\alpha^{n/2}}{2^n} = \left(\sqrt{\frac{\alpha}{4}}\right)^n
\end{aligned}$$

in diesem Fall. Die Wahrscheinlichkeit, dass der aktuelle String direkt nach der Initialisierung höchstens $(n/2)-1$ Bits mit Wert Eins hat, ist nach unten durch $(1/2) - O(1/\sqrt{n})$ beschränkt. Folglich haben wir

$$\mathrm{E}(T) = \Omega\left(\mathrm{E}\left(T_{(n/2)-1}\right)\right) = \left(\sqrt{\frac{\alpha}{4}}\right)^n$$

als untere Schranke für die erwartete Laufzeit. Wenn α nicht zu klein ist, also $\alpha \geq 4+\varepsilon$ für eine positive Konstante ε, ist das schon exponentiell groß. Wir haben zum Nachweis dieser unteren Schranke ausgenutzt, dass beim Wechsel von $(n/2)-1$ Einsen zu $(n/2)$ Einsen der Funktionswert kleiner wird und bei nicht zu kleinen Werten von α Verschlechterungen unwahrscheinlich sind.

Um eine exponentiell große untere Schranke für noch kleinere Werte für α nachzuweisen, brauchen wir eine etwas andere Beweisstrategie. Wenn $\alpha \leq 4$ ist, dann rückt α in gewisser Weise „in die Nähe von 1“, was einer rein zufälligen Irrfahrt entspricht. Wir hatten uns schon überlegt, dass eine rein zufällige Irrfahrt in der rechten Hälfte des Suchraums eher ungünstig ist. Wir betrachten zunächst wieder die Übergangswahrscheinlichkeit. Für $i \in \{(n/2)+2, (n/2)+3, \ldots, n-1\}$ haben wir

$$p_i^+ = \frac{n-i}{n}, p_i^- = \frac{i}{\alpha n}.$$

Wir betrachten nur $\mathrm{E}\left(T_{n-1}^+\right)$ und haben zunächst wie vorhin

$$\begin{aligned}
\mathrm{E}\left(T_{n-1}^+\right) &= \sum_{0\leq k\leq n-1} \frac{1}{p_k^+} \prod_{k<l\leq n-1} \frac{p_l^-}{p_l^+} \geq \sum_{(n/2)+2\leq k\leq n-1} \frac{1}{p_k^+} \prod_{k<l\leq n-1} \frac{p_l^-}{p_l^+} \\
&= \sum_{0\leq k\leq (n/2)-3} \frac{\prod\limits_{n-k\leq l<n} p_l^-}{\prod\limits_{n-1-k\leq l<n} p_l^+} = \sum_{0\leq k\leq (n/2)-3} \frac{\prod\limits_{n-k\leq l<n} l/(\alpha n)}{\prod\limits_{n-1-k\leq l<n} (n-l)/n} \\
&= \sum_{0\leq k\leq (n/2)-3} \frac{1}{\alpha^k} \cdot \frac{n!}{(n-k-1)!\cdot(k+1)!} \\
&= \sum_{0\leq k\leq (n/2)-3} \frac{\binom{n}{k+1}}{\alpha^k} = \alpha \sum_{0<k\leq (n/2)-2} \frac{\binom{n}{k}}{\alpha^k} \\
&\geq \alpha\left(\frac{(1+1/\alpha)^{n-4}}{2} - 1\right) = \Omega\left(\left(1+\frac{1}{\alpha}\right)^{n-4}\right)
\end{aligned}$$

für diesen Schritt. Damit haben wir insgesamt

$$\mathrm{E}(T) = \Omega\left(\left(\sqrt{\frac{\alpha}{4}}\right)^n + \left(1+\frac{1}{\alpha}\right)^{n-4}\right)$$

als untere Schranke für die erwartete Laufzeit. Mit $\alpha \leq 5{,}566$ haben wir $1 + 1/\alpha > 1{,}179$ und mit $\alpha > 5{,}566$ haben wir $\sqrt{\alpha/4} > 1{,}179$, also $\mathrm{E}(T) = \Omega(1{,}179^n)$ für jede Wahl von α. $\square$

Dass eine konstante Selektionsstrategie α bei der Funktion f_{K} nicht zum Erfolg führt, war intuitiv klar. Weniger offensichtlich ist, dass es überhaupt eine Selektionsstrategie α gibt, die zu einer Optimierung von f_{K} durch Algorithmus 8.4 in polynomieller Zeit mit großer Wahrscheinlichkeit führt. Die Idee für die Definition von α ist die folgende. Wenn man anfangs α sehr nahe bei 1 wählt, gleicht Algorithmus 8.4 einer rein zufälligen Irrfahrt. Dann ist die Wahrscheinlichkeit, innerhalb einer recht kleinen Anzahl von Generationen irgendwann einmal in die rechte Hälfte des Suchraums zu kommen, sehr groß. Wenn andererseits α dann auch nicht wieder zu klein ist, ist die Wahrscheinlichkeit, wieder in die linke Hälfte des Suchraums zu wechseln, sehr klein, weil die Differenz in den Funktionswerten am Übergang von der rechten in die linke Suchraumhälfte so groß ist. Wenn anschließend α wächst, kann in der rechten Hälfte des Suchraums das globale Maximum ebenso effizient gefunden werden wie bei der Funktion f_1. Wir formalisieren diese Ideen. Um eine Aussage nicht nur mit großer Wahrscheinlichkeit machen zu können, sondern auch eine Aussage über die erwartete Laufzeit, werden wir wie besprochen $\alpha(t) = 1$ wählen für alle $t > 2^n$.

Satz 8.15. *Sei $n \in \mathbb{N}$ gerade mit $n > 2$. Sei $s(n)$ ein Polynom mit $s(n) \geq 2en^4 \log n$. Die Wahrscheinlichkeit, dass Algorithmus 8.4 mit Selektionsstrategie*

$$\alpha(t) := 1 + \frac{t}{s(n)}$$

innerhalb der ersten $O(ns(n))$ Generationen das globale Maximum der Funktion $f_K\colon \{0,1\}^n \to \mathbb{R}$ findet, ist durch $1 - O(n^{-n})$ nach unten beschränkt. Die erwartete Laufzeit von Algorithmus 8.4 mit Selektionsstrategie

$$\alpha(t) := \begin{cases} 1 + t/s(n) & \textit{falls } t \leq 2^n, \\ 1 & \textit{sonst} \end{cases}$$

auf der Funktion $f_K\colon \{0,1\}^n \to \mathbb{R}$ ist durch $O(ns(n))$ nach oben beschränkt.

Beweis. Wir führen den Beweis ganz ähnlich wie den Beweis zu Satz 8.12. Wir teilen einen Lauf von Algorithmus 8.4 in zwei Phasen fester Länge auf. Wir zeigen, dass mit großer Wahrscheinlichkeit am Ende der ersten Phase der aktuelle String x mindestens $(n/2) + 1$ Bits mit Wert Eins enthält, also zur rechten Suchraumhälfte gehört, und in der zweiten Phase unter der

Voraussetzung, diese in der rechten Suchraumhälfte zu beginnen, mit großer Wahrscheinlichkeit das globale Maximum von f_{K} gefunden wird.

Die erste Phase hat Länge $s(n)/n + 2en^3 \log n$. Wir ignorieren die ersten $s(n)/n$ Generationen der ersten Phase und betrachten nur die darauf folgenden $2en^3 \log n$ Generationen. In diesem zweiten Teil der ersten Phase gilt

$$1 + \frac{1}{n} \leq \alpha(t) \leq 1 + \frac{2}{n}$$

während aller Generationen. Wir können unsere Überlegungen zu den Übergangswahrscheinlichkeiten aus Satz 8.14 übernehmen und haben darum für $i \in \{0, 1, \ldots, (n/2) + 1\}$, dass

$$\begin{aligned}
\mathrm{E}\left(T_i^+\right) &\leq \sum_{0 \leq j \leq i} \beta^{i-j+1} \frac{\binom{n}{j}}{\binom{n-1}{i}} = \sum_{0 \leq j \leq i} \beta^{j+1} \frac{\binom{n}{i-j}}{\binom{n-1}{i}} \\
&= \sum_{0 \leq j \leq i} \beta^{j+1} \frac{n!}{(i-j)! \cdot (n-i+j)!} \cdot \frac{i! \cdot (n-1-i)!}{(n-1)!} \\
&= \sum_{0 \leq j \leq i} \beta^{j+1} \frac{\binom{i}{j}}{\binom{n-i+j}{j}} \frac{n}{n-i}
\end{aligned}$$

gilt, wenn $\alpha(t) \leq \beta$ für alle betrachten Generationen gilt. Offensichtlich gilt $\mathrm{E}\left(T_i^+\right) \leq \mathrm{E}\left(T_{i+1}^+\right)$ für alle $i \in \{0, 1, \ldots, (n/2) - 1\}$. Wie wir uns bereits überlegt haben, können wir $\beta := 1 + 2/n$ einsetzen und haben dann

$$\begin{aligned}
\mathrm{E}\left(T_i^+\right) \leq \mathrm{E}\left(T_{(n/2)-1}^+\right) \leq \sum_{0 \leq j < n/2} \left(1 + \frac{2}{n}\right)^{j+1} \frac{\binom{(n/2)-1}{j}}{\binom{(n/2)+1+j}{j}} \frac{n}{(n/2)+1} \\
\leq 2 \sum_{0 \leq j < n/2} e = en
\end{aligned}$$

als obere Schranke. Wir benutzen Lemma 8.6 und haben

$$\mathrm{E}\left(T_{n/2}^+\right) = \frac{1}{p_{n/2}^+} + \frac{p_{n/2}^-}{p_{n/2}^+} \mathrm{E}\left(T_{(n/2)-1}^+\right) \leq 2 + en$$

als obere Schranke für die erwartete Anzahl Generationen für diesen Übergang in die rechte Hälfte des Suchraums. Insgesamt ist also die erwartete Anzahl Generationen, bis erstmals die rechte Hälfte des Suchraums erreicht wird, durch

$$\frac{n}{2} \cdot en + 2 + en \leq en^2$$

nach oben beschränkt. Durch Anwendung der Markov-Ungleichung wissen wir, dass die Wahrscheinlichkeit, nicht innerhalb der ersten $2en^2$ Generationen wenigstens einmal die rechte Suchraumhälfte zu erreichen, nach oben durch $1/2$ beschränkt ist. Unsere Analyse ist unabhängig von dem String, der als erstes bei der Initialisierung erzeugt wird, darum können wir die $2en^3 \log n$ Generationen der ersten Phase, die wir betrachten, als $n \log n$ Subphasen der Länge jeweils $2en^2$ auffassen. Die Wahrscheinlichkeit, in keiner dieser Subphasen die rechte Suchraumhälfte zu erreichen, ist durch $(1/2)^{n \log n} = n^{-n}$ nach oben beschränkt.

Nehmen wir nun an, dass Algorithmus 8.4 die rechte Suchraumhälfte erreicht. Wir setzen voraus, dass die Anzahl der Generationen zu diesem Zeitpunkt mindestens $t \geq s(n)/n$ ist. Frühere Besuche in der rechten Suchraumhälfte beachten wir nicht. Es ist $\alpha(t) \geq 1 + 1/n$, also ist die Wahrscheinlichkeit, die rechte Hälfte des Suchraums wieder zu verlassen durch

$$\frac{(n/2)+1}{n} \cdot \left(1 + \frac{1}{n}\right)^{7n^2 \ln n} = O\left(n^{-n}\right)$$

nach oben beschränkt. Die Wahrscheinlichkeit, die rechte Suchraumhälfte innerhalb von $ns(n)$ Generationen wieder zu verlassen, wenn in allen Generationen $\alpha(t) \geq 1 + 1/n$ gilt, ist durch $ns(n) \cdot O\left(n^{-n}\right) = O\left(n^{-n}\right)$ nach oben beschränkt. Wir haben jetzt also insgesamt gezeigt, dass mit Wahrscheinlichkeit $1 - O\left(n^{-n}\right)$ am Ende der ersten Phase der aktuelle String x mindestens $(n/2) + 1$ Bits mit Wert Eins enthält.

Die zweite Phase beginnt am Ende der ersten Phase und endet nach insgesamt $n \cdot s(n)$ Generationen. Wir gehen davon aus, am Anfang der zweiten Phase mit einem String x zu beginnen, der wenigstens $(n/2) + 1$ Bits mit Wert Eins enthält. Wir haben schon gesehen, dass die Wahrscheinlichkeit, innerhalb der zweiten Phase wieder in die linke Suchraumhälfte zu wechseln, durch $O\left(n^{-n}\right)$ nach oben beschränkt ist. Wir ignorieren den Anfang der zweiten Phase und kümmern uns nur um Generationen mit $t \geq (n-1)s(n)$. Wir haben dann $\alpha(t) \geq n$ für den Rest der zweiten Phase. Wir wollen jetzt eine obere Schranke für die erwartete Anzahl Generationen angeben, die bis zum Erreichen des globalen Maximum benötigt werden. Wir haben analog zu unseren Überlegungen im Beweis zu Satz 8.14

$$\begin{aligned} \mathrm{E}\left(T^{+}_{(n/2)-1}\right) &\leq \sum_{0 \leq j < n/2} (n+1)^{(n/2)-j} \cdot \frac{\binom{n}{j}}{\binom{n-1}{(n/2)-1}} \\ &\leq \sum_{0 \leq j < n/2} \binom{n}{j} n^{n-j} = \frac{(n+1)^n}{2} \end{aligned}$$

als obere Schranke für $\mathrm{E}\left(T^{+}_{(n/2)-1}\right)$. Das gibt

$$\mathrm{E}\left(T^{+}_{n/2}\right) = \frac{n}{n/2} + \frac{(n/2)/n}{(n/2)/n} \cdot \mathrm{E}\left(T^{+}_{(n/2)-1}\right) \leq 2 + \frac{(n+1)^n}{2}$$

als obere Schranke $\mathrm{E}\left(T^{+}_{n/2}\right)$ und darauf dann aufbauend

$$\begin{aligned}
\mathrm{E}\left(T^{+}_{(n/2)+1}\right) &\leq \frac{1}{((n/2)-1)/n} + \frac{((n/2)+1)/\left(n \cdot n^{7n^2 \ln n}\right)}{((n/2)-1)/n} \mathrm{E}\left(T^{+}_{n/2}\right) \\
&\leq \frac{2n}{n-2} + \frac{n+2}{2n^{(7n^2 \ln n)+1}} \frac{2n}{n-2} (n+1)^n \\
&\leq 6 + \frac{5(n+1)^n}{n^{(7n^2 \ln n)+1}} \leq 7
\end{aligned}$$

als obere Schranke für $\mathrm{E}\left(T^{+}_{(n/2)+1}\right)$, wobei wir $n > 3$ voraussetzen.

Wir wenden jetzt Lemma 8.7 mit $j = i - (n/2) - 1$ an und erhalten damit für $i \in \{(n/2)+2, (n/2)+3, \ldots, n-1\}$

$$\begin{aligned}
\mathrm{E}\left(T^{+}_{i}\right) &= \left(\sum_{0 \leq j \leq i-(n/2)-2} \frac{\prod\limits_{0 \leq k < j} p^{-}_{i-k}}{\prod\limits_{0 \leq k \leq j} p^{+}_{i-k}} \right) + \frac{\prod\limits_{0 \leq k \leq i-(n/2)-2} p^{-}_{i-k}}{\prod\limits_{0 \leq k \leq i-(n/2)-2} p^{+}_{i-k}} \cdot \mathrm{E}\left(T^{+}_{(n/2)+1}\right) \\
&= \left(\sum_{0 \leq j \leq i-(n/2)-2} \frac{\prod\limits_{i-j < k \leq i} p^{-}_{k}}{\prod\limits_{i-j \leq k \leq i} p^{+}_{k}} \right) + \frac{\prod\limits_{(n/2)+2 \leq k \leq i} p^{-}_{k}}{\prod\limits_{(n/2)+2 \leq k \leq i} p^{+}_{k}} \cdot 7
\end{aligned}$$

als obere Schranke für den Rest der rechten Hälfte des Suchraums. Für alle x mit $\|x\|_1 \geq (n/2)+2$ verhält sich Algorithmus 8.4 auf der Funktion f_{K} wie auf f_1. Die vom Betrag her größeren Funktionswerte spielen keine Rolle, weil die Differenzen zwischen den Funktionswerten gleich sind. Wir können darum unsere Ergebnisse über f_1 anwenden und haben

$$\begin{aligned}
\mathrm{E}\left(T_i^+\right) &\leq \left(\sum_{0 \leq j \leq i-(n/2)-2} \frac{\prod\limits_{i-j<k\leq i} k/(n\cdot n)}{\prod\limits_{i-j\leq k\leq i} (n-k)/n} \right) + \frac{\prod\limits_{(n/2)+2\leq k\leq i} k/(n\cdot n)}{\prod\limits_{(n/2)+2\leq k\leq i} (n-k)/k} \cdot 7 \\
&= \left(\sum_{0 \leq j \leq i-(n/2)-2} \frac{n^{-2j}\cdot i!/(i-j)!}{n^{-j-1}\cdot (n-i+j)!/(n-i-1)!} \right) \\
&\quad + \frac{n^{2i+n+2}\cdot i!/((n/2)+1)!}{n^{-i+(n/2)+1}\cdot ((n/2)-2)!/(n-i-1)!} \cdot 7 \\
&= \left(\sum_{0 \leq j \leq i-(n/2)-2} n^{1-j} \cdot \frac{\binom{i}{j}}{(n-i)\binom{n-i+j}{j}} \right) \\
&\quad + \frac{((n/2)-1)\binom{n}{(n/2)+1} n^{(n/2)+1}}{(n-i)\binom{n}{i} n^i} \cdot 7
\end{aligned}$$

als obere Schranke. Um eine obere Schranke für den zweiten Summanden angeben zu können, betrachten wir folgende Äquivalenzen für $i \in \{0,1,\ldots,n-2\}$.

$$\begin{aligned}
& (n-i)\binom{n}{i} n^i \leq (n-(i+1))\binom{n}{i+1} n^{i+1} \\
\Leftrightarrow\ & \frac{n-i}{n-i-1} \cdot \frac{n!}{i!\cdot(n-i)!} \cdot \frac{(i+1)!\cdot(n-i-1)!}{n!} \leq n \\
\Leftrightarrow\ & \frac{i+1}{n-i-1} \leq n
\end{aligned}$$

Da wir $i \in \{0,1,\ldots,n-2\}$ voraussetzen, ist die letzte Ungleichung offensichtlich wahr. Wir haben also bis jetzt insgesamt

$$\mathrm{E}\left(T_i^+\right) \leq \left(\sum_{0 \leq j \leq i-(n/2)-2} n^{1-j} \cdot \frac{\binom{i}{j}}{(n-i)\binom{n-i+j}{j}} \right) + 7$$

als obere Schranke. Um eine obere Schranke für alle $\mathrm{E}\left(T_i^+\right)$ mit $i \in \{(n/2)+2,(n/2)+3,\ldots,n-1\}$ zu bekommen, genügt es folglich, eine obere Schranke für $\mathrm{E}\left(T_{n-1}^{+}\right)$ zu finden. Es gilt

$$\begin{aligned}
\mathrm{E}\left(T_{n-1}^{+}\right) &\leq n\left(\sum_{0\leq j\leq (n/2)-3}\left(\frac{1}{n}\right)^{j}\frac{\binom{n-1}{j}}{\binom{j+1}{j}}\right)+7\\
&\leq n\left(\sum_{0\leq j\leq (n/2)-3}\left(\frac{1}{n}\right)^{j}\binom{n}{j}\right)+7\\
&\leq n\cdot\frac{(1+1/n)^{n}}{2}+7\leq\frac{en}{2}+7\leq 4n
\end{aligned}$$

für diesen letzten Schritt und somit für alle Schritte in der rechten Hälfte des Suchraums in dieser Phase. Folglich haben wir

$$\mathrm{E}\left(T_{i}\right)\leq\frac{n}{2}\cdot 4n=2n^{2}$$

als obere Schranke für die erwartete Anzahl Generationen, in der zweiten Phase das globale Maximum von f_{K} zu erreichen. Wir folgern mit Hilfe der Markov-Ungleichung, dass mit Wahrscheinlichkeit mindestens $1/2$ nach nicht mehr als $4n^2$ Generationen das globale Maximum erreicht ist. Wir betrachten hier wiederum nur die $s(n)\geq 2en^4\log n$ Generationen als mindestens $n^2\log n$ unabhängige Subphasen, die jeweils Länge mindestens $4n^2$ haben. Daraus folgt, dass das globale Maximum in wenigstens einer dieser Subphasen mit Wahrscheinlichkeit mindestens $1-O\left(n^{-n}\right)$ erreicht wird. Daraus folgt insgesamt, dass mit Wahrscheinlichkeit $1-O\left(n^{-n}\right)$ innerhalb der ersten $n\cdot s(n)$ Generationen das globale Maximum von f_{K} gefunden wird.

Um die obere Schranke für die erwartete Laufzeit nachzuweisen, betrachten wir den Fall, dass nach $n\cdot s(n)$ Generationen das globale Maximum noch nicht erreicht wurde. Dieser Fall hat Wahrscheinlichkeit $O\left(n^{-n}\right)$. Wir ignorieren alle Generationen, bis $t>2^n$ gilt. Dann haben wir $\alpha(t)=1$ für alle nachfolgenden Generationen. Wir haben uns im Beweis zu Satz 8.12 schon überlegt, dass dann die erwartete Anzahl Generationen bis zum Erreichen des Optimums durch $3\cdot 2^n$ nach oben beschränkt ist. Das gibt insgesamt

$$\mathrm{E}\left(T\right)=O\left(n\cdot s(n)\right)+3\cdot 2^{n}\cdot O\left(n^{-n}\right)=O\left(n\cdot s(n)\right)$$

wie behauptet. □

Die Funktion f_{K} ist also in der Tat ein zweites Beispiel, bei dem ein gut konfiguriertes Simulated Annealing einen optimal eingestellen Metropolis-Algorithmus schlägt und polynomielle Laufzeit erreicht sowohl im Durchschnitt als auch mit hoher Wahrscheinlichkeit, während der Metropolis-Algorithmus auch bei optimaler Parametrisierung exponentielle erwartete Laufzeit hat. Dabei haben wir das Resultat für f_{K} für eine Klasse von Selektionsstrategien gezeigt, die als natürlich gelten kann. Die „Temperatur" fällt einfach linear. Das plötzliche Steigen der Temperatur zum Schluss ist zwar künstlich, war aber für Optimierung in Polynomialzeit mit Wahrscheinlichkeit $1 - O\left(n^{-n}\right)$ auch nicht erforderlich. Die Frage von Jerrum und Sinclair (1997) bleibt aber offen. Auch wir haben hier kein natürliches Problem präsentieren können, für das sich ein so deutlicher Vorteil für Simulated Annealing nachweisen lässt.

9 Evolutionäre Algorithmen mit Rekombination

Wir haben in den vorangegangenen Kapiteln eine große Anzahl Erkenntnisse über den (1+1) EA und einige Varianten davon gewonnen. Wir hoffen natürlich, dass wir dabei Wirkprinzipien erkannt haben, wie sie auch bei „realistischeren“ evolutionären Algorithmen, also bei Instanzen des allgemeinen Prinzips „evolutionärer Algorithmus“, die in der Praxis von Ingenieuren eingesetzt werden, auftreten. Wir haben auch erkennen müssen, dass schon die Analyse eines so einfachen evolutionären Algorithmus, wie es der (1+1) EA ist, auf relativ einfach strukturierten, künstlichen Zielfunktionen sehr schwierig sein kann. Es wurde in den Kapitel 7 und 8 auch deutlich, dass die relativ bescheidenen Variationsmöglichkeiten beim (1+1) EA zu Varianten führen können, die sich auf einigen Funktionen völlig anders verhalten als ihr „Stammvater“ und die einen mitunter auch vor deutlich schwierigere Analyseprobleme stellen. Trotzdem wollen wir in diesem Kapitel versuchen, „realistischen“ evolutionären Algorithmen ein großes Stück näherzukommen. Wir wollen den in der Praxis wichtigen Suchoperator Crossover, den wir bis jetzt völlig aus unseren Überlegungen ausgeschlossen haben, untersuchen. Konkret wollen wir nachweisen, dass es Funktionen gibt, bei denen ein mehr oder minder typischer genetischer Algorithmus sowohl mit großer Wahrscheinlichkeit als auch im Durchschnitt effizient einen global maximalen Punkt findet, während nicht nur der (1+1) EA, sondern auch viele andere rein mutationsbasierte evolutionäre Algorithmen deutlich länger brauchen. Wir müssen zu diesem Zweck auf der Algorithmenseite zwei Dinge gleichzeitig tun. Wir müssen erstens einen evolutionären Algorithmus mit einer echten Population von Individuen untersuchen und zweitens Crossover als Suchoperator zulassen. Auf der Problemseite müssen wir eine klar strukturierte Zielfunktion angeben, von der wir annehmen und dann auch nachweisen können, dass Crossover sich essenziell effizienzsteigernd auswirkt.

Bevor wir uns diesen beiden Aufgaben konkret widmen, wollen wir versuchen zu verstehen, warum gerade Rekombination die Analyse evolutionärer Algorithmen entscheidend schwieriger macht. Wir haben gesehen, dass sich einfache mutationsbasierte evolutionäre Algorithmen gut als diskrete Markovketten modellieren und analysieren lassen. Führt man Rekombination ein, ändert sich die Situation. Weil Rekombination (typischerweise) auf zwei Individuen der aktuellen Population operiert, kommt man von einem linearen stochastischem System, das in gewisser Weise gut verstanden und analysierbar ist, zu einem quadratischen dynamischen System, das im allgemeinen viel schlechter handhabbar ist. Quadratische, dynamische Systeme führen zu Fragen, deren Beantwortung PSPACE-vollständig ist, die also von extremer Schwierigkeit sind (Arora, Rabani und Vazirani 1994). Betrachtet man Rekombination isoliert, ändert sich das allerdings. Es ist seit langem bekannt, dass eine Population bei alleiniger Anwendung von Crossover zu einem eindeutigen Fixpunkt konvergiert, der Equilibrium genannt wird und bei dem die Wahrscheinlichkeit für das Vorkommen eines Bitstrings $x = x_1 x_2 \cdots x_n$ gegeben ist durch die Produktwahrscheinlichkeit $\prod\limits_{1 \leq i \leq n} p_i$, wobei p_i die Wahrscheinlichkeit bezeichnet, in der initialen Population an Position i ein Bit mit Wert x_i zu finden (Geiringer 1944). Es ist sogar für die wichtigsten Crossoveroperatoren bekannt, wie schnell dieser Zustand approximativ erreicht wird (Rabani, Rabinovich und Sinclair 1998). Diese Überlegungen sind aber nicht mehr gültig, wenn Selektion eingeführt wird. Es gibt verschiedene Ansätze, diesem Problem zu entkommen. Man kann etwa für eine hinreichend lange Zeit Crossover ohne Selektion anwenden oder gleich die bekannte Grenzverteilung als erreicht annehmen und in jeder Generation die neue Population gemäß dieser Verteilung generieren (Mühlenbein 1998). Dieses Verfahren hat den Nachteil, dass implizit die Wichtigkeit der Selektion herabgesetzt wird, was wenigstens der Intuition klar widerspricht. Außerdem sind solche Algorithmen weit von „realistischen“ evolutionären Algorithmen entfernt, um die es uns hier ja geht. Ein anderer Ansatz ist eher physikalisch orientiert und betrachtet den evolutionären Algorithmus als ein komplexes dynamische System, dass anhand statistischer Merkmale wie dem durchschnittlichen Funktionswert der Population, Varianz etc. hinreichend genau beschrieben werden kann. Nimmt man für die Analyse zusätzlich an, dass die Populationsgröße gegen ∞ strebt, erhält man ein prinzipiell gut analysierbares System. An Stellen, wo Informationen benötigt werden, die aus den aktuell bekannten statistischen Größen nicht gewonnen werden können, nimmt man an, dass die Verteilung der Population unter all den Verteilungen, die den statistischen Informationen genügen, diejenige mit maximaler Entropie ist. Praktische Beispiele für dieses Vorgehen liefern beispielsweise Rattray und Shapiro (1996) und Prügel-Bennet und Shapiro (1997). Dieses Vorgehen führt dazu, dass evolutionäre Algorithmen untersucht werden, wie Physiker die „reale Welt“ untersuchen: als einen Gegenstandsbereich, der nicht vollständig erfasst werden kann und dem man darum mit Approximationen und Theorien zu Leibe rückt, die dann empirisch überprüft und gegebenenfalls falsifiziert werden. Dieser Ansatz kann ohne Zweifel wichtige und hilfreiche Einsichten vermitteln. Tatsächlich sind aber evolutionäre Algorithmen randomisierte Algorithmen und als solche mathematische Objekte, die durch eine mathematischen Analyse zumindest im Prinzip vollständig erschlossen werden können.

Darum kann eine Analyse „Wahrheiten“ über evolutionäre Algorithmen in letzter Konsequenz nur dann liefern, wenn sie nicht auf unbewiesenen Annahmen fußt, so plausibel diese Annahmen auch erscheinen mögen. In diesem Sinne wollen wir in diesem Kapitel zunächst den folgenden, noch recht einfachen genetischen Algorithmus untersuchen, der sich grob als Steady State GA mit uniformen Crossover charakterisieren lässt.

Algorithmus 9.1.

1. *Für alle* $i \in \{1, 2, \ldots, n\}$
   Wähle $x_i \in \{0,1\}^n$ *zufällig gleichverteilt.*
2. $r := 1$
3. *Mit Wahrscheinlichkeit* $1/(n \log n)$*:*
   $r := 0$
   Wähle $i_1, i_2 \in \{1, 2, \ldots, n\}$ *zufällig gleichverteilt.*
   Für alle $j \in \{1, 2, \ldots, n\}$
   Mit Wahrscheinlichkeit $1/2$*:*
   Setze das j*-te Bit in* y *gleich dem* j*-ten Bit in* x_{i_1}*.*
   Sonst setze das j*-te Bit in* y *gleich dem* j*-ten Bit in* x_{i_2}*.*
   Sonst wähle $y \in \{x_1, x_2, \ldots, x_n\}$ *zufällig gleichverteilt.*
4. *Für alle* $i \in \{1, 2, \ldots, n\}$*:*
   Mit Wahrscheinlichkeit $1/n$*:*
   Ersetze das i*-te Bit in* y *durch sein Komplement.* $r := 0$
5. *Falls* $r = 1$*, weiter bei* 3.
6. $m := \min\{f(x_i) \mid i \in \{1, 2, \ldots, n\}\}$
7. *Falls* $f(y) \geq m$
   Wähle $i \in \{j \mid f(x_j) = m\}$ *zufällig gleichverteilt.*
   Setze $x_i := y$*.*
8. *Weiter bei* 2.

Wir diskutieren kurz den Algorithmus. In Zeile 1 wird die Population der Größe n zufällig gleichverteilt initialisiert. Die Population besteht aus n Strings der Länge n, die mit $x_1, x_2, \ldots, x_n$ bezeichnet werden. In Zeile 2 wird ein Bit r auf 1 gesetzt. Wir erläutern die Funktion dieses Bits, wenn wir es in Zeile 5 benutzen. In Zeile 3 wird das so genannte uniforme Crossover realisiert. Zunächst werden zwei Eltern x_{i_1} und x_{i_2} zufällig gleichverteilt aus der Population gewählt. Üblicherweise wird solche Selektion in Abhängigkeit von den Funktionswerten unter der Zielfunktion f durchgeführt. Je größer dabei die Bevorzugung von Individuen mit großen Funktionswerten ist, desto „stärker“ nennt man den Selektionsmechanismus. Gänzlich von den Funktionswerten unabhängig zufällig gleichverteilt zu wählen, ist in sofern der

„schwächste“ denkbare Selektionsmechanismus. Strings mit kleinerem Funktionswert zu bevorzugen, schließen wir hier aus. Das widerspräche der Idee evolutionärer Algorithmen auf ähnliche Weise wie die Wahl einer Mutationswahrscheinlichkeit $p(n) > 1/2$, was wir in Kapitel 3 aus diesem Grund auch ausgeschlossen haben. Wir werden uns hier nicht mit anderen Selektionsmechanismen beschäftigen. Alle Ergebnisse dieses Kapitels gelten aber auch, wenn man an dieser Stelle einen beliebigen stärkeren Selektionsmechanismus verwendet, der allerdings nur von den Funktionswerten abhängen darf. Eine Crossoveroperation wird in Algorithmus 9.1 allerdings nur mit Wahrscheinlichkeit $1/(n \log n)$ durchgeführt, was ungewöhnlich unwahrscheinlich ist. Falls Crossover zum Einsatz kommt, merken wir uns das, in dem wir r auf Null setzen. Danach haben wir ein Kind y, das entweder Ergebnis des uniformen Crossover ist oder einfach ein zufällig gewähltes Elter. Dieser String y wird der gleichen Mutation unterworfen, wie sie auch beim (1+1) EA zum Einsatz kommt. Wir legen uns dabei auf die Mutationswahrscheinlichkeit $1/n$ fest. Falls wenigstens ein Bit tatsächlich mutiert, merken wir uns das, indem wir r auf Null setzen. In Zeile 5 prüfen wir, ob Crossover stattgefunden hat oder wenigstens ein Bit tatsächlich mutierte. Wenn beides nicht der Fall ist, sprechen wir von einer Replikation: Ein Elter aus der Population wurde ausgewählt und wird völlig unverändert als Kind beibehalten. Wir schließen in Algorithmus 9.1 solche Replikationen aus. Das ist ein etwas ungewöhnliches Vorgehen für genetische Algorithmen, das sich aber gut motivieren lässt.

Nach zufällig gleichverteilter Initialisierung starten genetische Algorithmen mit einer Population sehr unähnlicher Strings, man spricht von großer Diversität. Durch die Selektion, die typischerweise Individuen mit überdurchschnittlich großen Funktionswerten bevorzugt, nimmt diese Diversität aber rasch ab. Im Extremfall enthält die Population nur noch Kopien eines einzigen Individuums, was einer erfolgreichen Suche abträglich sein kann. Um das zu vermeiden, wird gelegentlich vorgeschlagen, keine Duplikate in der Population zuzulassen (Ronald 1998). Das erfordert, dass jedes Kind vor dem Einfügen in die aktuelle Population daraufhin getestet werden muss, ob es eventuell schon vorhanden ist. Durch Verwaltung eines Wörterbuchs für die aktuelle Population kann dieses Problem zwar effizient gelöst werden, einfacher und schneller jedoch ist die Vermeidung von Replikationen, wie wir sie hier vorschlagen. Es genügt ein lokal auf das Kind bezogener Test ohne Berücksichtigung der gesamten Population. Unsere Analyse wird im Anschluss zeigen, dass dieser einfache Mechanismus ausreicht, um positive Effekte zu erzielen.

In Zeile 6 kürzen wir aus Gründen der Bequemlichkeit den aktuell minimalen Funktionswert der Population mit m ab. Wenn der Funktionswert $f(y)$ des

Kindes y mindestens so groß wie m ist, dann ersetzt y ein Elter, das zufällig unter allen Eltern mit minimalem Funktionswert gewählt wird. Wie bei allen hier besprochenen evolutionären Algorithmen beschäftigen wir uns nicht mit der Wahl eines geeigneten Abbruchkriteriums und lassen das Verfahren formal wieder endlos lange laufen.

Es ist ganz offensichtlich, dass man bei der genauen Festlegung von Algorithmus 9.1 noch erheblich mehr Wahl- und Variationsmöglichkeiten hat als schon beim einfachen (1+1) EA. Wir werden uns hier vor allem mit der oben vorgestellten Grundform befassen, allerdings werden wir auch einige Variationen untersuchen. Um den Rahmen unserer Überlegungen formal abzustecken, geben wir jetzt zunächst noch eine allgemeinere, parametrisierte Fassung von Algorithmus 9.1 an, welche wir dann mit verschiedenen festen Einstellungen der Kontrollparameter untersuchen werden. Natürlich ist auch hier wiederum der Einsatz nicht-konstanter Parameterkontrolle denkbar. Das geht aber deutlich über den Rahmen dieser Arbeit hinaus.

Algorithmus 9.2.

0. *Wähle* $s(n) \in \mathbb{N}, p(n) \in (0; 1/2], q(n) \in [0; 1], u \in \{0, 1\}$.
1. *Für alle* $i \in \{1, 2, \dots, s(n)\}$
   Wähle $x_i \in \{0, 1\}^n$ *zufällig gleichverteilt.*
2. $r := 1$
3. *Mit Wahrscheinlichkeit* $q(n)$*:*
   $r := 0$
   Wähle $i_1, i_2 \in \{1, 2, \dots, s(n)\}$ *zufällig gleichverteilt.*
   Für alle $j \in \{1, 2, \dots, n\}$
      Mit Wahrscheinlichkeit $1/2$*:*
         Setze das j*-te Bit in* y *gleich dem* j*-ten Bit in* x_{i_1}.
         Sonst setze das j*-te Bit in* y *gleich dem* j*-ten Bit in* x_{i_2}.
   Sonst wähle $y \in \{x_1, x_2, \dots, x_{s(n)}\}$ *zufällig gleichverteilt.*
4. *Für alle* $i \in \{1, 2, \dots, n\}$*:*
   Mit Wahrscheinlichkeit $p(n)$*:*
      Ersetze das i*-te Bit in* y *durch sein Komplement.* $r := 0$
5. *Falls* $r \cdot u = 1$*, weiter bei* 3.
6. $m := \min\{f(x_i) \mid i \in \{1, 2, \dots, n\}\}$
7. *Falls* $f(y) \geq m$
   Wähle $i \in \{j \mid f(x_j) = m\}$ *zufällig gleichverteilt.*
   Setze $x_i := y$.
8. *Weiter bei* 2.

Wir lassen also zu, die Populationsgröße $s(n)$ zu wählen, die Wahrscheinlichkeit $q(n)$, mit der Crossover zum Einsatz kommt, die Mutationswahrscheinlichkeit $p(n)$, mit der ein Bit mutiert sowie einen Parameter $u \in \{0,1\}$, der bestimmt, ob Replikationen vermieden werden ($u = 1$) oder zugelassen sind ($u = 0$). Algorithmus 9.1 ergibt sich also als Algorithmus 9.2 mit $s(n) = n$, $q(n) = 1/(n \log n)$, $p(n) = 1/n$ und $u = 1$.

Wir wollen nachweisen, dass gerade der Einsatz von Crossover die Effizienz evolutionärer Algorithmen erheblich steigern kann. Dazu brauchen wir jetzt eine Zielfunktion, für die wir den konkreten Nachweis führen. Diese Zielfunktion soll eine Eigenschaft haben, die sie für mutationsbasierte evolutionäre Algorithmen schwierig macht, während Crossover diese Schwierigkeit verkleinert oder gar verschwinden lässt. Wir vergleichen Algorithmus 9.2 konkret mit dem (1+1) EA mit $p(n) = 1/n$, wollen aber zumindest anschaulich glaubhaft machen, dass mutationsbasierte Algorithmen im allgemeinen Schwierigkeiten mit dieser Funktion haben.

Ein Aspekt, der eine Zielfunktion für den (1+1) EA mit $p(n) = 1/n$ schwierig macht, ist die Notwendigkeit für „große Sprünge“. Wenn für eine Verbesserung das Überwinden eines großen Hammingabstandes nötig ist, muss man im Durchschnitt lange auf eine solche Mutation warten. Wir konzentrieren uns allein auf diesen Aspekt und definieren eine Familie von Zielfunktionen, die wir $f_{\mathrm{J},m}$ nennen (J für Jump), welche diese Eigenschaft in sehr isolierter und reiner Form in steuerbarer Ausgeprägtheit realisieren.

Definition 9.3. *Sei $m \in \{1, 2, \dots, n\}$. Die Funktion $f_{J,m}\colon \{0,1\}^n \to \mathbb{R}$ ist definiert durch*

$$f_{J,m}(x) := \begin{cases} m + \|x\|_1 & \text{falls } \|x\|_1 \leq n - m \text{ oder } \|x\|_1 = n, \\ n - \|x\|_1 & \text{sonst} \end{cases}$$

für alle $x \in \{0,1\}^n$.

Die Schwierigkeit der Funktion $f_{\mathrm{J},m}$ für den (1+1) EA hängt sehr stark von der Wahl des Parameters m ab. Für $m = 1$ ergibt sich einfach die Funktion $f_1 + 1$ (Abbildung 6), die sehr einfach zu optimieren ist und vom (1+1) EA in erwarteter Zeit $\Theta(n \log n)$ optimiert wird.

Für alle anderen Werte von m ist $f_{\mathrm{J},m}$ weder linear noch unimodal. Das globale Optimum ist für jeden Wert von m immer genau der Einsstring $\mathbf{1}$, lokal optimal sind stets alle Strings mit genau $n-m$ Einsen. Von diesen ist dann eine direkte Mutation zu $\mathbf{1}$ nötig, auf die mit Mutationswahrscheinlichkeit $1/n$ im Durchschnitt $n^m(1-1/n)^{m-n}$ Generationen gewartet werden muss. Schon für kleine Werte von m (etwa für $m = 5$ wie in Abbildung 7) ist das deutlich größer als $n \log n$. Schon für $m = 2$ ist die durchschnittliche Wartezeit für einen solchen „Sprung“ um den Faktor $n/\log n$ größer als $n \log n$.

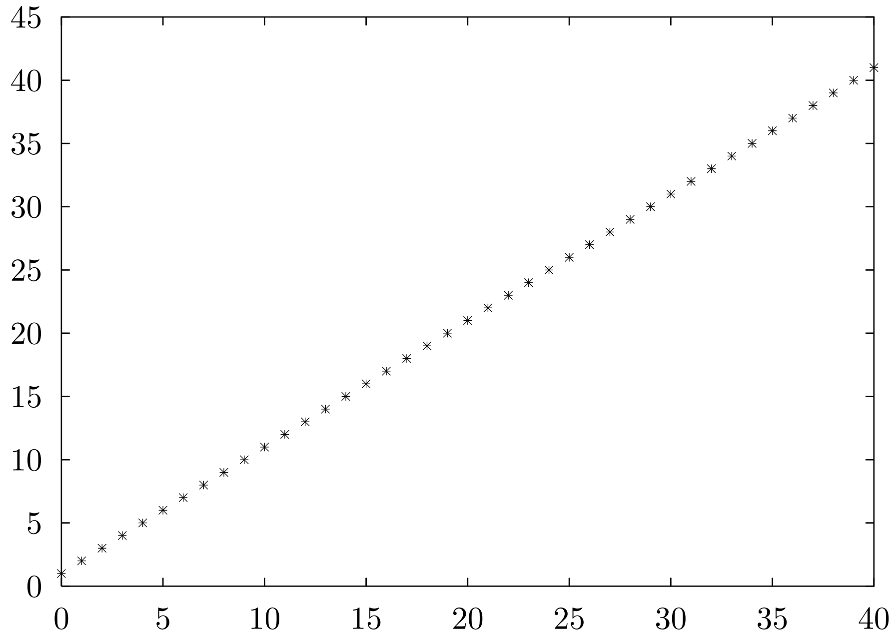

Abbildung 6: Die Funktion $f_{\mathrm{J},1}\colon \{0,1\}^{40} \to \mathbb{R}$.

Wählt man m groß (Abbildung 8), so wird die Funktion $f_{\mathrm{J},m}$ irreführend, wie wir das in Kapitel 6 kennengelernt haben. Für fast alle Strings ist es günstig, d. h. führt es zu einem größeren Funktionswert, die Anzahl der Einsen zu senken. Das globale Optimum bleibt aber wie besprochen der Einsstring $\mathbf{1}$.

Für $m = n$ schließlich erreicht diese Entwicklung ihren Höhepunkt (Abbildung 9). Die Funktion entspricht dann im wesentlichen der Funktion f_{T}, wobei lediglich die Rolle von Nullen und Einsen vertauscht ist und die absoluten Funktionswerte leicht verschoben sind. Wir zeigen nun zunächst, dass die durchschnittliche Laufzeit des (1+1) EA tatsächlich stark und direkt von der Wahl des Parameters m abhängt.

Satz 9.4. *Die erwartete Laufzeit des* $(1+1)$ *EA mit* $p(n) = 1/n$ *auf der Funktion* $f_{J,m}\colon \{0,1\}^n \to \mathbb{R}$ *beträgt* $\Theta\left(n^m + n \log n\right)$.

Beweis. Für $m = 1$ haben wir wie erwähnt $f_{\mathrm{J},1} = f_1 + 1$. Die Aussage folgt dann direkt aus Satz 3.4. Sei jetzt also $m > 1$. Die obere Schranke ist leicht zu zeigen, wir wählen die triviale Partitionierung

$$Q_i := \{x \in \{0,1\}^n \mid f_{\mathrm{J},m}(x) = i\}$$

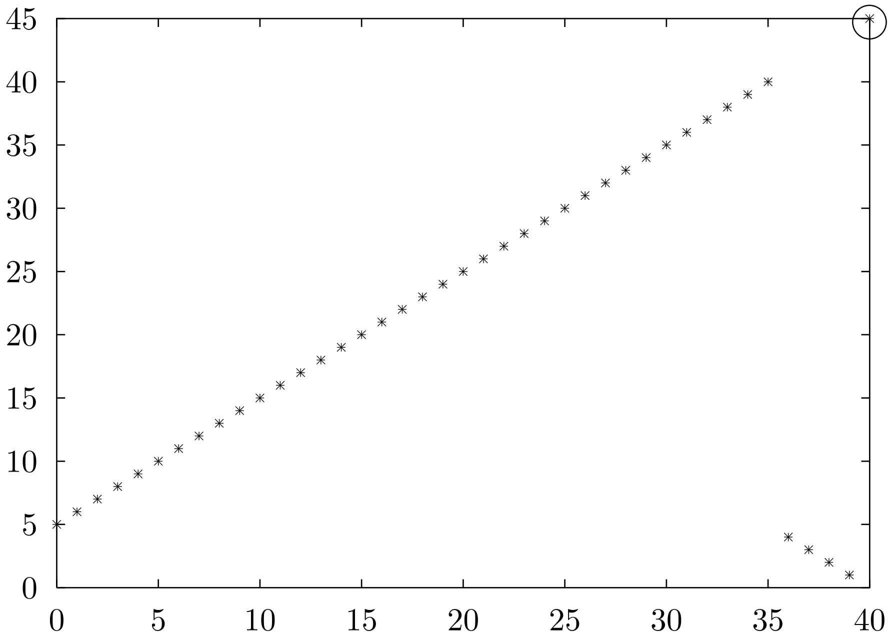

Abbildung 7: Die Funktion $f_{\mathrm{J},5}\colon \{0,1\}^{40} \to \mathbb{R}$.

mit $0 \le i \le n+m$ und betrachten im folgenden nur noch solche Werte von i, für die $Q_i \neq \emptyset$ gilt. Für alle diese $i \in \{0,1,\dots,n+m\} \setminus \{n, n+m\}$ haben wir, dass es jeweils genügt, die Anzahl von Bits mit Wert Eins in x um genau 1 zu erhöhen oder zu senken, um zu einem Kind y mit echt größerem Funktionswert zu kommen. Damit gilt also im Durchschnitt nach $O\left(n^2\right)$ Generationen $f_{\mathrm{J},m}(x) = n$ oder $f_{\mathrm{J},m}(x) = n+m$. Im zweiten Fall ist das globale Maximum erreicht, sei also $f_{\mathrm{J},m}(x) = n$. Dann gilt $\|x\|_1 = n-m$ und x ist lokal optimal. Der Hammingabstand vom globalen Maximum $\mathbf{1}$ beträgt genau m. Folglich wird in einer Generation das globale Maximum mit Wahrscheinlichkeit $(1/n)^m(1-1/n)^{n-m} \ge e^{-1}(1/n)^n$ erreicht. Die erwartete Laufzeit ist also insgesamt durch $O\left(n^2+n^m\right) = O\left(n^m\right)$ nach oben beschränkt.

Für die untere Schranke partitionieren wir den Suchraum etwas anders. Wir benutzen die Partitionierung in die drei Klassen

$$\begin{aligned}
P_1 &:= \{x \in \{0,1\}^n \mid n-m < \|x\|_1 < n\} \\
P_3 &:= \{\mathbf{1}\} \\
P_2 &:= \{0,1\}^n \setminus (P_1 \cup P_3)
\end{aligned}$$

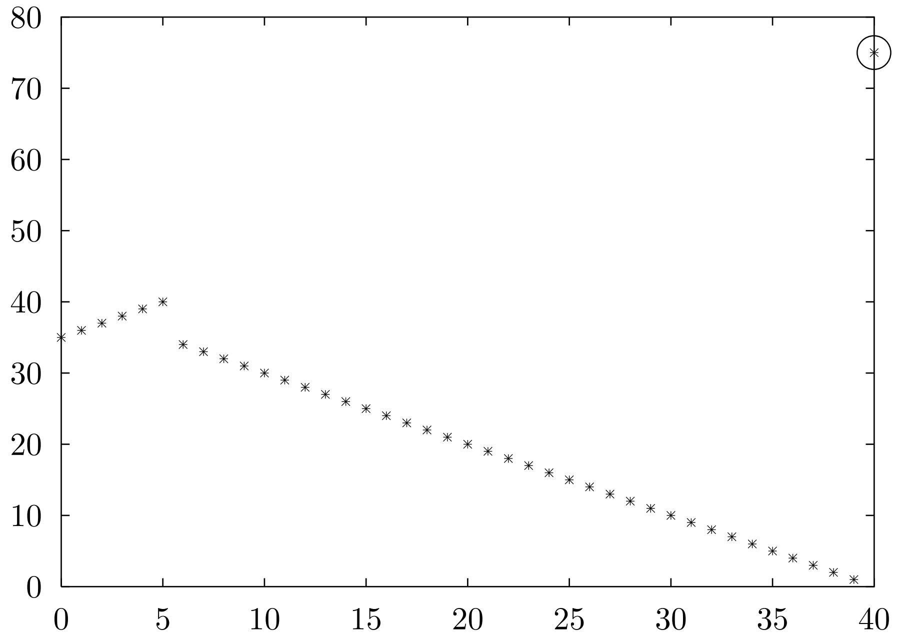

Abbildung 8: Die Funktion $f_{\mathrm{J},35}\colon \{0,1\}^{40} \to \mathbb{R}$.

und bemerken, dass

$$P_1 <_{f_{\mathrm{J},m}} P_2 <_{f_{\mathrm{J},m}} P_3$$

gilt. Wir sehen direkt, dass mit Wahrscheinlichkeit $1 - e^{-\Omega(n)}$ der aktuelle String x direkt nach der Initialisierung zu $P_1 \cup P_2$ gehört und wenigstens drei Bits mit Wert Null enthält. Analog zum Beweis von Satz 6.3 sehen wir, dass mit Wahrscheinlichkeit $1 - O(1/n)$ ein Punkt in P_2 erreicht wird. Alle Strings in P_2 haben mindestens Hammingabstand m vom globalen Maximum $\mathbf{1}$. Folglich ist die erwartete Anzahl Generationen, bis von P_2 aus das globale Maximum gefunden wird, durch $\Omega(n^m)$ nach unten beschränkt und wir haben insgesamt $\Omega(n^m)$ als untere Schranke für die erwartete Laufzeit. Zusammen mit den Ergebnissen für den Fall $m = 1$ haben wir insgesamt $\Theta(n^m + n \log n)$ als erwartete Laufzeit. $\square$

Satz 9.4 dient uns hier in erster Linie als Hilfsresultat, ist aber auch für sich alleine gesehen von Interesse. Man kann die Aussage als Hierarchieresultat für den (1+1) EA interpretieren, was in gewisser Weise eine interessante Antwortvariante auf das Klassifikationsproblem ist, mit dem wir uns ja in Kapitel 2 ausführlich beschäftigt haben. Wir formulieren ein solches Hierarchieresultat explizit als Folgerung aus Satz 9.4.

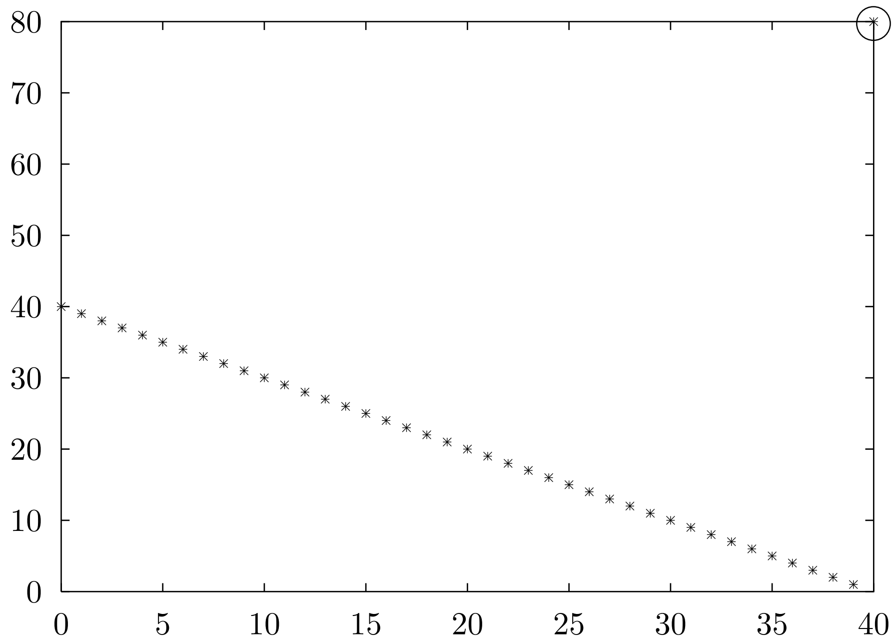

Abbildung 9: Die Funktion $f_{\mathrm{J},40}\colon \{0,1\}^{40} \to \mathbb{R}$.

Folgerung 9.5. *Sei $n \in \mathbb{N}$. Für $i \in \{0,1,\dots,n\}$ definieren wir die Menge F_i als Menge aller der Funktionen $f\colon \{0,1\}^n \to \mathbb{R}$, so dass der (1+1) EA mit $p(n) = 1/n$ die Funktion f im Durchschnitt in $O(n^i)$ Generationen optimiert. Es gilt*

$$F_0 \subsetneq F_1 \subsetneq F_2 \subsetneq \cdots \subsetneq F_{n-1} \subsetneq F_n.$$

Bevor wir uns wieder mit der Analyse von Algorithmus 9.2 beschäftigen, wollen wir noch klären, warum andere Mutationswahrscheinlichkeiten oder der Einsatz von Populationen bei der Optimierung von $f_{\mathrm{J},m}$ nicht hilfreich sind, warum also die erwartete Laufzeit des (1+1) EA mit $p(n) = 1/n$ typisch ist für mutationsbasierte evolutionäre Algorithmen überhaupt.

Nach zufälliger Initialisierung startet ein evolutionärer Algorithmus mit großer Wahrscheinlichkeit mit großem Hammingabstand zum globalen Maximum **1**. Dann findet er an jeder Stelle des Suchraums deutliche lokale Hinweise auf das lokale Maximum für Strings mit genau $n - m$ Einsen, wie wir uns schon überlegt haben. Er wird also typischerweise mit großer Wahrscheinlichkeit einen String mit Hammingabstand m vom globalen Maximum erreichen. Alle anderen Strings sind dann schlechter, werden also mit großer

Wahrscheinlichkeit nicht als „Zwischenschritte“ auf dem Weg zum eindeutigen globalen Maximum benutzt. Weil **1** der einzige global maximale String ist, dauert es auch mit anderen Mutationswahrscheinlichkeiten als $1/n$ lange, bis genau dieser String gefunden wird.

Nach diesem Exkurs zum (1+1) EA wollen wir uns jetzt der Analyse von Algorithmus 9.2 widmen. Wir konzentrieren uns auf die Funktion $f_{\mathrm{J},m}$ für kleine Werte von m. Konkret betrachten wir die Situation $m = O(\log n)$ und widmen zunächst dem Sonderfall, dass $m = O(1)$ gilt, unsere besondere Aufmerksamkeit.

Wir formulieren jetzt unser erstes Ergebnis für Algorithmus 9.2, wobei wir uns bei der Einstellung der Parameter auf die als Algorithmus 9.1 definierte Grundform konzentrieren. Wir werden unsere Beweismethoden an dieser Aussage entfalten. Wir kommen später zu Varianten und Verallgemeinerungen, werden aber die wesentliche Beweisarbeit an diesem Spezialfall leisten.

Satz 9.6. *Sei $m \in \mathbb{N}$ eine Konstante. Für jede Konstante $k \in \mathbb{N}$ gilt, dass die Wahrscheinlichkeit, dass Algorithmus 9.2 mit $s(n) = n$, $p(n) = 1/n$, $q(n) = 1/(n \log n)$ und $u = 1$ die Funktion $f_{J,m}\colon \{0,1\}^n \to \mathbb{R}$ in $O(n^2 \log n)$ Generationen optimiert, durch $1 - O\left(n^{-k}\right)$ nach unten beschränkt ist. Die Wahrscheinlichkeit, dass Algorithmus 9.2 mit dieser Parametrisierung die Funktion $f_{J,m}$ in $O(n^3)$ Generationen optimiert, ist durch $1 - e^{-\Omega(n)}$ nach unten beschränkt.*

Beweis. Wir teilen einen Lauf von Algorithmus 9.2 mit der angegebenen Parametrisierung in vier verschiedene Phasen auf und zeigen dass mit großer Wahrscheinlichkeit am Ende jeder Phase bestimmte Bedingungen erfüllt sind.

Die erste Phase umfasst nur die Initialisierung. Wir behaupten, dass mit Wahrscheinlichkeit $1 - e^{-\Omega(n)}$ die gesamte Population nur Strings mit höchstens $n - m$ Bits mit Wert Eins enthält. Weil m eine Konstante ist, ist die Wahrscheinlichkeit, dass ein bestimmtes Individuum höchstens $n - m$ Einsen enthält durch $1 - e^{-\Omega(n)}$ nach unten beschränkt. Weil die Population genau n Individuen enthält, gibt es in der Population mit Wahrscheinlichkeit höchstens $n \cdot e^{-\Omega(n)} = e^{-\Omega(n)}$ ein Individuum mit mehr als $n - m$ Bits mit Wert Eins.

Wir konzentrieren uns zunächst auf die erste Teilaussage in Satz 9.6. Dazu definieren wir, dass die zweite Phase Länge $c_1 n^2 \ln n$ hat. Dabei ist c_1 eine positive Konstante, deren Größe in Abhängigkeit von m und k passend gewählt wird. Wir nennen Phase 2 erfolgreich, wenn am Ende dieser Phase mindestens ein Individuum in der Population $\mathbf{1}$ ist, oder wenn alle Individuen genau $n-m$ Bits mit Wert Eins enthalten. Die Anzahl der Einsbits in der Population kann insgesamt nicht abnehmen. Wir ignorieren darum Generationen, in denen Crossover vorkommt oder mehr als ein Bit mutiert, ohne die Erfolgswahrscheinlichkeit der zweiten Phase dadurch zu vergrößern. Die Wahrscheinlichkeit, dass in einer Generation nur genau ein Bit mutiert, beträgt

$$(1-q(n))\binom{n}{1}p(n)\,(1-p(n))^{n-1} = \left(1-\frac{1}{n\log n}\right)\left(1-\frac{1}{n}\right)^{n-1} \geq \frac{1}{2e},$$

also haben wir in $c_1 n^2 \ln n$ Generationen mit Wahrscheinlichkeit $1-e^{-\Omega(n)}$ mindestens $(c_1/(4e))n^2 \ln n$ solche Generationen. Wir sind jetzt in einer Situation ähnlich dem Sammelkartenproblem (Coupon Collector's Problem, siehe Kapitel 4). Wir starten mit N (leeren) Eimern und wissen, dass die Wahrscheinlichkeit, nach $(\beta+1)N\ln N$ geworfenen Bällen noch einen ganz leeren Eimer zu finden, durch $N^{-\beta}$ nach oben beschränkt ist. Wir interpretieren die n^2 Bitpositionen der Population als $n^2 = N$ (leere) Eimer. Nach

$$(k+2)n^2\ln n = \left(\frac{k}{2}+1\right)N\ln N$$

solchen Generationen ist also jedes Bit in der gesamten Population mit Wahrscheinlichkeit mindestens $1-N^{-k/2} = 1-n^{-k}$ mindestens einmal von einer solchen Mutation eines einzelnen Bits betroffen. Die Wahrscheinlichkeit, in $4e(k+2)n^2\ln n$ Generationen insgesamt nicht mindestens $(k+2)n^2\ln n$ solche Generationen zu haben, ist exponentiell klein, wie wir uns schon überlegt haben. Mit $c_1 = 4e(k+2)$ haben wir also, dass die erste Phase mit Wahrscheinlichkeit $1-O\left(n^{-k}\right)$ erfolgreich ist.

Für die zweite Teilaussage verlängern wir die Länge der ersten Phase auf $8e\left(n^3+n^2\ln n\right)$. Dann haben wir mit Wahrscheinlichkeit $1-e^{-\Omega(n)}$ mindestens

$$2\left(n^3+n^2\ln n\right) = 2\left(\frac{n}{\ln n}+1\right)n^2\ln n = \left(\frac{n}{\ln n}+1\right)N\ln N$$

Generationen, in denen es genau eine Mutation eines einzelnen Bits gibt. Die Wahrscheinlichkeit, dass danach nicht alle Individuen der Population genau $n-m$ Bits mit Wert Eins enthalten, ist durch

$$n^{-n/\ln n} = e^{-n}$$

nach unten beschränkt. Also wird die zweite Phase mit Wahrscheinlichkeit $1-e^{-\Omega(n)}$ erfolgreich beendet, wenn sie Länge $8e\left(n^3+n^2\ln n\right) = O\left(n^3\right)$ hat.

Die dritte Phase hat Länge $c_2 n^2 \log n$ für eine positive Konstante c_2, die wir in Abhängigkeit von m und k passend wählen. Sie beginnt mit einer Population, die nur Individuen mit genau $n-m$ Bits mit Wert Eins enthält. Wir bezeichnen sie als erfolgreich, wenn am Ende dieser dritten Phase das globale Maximum $\mathbf{1}$ in der Population vorkommt oder für jede der n Positionen gilt, dass es höchstens $n/(4m)$ Individuen in der aktuellen Population gibt, die ein Bit mit Wert Null an der betrachteten Position haben. Die Idee dabei ist die folgende. Nehmen wir an, dass wir eine Population so generieren, dass wir jedes Individuum unter allen Strings mit genau $n-m$ Einsbits zufällig gleichverteilt auswählen. Dann sind die wenigen Bits mit Wert Null in der Population gut verteilt, bei Populationsgröße n erwarten wir nur m Bits mit Wert Null an jeder Position. Wählt man aus dieser Population zwei Individuen aus, so gibt es mit Wahrscheinlichkeit

$$\frac{\binom{n-m}{n}}{\binom{n}{m}} = \frac{(n-m)! \cdot (n-m)!}{(n-2m)! \cdot n!} = \prod_{i=0}^{m-1} \frac{n-m-i}{n-i} \geq \left(1 - \frac{m}{n}\right)^m \approx e^{-m^2/n}$$

keine Position, an der beide Individuen ein Bit mit Wert Null haben. Dann kann uniformes Crossover mit Wahrscheinlichkeit 2^{-2m} den Einsstring $\mathbf{1}$ erzeugen, was für konstantes m eine konstante Wahrscheinlichkeit ist. Also ist das globale Maximum von $f_{\mathrm{J},m}$ in dieser Situation leicht zu finden. Es gibt aber leider keine offensichtlichen, zwingenden Gründe dafür, dass in der Population am Ende der zweiten Phase die Bits mit Wert Null so schön gleichmäßig verteilt sind. Bei Experimenten beobachtet man in der Tat häufig, dass die mn Nullbits der Population sich an relativ wenigen Positionen konzentrieren. Dann ist die Wahrscheinlichkeit groß, zwei Individuen als Eltern beim Crossover zu verwenden, die beide an einer Position ein Nullbit aufweisen, so dass $\mathbf{1}$ mit Crossover nicht erzeugbar ist. Wir zeigen darum, dass in der dritten Phase unabhängig von der anfänglichen Verteilung der Nullen am Ende eine nicht zu starke Konzentration der Nullbits vorliegt mit großer Wahrscheinlichkeit. Konkret zeigen wir, dass noch $c_2 n^2 \log n$ Generationen mit Wahrscheinlichkeit $1 - e^{-\Omega(n)}$ an jeder Position höchstens $n/(4m)$ Nullbits stehen. Wir werden uns dabei auf die Effekte beschränken, die allein durch Mutationen verursacht werden. Wir nehmen zu unseren Ungunsten an, dass Crossover eine Konzentration von Nullbits begünstigt. Durch diese unrealistische Abschätzung vermeiden wir, die Auswirkungen von Crossover detailliert analysieren zu müssen.

Wir betrachten zunächst nur eine einzige Position, etwa die erste. Wir leiten eine obere Schranke für die Wahrscheinlichkeit her, dass am Ende der dritten Phase an dieser Position mehr als $n/(4m)$ Nullbits vorhanden sind. Wir erhalten dann eine Abschätzung für die Wahrscheinlichkeit, dass die dritte Phase insgesamt nicht erfolgreich beendet wird, indem wir diese obere Schranke mit n, der Anzahl Positionen, multiplizieren.

Wir betrachten eine einzelne Generation, sei z die Anzahl von Nullbits an der ersten Position am Anfang dieser Generation. Es ist natürlich $z \leq s(n) = n$. Die Anzahl Nullen an dieser Position kann in einer Generation um 1 größer werden, um 1 kleiner werden oder unverändert bleiben. Wir bezeichnen die Wahrscheinlichkeit, z um 1 zu vergrößern mit $p^+(z)$ und die Wahrscheinlichkeit, z um 1 zu verkleinern mit $p^-(z)$. Wir ignorieren (zu unseren Ungunsten) die Möglichkeit, $\mathbf{1}$ zu erzeugen. Dann ist klar, dass sowohl am Anfang als auch am Ende der betrachteten Generation, alle Individuen der aktuellen Population genau $n-m$ Einsbits enthalten. Sei A^+ bzw. A^- das Ereignis, z um 1 zu vergrößern bzw. zu verkleinern, sei also $\text{Prob}\,(A^+) = p^+(z)$ und $\text{Prob}\,(A^-) = p^-(z)$. Wir betrachten verschiedene Ereignisse, deren Wahrscheinlichkeiten sich leichter abschätzen lassen und die insgesamt eine Abschätzung von $p^+(z)$ und $p^-(z)$ erlauben.

B**:** Es wird uniformes Crossover durchgeführt. Es ist $\text{Prob}\,(B) = q(n) = 1/(n \log n)$.

C**:** Für die Ersetzung in Zeile 7 von Algorithmus 9.2 wird ein Individuum mit einem Einsbit an der ersten Position ausgewählt. Weil alle Individuen gleichen Funktionswert haben und wir zufällig gleichverteilt wählen, ist $\text{Prob}\,(C) = (n-z)/n$.

D**:** Für die Mutation in Zeile 4 wird in Zeile 3 ein Individuum mit einem Nullbit an der ersten Position ausgewählt. Weil wir zufällig gleichverteilt wählen, ist $\text{Prob}\,(D) = z/n$. Weil alle Individuen gleichen Funktionswert haben, gilt das nicht nur für diese schwache Selektion, sondern für beliebige Selektionsmechanismen, die rein funktionswertbasiert arbeiten.

E**:** Bei der Mutation in Zeile 4 mutiert das erste Bit nicht. Es ist $\text{Prob}\,(E) = 1 - p(n) = 1 - 1/n$.

F_i^+**:** Wir setzen voraus, dass an der ersten Position ein Nullbit steht und betrachten nur die restlichen $n-1$ Positionen. Bei der Mutation in Zeile 4 gibt es unter den $m-1$ Positionen, an denen jeweils ein Bit mit Wert Null steht, genau i Bits, die mutieren. Außerdem gibt es genau $i-1$ Bits unter den $n-m$ Bits mit Wert Eins, die mutieren. Es ist $\text{Prob}\left(F_i^+\right) = \binom{m-1}{i}\binom{n-m}{i}(1/n)^{2i}(1-1/n)^{n-2i}$. Weil Replikationen nicht vorkommen, brauchen wir den Fall $i = 0$ nicht zu betrachten.

G_i^+: Wir setzen voraus, dass an der ersten Position ein Einsbit steht und betrachten nur die restlichen $n-1$ Positionen. Bei der Mutation in Zeile 4 gibt es unter den m Positionen, an denen jeweils ein Bit mit Wert Null steht, genau i Bits, die mutieren. Außerdem gibt es genau $i-1$ Bits unter den $n-m-1$ Bits mit Wert Eins, die mutieren. Es ist $\text{Prob}\left(G_i^+\right) = \binom{m}{i}\binom{n-m-1}{i-1}(1/n)^{2i-1}(1-1/n)^{n-2i}$.

Offensichtlich ist mit diesen Festlegungen

$$A^+ \subseteq B \cup \left(\overline{B} \cap C \cap \left(\left(D \cap E \cap \bigcup_{1 \leq i < m} F_i^+\right) \cup \left(\overline{D} \cap \overline{E} \cap \bigcup_{1 \leq i \leq m} G_i^+\right)\right)\right)$$

und damit

$$\begin{aligned}
p^+(z) \quad \leq \quad & \frac{1}{n \log n} + \left(1 - \frac{1}{n \log n}\right) \frac{n-z}{n} \\
& \left(\frac{z}{n} \sum_{1 \leq i < m} \binom{m-1}{i} \binom{n-m}{i} \left(\frac{1}{n}\right)^{2i} \left(1 - \frac{1}{n}\right)^{n-2i}\right. \\
& \left. + \frac{n-z}{\sum}_{1 \leq i \leq m} \binom{m}{i} \binom{n-m-1}{i-1} \left(\frac{1}{n}\right)^{2i} \left(1 - \frac{1}{n}\right)^{n-2i}\right) \\
\leq \quad & \frac{1}{n \log n} + \left(1 - \frac{1}{n \log n}\right) \frac{n-z}{n} \\
& \left(\frac{z}{n}(m-1)(n-m)\frac{1}{n^2}\left(1 - \frac{1}{n}\right)^{n-2} + O\left(\frac{m^2}{n^2}\right)\right. \\
& \left. \frac{n-z}{n}\left(m\frac{1}{n^2}\left(1 - \frac{1}{n}\right)^{n-2} + O\left(\frac{m^2}{n^3}\right)\right)\right)
\end{aligned}$$

eine obere Schranke für $p^+(z)$. Um eine untere Schranke für $p^-(z)$ zu erhalten, gehen wir analog vor und betrachten zusätzlich die Ereignisse

F_i^-: Wir setzen voraus, dass an der ersten Position ein Nullbit steht und betrachten nur die restlichen $n-1$ Positionen. Bei der Mutation in Zeile 4 gibt es unter den $(m-1)$ Positionen, an denen jeweils ein Bit mit Wert Null steht, genau $i-1$ Bits die mutieren. Außerdem gibt es genau i Bits unter den $n-m$ Bits mit Wert Eins, die mutieren. Es ist $\text{Prob}\left(F_i^-\right) = \binom{m-1}{i-1}\binom{n-m}{i}(1/n)^{2i-1}(1-1/n)^{n-2i}$.

G_i^-: Wir setzen voraus, dass an der ersten Position ein Einsbit steht und betrachten nur die restlichen $n-1$ Positionen. Bei der Mutation in Zeile 4 gibt es unter den m Positionen, an denen jeweils ein Bit mit Wert Null steht, genau i Bits, die mutieren. Außerdem gibt es genau i Bits unter den $n-m-1$ Bits mit Wert Eins, die mutieren. Den Fall $i=0$ brauchen wir nicht zu betrachten, da er eine Replikation bedeutet. Es ist $\text{Prob}\left(G_i^-\right) = \binom{m}{i}\binom{n-m-1}{i}(1/n)^{2i}(1-1/n)^{n-2i-1}$.

Damit haben wir

$$A^- \supseteq \overline{B} \cap \overline{C} \cap \left(\left(D \cap \overline{E} \cap \bigcup_{1\leq i\leq m} F_i^-\right) \cup \left(\overline{D} \cap E \cap \bigcup_{1\leq i\leq m} G_i^-\right)\right)$$

und folglich

$$\begin{aligned}
p^-(z) \quad &\geq \quad \left(1-\frac{1}{n\log n}\right)\frac{z}{n} \\
&\qquad \left(\frac{z}{n}\sum_{1\leq i\leq m}\binom{m-1}{i-1}\binom{n-m}{i}\left(\frac{1}{n}\right)^{2i}\left(1-\frac{1}{n}\right)^{n-2i}\right. \\
&\qquad \left. +\frac{n-z}{n}\sum_{1\leq i\leq m}\binom{m}{i}\binom{n-m-1}{i}\left(\frac{1}{n}\right)^{2i}\left(1-\frac{1}{n}\right)^{n-2i}\right) \\
&\geq \quad \left(1-\frac{1}{n\log n}\right)\frac{z}{n} \\
&\qquad \left(\frac{z}{n}(n-m)\frac{1}{n^2}\left(1-\frac{1}{n}\right)^{n-2}\right. \\
&\qquad \left. +\frac{n-z}{n}m(n-m-1)\frac{1}{n^2}\left(1-\frac{1}{n}\right)^{n-2}\right)
\end{aligned}$$

als untere Schranke für $p^-(z)$. Wir setzen voraus, dass m eine Konstante ist. Nehmen wir zunächst an, dass $z \geq n/(8m)$ gilt. Dann haben wir

$$p^-(z) \geq p^-(z) - p^+(z) = \Omega\left(\frac{1}{n}\right)$$

und außerdem

$$p^-(z) = O\left(\frac{1}{n}\right),$$

also gilt auch

$$p^-(z) + p^+(z) = O\left(\frac{1}{n}\right).$$

Wir nennen eine Generation essenziell, wenn sich die Anzahl der Nullbits an der ersten Position in dieser Generation ändert. Die dritte Phase hat Länge $c_2 n^2 \log n$, also haben wir mit Wahrscheinlichkeit $1 - e^{-\Omega(n)}$ mindestens $c_2' n \log n$ essenzielle Generationen, wobei c_2' eine positive Konstante ist. Wir bezeichnen die bedingten Wahrscheinlichkeiten, z in einer essenziellen Generation zu vergrößern bzw. zu verkleinern mit $q^+(z)$ bzw. $q^-(z)$. Es ist

$$q^-(z) = \frac{p^-(z)}{p^-(z) + p^+(z)}, q^+(z) = \frac{p^+(z)}{p^-(z) + p^+(z)}.$$

Aus den Abschätzungen für $p^-(z)$ und $p^+(z)$ folgt

$$q^-(z) - q^+(z) = \frac{p^-(z) - p^+(z)}{p^-(z) + p^+(z)} = \Omega(1),$$

wir senken also mit Wahrscheinlichkeit $1 - e^{-\Omega(n)}$ in $c_2' n \log n$ essenziellen Generationen um wenigstens $c_2'' n$ für eine positive Konstante c_2''. Wir wählen c_2 so groß, dass wir c_2' so wählen können, dass $c_2'' \geq 1$ gilt. Wir werden also die Anzahl Nullbits an der ersten Position mit Wahrscheinlichkeit exponentiell nahe bei 1 weit genug senken, allerdings gelten diese Überlegungen nur unter der Bedingung, dass $z \geq n/(8m)$ gilt. Wir betrachten den ersten Zeitpunkt in der dritten Phase, zu dem $z < n/(8m)$ gilt. Wenn die Anzahl der Generationen der dritten Phase, die noch auf diesen Zeitpunkt folgen, durch $n/(8m)$ nach oben beschränkt ist, ist am Ende der dritten Phase z sicher durch $n/(8m) + n/(8m) = n/(4m)$ nach oben beschränkt. Andernfalls teilen wir die restlichen Generationen in Subphasen der Länge $n/(8m)$ ein. Wir betrachten eine solche Subphase. Wird zu irgendeinem Zeitpunkt in dieser Subphase z kleiner als $n/(8m)$, so gilt am Ende der Subphase $z < n/(8m) + n/(8m) = n/(4m)$. Andernfalls können wir annehmen, dass während der gesamten betrachteten Subphase $z \geq n/(8m)$ gilt. Dann können wir analog zur Beweisführung für die gesamte Phase zeigen, dass am Ende der Subphase mit Wahrscheinlichkeit $1 - e^{-\Omega(n)}$ die Anzahl der Einsen an erster Position z nach oben durch $n/(4m)$ beschränkt ist. Das gilt auf gleiche Weise für alle Subphasen, von denen es insgesamt nur $O(mn \log n)$ viele gibt. Folglich haben wir insgesamt gezeigt, dass die dritte Phase, die Länge $c_2 n^2 \log n$ hat, mit Wahrscheinlichkeit $1 - n \cdot e^{-\Omega(n)} = 1 - e^{-\Omega(n)}$ erfolgreich beendet wird.

Die vierte Phase hat Länge $c_3 n^2 \log n$, wobei c_3 eine positive Konstante ist, die wir in Abhängigkeit von m und k passend wählen. Wir müssen zunächst genau wie in der dritten Phase die Konzentration der Nullbits kontrollieren. Wir starten mit höchstens $n/(4m)$ Nullbits an jeder Position und wollen zeigen, dass mit Wahrscheinlichkeit $1 - e^{-\Omega(n)}$ zu jedem Zeitpunkt der vierten Phase an jeder Position höchstens $n/(2m)$ Nullbits vorkommen. Wir gehen wie bei der Analyse der dritten Phase vor und betrachten Subphasen der Länge $n/(4m)$. Subphasen, bei denen die Anzahl Nullbits an der ersten Position durch $n/(4m)$ nach oben beschränkt ist, können keinen Misserfolg verursachen. Bei den anderen Subphasen können wir analog zur dritten Phase $z \geq n/(4m)$ voraussetzen und dann wieder wie in der dritten Phase Chernoff-Schranken anwenden. Damit haben wir also gezeigt, dass mit Wahrscheinlichkeit $1 - e^{-\Omega(n)}$ die Anzahl Nullbits an jeder Position während der vierten Phase durch $n/(2m)$ nach oben beschränkt ist. Wir betrachten jetzt eine Position und wollen die Wahrscheinlichkeit, das globale Maximum zu finden, nach unten abschätzen. Für die Erzeugung des globalen Maximums ist es ausreichend, Crossover auszuführen (mit Wahrscheinlichkeit $q(n) = 1/(n \log n)$), zwei Strings als Eltern zu wählen, die an keiner Position beide ein Nullbit haben, durch uniformes Crossover $\mathbf{1}$ zu erzeugen, indem an den $2m$ Positionen, an denen einer der beiden Eltern ein Nullbit hat, das Einsbit des anderen Elter gewählt wird (mit Wahrscheinlichkeit 2^{-2m}) und schließlich das erzeugte Kind $\mathbf{1}$ durch Mutation nicht zu verändern (mit Wahrscheinlichkeit $(1 - p(n))^n = (1 - 1/n)^n$). Wir betrachten die Selektion der Eltern zum Crossover. Sei das erste Individuum gewählt, es enthält genau m Bits mit Wert Null. Nun wird zufällig gleichverteilt aus der Population das zweite Individuum gewählt. Die m Positionen, an denen das erste Elter ein Nullbit hat, sind für uns kritisch in dem Sinne, dass das zweite Elter an dieser Stelle kein Nullbit haben soll. An jeder Position ist die Anzahl Nullbits durch $n/(2m)$ nach oben beschränkt, folglich gibt es in der Population höchstens $n/(2m)$ Individuen, die an einer bestimmten von diesen m Positionen ebenfalls ein Nullbit haben. Folglich ist die Anzahl der Individuen in der gesamten Population, die auf diese Weise mit dem ersten gewählten Elter kollidieren, nach oben durch

$$m \cdot \frac{n}{2m} = \frac{n}{2}$$

beschränkt. Weil zufällig gleichverteilt aus allen n Individuen der Population ausgewählt wird, haben wir mit Wahrscheinlichkeit mindestens $1/2$ ein solches zweites Elter, das mit dem ersten an keiner Position ein Nullbit gemeinsam hat. Weil alle Individuen gleichen Funktionswert haben, gilt das für jeden Selektionsmechanismus, der rein funktionswertbasiert vorgeht. Wir erzeugen also in einer Generation mit Wahrscheinlichkeit

$$\frac{1}{n \log n} \cdot \frac{1}{2} \cdot \left(\frac{1}{2}\right)^{2m} \cdot \left(1 - \frac{1}{n}\right)^{n} = \Omega\left(\frac{1}{2^{2m} n \log n}\right) = \Omega\left(\frac{1}{n \log n}\right)$$

das globale Maximum. Folglich erzeugen wir in $c_3 n^2 \log n$ Generationen mit Wahrscheinlichkeit $1 - e^{-\Omega(n)}$ das globale Maximum.

Wir haben also insgesamt gezeigt, dass wir in $O(n^2 \log n)$ Generationen mit Wahrscheinlichkeit $1 - O\left(n^{-k}\right)$ das globale Maximum erreichen. Um die Erfolgswahrscheinlichkeit auf $1 - e^{-\Omega(n)}$ zu erhöhen, reicht es aus, alleine die Länge der zweiten Phase auf $8e\,(n^3 + n^2 \ln n)$ zu erhöhen, so dass wir also mit Wahrscheinlichkeit $1 - e^{-\Omega(n)}$ in $O(n^3)$ Generationen das globale Maximum erreichen. □

Es ist nicht schwer, auch eine Aussage über die erwartete Laufzeit von Algorithmus 9.1 auf der Funktion $f_{\mathrm{J},m}$ für konstante Werte von m zu machen. Es genügt eine einfache Erweiterung des Beweises von Satz 9.6.

Satz 9.7. *Sei $m \in \mathbb{N}$ eine Konstante. Algorithmus 9.2 mit $s(n) = n$, $p(n) = 1/n$, $q(n) = 1/(n \log n)$ und $u = 1$ optimiert die Funktion $f_{J,m}\colon \{0,1\}^n \to \mathbb{R}$ in erwarteter Zeit $O(n^2 \log n)$.*

Beweis. Um von einer Aussage über eine Anzahl von Generationen, die mit großer Wahrscheinlichkeit zum Erreichen eines globalen Optimums ausreicht, zu einer Aussage über die erwartete Laufzeit zu kommen, genügt es, die erwartete Anzahl benötigter Wiederholungen (im Falle von Fehlschlägen) abzuschätzen. Das setzt aber voraus, dass die Analyse der ursprünglichen Aussage von der initialen Population unabhängig ist. Das ist bei Satz 9.6 nicht der Fall. Um das zu ändern, lassen wir die Analyse der ersten Phase, also der zufälligen Initialisierung weg. Wir müssen die Analyse von Phase 2 so anpassen, dass sie für beliebige Startpopulationen gilt. Wenn die Startpopulation das globale Maximum enthält oder dieses zufällig erzeugt wird, ist nichts mehr zu zeigen. Andernfalls müssen wir uns jetzt zusätzlich um Individuen x mit $n - m < \|x\|_1 < n$ kümmern. Wir haben uns schon überlegt, dass $f_{\mathrm{J},m}$ sich in diesem Bereich des Suchraums wie $-f_1$ verhält. Wir können also die Analyse der zweiten Phase „verdoppeln“: nach einer ersten Subphase der Länge $O(n^2 \log n)$ haben alle String der Population dann mit Wahrscheinlichkeit $1 - O\left(n^{-k}\right)$ höchstens $n - m$ Bits mit Wert Eins und wir sind in der ursprünglichen Situation. Insgesamt wird also das globale Maximum weiter mit Wahrscheinlichkeit $1 - O\left(n^{-k}\right)$ erreicht. Wir wählen $k = 1$ und sehen, dass eine konstante Anzahl von Wiederholungen dieser $O(n^2 \log n)$ Generationen im erwarteten Fall ausreicht. □

Wir wollen jetzt basierend auf den Analysemethoden aus dem Beweis zu Satz 9.6 etwas allgemeinere Aussagen über Algorithmus 9.2 und die Funktion $f_{\mathrm{J},m}$ machen. Eine erste wichtige Beobachtung ist, dass sich nicht allzu viel ändert, wenn der Wert von m ein wenig mit n wächst.

Satz 9.8. *Sei $n \in \mathbb{N}, m \in \mathbb{N}$ mit $m = O(\log n)$. Sei δ eine Konstante mit $0 < \delta < 1$. Die Wahrscheinlichkeit, dass Algorithmus 9.2 mit $s(n) = n$, $p(n) = 1/n$, $q(n) = 1/\left(n \log^3 n\right)$ und $u = 1$ die Funktion $f_{J,m}\colon \{0,1\}^n \to \mathbb{R}$ in*

$$O\left(n^3 + n^2\left(\log^5 n + 2^{2m}\right)\right)$$

Generationen optimiert, ist nach unten durch $1 - e^{-\Omega\left(n^\delta\right)}$ beschränkt. Die erwartete Laufzeit von Algorithmus 9.2 mit dieser Parametrisierung ist durch

$$O\left(n \log^3 n \left(n \log^2 n + 2^{2m}\right)\right)$$

nach oben beschränkt.

Beweis. Wir beweisen zunächst die erste Teilaussage. Im Vergleich zum Beweis von Satz 9.6 ändert sich nichts wesentlich an der ersten und zweiten Phase. Für eine Erfolgswahrscheinlichkeit von $1 - e^{-\Omega(n)}$ in der zweiten Phase wählen wir die Länge der zweiten Phase mit $c_1 n^3$. Bei der Analyse der dritten Phase ändert sich an unserem Vorgehen prinzipiell nichts, weil m aber nicht mehr konstant ist, müssen wir etwas andere Abschätzungen für $p^-(z)$ und $p^+(z)$ vornehmen. Konkret ergibt sich

$$p^-(z) - p^+(z) = \Omega\left(\frac{1}{n \log^2 n}\right)$$

sowie

$$p^-(z) = O\left(\frac{\log n}{n}\right),$$

also auch

$$p^-(z) + p^+(z) = O\left(\frac{\log n}{n}\right).$$

Hier wird deutlich, warum wir die Crossoverwahrscheinlichkeit mit $q(n) = 1/\left(n \log^3 n\right)$ noch etwas kleiner gewählt haben. Für die bedingten Wahrscheinlichkeiten $q^-(z)$ und $q^+(z)$ haben wir dann

$$q^-(z) - q^+(z) = \Omega\left(\frac{1}{\log^3 n}\right),$$

wenn wir $z \geq n/(8m)$ voraussetzen. Wir wählen $c_2 n^2 \log^5 n$ als Länge der dritten Phase und haben, dass mit Wahrscheinlichkeit $1 - e^{-\Omega\left(n^\delta\right)}$ die dritte Phase erfolgreich abgeschlossen wird. In der vierten Phase gehen wir genauso vor. Die Wahrscheinlichkeit, in einer Generation das globale Maximum zu erzeugen, ist durch

$$\Omega\left(\frac{1}{n \log^3 n} \cdot 2^{-2m}\right)$$

nach unten beschränkt. Die Wahrscheinlichkeit, dass das globale Maximum in $\Theta\left(2^{2m} n \log^3 n\right)$ Generationen gefunden wird, ist also $\Omega(1)$. Die Wahrscheinlichkeit, es in $\Theta\left(2^{2m} n^2\right)$ Generationen zu finden, ist $1 - e^{-\Omega\left(n/\log^3 n\right)} = 1 - e^{-\Omega\left(n^\delta\right)}$. Folglich wird insgesamt mit Wahrscheinlichkeit $1 - e^{-\Omega\left(n^\delta\right)}$ das globale Maximum in

$$O\left(n^3 + n^2 \log^5 n + 2^{2m} n^2\right)$$

Generationen gefunden wird. Wir können die Analyse der zweiten Phase wie im Beweis zu Satz 9.7 für beliebige Startpopulationen verallgemeinern. Es genügt uns, eine konstante Erfolgswahrscheinlichkeit zu haben, weil dadurch die erwartete Anzahl von Wiederholungen durch eine Konstante nach oben beschränkt wird. Folglich reicht uns für die vierte Phase eine Länge von $\Theta\left(2^{2m} n \log^3 n\right)$ und wir haben

$$O\left(n^2 \log n + n^2 \log^5 n + 2^{2m} n \log^3 n\right) = O\left(n \log^3 n \left(n \log^2 n + 2^{2m}\right)\right)$$

als obere Schranke für die erwartete Laufzeit. □

Im Vergleich zum (1+1) EA mit $p(n) = 1/n$ ist Satz 9.8 von besonderem Interesse. Wir wissen, dass der (1+1) EA für die Optimierung der Funktion $f_{\mathrm{J},m}$ mit $m = \Theta(\log n)$ superpolynomiell viele Generationen braucht, während Algorithmus 9.2 nur polynomielle erwartete Laufzeit hat. Wir haben uns überlegt, dass $\Omega\left(n^m\right)$ eine allgemeine untere Schranke für mutationsbasierte evolutionäre Algorithmen mit Mutationswahrscheinlichkeit $1/n$ auf der Funktion $f_{\mathrm{J},m}$ ist. Insofern zeigt also vor allem Satz 9.8 besonders deutlich, dass der Einsatz von Crossover die Effizienz evolutionärer Algorithmen erheblich steigern kann.

Wodurch sich der genetische Algorithmus, wie wir ihn bisher betrachtet haben, von vielen anderen „handelsüblichen“ genetischen Algorithmen unterscheidet ist die Vermeidung von Replikationen. Typischerweise werden Replikationen zugelassen, sie können bei schwächeren Ersetzungsstrategien als der hier gewählten, die keine Verschlechterungen zulässt, auch wichtig sein, um nicht gute Individuen durch unglückliche Zufallsentscheidungen bei der Selektion wieder zu verlieren. Wir werden darum jetzt Algorithmus 9.2 mit $u = 0$, also bei Zulassung von Replikationen, betrachten. Wir beschränken dabei unsere Analyse wieder auf den Fall $m = O(1)$, eine Verallgemeinerung auf den Fall $m = O(\log n)$ ist möglich aber nicht von besonderem Interesse.

Satz 9.9. *Sei $m \in \mathbb{N}$ eine Konstante. Für jede monotone Funktion $a\colon \mathbb{N} \to \mathbb{R}^+$ gilt, dass die Wahrscheinlichkeit, dass Algorithmus 9.2 mit $s(n) = n$, $p(n) = 1/n$, $q(n) = 1/(n \log n)$ und $u = 0$ die Funktion $f_{J,m}\colon \{0,1\}^n \to \mathbb{R}$ in $O(a(n)n^3 \log n)$ Generationen optimiert, nach unten durch $1 - e^{-\Omega(a(n) \log n)}$ beschränkt ist.*

Beweis. Wir orientieren uns wiederum am Beweis von Satz 9.6 und präsentieren den Beweis hier als Modifikation davon. Für die ersten beiden Phasen ändert sich im wesentlichen nichts. Tatsächlichen haben wir in der zweiten Phase eine Wahrscheinlichkeit von $(1 - 1/(n \log n))(1 - 1/n)^n$ für eine Replikation, so dass sich hier Individuen, die bereits eine große Anzahl Bits mit Wert Eins haben, schneller in der Population ausbreiten können. Wir ignorieren diese für uns günstige Tendenz.

In der zweiten Phase ergibt sich eine wesentliche Änderung. Wir können die Ereignisse A^- und A^+ im wesentlichen unverändert modellieren, müssen aber nun die Ereignisse F_0^+ und G_0^-, die Replikationen implizieren, mitberücksichtigen. Damit bleibt zwar weiterhin

$$p^-(z) - p^+(z) = \Omega\left(\frac{1}{n}\right)$$

gültig, wir haben aber keine obere Schranke in der gleichen Größenordnung für die Wahrscheinlichkeit, eine essenzielle Generation zu haben. Das ändert wesentlich die Größenordnung der bedingten Wahrscheinlichkeiten $q^-(z)$ und $q^+(z)$ im Vergleich zum Beweis von Satz 9.6, wir haben

$$q^-(z) - q^+(z) = \Omega\left(\frac{1}{n}\right)$$

im Gegensatz zu $q^-(z) - q^+(z) = \Omega(1)$ wie beim Beweis von Satz 9.6. Es gilt hier allerdings auch weiterhin, dass eine essenzielle Generation mit Wahrscheinlichkeit $\Omega(1/n)$ vorkommt. Darum brauchen wir eine größere Anzahl essenzieller Generationen und müssen, um das zu erreichen, die Länge der dritten Phase erhöhen. Wir wählen $a(n) \cdot c_2 n^3 \log n$ als Länge der dritten Phase, wobei c_2 eine hinreichend große positive Konstante ist. Wir haben mit dieser Wahl dann in der zweiten Phase mit Wahrscheinlichkeit $1 - e^{-\Omega(n)}$ mindestens $a(n) \cdot c_2' n^2 \log n$ essenzielle Generationen, wobei c_2' eine positive Konstante ist. Wenn wir in $a(n) \cdot c_2' n^2 \log n$ essenziellen Generationen die Anzahl der Nullbits an der ersten Position insgesamt genau d Mal verringern, haben wir am Ende der zweiten Phase genau

$$z + a(n) \cdot c_2' n^2 \log n - d$$

Nullbits an der ersten Position, wenn wir voraussetzen, anfangs z Nullbits dort zu haben. Wir brauchen, dass

$$z + a(n) \cdot c_2' n^2 \log n - d \leq \frac{n}{4m}$$

ist. Wir haben

$$q^-(z) - q^+(z) = \Omega\left(\frac{1}{n}\right)$$

und natürlich $q^-(z) + q^+(z) = 1$, so dass wir

$$q^-(z) = \frac{1}{2} + \Omega\left(\frac{1}{n}\right)$$

haben. Wir wenden die Chernoff-Schranken an und erkennen, dass die Wahrscheinlichkeit, nach $a(n) \cdot c_2' n^2 \log n$ essenziellen Generationen an der ersten Position mehr als $n/(4m)$ Nullbits zu haben, nach oben durch $e^{-\Omega(a(n) \log n)}$ beschränkt ist.

Für die Kontrolle der Konzentration der Nullbits in der vierten Phase gehen wir analog zur dritten Phase vor. Für die Erzeugung des globalen Maximums spielen Replikationen keine Rolle. Um insgesamt auf die gewünschte obere Schranke für einen Fehlschlag zu kommen, müssen wir noch die Länge der zweiten Phase auf $c_1 a(n) n^2 \log n$ ändern, wobei c_1 eine passend gewählte positive Konstante ist. $\square$

Für $a(n) = n/\log n$ haben wir also, dass $O(n^4)$ Generationen mit Wahrscheinlichkeit $1 - e^{-\Omega(n)}$ zur Optimierung ausreichen. Mit $a(n) = \log n$ haben wir, dass $O\left(n^3 \log^2 n\right)$ Generationen ausreichen mit Wahrscheinlichkeit $1 - e^{-\Omega\left(\log^2 n\right)}$, was immer noch superpolynomiell schnell gegen 1 konvergiert.

Alle Ergebnisse dieses Kapitels beruhen wesentlich darauf, dass Crossover nur mit sehr kleiner Wahrscheinlichkeit (etwa $1/(n \log n)$) zum Einsatz kommt. Wir haben zum einen gezeigt, dass schon die Anwendung von Crossover mit einer derart verschwindenden Wahrscheinlichkeit ausreicht, um für $f_{\mathrm{J},m}$ mit $m = \Theta(\log n)$ die erwartete Laufzeit von superpolynomiell auf polynomiell zu senken. Trotzdem sind solch kleine Crossoverwahrscheinlichkeiten $q(n)$ sehr ungewöhnlich. Gerade bei genetischen Algorithmen geht man in der Regel davon aus, dass Crossover der wesentliche Suchoperator ist und mit großer Wahrscheinlichkeit (etwa $q(n) = \Omega(1)$) angewendet werden sollte. Wir können unsere Ergebnisse auf diesen Fall nicht verallgemeinern, so weit tragen unsere Beweismethoden uns nicht. Der Beweisschwerpunkt liegt in der Analyse der dritten Phase (Beweis zu Satz 9.6), wo wir nachweisen, dass die Effekte von Mutationen alleine ausreichen, für eine genügend gleichmäßige Verteilung der Nullbits zu sorgen. Wir nehmen dort sehr pessimistisch und unrealistisch an, dass jede Crossoveranwendung schädlich ist und die Konzentration der Nullbits verstärkt. Wir können uns das erlauben, weil die Wahrscheinlichkeit von Crossover mit $q(n) = 1/(n \log n)$ substanziell unter der Wahrscheinlichkeit einer hilfreichen Mutation mit $p^-(z) - p^+(z) = \Omega(1/n)$ liegt. Für den Beweis reicht offenbar $q(n) = o(1/n)$ aus. Um noch größere Crossoverwahrscheinlichkeiten handhaben zu können, ist es erforderlich, die durch Crossover verursachten Effekte genauer zu analysieren. Man muss dann nachweisen, dass sich Crossover bezüglich der Konzentration der Nullbits zumindest neutral verhält. Das ist in gewisser Weise anschaulich klar, leider aber nicht in jeder Situation tatsächlich der Fall. Man kann Populationen konstruieren, bei denen Crossover negative Effekte hat. Um solche Fälle handhaben zu können, sind Aussagen über die Verteilung der Nullbits in der Population erforderlich, was impliziert, dass man Abhängigkeiten zwischen verschiedenen Positionen kontrollieren muss. Eine Analysemethode zu finden, die das praktisch leistet, ist wichtiges Ziel zukünftiger Forschung.

10 Zusammenfassung und weiterführende Betrachtungen

Wir haben in dieser Arbeit eine ganze Reihe Erkenntnisse über evolutionäre Algorithmen gewonnen. Dabei standen vor allem „bekannte Tatsachen“ im Mittelpunkt unseres Interesses. Unter „bekannten Tatsachen“ seien in diesem Fall empirische Kenntnisse über evolutionäre Algorithmen verstanden, wie man sie im praktischen Umgang mit diesen Suchheuristiken in der Regel macht. Im Rahmen einer tieferen Durchdringung evolutionärer Algorithmen und auf dem Weg zu einer besseren Lehrbarkeit des Forschungsgebiets ist es notwendig, diesen empirischen Erkenntnissen eine sichere Basis, ein Fundament aus analytischen Gewissheiten zu geben. In diesem Ziel, von „Tatsachen“, die allgemein angenommen und geglaubt werden, zu bewiesenen Tatsachen zu kommen, liegt die Motivation dieser Arbeit. Wir wollen hier kurz rekapitulieren, auf welchen Wegen wir uns diesen Zielen genähert haben, was wir dabei konkret gelernt haben, welche Ergebnisse uns wichtig sind und welche offenen Fragen wir für die Zukunft für besonders erforschenswert halten.

In der Einleitung (Kapitel 1) haben wir evolutionäre Algorithmen in den Kontext der Suchheuristiken und so genannten Black Box Optimierung eingeordnet, ihre historischen Wurzeln angedeutet und unsere Perspektive festgelegt: Fragen nach der Effizienz der Suche bei der Optimierung im Suchraum $\{0,1\}^n$ von einfachen, statischen Funktionen $f\colon \{0,1\}^n \to \mathbb{R}$. Wir haben die erwartete Laufzeit und die Wahrscheinlichkeit, mit der eine Optimierung in T Generationen gelingt, als Maße für die Effizienz festgelegt und Beziehungen zwischen diesen zwei Effizienzmaßen charakterisiert. Als wesentliches und für die Praxis relevanteres Effizienzmaß haben wir die Wahrscheinlichkeit, mit der die Optimierung in einer gegebenen Anzahl T von Generationen gelingt, erkannt. Bevor wir uns konkreten Algorithmen und Funktionen zugewandt haben, haben wir im zweiten Kapitel versucht, grundsätzliche Beschränkungen der Optimierung in Black Box Szenarien zu erhellen. Dabei drückte sich eine wichtige Erkenntnis im NFL-Theorem (Wolpert und Macready 1997) aus: gemittelt über alle Zielfunktionen ist keine sinnvolle Optimierung möglich. Wir haben eingeschränkte Black Box Szenarien entworfen und diskutiert, warum in der Praxis nur eingeschränkte Black Box Szenarien vorkommen können. Außerdem haben wir anhand eines sehr kleinen Beispiels ausführlich und ganz konkret gezeigt, dass es in derart eingeschränkten Szenarien sehr wohl „bessere“ und „schlechtere“ Optimierverfahren gibt (Droste, Jansen und Wegener 1999). Das führte uns zu der verwandten Fragestellung, für welche

Funktionenklassen denn evolutionäre Algorithmen gute Optimierer sind und somit zum Problem der Klassifikation. Wir haben grundsätzlich zwischen deskriptiven und analytischen Klassifikationen unterschieden, Beispiel für beide Typen von Klassifikationen kennengelernt und ihre Schwächen exemplarisch aufgezeigt (Jansen 2000). Wir haben uns klar gemacht, dass es ernsthafte Beschränkungen der Realisierbarkeit von „perfekten“ Klassifikationen gibt und diskutiert, worin besondere Schwierigkeiten des Klassifikationsproblems bestehen. Davon ausgehend haben wir unser Vorgehen motiviert, das sehr viel bescheidener ist: Wir haben im Anschluss ausschließlich ganz konkrete evolutionäre auf konkreten Zielfunktionen untersucht.

In Kapitel 3 haben wir den (1+1) EA als einfachsten evolutionären Algorithmus kennengelernt und an zwei wichtigen Beispielen Untersuchungsmethoden erprobt. Wir haben außerdem die zwei Beispielfunktionen, f_1 und f_{B}, beides lineare Funktionen, dazu benutzt, die übliche Wahl für die Mutationswahrscheinlichkeit $p(n) = 1/n$ zu motivieren (Droste, Jansen und Wegener 1998a). Diese einführenden und motivierenden Ergebnisse haben zu einem zentralen Resultat im folgenden Kapitel 4 geführt, der erwarteten Laufzeit des (1+1) EA mit Mutationswahrscheinlichkeit $p(n) = 1/n$ auf der großen, natürlichen und wichtigen Klasse der linearen Funktionen (Droste, Jansen und Wegener 1998d). Von dort aus bot sich der Schritt zu unimodalen Funktionen an (Kapitel 5), die eine echte Obermenge der vollständig nicht-trivialen linearen Funktionen sind und wegen des Fehlens lokaler Maxima als einfach vermutet worden sind. Zentrales Ergebnis dieses Kapitels ist die Erkenntnis, dass es unimodale Funktionen gibt, für die der (1+1) EA exponentielle erwartete Laufzeit hat (Droste, Jansen und Wegener 1998b). Nützliches praktisches Werkzeug aus diesem Kapitel sind lange k-Pfade (Horn, Goldberg und Deb 1994; Rudolph 1997a), die wir später noch einmal benutzt haben. In Kapitel 6 haben wir schließlich untersucht, wie schwierig (im Sinne der erwarteten Laufzeit) Funktionen für den (1+1) EA überhaupt werden können. Wir konnte die obere Schranke n^n mit sehr einfachen Mitteln nachweisen und trotzdem explizit Funktionen geben, die asymptotisch diese Laufzeit im Durchschnitt erfordern (Droste, Jansen und Wegener 1998a).

In Kapitel 7 haben wir vorsichtige erste Schritte über die Analyse des (1+1) EA mit Standardmutationswahrscheinlichkeit $p(n) = 1/n$ hinaus gemacht. Wir haben uns zum einen mit einer einfacheren Mutation auseinandergesetzt, bei der in jeder Generation genau ein Bit mutiert und gesehen, dass diese Mutation bei naiver Implementierung schneller und mit weniger Zufallsbits zu berechnen ist, auf vielen Funktionen ähnliches Verhalten wie der (1+1) EA zeigt und analytisch leichter zu handhaben ist. Wir konnten aber auch zeigen, dass auf manchen Beispielen erheblich anderes Verhalten zu beobachten ist (Droste, Jansen und Wegener 1998a). Als zweiten Aspekt haben wir andere Werte für die Mutationswahrscheinlichkeit untersucht. Wir konnten in einem praktisch relevanten Sinn nachweisen, dass die übliche Mutationswahrscheinlichkeit $1/n$ bei weitem nicht optimal sein kann (Jansen und Wegener 2000b). Wir haben ausgehend von dieser Erkenntnis und der Einsicht, dass das Einstellen der passenden Mutationswahrscheinlichkeit ein schwieriges Problem ist, einen Algorithmus mit dynamischer Kontrolle der Mutationswahrscheinlichkeit vorgeschlagen und seine Robustheit an verschiedenen Beispielen untermauert. Wir haben aber auch gezeigt, dass dieser Algorithmen dem (1+1) EA mit Mutationswahrscheinlichkeit $p(n) = 1/n$ deutlich unterlegen sein kann (Droste, Jansen und Wegener 2000a).

In Kapitel 8 sind wir ähnlichen Ideen gefolgt und haben uns mit alternativen Selektionsmechanismen auseinandergesetzt. Zum einen haben wir gesehen, dass schon vermeintlich winzige Unterschiede im Selektionsmechanismus enorme Auswirkungen auf die Effizienz der Suche haben können (Droste, Jansen und Wegener 1998a). Außerdem haben wir uns mit einen randomisierten Selektionsmechanismus auseinandergesetzt, der in Abhängigkeit von einem Parameter und der Differenz der betrachteten Funktionswerte auch Verschlechterungen in den Funktionswerten akzeptiert. Nach Einsatz des vereinfachten Mutationsoperators aus Kapitel 7 haben wir den Algorithmus je nach Art der Parameterkontrolle als Instanz des Metropolis-Algorithmus bzw. von Simulated Annealing identifiziert. Wir konnten anhand einer einfachen Funktion und mit einfachen beweistechnischen Mitteln zeigen, dass Simulated Annealing mit passender „Abkühlstrategie“ selbst einem optimal eingestellten Metropolis-Algorithmus bei weitem überlegen sein kann. Wir haben anhand zweier Beispielfunktionen aber auch erkannt, dass ein „Abkühlen“ nicht per se sinnvoller ist als ein „Erwärmen“ im Laufe einer Optimierung (Droste, Jansen und Wegener 2000a).

In Kapitel 9 schließlich haben wir den bis dahin nicht berücksichtigten Suchoperator Crossover in unsere Überlegungen miteinbezogen. Wir weisen an einer Familie von Beispielfunktionen nach, dass der Einsatz von uniformen Crossover schon mit sehr geringer Wahrscheinlichkeit die Effizienz der Suche evolutionärer Algorithmen erheblich steigern kann (Jansen und Wegener 1999). Die verwendete Familie von Beispielfunktionen lieferte uns außerdem noch ein Hierarchieresultat, das im Sinne einer Klassifikation verstanden werden kann und demonstriert, dass es neben sehr einfachen und sehr schwierigen Funktionen auch jeweils (alle) Zwischenstufen gibt (Droste, Jansen und Wegener 1998a).

Wir haben unsere Ergebnisse für ganz konkrete evolutionäre Algorithmen erzielt. Das liegt zum einen daran, dass das Paradigma „evolutionärer Algorithmus“ so groß und unscharf ist, dass es schwer fällt, abzugrenzen und zu einem gegebenen Algorithmus zu entscheiden, ob er noch als Instanz dieses Paradigmas aufgefasst werden kann. Zum anderen haben wir aber auch an verschiedenen Stellen gesehen, dass schon verhältnismäßig kleine Änderungen an einem evolutionären Algorithmus recht dramatische Auswirkungen in seinem Suchverhalten zeitigen können. Was unseren Ergebnissen außerdem noch gemeinsam ist, sind die Beispielfunktionen, an denen wir unsere Aussagen ganz konkret festgemacht haben. Diese Funktionen haben wichtige Aufgaben: Sie sind derart konstruiert, dass sie bestimmte Effekte in konzentrierter Form repräsentieren, so dass eine Analyse besonders erleichtert wird und die Aussage deutlich und anschaulich gemacht werden kann. Sie stützen darum die Vorstellung, vermitteln intuitive Einsicht und haben zweifellos wichtige pädagogische Funktion. Sie sind auch insofern ganz sicher nicht pathologisch, als sie alle kurze Definitionen haben, effizient auswertbar sind und einfache und verständliche Struktur haben. Sie sind aber alle ganz deutlich als konstruiert und „künstlich“ erkennbar, sie sind nicht Instanzen eines „natürlichen Problems“. Es ist ganz ohne Zweifel ein wichtiger nächster Schritt in der theoretischen Analyse evolutionärer Algorithmen, Analyseergebnisse für Instanzen natürlicher Probleme zu finden.

Was unter einem natürlichen Problem zu verstehen ist, kann prinzipiell nicht ganz eindeutig definiert werden. Im allgemeinen wird man ein Problem als natürliches Problem anerkennen, wenn es wohlbekannt ist, wenn es von (praktischer) Bedeutung ist und wenn es ein grundlegendes Problem ist. Von den Probleminstanzen, die wir betrachten, verlangen wir zusätzlich, dass sie durch eine nicht zu künstliche und konstruierte Instanziierung des Problems entstehen. In diesem Sinne kann man sicher das Erfüllbarkeitsproblem SAT als natürliches Problem benennen. Wir wollen hier anstelle von allgemeinen Überlegungen zu möglichen zukünftigen Forschungsrichtungen am Beispiel des Erfüllbarkeitsproblems exemplarisch zeigen, wie der Schritt zur Analyse von Instanzen natürlicher Probleme aussehen kann.

Das Erfüllbarkeitsproblem SAT ist ein schweres Problem, es ist das historisch erste Problem, für das nachgewiesen werden konnte, dass es NP-vollständig ist (Cook 1971). Eine Probleminstanz besteht aus einer Menge von Variablen $\{x_1, x_2, \ldots, x_n\}$, die mit 0 oder 1 belegt werden können, und einer Menge von Klauseln $\{u_1, u_2, \ldots, u_m\}$, wobei eine Klausel eine Disjunktion einiger Literale ist. Es soll entschieden werden, ob es eine Belegung der Variablen gibt, so dass alle Klauseln gleichzeitig erfüllt sind. Offenbar kann SAT auch leicht als Optimierungsproblem, dann MAXSAT genannt, aufgefasst werden:

Finde eine Belegung, so dass möglichst viele Klauseln gleichzeitig erfüllt sind. Für die Variante MAX-3-SAT, bei der jeder Klausel höchstens drei Literale enthält, ist bekannt, dass auch das Finden einer Belegung, welche eine Anzahl von Klauseln erfüllt, die bis auf einen kleinen Faktor an die Anzahl überhaupt gleichzeitig erfüllbarer Klauseln herankommt, unter der Voraussetzung $\mathrm{P} \neq \mathrm{NP}$ nicht in Polynomialzeit möglich ist (Aurora, Lund, Motwani, Sudan und Szegedy 1992; Håstad 1997).

Wir werden hier eine Instanz von MAX-3-SAT präsentieren und argumentieren, warum evolutionäre Algorithmen für diese Instanz mit Wahrscheinlichkeit sehr nahe bei 1 nicht in Polynomialzeit eine Lösung finden. Wir werden zeigen, dass sogar eine gute Approximation in Polynomialzeit mit großer Wahrscheinlichkeit nicht gelingt. Für mutationsbasierte evolutionäre Algorithmen führen wir sogar einen formalen Beweis (Droste, Jansen und Wegener 2000b). Dieser Beweis sollte nicht so missverstanden werden, dass wir argumentieren, dass evolutionäre Algorithmen zum Lösen von Erfüllbarkeitsproblemen grundsätzlich ungeeignet wären. Zum einen basiert die hier verwendete Probleminstanz auf einer Instanz von Papadimitriou (1994), die für die meisten Heuristiken für SAT schwierig ist. Zum anderen ist es aktuell so, dass der in Hinsicht auf durchschnittliche Worst Case Laufzeit beste Algorithmus für MAX-3-SAT ein evolutionärer Algorithmus mit problemspezifischer Mutation und geeigneter Neustartstrategie ist (Schöning 1999).

Wir merken an, dass man leicht jede MAX-3-SAT Instanz in ein multivariates Polynom vom Grad 3 transformieren kann. Wir geben also hier ein Polynom mit kleinem Grad an, das von evolutionären Algorithmen auch mit Hilfe von Neustarts nicht effizient optimiert werden kann. Das unterscheidet diese Funktion von f_{Q}, wo wir zwar sogar nur Polynomgrad 2 hatten, eine Optimierung mit Hilfe von Neustarts aber im Durchschnitt in Zeit $O(n \log n)$ gelingt.

Wir beginnen die Konstruktion unserer Zielfunktion wie erwähnt basierend auf einer MAX-3-SAT Instanz von Papadimitriou (1994) über den Variablen $\{x_1, x_2, \ldots, x_n\}$. Sie besteht aus n Klauseln der Länge 1 und $n(n-1)(n-2)$ Klauseln der Länge 3, im einzelnen sind das die Klauseln

- x_i für alle $i \in \{1, 2, \ldots, n\}$ sowie
- $x_i \vee \overline{x_j} \vee \overline{x_k}$ mit $(i, j, k) \in \{1, 2, \ldots, n\}^3$, wobei $i \neq j$, $i \neq k$ und $j \neq k$ gilt.

Wir bemerken zunächst, dass eine optimale Belegung der Variablen offensichtlich ist: setzt man alle Variablen $x_i = 1$, so sind alle $n(n-1)(n-2) + n$

Klauseln gleichzeitig erfüllt. Es gibt natürlich auch keine andere Belegung, die alle Klauseln erfüllt. Für einen Suchalgorithmus, der im Black Box Szenarium eine Optimierung versucht, sieht die Situation aber anders aus. Bei zufälligem Startpunkt sind etwa die Hälfte der Variablen auf 0 gesetzt. Dann ist es günstig, ein Literal auf 0 zu setzen, weil dadurch tendenziell mehr Klauseln erfüllt werden. Wenn man einem Optimierverfahren die Struktur der Probleminstanz mitgibt, ändert das natürlich die Situation. Alle Klauseln sind so genannte Horn-Klauseln, das bedeutet, sie enthalten höchstens ein nicht-negiertes Literal. Solche Klauseln entstehen typischerweise bei der Formulierung von Datenbankanfragen. Es ist seit langem bekannt, dass SAT für Horn-Klauseln effizient lösbar ist (Garey und Johnson 1979). Weil wir hier aber wie immer von Optimierung im Black Box Szenarium sprechen, spielt das bei unseren Überlegungen keine Rolle.

Wir formulieren diese MAXSAT-Instanz als Funktion f_{S}. Dabei gibt $f_{\mathrm{S}}(x)$ an, wieviele Klauseln der MAXSAT-Instanz die Belegung $x \in \{0,1\}^n$ der Variablen $x_1, x_2 \ldots, x_n$ gleichzeitig erfüllt.

Definition 10.1. *Die Funktion $f_S\colon \{0,1\}^n \to \mathbb{R}$ ist definiert durch*

$$f_S(x) := \sum_{1\le i\le n} x_i + \sum_{1\le i\le n} \sum_{\substack{1\le j\le n\\ j\ne i}} \sum_{\substack{1\le k\le n\\ k\ne i, k\ne j}} (1-(1-x_i)\,x_j x_k)$$

für alle $x \in \{0,1\}^n$.

Die Funktion f_{S} ist offensichtlich symmetrisch, wir können sie darum (wie wir das immer für symmetrische Funktionen tun) anschaulich graphisch darstellen (Abbildung 10).

Weil wir noch einige Überlegungen zu f_{S} vornehmen wollen und später gezielte Modifikationen vornehmen werden, um bestimmte Effekte zu verstärken, ist es hilfreich, eine Funktion $g_{\mathrm{S},n}\colon \mathbb{R} \to \mathbb{R}$ definieren, über die man leichter sprechen kann, und für die $g_{\mathrm{S},n}(i) = f_{\mathrm{S}}(x)$ für alle $x \in \{0,1\}^n$ mit $\|x\|_1 = i$ gilt.

Definition 10.2. *Sei $n \in \mathbb{N}$. Die Funktion $g_{S,n}\colon \mathbb{R} \to \mathbb{R}$ ist definiert als*

$$g_{S,n}(s) := s^3 - (n+1)s^2 + (n+1)s + n(n-1)(n-2)$$

für alle $s \in \mathbb{R}$.

In Abbildung 10 haben wir den dargestellten Abschnitt des Definitionsbereichs etwas erweitert und die Funktion f_{S} mit der Funktion $g_{\mathrm{S},40}$ hinterlegt. Wir vergewissern uns zunächst, dass $g_{\mathrm{S},n}$ auch die gewünschte Übereinstimmung mit f_{S} garantiert.

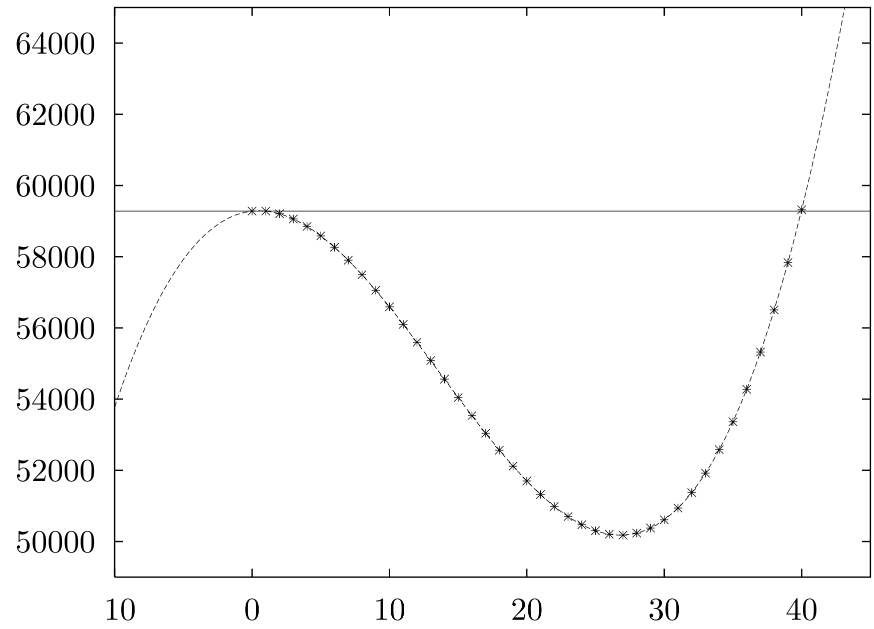

Abbildung 10: Die Funktion $f_{\mathrm{S}}\colon \{0,1\}^{40} \to \mathbb{R}$.

Lemma 10.3. *Für alle $n \in \mathbb{N}$ und alle $x \in \{0,1\}^n$ gilt*

$$f_S(x) = g_{S,n}(\|x\|_1).$$

Beweis. Es ist

$$\begin{aligned}
f_{\mathrm{S}}(x) &= \sum_{1\le i\le n} x_i + \sum_{1\le i\le n} \sum_{\substack{1\le j\le n\\ j\ne i}} \sum_{\substack{1\le k\le n\\ k\ne i, k\ne j}} (1-(1-x_i)\,x_jx_k) \\
&= \sum_{1\le i\le n} x_i + \sum_{1\le i\le n} \sum_{\substack{1\le j\le n\\ j\ne i}} \sum_{\substack{1\le k\le n\\ k\ne i, k\ne j}} 1 - x_jx_k + x_ix_jx_k \\
&= \|x\|_1 + n(n-1)(n-2) - 2(n-2) \sum_{1\le j<k\le n} x_jx_k \\
&\qquad +6 \sum_{1\le i<j<k\le n} x_ix_jx_k \\
&= \|x\|_1 + n(n-1)(n-2) - 2(n-2)\binom{\|x\|_1}{2} + 6\binom{\|x\|_1}{3} \\
&= \|x\|_1^3 - (n+1)\|x\|_1^2 + (n+1)\|x\|_1 + n(n-1)(n-2) \\
&= g_{\mathrm{S},n}(\|x\|_1)
\end{aligned}$$

wie behauptet. $\square$

Man sieht direkt, dass $f_{\mathrm{S}}(\mathbf{1}) = n(n-1)(n-2)+n$ global maximal ist, während $f_{\mathrm{S}}(\mathbf{0}) = n(n-1)(n-2)$ das einzige andere lokale Maximum ist. Das lokale Minimum von $g_{\mathrm{S},n}$ wird für

$$s = \frac{n+1}{3} + \frac{\sqrt{n^2-n-2}}{3} \approx \frac{2}{3}(n+1)$$

angenommen. Darum befindet sich die höchstens polynomiell große Population eines evolutionären Algorithmus mit Wahrscheinlichkeit $1 - e^{-\Omega(n)}$ „links" vom lokalen Minimum und die Situation entspricht im wesentlichen der Situation bei der Funktion f_{T}. Alle lokalen Hinweise dirigieren den Algorithmus in Richtung des lokalen Maximums weg vom globalen Maximum, das darum schwer zu finden ist. Während f_{T} (siehe auch Kapitel 6) aber eine hochgradig künstliche Funktion ist, haben wir uns überlegt, dass f_{S} Instanz eines natürlichen Problem ist.

In der Praxis gibt man sich bei schwierigen Zielfunktionen oft auch mit einer guten Approximation zufrieden. Wir hatten erwähnt, dass MAX-3-SAT nicht gut approximierbar ist. Das gilt allerdings natürlich nicht für jede Probleminstanz. Insbesondere ist f_{S} gut approximierbar auch für einfache evolutionäre Algorithmen: der (1+1) EA findet in erwarteter Zeit $\Theta(n \log n)$ ein lokales Maximum von f_{S}, mit großer Wahrscheinlichkeit $\mathbf{0}$ (mit sehr kleiner Wahrscheinlichkeit das globale Maximum $\mathbf{1}$). Dann gilt für die Güte der gefundenen Lösung

$$\frac{n(n-1)(n-2)+n}{n(n-1)(n-2)} = 1 + \frac{1}{n^2-3n+2},$$

was rasch gegen 1 konvergiert. Mit einer Lösung, deren Qualität derart dicht bei der einer optimalen Lösung liegt, wird man in der Praxis zufrieden sein. Man könnte darum argumentieren wollen, dass die Funktion f_{S} unter praktischen Aspekten für evolutionäre Algorithmen nicht schwer zu optimieren ist. Um diesem Argument zu begegnen, führen wir eine Variante von f_{S} ein, für die wir nachweisen werden, dass hier nicht nur das globale Maximum sondern auch gute Approximationen schwer zu finden sind.

Definition 10.4. *Sei $r \in \mathbb{N}$. Die Funktion $f_{S,r}\colon \{0,1\}^n \to \mathbb{R}$ entsteht wie die Funktion f_S aus der genannten SAT-Instanz, wobei wir die Klausel x_i für alle $i \in \{1,2,\dots,n\}$ genau r-mal vorkommen lassen. Analog zur Funktion $g_{S,n}$ definieren wir die Funktion $g_{S,n,r}\colon \mathbb{R} \to \mathbb{R}$ durch*

$$g_{S,n,r}(s) := s^3 - (n+1)s^2 + (n+r)s + n(n-1)(n-2)$$

für alle $s \in \mathbb{R}$.

Wählt man für r substanziell größere Werte als 1, ändert sich die Funktion $f_{S,r}$ im Vergleich zur Funktion f_S erheblich. Das globale Maximum von $g_{S,n,r}$ liegt offensichtlich weiterhin bei n (für $0 \leq s \leq n$), aber es gibt ein zweites lokales Maximum jetzt nicht mehr bei 0 sondern bei

$$s_{\max}(n,r) = \frac{n+1}{3} - \sqrt{\left(\frac{n+1}{3}\right)^2 - \frac{n+r}{3}}.$$

Für nicht zu kleine Werte von r rückt also das lokale Maximum näher an das globale Maximum heran. Außerdem liegt jetzt das lokale Minimum nicht mehr bei etwa $(2/3)(n+1)$, wie das bei $g_{S,n}$ der Fall ist, sondern bei

$$s_{\min}(n,r) = \frac{n+1}{3} + \sqrt{\left(\frac{n+1}{3}\right)^2 - \frac{n+r}{3}},$$

was für nicht zu kleine Werte von f merklich weiter „nach links" rutscht. Wir argumentieren, dass die Funktion $f_S = f_{S,1}$ für evolutionäre Algorithmen schwierig ist, weil sie ähnlich wie f_T mit großer Wahrscheinlichkeit lokal nur „falsche Hinweise" gibt. Das gilt natürlich nur, wenn die Population vollständig „links" des lokalen Minimums ist. Wenn die Position des lokalen Minimums in die Nähe von $n/2$ rückt, wird die Funktion für evolutionäre Algorithmen einfach, jedenfalls wenn man Neustartstrategien berücksichtigt. Die Länge ist dann ähnlich wie bei der Funktion f_Q (siehe Kapitel 6). Wir betrachten darum die Funktion $f_{S,r}$ nur für solche Werte von r, bei denen das lokale Minimum der Funktion $g_{S,n,r}$ für einen Wert $s_{\min}(n,r)$ mit

$$s_{\min}(n,r) \geq \left(\frac{1}{2} + \varepsilon\right) n$$

angenommen wird. Damit beschränken wir r durch

$$r \leq \left\lceil \left(\frac{1}{4} - \varepsilon\right) \left(n^2 - n\right) \right\rceil,$$

wobei ε eine positive Konstante ist. Um zu zeigen, dass $f_{S,r}$ auch nicht gut approximativ lösbar ist, wollen wir jetzt erreichen, dass die Güte einer lokal maximalen Lösung, also der Quotient

$$\frac{g_{S,n,r}(n)}{g_{S,n,r}\left(s_{\max}(n,r)\right)}$$

unter dieser Bedingung für r möglichst groß wird. Wir setzen vereinfachend $r := \alpha n^2$. Dann haben wir

$$g_{S,n,r}(n) = \Theta\left(n^3\right)$$

und

$$g_{\mathrm{S},n,r}\left(s_{\max}(n,r)\right) = \Theta\left(n^3\right),$$

also genügt es, Terme der Größenordnung n^3 zu betrachten. Das gibt dann

$$(1+\alpha)n^3$$

für n und

$$\left(1+\beta-\beta^2+\beta^3\right)n^3$$

für $s_{\max}(n,r)$, wobei

$$\beta = \frac{1}{3} - \sqrt{\frac{1}{9} - \frac{\alpha}{r}}$$

gilt. Wir haben also den Quotienten

$$\frac{1+\alpha}{1+\beta-\beta^2+\beta^3}.$$

Für $\alpha = 0{,}25$ ergibt sich für diesen Quotienten etwa 1,0931, die optimale Lösung ist also um etwa 9,31% besser als die nur lokal optimale Lösung, die leicht zu finden ist.

Für den (1+1) EA ist aus den Überlegungen aus Kapitel 6 klar, dass mit großer Wahrscheinlichkeit $n^{O(1)}$ Generationen zur Optimierung von $f_{\mathrm{S},0{,}25n^2}$ nicht ausreichen. Wir wollen zeigen, dass das auch für eine große Klasse mutationsbasierter evolutionärer Algorithmen gilt. Dazu definieren wir zunächst exakt, welche Algorithmen wir in unsere Überlegungen einschließen können.

Definition 10.5. *Ein* mutationsbasierter EA *benutzt eine Population der Größe $s(n)$, die zufällig gleichverteilt initialisiert wird. Die Populationsgröße $s(n)$ ist polynomiell in n beschränkt. In jeder Generation wird aus der Population eine Anzahl Kinder generiert. Das geschieht mittels Selektion aus den Eltern, wobei für alle Eltern x, y mit $f(x) \geq f(y)$ stets gilt, dass x mit mindestens gleicher Wahrscheinlichkeit wie y selektiert wird. Auf die gewählten Eltern kann bitweise Mutation mit Mutationswahrscheinlichkeit $p(n) \in [0; 1/2]$ durchgeführt werden. Aus den Eltern und Kindern wird die nächste Generation selektiert, wobei für alle Eltern x, y mit $f(x) \geq f(y)$ und alle Kinder x', y' mit $f(x') \geq f(y')$ gilt, dass x mit mindestens gleicher Wahrscheinlichkeit wie y und x' mit mindestens gleicher Wahrscheinlichkeit wie y' selektiert wird.*

Basierend auf dieser recht allgemeinen Definition, die eine Vielzahl verschiedener mutationsbasierter evolutionärer Algorithmen einschließt, können wir jetzt unser Resultat formulieren.

Satz 10.6. *Die Wahrscheinlichkeit, dass ein mutationsbasierter EA im Sinne von Definition 10.5, der insgesamt* $e^{o(\sqrt{n})}$ *Individuen generiert, die Funktion* $f_{S,0,25n^2}\colon \{0,1\}^n \to \mathbb{R}$ *optimiert, ist durch* $e^{-\Omega(\sqrt{n})}$ *nach oben beschränkt.*

Beweis. Wir schätzen die Wahrscheinlichkeit, das globale Optimum $\mathbf{1}$ der Funktion $f_{\mathrm{S},0,25n^2}$ zu erreichen, nach oben ab durch die Wahrscheinlichkeit, einen String mit mindestens $((1/2)+\varepsilon)n$ Bits mit Wert Eins zu erreichen, dabei ist ε eine positive Konstante. Weil $f_{\mathrm{S},0,25n^2}$ symmetrisch ist, können wir in der Population stets Strings x durch $1^{\|x\|_1}0^{n-\|x\|_1}$ ersetzen, ohne etwas wesentlich zu ändern.

Wir vermuten, dass es für die Optimierung günstiger ist, einen String $x = 0^{n-s}1^s$ in der Population zu haben als einen String $x' = 0^{n-s'}1^{s'}$, wenn $s > s'$ gilt. Wir beweisen die etwas schwächere Aussage, dass die Wahrscheinlichkeit, einen String mit mindestens s'' Einsbits durch Mutation zu erreichen, nicht kleiner ist, wenn x als Elter gewählt ist. Wir betrachten x und x' genauer.

$$\begin{array}{rcl} x & = & \boxed{0\cdots0}\boxed{1\cdots1}\boxed{1\cdots1} \\ x' & = & \underbrace{\boxed{0\cdots0}}_{n-s}\underbrace{\boxed{0\cdots0}}_{s-s'}\underbrace{\boxed{1\cdots1}}_{s'} \end{array}$$

Eine Mutation wirkt sich sowohl auf den vorderen $n-s$ Bits als auch auf den hinteren s' Bits bei x und x' gleich aus. Unabhängig von diesen Positionen werden die $s-s'$ Bits an den Positionen $n-s+1, n-s+2, \ldots, n-s'$ mit Wahrscheinlichkeit $p(n)$ mutiert. Weil $p(n) \leq 1/2$ gilt, ist für jedes $d \in \mathbb{N}_0$, die Wahrscheinlichkeit, höchstens d dieser Bits zu mutieren, mindestens so groß wie die Wahrscheinlichkeit, mindestens $(s-s')-d$ Bits zu mutieren. Daraus folgt direkt die Behauptung.

Solange alle Individuen „links des lokalen Minimums“ und „rechts des nur lokalen Maximums“ sind, kann Selektion gemäß unseren Voraussetzungen nur Strings mit einer kleineren Anzahl Einsbits bevorzugen. Wir haben $r = 0{,}25n^2$, folglich $g_{\mathrm{S},n,0,25n^2}$ monoton fallend in $[n/6; ((1/2)+\varepsilon)n]$. Wenn wir annehmen, dass alle Individuen der Population Anzahlen von Einsbits haben, die innerhalb dieses Intervalls liegen, kann die Selektion bestenfalls zufällig gleichverteilt wählen.

Wir ersetzen jetzt gedanklich in der Population jedes Individuum mit weniger als $((1+\varepsilon)/2)n$ Einsbits durch ein Individuum mit genau $((1+\varepsilon)/2)n$

Einsbits. Wir haben uns überlegt, dass das die Erfolgswahrscheinlichkeit des evolutionären Algorithmus nur vergrößern kann. Nehmen wir an, dass ein Individuum x^* mit $((1/2)+\varepsilon)n$ Einsbits erzeugt wird. Wir verfolgen in dieser Situation einen historienbasierten Ansatz, wie man ihn bei Rabani, Rabinovich und Sinclair (1998) findet. Dieses Individuum x^* hat eine Reihe von „Vorfahren" $x_0, x_1, \ldots$, so dass x_0 zur initialen Population gehört und x_{i+1} aus x_i durch Mutation entstanden ist. Gemäß unserer Annahmen haben wir $\|x_0\|_1 = ((1+\varepsilon)/2)n$, $\|x_i\|_1 \geq ((1+\varepsilon)/2)n$ für alle $i \geq 0$ und $\|x^*\|_1 \geq ((1/2)+\varepsilon)n$. Dann gibt es einen Index i^*, so dass $\|x_i\|_1 > ((1+\varepsilon)/2)n$ für alle $i > i^*$ gilt. Wir lassen den Anfang der Sequenz weg und betrachten die Folge $x_0, x_1, \ldots, x_{j^*}$ mit $x_{j^*} = x^*$, $\|x_0\|_1 = ((1+\varepsilon)/2)n$ und $\|x_i\|_1 > ((1+\varepsilon)/2)n$ für alle $i > 0$. Wir können voraussetzen, dass Individuum x_i direkt von x_{i+1} durch Mutation entstanden ist.

Die Strings $x_0, x_1, \ldots, x_{j^*-1}$ enthalten insgesamt mindestens $((1+\varepsilon)/2)nj^*$ Einsbits und höchstens $((1-\varepsilon)/2)nj^*$ Nullbits. Wir suchen eine obere Schranke für die Wahrscheinlichkeit, beginnend mit x_0 schließlich bei x_{j^*} anzukommen. Alle nj^* Bits mutieren mit Mutationswahrscheinlichkeit $p(n)$. Eine notwendige Bedingung, um bei x_{j^*} anzukommen, ist es, dass insgesamt mindestens $n\varepsilon/2$ Bits mehr ihren Wert von Null auf Eins ändern als Bits ihren Wert von Eins auf Null ändern. Dazu ist es erforderlich, dass entweder höchstens $nj^*p(n)/2$ Bits mit Wert Eins mutieren oder mindestens $nj^*p(n)/2$ Bits mit Wert Null. Andernfalls ist die Anzahl Bits, die von Eins nach Null mutieren größer als die Anzahl Bits, die von Null nach Eins mutieren, so dass die Anzahl Einsbits insgesamt abnimmt, was einen Misserfolg impliziert. Wir erinnern uns, dass ε eine positive Konstante ist. Daraus folgt

$$\frac{nj^*p(n)}{2} \leq (1-\delta_1) \cdot \left(\frac{1}{2} + \frac{\varepsilon}{2}\right) nj^*p(n)$$

für eine positive Konstante δ_1. Dabei ist $((1/2)+\varepsilon)nj^*p(n)$ eine untere Schranke für die erwartete Anzahl Bits, die ihren Wert von Null nach Eins wechseln. Außerdem haben wir

$$\frac{nj^*p(n)}{2} \geq (1+\delta_2) \cdot \left(\frac{1}{2} - \frac{\varepsilon}{2}\right) nj^*p(n)$$

für eine positive Konstante δ_2. Dabei ist $((1/2)-\varepsilon)nj^*p(n)$ eine obere Schranke für die erwartete Anzahl Bits, die ihren Wert von Eins nach Null wechseln. Folglich können wir durch Anwendung der Chernoff-Schranken die Wahrscheinlichkeit für beide Ereignisse durch $e^{-\Omega(nj^*p(n))}$ nach oben abschätzen. Falls

$$p(n) = \Omega\left(\frac{1}{j^*\sqrt{n}}\right)$$

gilt, ist diese Wahrscheinlichkeit exponentiell klein. Wenn andererseits

$$p(n) = O\left(\frac{1}{j^*\sqrt{n}}\right)$$

gilt, haben wir $nj^*p(n)/2 = O(\sqrt{n})$. In diesem Fall nutzen wir aus, dass mindestens $\varepsilon n/2$ Bits mit Wert Null im Erfolgsfall mutieren müssen. Wir wenden wieder Chernoff-Schranken an und sehen, dass die Wahrscheinlichkeit hierfür durch $e^{-\Omega\left(\sqrt{n}\right)}$ nach oben beschränkt ist. Folglich ist die Erfolgswahrscheinlichkeit insgesamt durch $e^{-\Omega\left(\sqrt{n}\right)}$ nach oben beschränkt, wenn die Anzahl erzeugter Individuen insgesamt durch $e^{o\left(\sqrt{n}\right)}$ beschränkt ist. $\square$

Wir vermuten, dass evolutionäre Algorithmen mit (uniformen) Crossover die Funktion $f_{\mathrm{S},0,25n^2}$ auch nicht effizienter optimieren. Für einen Beweis fehlen uns aber — wie in Kapitel 9 ausführlich erläutert — die passenden Beweismethoden. So zeigt sich an dieser Stelle, wie eng die verschiedenen bestehenden Wissenlücken und Analysedefizite zusammenhängen und wie groß der Bedarf nach weiterführender Forschung auf diesem spannenden und lebendigen Teilgebiet der Algorithmik ist.

A Einige mathematische Grundlagen

In diesem Anhang geben wir einige Definitionen und Sätze wieder, die alle wohlbekannt und recht grundlegend sind. Es handelt sich dabei stets um mehr oder minder einfache Gegenstände der Mathematik, die für diese Arbeit wichtig sind und verschiedentlich verwendet werden. Konkret geht es um einige Grundsätzlichkeiten im Zusammenhang mit diskreten Wahrscheinlichkeitsräumen, Notationen für das asymptotische Verhalten von Funktionen und einige nützliche Formeln und Abschätzungen. Dieser Anhang versteht sich als Serviceleistung für den Leser, der die eine oder andere Definition oder Formel nachschlagen möchte. Es geht natürlich nicht darum, eine Einführung in die berührten Themenbereiche zu geben oder das Studium geeigneter Literatur zu ersetzen. Als grundlegende Literatur seien hier Feller (1968) und Feller (1971), Graham, Knuth und Patashnik (1989), Motwani und Raghavan (1995) und Cormen, Leiserson und Rivest (1990) empfohlen. Die wesentlichen komplexitätstheoretischen Grundlagen, die hier nicht wiederholt werden, findet man in verständlicher und anschaulicher Form in Garey und Johnson (1979), eine kurze Zusammenfassung unter besonderer Berücksichtigung der im Rahmen dieser Arbeit natürlich einschlägigen Aspekte Optimierung und Approximation findet man auch bei Jansen (1998).

Wir beginnen mit den grundlegenden Definitionen für diskrete Wahrscheinlichkeitsräume.

Definition A.1. *Ein* diskreter Wahrscheinlichkeitsraum *ist ein Paar* (Ω, p)*, wobei* Ω *eine abzählbare Menge ist und* p *eine Abbildung* $p\colon \Omega \to [0;1]$*, so dass*

$$\sum_{\omega \in \Omega} p(\omega) = 1$$

gilt. Ein Ereignis *ist eine Teilmenge* $A \subseteq \Omega$*. Ein Element* $a \in \Omega$ *heißt* Elementarereignis *und hat* Wahrscheinlichkeit $p(a)$*. Die Wahrscheinlichkeit eines Ereignisses ist als Summe der Wahrscheinlichkeiten der Elementarereignisse*

$$\mathit{Prob}\,(A) = \sum_{a \in A} p(a)$$

definiert. Für paarweise disjunkte Ereignisse $A_1, A_2, \ldots, A_n$ *gilt*

$$\mathit{Prob}\left(\bigcup_{1 \le i \le n} A_i\right) = \sum_{1 \le i \le n} \mathit{Prob}\,(A_i)\,.$$

Aus dieser Definition lassen sich einige nützliche Aussagen ableiten, die wir kurz festhalten. Sie erweisen sich bei vielen Abschätzungen als hilfreich.

Lemma A.2. *Sei* (Ω, p) *ein Wahrscheinlichkeitsraum,* $n \in \mathbb{N}$. *Für beliebige Ereignisse* $A_1, A_2, \dots A_n \subseteq \Omega$ *gilt:*

$$\begin{aligned} \mathit{Prob}\left(\overline{A_1}\right) &= 1 - \mathit{Prob}\,(A_1) \\ A_1 \subseteq A_2 &\Rightarrow \mathit{Prob}\,(A_1) \leq \mathit{Prob}\,(A_2) \\ \mathit{Prob}\,(A_1 \cup A_2) &= \mathit{Prob}\,(A_1) + \mathit{Prob}\,(A_2) - \mathit{Prob}\,(A_1 \cap A_2) \\ &\leq \mathit{Prob}\,(A_1) + \mathit{Prob}\,(A_2) \\ \mathit{Prob}\left(\bigcup_{1 \leq i \leq n} A_i\right) &\leq \sum_{i \geq 1} \mathit{Prob}\,(A_i) \end{aligned}$$

Ein wichtiger bis jetzt noch nicht eingeführter Begriff ist der Erwartungswert.

Definition A.3. *Sei* (Ω, p) *ein diskreter Wahrscheinlichkeitsraum, sei* B *eine beliebige nicht-leere Menge. Eine Funktion* $X\colon \Omega \to B$ *heißt* Zufallsfunktion, *gilt* $B = \mathbb{R}$, *so heißt* X *auch* Zufallsvariable. *Der* Erwartungswert *einer Zufallsvariablen* X *ist als*

$$E(X) = \sum_{\omega \in \Omega} X(\omega) \cdot p(\omega)$$

definiert.

Einige nützliche Aussagen über den Erwartungswert notieren wir wiederum als Lemma ohne Beweis.

Lemma A.4. *Sei* (Ω, p) *ein diskreter Wahrscheinlichkeitsraum, seien* X *und* Y *zwei zugehörige Zufallsvariable. Seien* $\alpha, \beta \in \mathbb{R}$. *Dann gilt:*

$$\begin{gathered} E(\alpha \cdot X + \beta \cdot Y) = \alpha \cdot E(X) + \beta \cdot E(Y) \\ (\forall \omega \in \Omega\colon X(\omega) \leq Y(\omega)) \Rightarrow E(X) \leq E(Y) \\ (\forall \gamma \in \mathbb{R}\colon \mathit{Prob}\,(X \geq \gamma) \leq \mathit{Prob}\,(Y \geq \gamma)) \Rightarrow E(X) \leq E(Y) \\ (X : \Omega \to \{0,1\}) \Rightarrow E(X) = \mathit{Prob}\,(X = 1) \\ (X : \Omega \to \mathbb{N}) \Rightarrow E(X) = \sum_{i \geq 1} \mathit{Prob}\,(X \geq i) \\ \left(X : \Omega \to \mathbb{R}_0^+\right) \Rightarrow \sum_{i \geq 1} \mathit{Prob}\,(X \geq i) \leq E(X) \leq \sum_{i \geq 0} \mathit{Prob}\,(X \geq i) \end{gathered}$$

Wir führen jetzt noch die wichtigen Konzepte der bedingten Wahrscheinlichkeiten und der Unabhängigkeit ein.

Definition A.5. *Sei* (Ω, p) *ein diskreter Wahrscheinlichkeitsraum, seien* $A, B \subseteq \Omega$ *zwei Ereignisse, sei* $\mathit{Prob}\,(B) > 0$*. Die* bedingte Wahrscheinlichkeit *von* A *unter* B *definieren wir als*

$$\mathit{Prob}\,(A \mid B) := \frac{\mathit{Prob}\,(A \cap B)}{\mathit{Prob}\,(B)}.$$

Ereignisse $A_1, A_2, \dots$ *heißen* unabhängig*, wenn*

$$\mathit{Prob}\left(\bigcap_{i \geq 1} A_i\right) = \prod_{i \geq 1} \mathit{Prob}\,(A_i)$$

gilt. Sei X *eine Zufallsvariable. Der* bedingte Erwartungswert *von* X *unter* B *ist definiert durch*

$$E(X \mid B) = \sum_{b \in B} \frac{p(b)}{\mathit{Prob}\,(B)} \cdot X(b).$$

Die Zufallsvariable X *heißt* unabhängig *vom Ereignis* B*, wenn*

$$\forall U \subseteq X[\Omega]\colon \mathit{Prob}\,(\{X \in U\} \cap B) = \mathit{Prob}\,(X \in U)\, \mathit{Prob}\,(B)$$

gilt.

Auch hier notieren wir einige nützliche Folgerungen.

Lemma A.6. *Sei* (Ω, p) *ein diskreter Wahrscheinlichkeitsraum, sei* $X \geq 0$ *eine nicht-negative Zufallsvariable. Seien* $A_1, A_2, \dots, A_n \subseteq \Omega$ *Ereignisse mit* $\mathit{Prob}\,(A_i) > 0$ *für alle* $i \in \{1, 2, \dots, n\}$*. Dann gilt:*

$$(X \text{ und } A_1 \text{ unabhängig}) \Rightarrow E(X \mid A_1) = E(X)$$

$$(A_1, A_2, \dots, A_n \text{ Partition von } \Omega) \Rightarrow E(X) = \sum_{1 \leq i \leq n} \mathit{Prob}\,(A_i) \cdot E(X \mid A_i)$$

$$\left(\bigcup_{1 \leq i \leq n} A_i = \Omega\right) \Rightarrow E(X) \leq \sum_{1 \leq i \leq n} \mathit{Prob}\,(A_i) \cdot E(X \mid A_i)$$

Wir definieren jetzt noch drei im Rahmen dieser Arbeit wichtige Verteilungen und notieren die Erwartungswerte dazu.

Definition A.7. *Sei* (Ω, p) *ein diskreter Wahrscheinlichkeitsraum. Eine Zufallsvariable* $X\colon \Omega \to B$ *mit* $|B| = k \in \mathbb{N}$ *heißt* gleichverteilt*, wenn*

$$\forall b \in B\colon \mathit{Prob}\,(X = b) = \frac{1}{k}$$

gilt.

Sei $n \in \mathbb{N}$, $q \in (0;1)$. *Eine Zufallsvariable* $X\colon \Omega \to \{0, 1, \ldots, n\}$ *heißt* binomialverteilt *mit Parametern* n *und* q, *wenn*

$$\forall i \in \{0, 1, \ldots, n\}\colon \mathit{Prob}\,(X = i) = \binom{n}{i} q^i (1-q)^{n-i}$$

gilt.

Sei $\lambda \in \mathbb{R}^+$. *Eine Zufallsvariable* $X\colon \Omega \to \mathbb{N}_0$ *heißt* poissonverteilt *mit Parameter* λ, *wenn*

$$\forall i \in \mathbb{N}_0\colon \mathit{Prob}\,(X = i) = \frac{\lambda^i}{e^\lambda \cdot i!}$$

gilt.

Satz A.8. *Sei* (Ω, p) *ein diskreter Wahrscheinlichkeitsraum,* X *eine Zufallsvariable. Ist* $X\colon \Omega \to B$ *mit* $|B| = k \in \mathbb{N}$ *gleichverteilt, so gilt für den Erwartungswert von* X

$$E\,(X) = \frac{1}{|B|} \cdot \sum_{b \in B} X(b).$$

Ist $X\colon \Omega \to \{0, 1, \ldots, n\}$ *binomialverteilt mit den Parametern* $n \in \mathbb{N}$ *und* $q \in (0;1)$, *so gilt für den Erwartungswert von* X

$$E\,(X) = n \cdot q.$$

Ist $X\colon \Omega \to \mathbb{N}_0$ *poissonverteilt mit dem Parameter* $\lambda \in \mathbb{R}^+$, *so gilt für den Erwartungswert von* X

$$E\,(X) = \lambda.$$

Beweise zu Satz A.8 findet man beispielsweise bei Feller (1968), Abschnitt IX.3.

Eine (für uns) wichtige Eigenschaft der Poissonverteilung ist, dass sie für große n eine gute Näherung der Binomialverteilung ist (Feller (1968), Abschnitt IV.5).

Satz A.9 (Grenzwertsatz von Poisson). *Seien* $n \in \mathbb{N}$ *und* $p(n) \in (0;1)$ *so gegeben, dass* $\lim\limits_{n\to\infty} n \cdot p(n) = \lambda \in \mathbb{R}^+$ *gilt. Dann gilt*

$$\lim_{n\to\infty} \binom{n}{i} p(n)^i (1-p(n))^{n-i} = \frac{\lambda^i}{e^\lambda \cdot i!}$$

für alle $i \in \{0, 1, \ldots, n\}$.

Wir brauchen an verschiedenen Stellen Abschätzungen für die Wahrscheinlichkeit, dass eine Zufallsvariable X erheblich von ihrem Erwartungswert $\mathrm{E}(X)$ abweicht. Eine einfach zu beweisende Abschätzung, die in vielen Situationen gilt, ist die so genannte Markov-Ungleichung. Man findet einen Beweis etwa bei Motwani und Raghavan (1995), Satz 3.2.

Satz A.10 (Markov-Ungleichung). *Sei* (Ω, p) *ein diskreter Wahrscheinlichkeitsraum, sei* $t \in \mathbb{R}^+$, $X \colon \Omega \to \mathbb{R}_0^+$ *eine nicht-negative Zufallsvariable. Dann gilt*

$$\mathit{Prob}\,(X \geq t \cdot E(X)) \leq \frac{1}{t}.$$

Eine wesentliche schärfere Aussage als die (relativ schwache) Markov-Ungleichung liefern Chernoff-Schranken, die allerdings auch weitere Voraussetzungen machen. Auch hierzu findet man bei Motwani und Raghavan (1995) einen Beweis (Kapitel 4.1).

Satz A.11 (Chernoff-Schranken). *Sei* (Ω, p) *ein diskreter Wahrscheinlichkeitsraum, seien* $X_1, X_2, \ldots, X_n \colon \Omega \to \{0,1\}$ *unabhängige Zufallsvariablen mit* $0 < \mathit{Prob}\,(X_i = 1) < 1$ *für alle* $i \in \{1, 2, \ldots, n\}$. *Sei* $X := \sum\limits_{1\leq i\leq n} X_i$. *Es gilt*

$$\mathit{Prob}\,(X > (1+\delta)E(X)) < \left(\frac{e^\delta}{(1+\delta)^{1+\delta}}\right)^{E(x)}$$

für alle $\delta > 0$ *und*

$$\mathit{Prob}\,(X < (1-\delta)E(X)) < e^{-E(x)\delta^2/2}$$

für alle δ *mit* $0 < \delta \leq 1$.

Ein wichtiges Rechenhilfsmittel ist die Stirlingsche Formel, die wir hier nicht möglichst exakt, sondern als obere und untere Schranke notieren, was für unsere Abschätzungen etwas hilfreicher ist. Einen Beweis findet man zum Beispiel bei Graham, Knuth und Patashnik (1989), Kapitel 9.

Satz A.12. *Für alle $n \in \mathbb{N}$ gilt*

$$\frac{\sqrt{2\pi n}n^n}{e^n} < n! < \frac{\sqrt{3\pi n}n^n}{e^n}$$

Die Stirlingsche Formel ist vor allem zur Abschätzung von Binomialkoeffizienten hilfreich. Wir halten eine einfache Folgerung fest.

Folgerung A.13. *Für alle $n \in \mathbb{N}, k \in \{0, 1, \ldots, n\}$ gilt*

$$\left(\frac{n}{e}\right)^k \leq \binom{n}{k} = \frac{n!}{k! \cdot (n-k)!} \leq \frac{n^k}{k!}.$$

Gelegentlich hilfreich ist der binomische Satz, für den man einen Beweis ebenfalls bei Graham, Knuth und Patashnik (1989) (Kapitel 5.1) findet.

Satz A.14 (Binomischer Satz). *Für alle $n \in \mathbb{N}$, $a, b \in \mathbb{R}$ gilt*

$$\sum_{0 \leq i \leq n} \binom{n}{i} a^i b^{n-i} = (a+b)^n.$$

Für Rechnungen ist es oft hilfreich, einige Abschätzungen zur Hand zu haben. Zum einen ist in dieser Hinsicht die harmonische Summe für uns wichtig. Einen Beweis für die einfache Formulierung im folgenden Satz findet man zum Beispiel bei Graham, Knuth und Patashnik (1989) (Kapitel 6.3).

Satz A.15 (Harmonische Summe). *Für alle $n \in \mathbb{N}$ gilt*

$$\ln n \leq \sum_{1 \leq i \leq n} \frac{1}{i} \leq (\ln n) + 1.$$

Ähnlich entscheidend ist für uns die Konvergenz der Reihe $\sum 1/i^2$, wobei es uns auf das exakte Resultat im Rahmen dieser Arbeit nicht ankommt. Eine einfache Herleitung für die Abschätzung, die man im nächsten Satz findet, geben zum Beispiel Dörfler und Peschek (1998) (Kapitel 11.2, Beispiel 4).

Satz A.16.

$$1 < \sum_{i \geq 1} \frac{1}{i^2} \leq 2$$

Schließlich führen wir noch eine Abschätzung der Ausdrücke $(1 - 1/n)^{n-1}$ und $(1 - 1/n)^n$ auf, die uns an vielen Stellen als Wahrscheinlichkeit dafür, dass (fast) alle Bits in einem Schritt bei Mutationswahrscheinlichkeit $1/n$ nicht mutieren, begegnen (Motwani und Raghavan (1995), Anhang B).

Satz A.17. *Für alle $n \in \mathbb{N}$ gilt*

$$\left(1 - \frac{1}{n}\right)^n < \frac{1}{e} < \left(1 - \frac{1}{n}\right)^{n-1}.$$

Als letztes wichtiges Werkzeug liefern wir hier noch Definitionen, um das asymptotische Verhalten von Funktionen $f\colon \mathbb{N} \to \mathbb{R}_0^+$ zu beschreiben. Wir brauchen für unsere Ergebnisse tatsächlich eine etwas allgemeinere Form für Funktionen $f\colon A \to \mathbb{R}_0^+$, wobei $A \subseteq \mathbb{N}$ eine nicht-endliche Teilmenge von $\mathbb{N}$ ist. Unsere Definitionen unterscheiden sich dadurch nicht wesentlich von den üblichen Definitionen, wie man sie zum Beispiel bei Garey und Johnson (1979) oder Cormen, Leiserson und Rivest (1990) findet.

Definition A.18. *Seien f, g zwei Funktionen $f\colon A \to \mathbb{R}_0^+$ und $g\colon A \to \mathbb{R}_0^+$, wobei $A \subseteq \mathbb{N}$ nicht-endliche Kardinalität hat. Wir schreiben*

$$f(n) = O\left(g(n)\right),$$

wenn es eine Konstante $n_0 \in A$ und eine Konstante $c \in \mathbb{R}^+$ gibt, so dass für alle $n \in A$ mit $n > n_0$ stets $f(n) \leq c \cdot g(n)$ gilt. Wir sagen, f wächst höchstens so schnell wie *g.*

Wir schreiben

$$f(n) = \Omega\left(g(n)\right),$$

wenn es eine Konstante $n_0 \in A$ und eine Konstante $c \in \mathbb{R}^+$ gibt, so dass für alle $n \in A$ mit $n > n_0$ stets $f(n) \geq c \cdot g(n)$ gilt. Wir sagen, f wächst mindestens so schnell wie *g.*

Wir schreiben

$$f(n) = \Theta\left(g(n)\right),$$

wenn $f = O\left(g(n)\right)$ und $f = \Omega\left(g(n)\right)$ gilt. Wir sagen, f wächst ebenso schnell wie *g.*

Wir schreiben

$$f(n) = o\left(g(n)\right),$$

wenn $\lim\limits_{\substack{n\to\infty \\ n\in A}} f(n)/g(n) = 0$ *gilt. Wir sagen,* f wächst langsamer als g.

Wir schreiben

$$f(n) = \omega\left(g(n)\right),$$

wenn $g(n) = o\left(f(n)\right)$ *gilt. Wir sagen,* f wächst schneller als g.

Wir lassen auch eine etwas ungenaue Schreibweise wie $O(n) = O(n^2)$ zu und meinen damit, dass für alle Funktionen f, für die $f(n) = O(n)$ gilt, auch $f(n) = O(n^2)$ gilt. Die Notation aus Definition A.18 ist so üblich, die Verwendung des Gleichheitszeichens ist allerdings etwas unglücklich. So beschreibt in diesem Kontext das Symbol „=" im allgemeinen keine symmetrische Relation: natürlich folgt aus $n = O(n^2)$ nicht $O(n^2) = n$ und aus $O(n) = O(n^2)$ auch nicht $O(n^2) = O(n)$. Etwas stringenter ist es, von Funktionenmengen zu sprechen, also etwa $O(g(n))$ als Menge aller Funktionen f aufzufassen, so dass $f = O(g(n))$ im Sinne von Definition A.18 gilt. Dann sollte man natürlich besser $f \in O(g(n))$ und $O(n) \subseteq O(n^2)$ schreiben. Wir bleiben hier allerdings bei der üblichen Schreibweise.

B Übersicht der betrachteten Beispielfunktionen

Wir haben an verschiedenen Stellen erwähnt und erläutert, dass die im Rahmen dieser Arbeit verwendeten Funktionen in gewisser Weise alle künstlich, aber nicht pathologisch sind. Sie haben uns geholfen, den Aspekt des gerade untersuchten evolutionären Algorithmus, auf den es uns ankam, besonders deutlich hervorzuheben, ihn so erkennbar, analysierbar und verständlich zu machen. In diesem Sinne haben unsere Beispielfunktionen alle pädagogische Funktion und wir nehmen an, dass sie auch in Zukunft bei der Untersuchung evolutionärer Algorithmen eine hilfreiche Rolle spielen können. Darum stellen wir noch einmal kurz und knapp die verwendeten Funktionen zusammen und geben ihre wesentlichen Eigenschaften an.

f_1: Definition 3.2 (Seite 47), Abbildung 1 (Seite 48). Einfache symmetrische, lineare Funktion, gut analysierbar für viele Algorithmen.

f_{B}: Definition 3.6 (Seite 52). Lineare Funktion, Extremgegenpunkt zu f_1 in Bezug auf Gewichte, insofern wichtige lineare Funktion.

f_{R}: Definition 2.12 (Seite 37). Strukturell einfache Funktion, Gegenbeispiel für Fitness Distance Correlation, gut optimierbares Gegenbeispiel für durchschnittsbasierte Maße, gut analysierbar für viele evolutionäre Algorithmen.

f_{L}: Definition 5.6 (Seite 81). Nicht-lineare, unimodale Funktion, gut analysierbar für viele Algorithmen, mit bitweiser Mutation mit Mutationswahrscheinlichkeit $p(n) = 1/n$ optimierbar in $\Theta\left(n^2\right)$ Schritten mit überwältigender Wahrscheinlichkeit.

$f_{\mathrm{P},k}$: Definition 5.11 (Seite 87). Schwierige unimodale Funktion, unimodale Einbettung des langen k-Pfades (Definition 5.8).

$f_{\mathrm{N},a}$: Definition 2.14 (Seite 39). Strukturell sehr einfache, sehr schwierige Funktion, gut analysierbar für viele Algorithmen.

$f_{\mathrm{C},a}$: Definition 2.9 (Seite 34). Ähnlich wie $f_{\mathrm{N},a}$ aber mit 2 „Nadeln“, zusammen mit $\overline{f_{\mathrm{C},a}}$ als Gegenbeispiel für Epistasie benutzt.

$\overline{f_{\mathrm{C},a}}$: Definition 2.10 (Seite 35). „Gegenfunktion“ zu $f_{\mathrm{C},a}$, sehr einfach optimierbar, zusammen mit $f_{\mathrm{C},a}$ als Gegenbeispiel für Epistasie benutzt.

$\boldsymbol{f_{\mathrm{T}}}$**:** Definition 6.2 (Seite 94). Strukturell einfache, sehr schwierige Funktion, wie f_1, aber mit globalem Maximum bei $\mathbf{0}$, sehr irreführend (deceptive), gut analysierbar für viele Algorithmen.

$\boldsymbol{f_{\mathrm{Q}}}$**:** Definition 6.4 (Seite 97), Abbildung 3 (Seite 97). Für den (1+1) EA mit $p(n) = 1/n$ schwierigste Funktion bezogen auf die erwartete Laufzeit, im wesentlichen Quadrat von f_1, im Grunde aber auch für den (1+1) EA mit $p(n) = 1/n$ leicht optimierbar.

$\boldsymbol{f_{\mathrm{J},m}}$**:** Definition 9.3 (Seite 185), Abbildungen 6–9 (Seiten 186–189). Strukturell einfache, symmetrische Funktion, Kombination von f_1 und $-f_1$ mit skalierbarer Sprungstelle, einstellbare Schwierigkeit.

$\boldsymbol{f_{\mathrm{M}}}$**:** Definition 7.8 (Seite 113). Verbindung von Ideen aus f_1, f_{R} und $f_{\mathrm{J},m}$, überwiegend von technischem Interesse in Kapitel 7.

$\boldsymbol{f_{\mathrm{F}}}$**:** Definition 7.20 (Seite 139). Verbindung von Ideen aus f_1, f_{R}, $f_{\mathrm{J},m}$, f_{M} und $f_{\mathrm{P},k}$, überwiegend von technischem Interesse in Kapitel 7.

$\boldsymbol{f_{\mathrm{H}}}$**:** Definition 8.9 (Seite 159), Abbildung 4 (Seite 159). Modifikation von $f_{\mathrm{J},m}$ mit für Algorithmus 8.4 geeignet veränderter „Sprunggröße“. Nur für Kapitel 8 von Interesse.

$\boldsymbol{f_{\mathrm{K}}}$**:** Definition 8.13 (Seite 169), Abbildung 5 (Seite 170). Im wesentlichen Modifikation von f_{Q}, für Algorithmus 8.4 geeignet veränderte Funktionswerte. Nur für Kapitel 8 von Interesse.

$\boldsymbol{f_{\mathrm{S}}}$**:** Definition 10.1 (Seite 209), Abbildung 10 (Seite 210). Symmetrische, schwierig zu optimierende Funktion, gut analysierbar für viele Algorithmen, Funktionsfassung der SAT-Instanz von Papadimitriou (1994).

Literatur

L. Altenberg (1997). NK fitness landscapes. In *Handbook of Evolutionary Computation*, B2.7.2. Oxford University Press, Oxford.

S. Arora, Y. Rabani und U. Vazirani (1994). Simulating quadratic dynamical systems is PSPACE-complete. In *Proceedings of the 26th Annual ACM Symposium on Theory of Computing* (*STOC '94*), 459–467.

S. Aurora, C. Lund, R. Motwani, M. Sudan und M. Szegedy (1992). Proof verification and hardness of approximation results. In *Proceedings of the 33rd Annual IEEE Symposium on Foundations of Computer Science* (*FOCS '92*), 14–23.

T. Bäck (1993). Optimal mutation rates in genetic search. In *Proceedings of the Fifth International Conference on Genetic Algorithms* (*ICGA '93*), 2–8. Morgan Kaufmann, San Mateo, CA.

T. Bäck (1996). *Evolutionary Algorithms in Theory and Practice*. Oxford University Press, New York, NY.

T. Bäck (1998). An overview of parameter control methods by self-adaptation in evolutionary algorithms. *Fundamenta Informaticae 35*, 51–66.

R. E. Bryant (1986). Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers C-35*, 677–693.

S. A. Cook (1971). The complexity of theorem-proving procedures. In *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing* (*STOC '71*), 151–158.

T. H. Cormen, C. E. Leiserson und R. L. Rivest (1990). *Introduction to Algorithms*. McGraw-Hill, New York, NY.

Y. Davidor (1991). Epistasis variance: A viewpoint on GA-hardness. In G. J. E. Rawlins (Hrsg.), *Proceedings of the 1st Workship on Foundations of genetic algorithms* (*FOGA '91*), 23–35. Morgan Kaufman, San Mateo, CA.

W. Dörfler und W. Peschek (1998). *Einführung in die Mathematik für Informatiker*. Hanser, München.

S. Droste, T. Jansen und I. Wegener (1998a). On the analysis of the (1+1) evolutionary algorithm. Reihe CI 21/98, SFB 531, Universität Dortmund, Dortmund. eingereicht: Theoretical Computer Science.

S. Droste, T. Jansen und I. Wegener (1998b). On the optimization of unimodal functions with the (1+1) evolutionary algorithm. In A. E. Eiben, T. Bäck, M. Schoenauer und H.-P. Schwefel (Hrsg.), *Proceedings of the 5th*

Parallel Problem Solving from Nature (*PPSN V*), Band 1498 der Reihe *Lecture Notes in Computer Science*, 47–56. Springer, Berlin.

S. Droste, T. Jansen und I. Wegener (1998c). A rigorous complexity analysis of the (1+1) evolutionary algorithm for linear functions with Boolean inputs. In D. B. Fogel, H.-P. Schwefel, T. Bäck und X. Yao (Hrsg.), *Proceedings of the IEEE International Conference on Evolutionary Computation* (*ICEC '98*), 499–504. IEEE Press.

S. Droste, T. Jansen und I. Wegener (1998d). A rigorous complexity analysis of the (1+1) evolutionary algorithm for separable functions with Boolean inputs. *Evolutionary Computation 6*(2), 185–196.

S. Droste, T. Jansen und I. Wegener (1999). Perhaps not a free lunch but at least a free appetizer. In W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela und R. E. Smith (Hrsg.), *Proceedings of the First Genetic and Evolutionary Computation Conference* (*GECCO '99*), 833–839. Morgan Kaufmann, San Francisco, CA.

S. Droste, T. Jansen und I. Wegener (2000a). Dynamic parameter control in simple evolutionary algorithms (extended abstract). akzeptiert für: Foundations of Genetic Algorithms (FOGA 2000).

S. Droste, T. Jansen und I. Wegener (2000b). A natural and simple function which is hard for all evolutionary algorithms. akzeptiert für: Third Asia-Pacific Conference on Simulated Evolution And Learning (SEAL 2000).

W. Feller (1968). *An Introduction to Probability Theory and Its Applications. Volume I.* Wiley, New York, NY.

W. Feller (1971). *An Introduction to Probability Theory and Its Applications. Volume II.* Wiley, New York, NY.

D. B. Fogel (1995). *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence.* IEEE Press, Piscataway, NJ.

L. J. Fogel (1999). *Intelligence Through Simulated Evolution: Forty Years of Evolutionary Programming.* Wiley, New York, NY.

C. Fonlupt, D. Robilliard und P. Preux (1998). A bit-wise epistasis measure for binary search space. In A. E. Eiben, T. Bäck, M. Schoenauer und H.-P. Schwefel (Hrsg.), *Proceedings of the 5th Parallel Problem Solving from Nature* (*PPSN V*), Band 1498 der Reihe *Lecture Notes in Computer Science*, 47–56. Springer, Berlin.

M. R. Garey und D. S. Johnson (1979). *Computers and Intractability. A Guide to the Theory of NP-Completeness.* W. H. Freeman Company, New York, NY.

J. Garnier und L. Kallel (2000). erscheint in L. Kallel, B. Naudts und A. Rogers: Theoretical Aspects of Natural Computing, in der Serie Natural Computing bei Springer, Berlin.

J. Garnier, L. Kallel und M. Schoenauer (1999). Rigorous hitting times for binary mutations. *Evolutionary Computation 7*(2), 173–203.

H. Geiringer (1944). On the probability theory of linkage in Mendelian herity. *Annals of Mathematical Statistics 15*, 25–57.

D. E. Goldberg (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addision-Wesley, Reading, MA.

R. L. Graham, D. E. Knuth und O. Patashnik (1989). *Concrete Mathematics*. Addison-Wesley, Reading, MA.

J. Håstad (1997). Some optimal inapproximability results. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing* (*STOC '97*), 1–10.

J. H. Holland (1992). *Adaption in Natural and Artificial Systems*. MIT Press, Cambridge, MA.

J. Horn, D. E. Goldberg und K. Deb (1994). Long path problems. In Y. Davidor, H.-P. Schwefel und R. Männer (Hrsg.), *Proceedings of the 3rd Parallel Problem Solving From Nature* (*PPSN III*), Band 866 der Reihe *Lecture Notes in Computer Science*, 149–158. Springer, Berlin.

T. Jansen (1998). Introduction to the theory of complexity and approximation algorithms. In E. W. Mayr, H. J. Prömel und A. Steger (Hrsg.), *Lectures on Proof Verification and Approximation Algorithms*, Band 1367 der Reihe *Lecture Notes in Computer Science*, 5–28. Springer, Berlin.

T. Jansen (2000). On classifications of fitness functions. In L. Kallel, B. Naudts und A. Rogers (Hrsg.), *Theoretical Aspects of Evolutionary Computing*, Natural Computing, 377–390. Springer, Berlin.

T. Jansen und I. Wegener (1999). On the analysis of evolutionary algorithms — a proof that crossover really can help. In J. Nešetřil (Hrsg.), *Proceedings of the 7th Annual European Symposium on Algorithms* (*ESA '99*), Band 1643 der Reihe *Lecture Notes in Computer Science*, 184–193. Springer, Berlin.

T. Jansen und I. Wegener (2000a). Evolutionary algorithms — how to cope with plateaus of constant fitness and when to reject strings of the same fitness. eingereicht: IEEE Transactions on Evolutionary Computation.

T. Jansen und I. Wegener (2000b). On the choice of the mutation probability for the (1+1) EA. In M. Schoenauer, K. Deb, G. Rudolph, X. Yao,

E. Lutton, J. J. Merelo und H.-P. Schwefel (Hrsg.), *Proceedings of the 6th Parallel Problem Solving from Nature* (*PPSN VI*), Band 1917 der Reihe *Lecture Notes in Computer Science*, 89–98. Springer, Berlin.

M. Jerrum und A. Sinclair (1997). The Markov chain Monte Carlo method: an approach to approximate counting and integration. In D. S. Hochbaum (Hrsg.), *Approximation Algorithms for NP-hard Problems*, 482–520. PWS Publishers, Boston, MA.

M. Jerrum und G. B. Sorkin (1998). The Metropolis algorithm for graph bisection. *Discrete Applied Mathematics 82*, 155–175.

T. Jones (1995). *Evolutionary Algorithms, Fitness Landscapes and Search.* Ph. D. thesis, University of New Mexico, Alburquerque, NM.

T. Jones und S. Forrest (1995). Fitness distance correlation as a measure of problem difficulty for genetic algorithms. In L. J. Eshelman (Hrsg.), *Proceedings of the 6th International Conference on Genetic Algorithms* (*ICGA '95*), 184–192. Morgan Kaufman, San Mateo, CA.

S. A. Kauffman (1993). *The Origins of Order: Self-Organization and Selection in Evolution.* Oxford University Press, Oxford.

S. Kirkpatrick, C. D. Gelatt und M. P. Vecchi (1983). Optimization by simulated annealing. *Science 220*, 671–680.

R. Menke (1998). unveröffentlicht.

N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller und E. Teller (1953). Equation of state calculation by fast computing machines. *Journal of Chemical Physics 21*, 1087–1092.

R. Motwani und P. Raghavan (1995). *Randomized Algorithms.* Cambridge University Press, Cambridge.

H. Mühlenbein (1992). How genetic algorithms really work. Mutation and hillclimbing. In R. Männer und R. Manderick (Hrsg.), *Proceedings of the 2nd Parallel Problem Solving from Nature* (*PPSN II*), 15–25. North-Holland, Amsterdam, Niederlande.

H. Mühlenbein (1998). The equation for the response to selection and its use for prediction. *Evolutionary Computation 5*(3), 303–346.

B. Naudts und L. Kallel (1999). Comparison of summary statistics of fitness landscapes. erscheint in der Zeitschrift IEEE Transactions on Evolutionary Computation.

B. Naudts, D. Suys und A. Verschoren (1997). Epistasis as a basic concept in formal landscape analysis. In T. Bäck (Hrsg.), *Proceedings of the*

7th International Conference on Genetic Algorithms (*ICGA '97*), 65–72. Morgan Kaufman, San Francisco, CA.

C. H. Papadimitriou (1994). *Computational Complexity.* Addison-Wesley, Reading, MA.

C. H. Papadimitriou, A. A. Schäffer und M. Yannakakis (1990). On the complexity of local search (extended abstract). In *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing* (*STOC '90*), 438–445.

A. Prügel-Bennet und J. L. Shapiro (1997). The dynamics of a genetic algorithm for simple ising systems. *Physica D 104*, 75 – 114.

R. J. Quick, V. J. Rayward-Smith und G. D. Smith (1998). Fitness distance correlation and ridge functions. In A. E. Eiben, T. Bäck, M. Schoenauer und H.-P. Schwefel (Hrsg.), *Proceedings of the 5th Parallel Problem Solving From Nature* (*PPSN V*), Band 1498 der Reihe *Lecture Notes in Computer Science*, 77–86. Springer, Berlin.

Y. Rabani, Y. Rabinovich und A. Sinclair (1998). A computational view of population genetics. *Random Structures and Algorithms 12*(4), 314–334.

M. Rattray und J. L. Shapiro (1996). The dynamics of a genetic algorithm for a simple learning problem. *Journal of Physics A 29*, 7451–7453.

I. Rechenberg (1994). *Evolutionsstrategie '94.* Frommann-Holzboog, Stuttgart.

S. Ronald (1998). Duplicate genotypes in a genetic algorithm. In D. B. Fogel, H.-P. Schwefel, T. Bäck und X. Yao (Hrsg.), *Proceedings of the IEEE International Conference on Evolutionary Computation* (*ICEC '98*), 793–798. IEEE Press.

G. Rudolph (1997a). *Convergence Properties of Evolutionary Algorithms.* Verlag Dr. Kovač, Hamburg.

G. Rudolph (1997b). How mutation and selection solve long path-problems in polynomial expected time. *Evolutionary Computation 4*(2), 195–205.

G. Rudolph (1999). Self-adaptation and global convergence: A counter-example. In *Proceedings of the Congress on Evolutionary Computation* (*CEC '99*), 646–651. IEEE Press.

R. Salomon (1996). Re-evaluating genetic algorithm performance under coordinate rotation of benchmark functions. A survey of some theoretical and practical aspects of genetic algorithms. *BioSystems 36*, 263–278.

U. Schöning (1999). A probabilistic algorithm for k-SAT and constraint satisfaction problems. In *Proceddings of the 40th Annual IEEE Symposium* *on Foundations of Computer Science* (*FOCS '99*), 410–414. IEEE Press, Piscataway, NJ.

H.-P. Schwefel (1995). *Evolution and Optimum Seeking*. Wiley, New-York, NY.

A. Sinclair (1993). *Algorithms for Random Generation and Counting: A Markov Chain Approach*. Birkhäuser, Boston.

G. B. Sorkin (1991). Efficient simulated annealing on fractal energy landscapes. *Algorithmica 6*, 367–418.

G. Tao und Z. Michalewicz (1998). Inver-over operator for the TSP. In A. E. Eiben, T. Bäck, M. Schoenauer und H.-P. Schwefel (Hrsg.), *Proceedings of the 5th Parallel Problem Solving from Nature* (*PPSN V*), Band 1498 der Reihe *Lecture Notes in Computer Science*, 803–812. Springer, Berlin.

R. K. Thompson und A. H. Wright (1996). Additively decomposable fitness functions. Technical report, University of Montana, Computer Science Department.

I. Wegener (2000). *Branching Programs and Binary Decision Diagrams — Theory and Applications*. SIAM. Monographs in Discrete Mathematics and Applications.

E. D. Weinberger (1996). NP completeness of Kauffman's *n-k* model, a tuneably rugged fitness landscape. Technical Report SFI-TR-96-02-003, The Sante Fe Institute, Santa Fe, NM.

D. H. Wolpert und W. G. Macready (1997). No free lunch theorem for optimization. *IEEE Transactions on Evolutionary Computation 1*(1), 67–82.