Populationsbasierte Optimierung durch das Lernen von Interaktionen mit Bayes'schen Netzen

Dissertation
zur Erlangung des Doktorgrades (Dr.rer.nat.)
der Mathematisch-Naturwissenschaftlichen Fakultät
der Rheinischen Friedrich-Wilhems-Universität Bonn

vorgelegt von
Thilo Mahnig
aus
Minneapolis

Bonn, März 2001

Angefertigt mit Genehmigung der Mathematisch-Naturwissenschaftlichen Fakultät der Rheinischen Friedrich-Wilhelms-Universität Bonn

1. Referent: Prof. Joachim Buhmann
2. Referent: Prof. Michael Clausen
Tag der Promotion: 10. April 2001

Ich widme diese Arbeit meiner Frau Andrea.

Inhaltsverzeichnis

Abbildungsverzeichnis

Algorithmenverzeichnis

Danksagung

Hiermit möchte ich zunächst Dr. Heinz Mühlenbein für die vielfältige Unterstützung bei der Anfertigung dieser Arbeit danken. Lieber Heinz, vielen Dank nicht nur für die wissenschaftliche Anleitung und die unzähligen Ideen, sondern auch für die Umstände und die freundschaftliche Atmosphäre, unter denen ich diese Arbeit erstellen konnte!

Weiterhin gilt mein Dank Prof. Buhmann als Betreuer an der Universität Bonn. Lieber Prof. Buhmann, ich bedanke mich herzlichst für die Gespräche und die zahlreichen Anregungen, die in diese Arbeit eingeflossen sind sowie die Erweiterung meines Horizonts in der *Reading Group*.

Dank gebührt weiterhin Prof. Clausen, der sich bereiterklärte, das Zweitgutachten zu erstellen, meinem Vater für das Korrekturlesen der Arbeit und meinen Eltern und Schwiegereltern für ihren Zuspruch.

Am Meisten bedanke ich mich aber bei Andrea für ihre moralische Unterstützung und für tausend andere Dinge …

1 Einführung

Die zugrunde liegende Aufgabenstellung in dieser Arbeit ist die Optimierung einer (reellwertigen) Funktion mit diskretem Wertebereich.

Aufgabe: *Gegeben sei die Funktion*

$$f(x) : \mathcal{R} \to \mathbb{R} \tag{1.1}$$

für diskretes $\mathcal{R} = \prod_{i=1}^{n} \mathcal{R}_i$ *und* $\mathcal{R}_i = \{0, \ldots, l_i\}$, *also* $x = (x_1, \ldots, x_n)$ *mit* $x_i \in \mathcal{R}_i$. *Finde ein* $x_{\max} \in \mathcal{R}$ *mit*

$$f(x_{\max}) = \max_{x \in \mathcal{R}} f(x) =: f_{\max} \tag{1.2}$$

Das Maximum muss nicht eindeutig sein. Die zu optimierende Funktion heißt auch ***Fitness-Funktion****.*

Ohne weitere Annahmen an die Funktion f erfordert die Suche nach dem Maximum für binäre Variablen einen Aufwand von 2^n Schritten. Daher begnügt man sich in der Praxis mit Heuristiken, die das Maximum möglichst gut approximieren.

In dieser Arbeit werden populationsbasierte Heuristiken untersucht. Eines der bekanntesten Beispiele sind die genetischen Algorithmen. Genetische Algorithmen haben sich zwar in der Praxis in vielen Fällen als Optimierungsverfahren bewährt. Durch ihren hohen stochastischen Anteil gibt es aber wenig Resultate, die verlässliche Aussagen über das Verhalten ermöglichen.

In den Anfängen wurden diese Algorithmen manchmal als „optimale" Suchalgorithmen angesehen, und das Schema-Theorem von Holland (1975) schien zu beweisen, dass genetische Algorithmen parallel überdurchschnittliche „Schemata" (bestimmte Teilräume des Suchraumes) durchsuchen.

Mit der Entdeckung der sogenannten *Deceptive*-Funktionen (Goldberg, 1989a) stellte man aber fest, dass genetische Algorithmen „getäuscht" werden können und zu einem *lokalen* Optimum gedrängt werden. Zur Analyse hat Goldberg die Walsh-Transformationen von Bethke (1981) verwendet. Diese erlaubt es aber nur, eine Aussage über das Verhalten einer gleichverteilten Population zu machen. Mit dem „No-Free-Lunch"-Theorem (Wolpert and Macready, 1997) musste die Hoffnung auf einen optimalen Optimierungsalgorithmus endgültig begraben werden. Es kommt daher darauf an, für einen Algorithmus den Teil des Suchraumes zu finden, in dem er gute Resultate liefert.

Die zentrale Idee der in dieser Arbeit vorgeschlagenen Algorithmen ist es, die Population durch eine *Verteilung* zu beschreiben. Diese Verteilung wird dabei durch eine polynomielle Zahl an Parametern dargestellt. Es entsteht dabei ein iterativer Prozess, der folgendermassen vorgeht: Zunächst werden zufällig Punkte generiert. Dann wird anhand der zu optimierenden Funktion davon eine Teilmenge selektiert. Weiterhin wird die Verteilung der selektierten Punkte geschätzt und durch eine möglichst kleine Zahl an Parametern möglichst gut approximiert. Im letzten Schritt werden neue Punkte aus dieser Verteilung gezogen und zu Schritt 2 gesprungen.

Mühlenbein hat 1996 den *UMDA* vorgeschlagen (Mühlenbein and Paaß, 1996) als Approximation eines einfachen genetischen Algorithmus. Dieses Verfahren hat den Vorteil, dass es mathematisch einfacher zu analysieren ist. In den folgenden Jahren haben Mühlenbein und ich in mehreren Publikationen diesen Algorithmus analysiert. Dabei zeigte sich, dass die durchschnittliche Fitness, berechnet über die univariaten Randverteilungen der Population, das Verhalten des *UMDA* (und somit eines einfachen genetischen Algorithmus) vollständig beschreiben. Diese Art der Analyse fand sich bisher nicht in der Literatur der genetischen Algorithmen. Sie transformiert das *diskrete* Optimierungsproblem in ein *kontinuierliches* Problem, parametrisiert über die Randverteilungen.

In dieser Arbeit zeige ich neben einer Verallgemeinerung der Resultate über den binären Fall hinaus auch einen Satz, der eine Korrespondenz zwischen den Maxima dieser beiden Optimierungsprobleme aufzeigt. Die Lösungen, die man durch Optimierung des kontinuierlichen Problems gewinnt, lassen sich alle auf Maxima im Ursprungsraum abbilden.

Dem *UMDA* liegt eine bestimmte Verteilung zugrunde, nämlich das Produkt der univariaten Randverteilungen. Aus der Struktur dieser Verteilung kann man direkt die Schwäche des Algorithmus ablesen, nämlich das Fehlen jeglicher Interaktion zwischen den Parametern. Er ist aber auch Repräsentant der oben beschriebenen Klasse von Algorithmen, die Populationen durch Verteilungen beschreiben und neue Individuen direkt aus diesen Verteilungen generieren. Es zeigt sich, dass die Boltzmann- (bzw. Gibbs)-Verteilung dafür geeignet ist. Mit unendlicher Populationsgröße konvergiert das Verfahren theoretisch gegen die Gleichverteilung auf den Maxima.

Parallel zu den Arbeiten am *UMDA* entstanden auch mehrere Publikationen, die den *FDA (Factorized Distribution Algorithm)* einführten und analysierten. Die entscheidende Idee beim *FDA* war es, die Boltzmann-Verteilung zu faktorisieren, um eine polynomielle Anzahl an Parametern zu behalten. Die Faktorisierung kann man dabei aus einer additiven Zerlegung der zu optimierenden Funktion berechnen. Diese Art der Optimierung wurde in (Mühlenbein et al., 1999) vorgeschlagen und hat ein breites Feld eröffnet, die Optimierung durch Schätzen von Verteilungen.

Für die Konvergenz ist es wesentlich, dass man ein Selektionsverfahren verwendet, mit dem die Population für die Dauer des Verfahrens eine Boltzmann-Verteilung bleibt. Dafür lässt sich Boltzmann-Selektion verwenden. Da Boltzmann-Selektion aber einen Parameter benötigt (ein Abkühlungsverfahren), der nicht einfach zu bestimmen war, wurde in bisherigen Anwendungen des *FDA Truncation Selection* benutzt. *Truncation Selection* ist ein aus der Züchtung bekanntes robustes Optimierungsverfahren, das insbesondere auch invariant ist unter monotonen linearen Transformationen der zu optimierenden Funktion.

In dieser Arbeit ermöglicht eine hier erstmals vorgestellte Reihenentwicklung der durchschnittlichen Fitness als Momente der Fitness-Verteilung ein neues adaptives Abkühlungsverfahren, welches als Imitation von *Truncation Selection* konzipiert wurde und ebenfalls auf einen Optimierungsalgorithmus führt, der invariant unter linearen monotonen Transformationen der Fitness-Funktion ist. Dieses Abkühlungsverfahren macht die Nutzung der Boltzmann-Selektion überhaupt erst praktikabel.

Wenn die Zerlegung der Funktion nicht mehr bekannt ist, muss sie aus den Daten gelernt werden. Dazu wird die Theorie der graphischen Modelle benötigt, speziell die der Bayes'schen Netzwerke. Die Sätze dazu finden sich alle in der Literatur (Lauritzen, 1996), die für diese Theorie nötigen Beweise wurden aber in dieser Arbeit noch einmal geführt, weil sie für diesen Spezialfall recht kurz sind und ein besseres Verständnis für die Zusammenhänge ermöglichen.

Es gab in der Literatur nach der Vorstellung des *FDA* mehrere Ansätze, diesen mit einem Lernverfahren zu verbinden (Pelikan et al., 1999; Ortiz et al., 1999). In dieser Arbeit schlage ich mit Hilfe der Informationstheorie die Brücke zwischen den durch die Bayes'schen Netze beschriebenen Faktorisierungen und den Faktorisierungen, die sich aus der additiven Zerlegung der Fitness-Funktion ergeben. Wieder gibt es eigentlich nur dann ein theoretisches Fundament, wenn man im Raum der Boltzmann-Verteilungen ist, also mit Boltzmann-Selektion arbeitet.

Der so gewonnene Algorithmus *LFDA (Learning Factorized Distribution Algorithm)* wird danach in dieser Arbeit erstmals auf ein schwieriges Optimierungsproblem angesetzt, die Graph Bipartitionierung. Wegen der Nebenbedingungen kann der Algorithmus nicht direkt verwendet werden, sondern als Meta-Heuristik, bei der die Lösungen noch mit einem Hillclimber gültig gemacht und optimiert werden. Sowohl der *UMDA* als auch der *LFDA* zeigen an diesem Problem, dass sie mit neuen Verfahren aus diesem Gebiet konkurrieren können.

Struktur der Arbeit

Die Arbeit ist wie folgt gegliedert. In Kapitel 2 wird kurz der einfache genetische Algorithmus mit den wesentlichen Bestandteilen Selektion, Rekombination und Mutation vorgestellt. In Kapitel 3 wird der *UMDA* definiert und analysiert. Danach werden Zerlegungssätze für die Fitness-Funktion gezeigt, die es erlauben, die durchschnittliche Fitness von Population effizient zu berechnen. Damit lässt sich das Verhalten des *UMDA* beschreiben über die durchschnittliche Fitness, parametrisiert nach den univariaten Randverteilungen der Population.

Es zeigt sich ferner, dass man die Mutation in der Theorie der genetischen Algorithmen abbilden kann auf die Berechnung der Wahrscheinlichkeiten mit Hilfe der Bayes'schen Regel und einer A-Priori-Verteilung, deren Parameter der Mutationsrate entspricht. Der so modifizierte *UMDA* ist weniger anfällig gegen Fixation eines Parameters und daher robuster.

In Kapitel 4 wird erläutert, wieso die Boltzmann (bzw. Gibbs)-Verteilung eine geeignete Suchverteilung ist, wenn man Optimierung über allgemeine Verteilungen machen will. Dann wird der *FDA (Factorized Distribution Algorithm)* vorgestellt, der eine Faktorisierung der Verteilung benutzt. Neben einem Fokussierungssatz für unendliche Populationsgröße folgt in diesem Kapitel die Reihenentwicklung der durchschnittlichen Fitness der Boltzmann-Verteilung und darauf aufbauend das adaptive Abkühlungsverfahren.

In Kapitel 5 wird in die Theorie der Bayes'schen Netzwerke eingeführt. Ein Abschnitt stellt dabei Theorie des Lernens des Modells vor und vergleicht zwei gängige Maße für die Güte eines Netzwerkes bei gegebenen Daten.

Der *FDA* zusammen mit der Theorie der graphischen Modelle liefert den *LFDA (Learning Factorized Distribution Algorithm)*, der in Kapitel 6 vorgestellt wird. Hier wird in jeder Generation die Verteilung aus den selektierten Punkten gemäß der Theorie aus Kapitel 5 geschätzt. In Testläufen mit einigen für genetische Algorithmen sehr schwierigen Problemen stellt der *LFDA* seine Leistungsfähigkeit unter Beweis.

In Kapitel 7 wird auf die Implementierung der Algorithmen eingegangen und die nummerische Komplexität von *UMDA*, *FDA* und *LFDA* behandelt. Die Skalierung hängt von einer Reihe von Stellparametern ab. In Kapitel 8 werden daher empirisch gewonnene Skalierungsergebnisse vorgestellt.

In Kapitel 9 erfolgt als Anwendung die Graph Bipartitionierung. Dieses Problem ist NP-vollständig, polynomielle Approximationssalgorithmen existieren nur für dichte Probleme. Zur Optimierung wurde der *LFDA* zusammen mit einem modifizierten Kernighan-Lin-Verfahren zur lokalen Optimierung eingesetzt. In den Benchmark-Problemen (die geringe Dichte haben) zeigt sich die Stärke dieses Verfahrens im Vergleich mit Ergebnissen anderer Autoren.

In Kapitel 10 werden schließlich noch zwei andere Optimierungsansätze kurz vorgestellt. Zum einen lassen sich die Parameter der einfachen Produktverteilung des *UMDA* auch über die Minimierung der Kullback-Leibler-Divergenz bestimmen, und es ergeben sich die *Mean-Field*-Gleichungen. Eine ganz andere Möglichkeit ist das dynamische Programmieren. Es zeigt sich hier, dass man genau in den Fällen, in denen eine exakte Faktorisierung möglich ist, auch mit Hilfe von dynamischem Programmieren in polynomiellem Aufwand das globale Optimum finden kann.

Im Anhang folgt ein Abschnitt, in dem neben einem Symbolverzeichnis in der Arbeit gebrauchte Definitionen vorgestellt werden.

Die beiden zentralen Algorithmen, der *UMDA* mit Mutation sowie der *LFDA*, sind zwei Basisalgorithmen, die grundsätzlich für die hier behandelte Klasse von Optimierungsproblemen benutzt werden können. Der *UMDA* ist dabei einfach zu implementieren, hat relativ wenig Stellparameter und kann auch multi-modale Funktionen optimieren. Sein Verhalten ist durch die hier gezeigte Analyse der Fitness-Landschaft weitgehend verstanden. Wenn Interaktionen zwischen den Variablen aber das Finden einer guten Lösung verhindern, kann der *LFDA* eingesetzt werden. Er ist sowohl in der Implementierung als auch in der Laufzeit deutlich komplexer als der *UMDA*.

2 Genetische Algorithmen

Als Startpunkt der vorliegenden Arbeit dienen die genetischen Algorithmen. In diesem Kapitel steht daher eine kurze Einführung, ohne auf die vielen Varianten, die im Laufe der Zeit entstanden sind, einzugehen.

Genetische Algorithmen wurden von Holland in den 60er Jahren erfunden und in seinem Buch (Holland, 1975) genauer beschrieben. Seitdem haben sie eine grosse Verbreitung erfahren. Ihre Popularität verdanken sie neben der Einfachheit des Algorithmus (und damit der Implementierung) der Tatsache, dass sie in vielen Fällen erfolgreich eingesetzt wurden.

Sie sind ursprünglich inspiriert durch die biologische Evolution und nutzen deren Prinzipien, nämlich *Selektion*, *Rekombination* und *Mutation*, um Optimierungsaufgaben zu lösen. Mühlenbein (1998) hat allerdings darauf hingewiesen, dass sich die Metaphern (und Formeln) der *Züchtung* viel besser eignen, um eine Optimierung einer gegebenen Fitness-Funktion zu erreichen. So haben die Züchter unter Verwendung spezieller statistischer Verfahren den Milchertrag von Kühen um mehr als 50% gesteigert (Falconer, 1981).

2.1 Beschreibung eines einfachen genetischen Algorithmus

Ein einfacher genetischer Algorithmus (*SGA*, *simple genetic algorithm*) sieht folgendermaßen aus:

Algorithmus 2.1: Genetischer Algorithmus *SGA*

```
Erzeuge eine Initialpopulation mit N Individuen
do {
    Selektiere N̂ ≤ N Individuen anhand der Fitness
    Erzeuge eine neue Population aus der selektierten mittels Rekom-
    bination und Mutation
} until (Abbruchkriterium erfüllt)
```

Die Population $\mathfrak{X}$ besteht dabei aus N Individuen $\mathfrak{X}^i$ (ein Verzeichnis der verwendeten Symbole findet sich in Anhang B). Diese Individuen $\mathfrak{X}^i$ haben n **Loci** (Positionen), an denen jeweils ein **Gen** $\mathfrak{X}_j^i$ steht. Die möglichen Belegungen der Gene heißen **Allele** und werden durch ganze Zahlen kodiert.

$$\mathfrak{X}^i = (\mathfrak{X}_1^i, \ldots, \mathfrak{X}_n^i) \quad \text{mit } \mathfrak{X}_j^i \in \{0, \ldots, l_i\} \tag{2.1}$$

Individuen lassen sich also mit Elementen des Suchraumes $\mathcal{R}$ identifizieren. Über die Fitness-Funktion f, die auf $\mathcal{R}$ definiert ist, kann man jedem Individuum $\mathfrak{X}^i$ die **Fitness** $f(\mathfrak{X}^i)$ zuordnen.

Die drei zentralen Komponenten **Selektion**, **Rekombination** und **Mutation** werden nun einzeln vorgestellt.

2.1.1 Selektion

Die Selektion ist das fokussierende Element bei den genetischen Algorithmen und die einzige Stelle, bei der die zu optimierende Funktion ein Rolle spielt. Aus den ursprünglichen N Individuen $\mathfrak{X}^i$ werden anhand der Fitness $\hat{N}$ Individuen $\hat{\mathfrak{X}}^i$ ausgewählt, wobei manche auch mehrfach genommen werden können.

Einige gebräuchliche Selektionsverfahren sind die folgenden:

Proportionale Selektion: Individuen $\mathfrak{X}^i$ werden als Elternteil ausgewählt mit Wahrscheinlichkeit $p(\mathfrak{X}^i) = \frac{f(\mathfrak{X}^i)}{\sum_j f(\mathfrak{X}^j)} \propto f(\mathfrak{X}^i)$, also proportional zur Fitness. Nachteil dieser Methode ist, dass der Selektionsdruck im Laufe des Verfahrens *abnimmt*: Wenn alle Individuen nahe am Optimum sind, spielen kleine Fitness-Gewinne keine Rolle mehr. Außerdem ist das Verfahren nicht invariant unter Translationen der Fitnessfunktion: $g(x) := f(x) + 1000$ erzeugt einen deutlich geringeren Selektionsdruck. Es ist aber invariant unter Multiplikation der Funktion mit einer Konstanten.

Exponentielle Selektion: Mit $\beta > 0$ ist die Wahrscheinlichkeit der Selektion proportional zu $e^{\beta f(\mathfrak{X}^i)}$. Hier gibt es auch gegen Ende des Verfahrens noch einen hohen Selektionsdruck, da das exponentielle Wachstum die Landschaft streckt. Das Verfahren ist invariant unter Translation $f(x)+c$, aber in der Regel nicht unter $c \cdot f(x)$. Es wurde bisher kaum eingesetzt, weil die Wahl des Faktors β äußerst schwierig ist. In Abschnitt 4.3 wird aber eine Lösung dafür vorgestellt.

***Truncation Selection*:** Die $\hat{N} = \tau \cdot N$ besten Individuen werden ausgewählt. Eltern werden gleichverteilt aus dieser Menge gezogen. Da hier nur die Sortierung nach Fitnesswerten eine Rolle spielt, ist dieses Selektionsverfahren invariant unter jeder monotonen Transformation. Züchter verwenden *Truncation Selection*.

2.1.2 Rekombination

Rekombination *(Crossover)* erzeugt aus (in der Regel) zwei Elternindividuen ein neues Individuum. Wegen der Inspiration durch die natürliche Evolution wurde anfangs bei genetischen Algorithmen in der Regel **1-Punkt-Rekombination** verwendet (Abbildung 2.1).

Eltern	00	0110
	10	1011
Kind	00	1011

Abbildung 2.1: Beispiel für 1-Punkt-Rekombination

Hierbei wird zufällig eine Position l festgelegt. Im Kind werden die ersten l Gene von dem ersten Elternteil genommen und die restliche Gene von dem zweiten.

Die Überlegung hinter dieser Art von Rekombination im Hinblick auf Optimierung ist es, dass benachbarte Gene vermutlich gemeinsam zu höherer Fitness beitragen und somit zusammen übernommen werden sollten. Mit dem *FDA* wird in Kapitel 4 ein viel effizienteres Verfahren vorgestellt, um Wissen über Abhängigkeiten zwischen (zum Beispiel benachbarten) Variablen zu kodieren und zu verwerten. Im Allgemeinen gibt es aber keinen Grund, solch eine Annahme zu treffen, daher kann man auch **Uniform Crossover** (Syswerda, 1989) verwenden (Abbildung 2.2).

Eltern	000000
	111111
Kind	010011

Abbildung 2.2: Beispiel für Uniform Crossover

Hierbei wird für jedes Gen neu ausgewürfelt, von welchem Elternteil es kopiert werden soll.

Als Erweiterung gibt es noch **Gene Pool Recombination** (Mühlenbein and Voigt, 1996), bei dem nicht zwei Eltern gewählt werden, sondern die Gesamtheit der selektierten Individuen genommen wird. Für jedes Gen wird ein neues Elternindividuum aus der selektierten Population gewählt. Betrache als Beispiel die Spalte der selektierten Indiviuen in Tabelle 2.1. Ein neues Individuum hat also für das erste Bit die Wahl zwischen 3 Mal der Eins und 2 Mal der Null. Das zweite Bit wird nur dann auf Null gesetzt, wenn hierfür das fünfte (selektierte) Individuum ausgewählt wird und so weiter.

2.1.3 Mutation

Der Mutationsoperator sorgt für Variabilität in der Population, in dem er zufällig Fehler in die Kopien einbaut. Mit einer Mutationswahrscheinlichkeit μ pro Individuum und Gen wird in der neu erzeugten Population das erzeugte Allel zufällig durch ein anderes ersetzt. Als Mutationsrate hat sich $\mu = 1/n$, also das Inverse der Anzahl an Genen, bewährt (Mühlenbein, 1991, 1992).

2.2 Ein einfaches Beispiel

Mit binärem Definitionsbereich sei die betrachtete Optimierungsfunktion die Funktion *OneMax*:

$$\mathit{OneMax}(x) := \sum_{i=1}^{n} x_i \tag{2.2}$$

Das globale Optimum ist $(1, \ldots, 1)$ mit Fitness-Wert n. Man kann das Optimum aber auch auf einen beliebigen Punkt x^* legen durch Wahl von

$$\mathit{OneMax}_{x^*}(x) := \sum_{i=1}^{n} |x_i - (1 - x_i^*)| \tag{2.3}$$

Es ändert sich aber nichts an dem Verhalten der Verfahren durch diese Wahl.

Indivi.	Fit	Selekt.	Fit	Neu	Fit.
111010	4⋆	111010	4	111111	6
000100	1	011010	3	110011	4
100110	3	110111	5	011111	5
011010	3⋆	011101	4	100111	4
110111	5⋆	100111	4	011010	3
000101	2			110011	4
011101	4⋆			111110	5
100101	3			001101	3
100111	4⋆			110011	4
100001	2			010011	3
$\bar{f}$	3.1		4		4.1

Tabelle 2.1: Beispiellauf für den genetischen Algorithmus

Zur Illustration sei hier ein Beispiellauf ohne Mutation vorgestellt (Tabelle 2.1). Die Startpopulation in Spalte 1 ist zufällig erzeugt. Die durchschnittliche Fitness dieser Population ist $\bar{f}=3.1$. In diesem Beispiel wurden einfach die fünf besten Individuen zur Zucht ausgewählt (mit einem Stern markiert). Da es mehrere Individuen mit Fitness 3 gibt, entschied der Zufall über die Wahl.

Die neue Population entstand durch *Uniform Crossover* aus den selektierten Individuen. So entstand z.B. das zweite neue Individuum aus dem ersten und fünften Selektiertem. Für jede Bitposition wird zufällig ein Elternteil ausgesucht und das entsprechende Bit kopiert. So musste also das 1. und 5. Bit im zweiten neuen Individuum auf 1 gesetzt werden, weil beide Elternteile dort ein 1-Bit haben.

Die durchschnittliche Fitness stieg bei diesem Vorgang auf $\bar{f}=4.1$ und das globale Optimum ist nun in der Population vertreten. Weitere Iterationen würden bewirken, dass die Population irgendwann nur noch aus Kopien des Optimums besteht.

2.3 Zusammenfassung

Genetische Algorithmen stellen eine einfach zu verstehende und einfach zu implementierende Klasse von Optimierungsalgorithmen dar. Sie lassen sich auch ohne größeres Wissen über die zu behandelnde Funktion implementieren (auch wenn sie sehr stark von solchem Wissen profitieren können). Es ist zum Beispiel nicht nötig, Ableitungen der Funktion zu kennen.

Trotz des einfachen Prinzips ist das Verständnis des Verhaltens von genetischen Algorithmen nicht so einfach, wie Kapitel 3 zeigen wird.

3 *UMDA* und dynamische Verteilungen

Die mathematische Analyse der genetischen Algorithmen ist sehr schwierig, weil der Zustandsraum so groß ist. Die Ursache hierfür ist der Rekombinations-Operator. Eine exakte Untersuchung könnte den Algorithmus als (in der Regel homogene) Markovkette modellieren, in der die *gesamte Population* einen Zustand darstellt, und dann die Übergangswahrscheinlichkeiten über die Operatoren (Mutation, Rekombination, Selektion) bestimmen.

Es ist klar, dass diese Vorgehensweise nur bei extrem kleinen Problemgrößen möglich ist. Eine vernünftige Analyse kann daher nur gelingen, wenn die Anzahl der Parameter drastisch reduziert wird.

In diesem Kapitel wird der *UMDA* vorgestellt, ein Algorithmus, der einen einfachen genetischen Algorithmus approximiert. Der *UMDA* parametrisiert die Verteilung, die die Population beschreibt, durch eine kleine Zahl von reellen Parametern. Es entstehen natürlich ständig neue Varianten von genetischen Algorithmen, die Idee ist aber, dass das Verhalten dieser Algorithmen *im Kern* durch das Verhalten des *UMDA* beschrieben wird.

Dieser Algorithmus lässt sich analysieren, und es zeigt sich, dass sein Verhalten beschrieben werden kann durch die *durchschnittliche Fitness* als Funktion der Zustandsvariablen des *UMDA*.

3.1 Der Univariate Marginal Distribution Algorithm *UMDA*

Die Grundidee des *UMDA* (Mühlenbein and Paaß, 1996) entstand aus folgender Beobachtung. Geiringer (1944) hat gezeigt, dass alle vollständigen Rekombinationsverfahren die gleiche Grenzverteilung haben. Ein vollständiges Verfahren ist dabei eines, bei dem für jede Teilmenge s von I die Wahrscheinlichkeit des Austausches der Gene aus s größer Null ist.

Definition 3.1. *Die* ***einfache Produktverteilung*** *ist eine Verteilung, für die*

$$p(X) = \prod_{i=1}^{n} p_i(X_i) \tag{3.1}$$

gilt. Man sagt auch, die Verteilung ist in Robbins Proportions *(Robbins, 1918).*

Für binäre Variablen ist die Verteilung in jeder Variablen also $p_i(X_i) = q_i^{x_i}(1-q_i)^{1-x_i}$ mit reellem Parameter q_i. Dies ist eine univariate Verteilung (mit einer Variable), daher der Name *UMDA*, *univariate marginal distribution algorithm.*

Der folgende Satz ist eine wesentliche Beobachtung, die die Grundlage für die Analyse der genetischen Algorithmen liefert.

Satz 3.2 (Geiringer). *Die Rekombination verändert die univariaten Randverteilungen nicht, d.h.* $p_i(x_i, t+1) = p_i(x_i, t)$. *Die Grenzverteilung jedes vollständigen Rekombinationsverfahrens ist die einfache Produktverteilung.*

Die Konvergenz zu der Grenzverteilung erfolgt dabei sehr schnell. Alle vorgestellten Rekombinationsverfahren aus Kapitel 2 verhalten sich also im Grenzwert (nach vielen Iterationen) identisch.

Wenn man (unter Vernachlässigung der Mutation) nach jedem Selektionsschritt nicht *einmal*, sondern (unendlich) oft Rekombination betreibt, so erhält man eine Verteilung der Gene in der Population, die in *Robbins Proportions* stehen.

Wenn man nun unterstellt, dass sich das grundlegende Verhalten des genetischen Algorithmus nicht ändert durch die Anzahl an Rekombinationszyklen in jedem einzelnen Schritt, so kann man auch einen Algorithmus betrachten, der *unendlich* oft Rekombination durchführt nach der Selektion. Nach dem Satz von Geiringer kann man dies schneller erzielen, in dem man direkt Individuen nach der einfachen Produktverteilung erzeugt.

Dies führt zu einem genetischen Algorithmus mit folgenden Eigenschaften:

- Keine Mutation (das wird in 3.8 noch genauer diskutiert),
- *Uniform Crossover* (Syswerda, 1989), da durch andere Rekombinationsverfahren schwierig zu beschreibende Interaktionen zwischen den Genen entstehen, und
- *Gene Pool Recombination:* (Mühlenbein and Voigt, 1996) Es werden nicht zwei Eltern ausgesucht, sondern jedes selektierte Individuum kann zu jedem Kind beitragen. Für jedes Gen wird neu entschieden, aus welchem selektierten Individuum es kopiert wird.

Das Verhalten dieses Algorithmus lässt sich vollständig durch die univariaten Randverteilungen (für binäre Variablen einfach die Bithäufigkeiten, siehe Abschnitt A.2 im Anhang) in der selektierten Population beschreiben. Diese Beobachtung führt zum *UMDA* von Mühlenbein & Paaß (1996).

Obwohl die Verteilung der selektierten Punkte in der Regel keine einfache Produktverteilung ist, wird diese Verteilung durch Schritt 5 erzwungen. Die Auswirkungen davon werden noch ausführlich diskutiert.

Beispiel 3.1. Betrachte Tabelle 2.1 auf Seite 10. Da in der Spalte der selektierten Individuen das erste Bit drei Mal 1 ist und zwei Mal 0, gilt $p(X_1{=}1) = {}^3\!/_5$. Die gleiche

Algorithmus 3.1: *UMDA* – Univariate Marginal Distribution Algorithm

Erzeuge eine zufällige Ausgangspopulation mit N Individuen. $t \Leftarrow 0$.
do {
  Selektiere $\hat{N} \leq N$ Punkte.
  Berechne Randverteilungen $p_i(x_i, t)$ aus den selektierten Punkten.
  Erzeuge neue Punkte gemäß $p(x, t+1) = \prod_{i=1}^{n} p_i(x_i, t)$.
  $t \Leftarrow t + 1$.
} **until** (Abbruchkriterium erreicht)

Wahrscheinlichkeit ergibt sich auch dafür, dass das dritte, vierte und fünfte Bit gleich 1 ist. Für das zweite Bit folgt $p(X_2 = 1) = 4/5$. Die Wahrscheinlichkeit, das Optimum zu erzeugen (also alle Bits 1), steigt demnach von $0.5^5 \approx 0.031$ für die Startpopulation mit Gleichverteilung auf $0.8 \cdot 0.6^4 \approx 0.104$. Das Optimum ist also bei 10 Individuen im Mittel bereits ein Mal vertreten.

Man sieht auch, dass die Erzeugung neuer Individuen nach *Gene Pool Recombination* aus Abschnitt 2.1.2 in der Auswirkung identisch zu diesem Verfahren ist.

In der Informationstheorie ist die Reduzierung einer Folge von identisch verteilten diskreten Variablen auf einen Parameter als *Methode der Typen* (Csiszár and Körner, 1981) bekannt. Im *UMDA* wird also für jede Variable x_i der Typ bestimmt. Die Gesetze der großen Zahlen, angewandt auf diese Theorie, sagen dann, dass für eine lange Folge (also großes N) der Typ durch die Verteilung der Variable angenähert wird (Cover and Thomas, 1991). Daraus lassen sich prinzipiell also Aussagen über benötigte Populationsgrößen ableiten. Die Sätze gehen aber davon aus, dass die einzelnen Sample unabhängig voneinander sind. Genau diese Unabhängigkeit wird aber durch die Selektion zerstört.

3.2 Parametrische Verteilungen

Für den *UMDA* wird nur die einfache Produktverteilung benötigt. Für eine bessere Notation und im Hinblick auf spätere Erweiterungen wird dennoch der generelle Fall betrachtet, dass die Verteilung p parametrisiert ist.

Definition 3.3. *Die Abbildung*

$$\begin{aligned} \tilde{p} :\ & \mathcal{R} \times Q && \to \mathbb{R}, \\ & (x, q) && \mapsto \tilde{p}(x, q) \end{aligned} \qquad \text{mit } \forall q \in Q : \sum_{x \in \mathcal{R}} \tilde{p}(x, q) = 1 \tag{3.2}$$

*heißt **parametrische Verteilung auf** $\mathcal{R}$ mit Parameterraum $Q \subseteq \mathbb{R}^{\tilde{n}}$. Begriffe wie Stetigkeit und Differenzierbarkeit beziehen sich dabei darauf, dass $\tilde{p}(x, q)$ für jedes (feste) x in den Parametern q stetig beziehungsweise differenzierbar ist. Die **Randverteilungen** für festes q werden $\tilde{p}_s(x_s, q)$ notiert.*

*Die **parametrische durchschnittliche Fitness** zu einer parametrischen Verteilung $\tilde{p}$ und einer Funktion f ist definiert durch*

$$W_f(q) := W_{f,\tilde{p}}(q) = \sum_{x \in \mathcal{R}} f(x)\tilde{p}(x,q) \tag{3.3}$$

*Die durch die Funktion $W_f(q)$ aufgespannte Untermannigfaltigkeit im $\mathbb{R}^{\tilde{n}+1}$ heißt **Fitnesslandschaft**.*

*Eine Abbildung $\tilde{P}(p)$ vom Raum der Verteilungen $\mathcal{R} \to \mathbb{R}$, die Wertebereich Q hat, heißt **Parameterschätzer**. Falls*

$$\tilde{P}(\tilde{p}(\cdot, q)) = q, \tag{3.4}$$

*so heißt der Parameterschätzer **selbsterhaltend**.*

Ein Parameterschätzer lässt sich durch ein Abstandsmaß im Raum der Verteilungen immer definieren, indem man einfach den Parameter wählt, der den gegebenen Abstand minimiert. Das ist aber im Allgemeinen eine sehr aufwändige Operation. Man wird sich also für Parameterschätzer interessieren, die effizient berechenbar sind.

Beispiel 3.2.

$$\tilde{p}^{triv}(x,q) = q_{\check{x}} \qquad \text{mit } Q = [0,1]^{2^n}, \sum_{x \in \mathcal{R}} q_{\check{x}} = 1 \tag{3.5}$$

Bei dieser parametrischen Verteilung ist der Parameter einfach die Dichte des dazugehörenden Punktes. Der (selbsterhaltende) Parameterschätzer ist einfach $\tilde{P}(p) = (p(x))_x$.

Beispiel 3.3. Im binären Raum mit $\mathcal{R}_i = \{0,1\}$ lassen sich die einfachen Produktverteilungen parametrisieren über $Q = [0,1]^n$ durch

$$\tilde{p}^{univ}(x,q) := \prod_{i=1}^{n} q_i^{x_i}(1-q_i)^{1-x_i} \tag{3.6}$$

Dabei ist $q_i = p_i(1) = p(X_i = 1)$ der selbsterhaltende Parameterschätzer. Wie sich noch zeigen wird, kann man in diesem Fall die durchschnittliche Fitness für zerlegbare Funktionen effizient (also in polynomieller Zeit) ausrechnen.

3.2.1 Die parametrische einfache Produktverteilung

Für einen größeren (diskreten) Wertebereich gilt als Erweiterung von Beispiel 3.3:

Definition 3.4. *Mit $\mathcal{R}_i = \{0, \ldots, l_i\}$ gilt $Q^u \subset \prod_{i=1}^{n} \prod_{y_i=1}^{l_i} [0,1]$. Dann parametrisiert*

$$\tilde{p}^u(x,q) := \prod_{i=1}^{n} \left[\prod_{y_i=1}^{l_i} q_{i,y_i}^{\delta(x_i=y_i)} \right] \left(1 - \sum_{y_i=1}^{l_i} q_{i,y_i}\right)^{\delta(x_i=0)} \tag{3.7}$$

die einfache Produktverteilung mit der Konvention $0^0 = 1$, wobei δ die Dirac-Verteilung darstellt. Dabei gelten für die Parameter die Nebenbedingung

$$Q^u := \{q \in \prod_{i=1}^{n} [0,1]^{l_i} \mid \forall i : \sum_{y_i=1}^{l_i} q_{i,y_i} \leq 1\} \tag{3.8}$$

Die dazugehörende durchschnittliche Fitness ist

$$W_f^u(q) = \sum_{x \in \mathcal{R}} f(x) \tilde{p}^u(x,q) \tag{3.9}$$

und der selbsterhaltende Parameterschätzer

$$q_{i,y_i} = p_i(y_i) = p(X_i = y_i) \tag{3.10}$$

Definiere weiterhin $\mathcal{R}_i^u := \{1, \ldots, l_i\}$ und analog als Produkt $\mathcal{R}_J^u$ mit $J \subseteq I$.

Der hier verwendete Parameterschätzer entspricht dabei dem *Maximum Likelihood* Schätzer aus Anhang A.4. In Abschnitt 3.8 wird die Möglichkeit behandelt, den Bayes'schen Schätzer zur Bestimmung der Parameter zu verwenden.

Diese Verteilung entspricht den *Robbins Proportions* aus der Populationsgenetik. Man beachte, dass diese Verteilung in jedem Argument q_{i,y_i} eine *lineare* Funktion ist. Die Notation ist relativ kompliziert, es gilt aber für binäre Funktionen

$$\tilde{p}^u(x,q) = \prod_{i=1}^{n} q_{i,1}^{x_i} (1 - q_{i,1})^{1-x_i} \tag{3.11}$$

$$W_f^u(q) = \sum_{x} f(x) \prod_{i=1}^{n} q_{i,1}^{x_i} (1 - q_{i,1})^{1-x_i} \tag{3.12}$$

Summieren über alle x mit einem x_i fest liefert wieder im allgemeinen Fall die Randverteilungen:

$$\forall 1 \leq y_i \leq l_i : \quad p_i(y_i) = q_{i,y_i}; \qquad p_i(0) = 1 - \sum_{y_i=1}^{l_i} q_{i,y_i} \tag{3.13}$$

und somit

$$\tilde{p}^u(x,q) = \prod_{i=1}^{n} \tilde{p}_i^u(x_i, q) \tag{3.14}$$

Diese Gleichung ist natürlich als Identität aufzufassen und nicht als Definition, da $\tilde{p}^u$ auf beiden Seiten vorkommt. Es lässt sich nun auch die Ableitung darstellen:

$$\frac{\partial \tilde{p}^u(x,q)}{\partial q_{i,y_i}} = \begin{cases} 0 & \text{wenn } x_i \neq y_i \text{ und } x_i \neq 0 \\ -\prod_{j \neq i} \tilde{p}_j^u(x_j, q) & \text{wenn } x_i = 0 \\ \prod_{j \neq i} \tilde{p}_j^u(x_j, q) & \text{wenn } x_i = y_i \end{cases} \tag{3.15}$$

Für die Darstellung in Definition 3.4 gilt, dass $q_{i,0}$ wegen der Nebenbedingung $p_i(0) = 1 - \sum_{y_i=1}^{l_i} p_i(y_i)$ nicht als Parameter genommen wurde. Es besteht aber auch die Möglichkeit, diese $q_{i,0}$ hinzuzunehmen und die Nebenbedingungen an Q entsprechend anzupassen:

Definition 3.5. *Mit $\mathcal{R}_i = \{0, \ldots, l_i\}$ gilt $Q^{u0} \subset \prod_{i=1}^{n} \prod_{y_i=0}^{l_i} [0,1]$ und*

$$\tilde{p}^{u0}(x, \tilde{q}) := \prod_{i=1}^{n} \tilde{q}_{i,x_i} \tag{3.16}$$

Dabei gilt für die Parameter die Nebenbedingung

$$Q^{u0} := \{q \in \prod_{i=1}^{n} [0,1]^{l_i+1} \mid \forall i : \sum_{y_i=0}^{l_i} q_{i,y_i} = 1\} \tag{3.17}$$

Die dazugehörende durchschnittliche Fitness ist

$$W_f^{u0}(\tilde{q}) = \sum_{x \in \mathcal{R}} f(x) \tilde{p}^{u0}(x, \tilde{q}) \tag{3.18}$$

und der selbsterhaltende Parameterschätzer

$$\tilde{q}_{i,y_i} = p_i(y_i) = p(X_i = y_i) \tag{3.19}$$

Diese zwei Darstellungen beschreiben also zwei verschiedene Fitnesslandschaften der gleichen Verteilung.

Anmerkung 3.6. Der Raum Q^u ist in den Raum Q^{u0} durch $\vec{Q}$ wie folgt eingebettet:

$$\vec{Q}(q) = \tilde{q} \text{ mit } \tilde{q}_{i,y_i} = \begin{cases} 1 - \sum_{y_i=1}^{l_i} q_{i,y_i} & \text{für } y_i = 0 \\ q_{i,y_i} & \text{sonst} \end{cases} \tag{3.20}$$

Dabei gilt

$$\tilde{p}^u(x, q) = \tilde{p}^{u0}(x, \vec{Q}(q)) \qquad \text{und} \qquad W_f^u(q) = W_f^{u0}(\vec{Q}(q)) \tag{3.21}$$

3.2.2 Ableitungen der durchschnittlichen Fitness

Von besonderem Interesse sind die Ableitungen der durchschnittlichen Fitness. Der folgende Satz ist aus (Mühlenbein and Mahnig, 1999a). Mühlenbein hat in (Mühlenbein, 1998) bereits einen Beweis für binäre Allele geführt.

Lemma 3.7. *Für eine Funktion f und eine Verteilung p setze*

$$\bar{f}_i(x_i, p) := \sum_{x | X_i = x_i} f(x) \prod_{j \neq i} p_j(x_j) \tag{3.22}$$

$$F_i(x_i, p) := \bar{f}_i(x_i, p) - W_f(p), \tag{3.23}$$

Dann gilt für die einfache Produktverteilung $p(x) = \tilde{p}^u(x, q)$

$$\frac{\partial W_f^u(q)}{\partial q_{i,x_i}} = \bar{f}_i(x_i, p) - \bar{f}_i(0, p) \qquad x_i \in \{1, \ldots, l_i\} \tag{3.24}$$

$$= F_i(x_i, p) - F_i(0, p) \tag{3.25}$$

sowie

$$F_i(x_i, p) = \frac{\partial W_f^u(q)}{\partial q_{i,x_i}} - \sum_{x_i=1}^{l_i} \frac{\partial W_f^u(q)}{\partial q_{i,x_i}} q_{i,x_i} \tag{3.26}$$

Beweis: Definiere als Abkürzung

$$\bar{q}_i(x) := \prod_{j \neq i} p_j(x_j) \tag{3.27}$$

Wähle $i \in I$.

$$W_f^u(q) = \sum_{y_i=0}^{l_i} \sum_{x|x_i=y_i} f(x) p_i(x_i) \prod_{j \neq i} p_j(x_j) \tag{3.28}$$

$$= \sum_{y_i=1}^{l_i} \sum_{x|x_i=y_i} f(x) p_i(x_i) \bar{q}_i(x) + \sum_{x|x_i=0} f(x) \left(1 - \sum_{y_i=1}^{l_i} p_i(y_i)\right) \bar{q}_i(x) \tag{3.29}$$

$$= \sum_{y_i=1}^{l_i} q_{i,y_i} \left[\sum_{x|x_i=y_i} f(x) \bar{q}_i(x) - \sum_{x|x_i=0} f(x) \bar{q}_i(x) \right] + \sum_{x|x_i=0} f(x) \bar{q}_i(x)$$

$$= \sum_{y_i=1}^{l_i} q_{i,y_i} \left[\bar{f}_i(y_i, p) - \bar{f}_i(0, p) \right] + \bar{f}_i(0, p) \tag{3.30}$$

Weil $\bar{f}_i$ von q_{i,x_i} unabhängig ist, folgen daraus (3.24) und (3.25).
Weiter gilt dann

$$W_f^u(q) = \bar{f}_i(0, p) + \sum_{x_i=1}^{l_i} \left[\bar{f}_i(x_i, p) - \bar{f}_i(0, p) \right] q_{i,x_i} \tag{3.31}$$

$$= \bar{f}_i(0, p) + \sum_{x_i=1}^{l_i} \frac{\partial W_f^u(q)}{\partial q_{i,x_i}} q_{i,x_i} \tag{3.32}$$

und somit (3.26). (Lemma 3.7) ■

Anmerkung 3.8. Es ist naheliegend, eine entsprechende Gleichung für den Parameterraum Q^{u0} zu bestimmen. Es ergibt sich dabei die Darstellung

$$\frac{\partial W_f^{u0}}{\partial \tilde{q}_{i,x_i}} = \bar{f}_i(x_i, \tilde{p}) \tag{3.33}$$

Man muss aber in der Interpretation dieser Gleichung sehr vorsichtig sein, da die $\tilde{q}_{i,x_i}$ wegen der Nebenbedingung keine echten Parameter der Funktion W_f^{u0} sind. Man fasst hier W_f^{u0} als Polynom in den Variablen $\tilde{q}_{i,x_i}$ auf und bildet so die Ableitung, ohne sich für die Nebenbedingung zu interessieren.

Betrachte als konkretes Beispiel die Funktion $f(x_1, x_2) = x_1 + 2x_2$. Die durchschnittliche Fitness bestimmt sich nach (3.18) zu

$$W_f^{u0}(\tilde{q}) = 1 \cdot \tilde{q}_{1,1}\tilde{q}_{2,0} + 2 \cdot \tilde{q}_{1,0}\tilde{q}_{2,1} + 3 \cdot \tilde{q}_{1,1}\tilde{q}_{2,1}, \tag{3.34}$$

wobei $\tilde{q}_{1,0} + \tilde{q}_{1,1} = 1$ und $\tilde{q}_{2,0} + \tilde{q}_{2,1} = 1$ gelten. Damit ergibt sich zum Beispiel die Ableitung $D := \partial W_f^{u0}/\partial\tilde{q}_{1,1} = \tilde{q}_{2,0} + 3\tilde{q}_{2,1}$. Unter Berücksichtung der Nebenbedingung $\tilde{q}_{2,0} + \tilde{q}_{2,1} = 1$ folgt $D = 1 + 2\tilde{q}_{2,1}$.

Wenn man aber *zuerst* die Nebenbedingung in (3.34) einsetzt, erhält man

$$W_f^{u0} = 1 \cdot \tilde{q}_{1,1}(1 - \tilde{q}_{2,1}) + 2 \cdot (1 - \tilde{q}_{1,1})\tilde{q}_{2,1} + 3 \cdot \tilde{q}_{1,1}\tilde{q}_{2,1} = \tilde{q}_{1,1} + 2\tilde{q}_{2,1} \tag{3.35}$$

und somit $D = 1$. Dies ist offensichtlich ein Widerspruch. Das liegt daran, dass man Gleichungen wie (3.34) nicht ableiten darf nach Variablen, die durch Nebenbedingungen gekoppelt sind.

Korrekterweise muss man die Variablen $\tilde{q}_{i,x_i}$ durch l_i Parameter beschreiben und dann nach diesen Parametern ableiten. Genau das passiert aber in Q^u und liefert dann obige Darstellung.

Mit Hilfe dieser Theorie soll im nächsten Abschnitt für eine bestimmte Klasse von Funktionen gezeigt werden, dass genetische Algorithmen insbesondere am Anfang Schwierigkeiten haben. Dies sind Funktionen, die invariant unter Permutationen der Wertemenge sind.

Definition 3.9. *Eine Funktion heißt* ***invariant unter Permutation der Wertemenge****, wenn alle $\mathcal{R}_i$ gleich sind und für jede Permutation σ von $\mathcal{R}_i$ gilt:*

$$f(x_1, \dots, x_n) = f(\sigma(x_1), \dots, \sigma(x_n)) \qquad =: f(\sigma(x)) \tag{3.36}$$

Für binären Wertebereich sind dies die Funktionen, die unter Bitinversion gleich bleiben.

Es gibt in der diskreten Optimierung eine ganze Reihe von Problemen, die diese Symmetrieeigenschaft haben, wie zum Beispiel das bipolare Ising-Modell, die Graph-Partitionierung oder (je nach Kodierung) das Traveling-Salesman-Problem.

Es gilt nun das folgende

Lemma 3.10. *Es sei $f(x)$ eine Funktion und p eine Verteilung, die invariant unter Permutation der Wertemenge seien (dies gilt insbesondere für die Gleichverteilung). Für jedes i ist dann $\bar{f}_i(\cdot, p)$ konstant:*

$$\bar{f}_i(x_i, p) = \bar{f}_i(y_i, p) \tag{3.37}$$

Beweis: Sei σ eine Permutation. Dann gilt

$$\bar{f}_i(x_i,p) = \sum_{x|X_i=x_i} f(x) \prod_{j\neq i} p_j(x_j) = \sum_{x|X_i=x_i} f\left(\sigma^{-1}(x)\right) \prod_{j\neq i} p_j\left(\sigma^{-1}(x_j)\right) \tag{3.38}$$

$$= \sum_{x|\sigma(X_i)=\sigma(x_i)} f(x) \prod_{j\neq i} p_j(x_j) = \bar{f}_i(\sigma(x_i),p) \tag{3.39}$$

(Lemma 3.10) ■

3.3 Dynamische Verteilungen

Man kann die probabilistischen Algorithmen auffassen als Operatoren auf dem Raum der parametrischen Verteilungen. Sie führen also eine Suche im *Parameterraum* durch. Dabei entsteht eine Folge von Verteilungen, nämlich für jede Generation eine. Im Folgenden werden folgende Abkürzungen für zeitabhängige Verteilungen verwendet.

Definition 3.11. *Es sei $q(t)$ eine Folge von Parametern, wobei als Definitionsbereich für t entweder die natürlichen Zahlen oder die positive reelle Halbachse gewählt wird. Dann definiere jedes Objekt, das von einer Verteilung abhängt, als abhängig von der Zeit t, wobei die Parameterfolge in der Notation unterdrückt wird. Es ergibt sich also*

$$\begin{aligned} p(x,t) &:= \tilde{p}(x,q(t)) \\ W_f(t) &:= W_f(q(t)) \\ \bar{f}_i(y_i,t) &:= \bar{f}_i(y_i,q(t)) \\ F_i(y_i,t) &:= \bar{f}_i(y_i,q(t)) - W_f(q(t)) \end{aligned} \tag{3.40}$$

Es stellt sich nun die Frage, wie die Algorithmen auf die betrachteten Größen einwirken und ob das Verhalten der Algorithmen allein aus den oben genannten Größen beschrieben werden kann. Eine vollständige Beantwortung dieser Frage ist außerordentlich schwierig, da der Zustandsraum (eine Population) exponentiell groß ist, nämlich $O(c^{n\cdot N})$. Daher werde ich mich auf Spezialfälle oder Approximationen einschränken.

3.3.1 Die Dynamik von *UMDA* mit proportionaler Selektion

Evolutionsbiologen verwenden in ihren Analysen proportionale Selektion. Wie sich zeigt, kann man dann Antworten auf die Frage der Dynamik finden. Proportionale Selektion ändert die Häufigkeiten der Individuen nach der Formel

$$p^s_f(x,p(\cdot)) = \frac{p(x)\cdot f(x)}{\sum_y p(y)f(y)} \tag{3.41}$$

Für *UMDA* ist die Startverteilung in Generation t gleich $p(x,t) = \tilde{p}^u(x,q(t))$. Damit ergibt sich

$$p^s(x,t) := p_f^s(x,p(\cdot,t)) = p(x,t)\frac{f(x)}{W_f(t)} \tag{3.42}$$

Dies liefert durch Summieren folgende Gleichung für die Randverteilungen:

$$p_i^s(x_i,t) = p_i(x_i,t)\frac{\bar{f}_i(x_i,t)}{W_f(t)} \tag{3.43}$$

Beim *UMDA* sind die univariaten Randverteilungen der neuen Generation gleich der entsprechenden Randverteilungen der selektierten Population. Es ergibt sich

Lemma 3.12. *Die Dynamik der Parameter aus Q^{u0} bei UMDA mit proportionaler Selektion ist für $x_i \in \{0,\ldots,l_i\}$ gegeben durch*

$$\tilde{q}_{i,x_i}(t+1) = \tilde{q}_{i,x_i}(t) + \tilde{q}_{i,x_i}(t)\frac{F_i(x_i,t)}{W_f^{u0}(t)} \tag{3.44}$$

beziehungsweise in dem Parameterraum Q^u für $x_i \in \{1,\ldots,l_i\}$

$$q_{i,x_i}(t+1) = q_{i,x_i}(t) + q_{i,x_i}(t)\frac{F_i(x_i,t)}{W_f^u(t)} \tag{3.45}$$

$$= q_{i,x_i}(t) + \frac{q_{i,x_i}(t)}{W_f^u(t)}\left(\frac{\partial W_f^u}{\partial q_{i,x_i}} - \sum_{y_i=1}^{l_i} q_{i,y_i}(t)\frac{\partial W_f^u}{\partial q_{i,y_i}}\right) \tag{3.46}$$

Für binäre Variablen vereinfacht sich dies zu

$$q_i(t+1) = q_i(t) + \frac{q_i(t)\big(1-q_i(t)\big)}{W_f^u(t)} \cdot \frac{\partial W_f^u}{\partial q_i} \tag{3.47}$$

Beweis: Gleichungen (3.44) und (3.45) folgen direkt aus (3.43). Die Darstellung mit Hilfe der Ableitungen ergeben sich durch Lemma 3.7. (Lemma 3.12) ■

Für binäre Variablen erfolgte der Beweis bereits in (Mühlenbein, 1998).

Diese dynamischen Gleichungen sind von Wright (1932) in der evolutionären Biologie postuliert worden und als „Wright's equations“ bekannt. Die Anwendung der Gleichung auch im Zusammenhang der adaptiven Fitness-Landschaft wird in Abschnitt 3.6 diskutiert.

Anmerkung 3.13. Wenn man W_f^{u0} als Polynom in den Variablen $\tilde{q}_{i,x_i}$ auffasst wie in Gleichung (3.33), so kann man (3.44) auch schreiben als

$$\tilde{q}_{i,x_i}(t+1) = \frac{\tilde{q}_{i,x_i}(t)}{W_f^{u0}(t)} \cdot \frac{\partial W_f^{u0}}{\partial \tilde{q}_{i,x_i}} \tag{3.48}$$

Um Nutzen aus den Gleichungen aus Lemma 3.12 zu ziehen, benötigt man noch eine Möglichkeit, die durchschnittliche Fitness zu berechnen. Dies geschieht in Abschnitt 3.4. Zunächst gilt:

Korollar 3.14. *Für binäre Variablen macht UMDA mit proportionaler Selektion gewichteten Gradientenaufstieg im Raum der durchschnittlichen Fitness. Wenn der Gradient der durchschnittlichen Fitness verschwindet, ist der Fortschritt minimal und nur durch Effekte der endlichen Populationsgröße zu überwinden.*

UMDA macht keinen exakten Gradientenaufstieg, weil in Gleichung (3.47) vor dem Gradienten von W_f^u noch der Vorfaktor $q_i(t)\big(1-q_i(t)\big)$ steht, der nötig ist, um die Parameter im Intervall $[0,1]$ zu behalten. Dadurch werden die Gradienten vom Rand „weggebogen". Diese Dynamik ist deutlich aufwändiger zu analysieren, vergleiche Abschnitt 3.7.4.

Es stellt sich nun die Frage, ob man die problematischen Punkte mit verschwindendem Gradienten erkennen kann. Mit Lemma 3.10 identifiziert das folgende Korollar eine Kategorie:

Korollar 3.15. *Für Funktionen, die invariant unter Permutation der Wertemenge sind (also zum Beispiel binären Funktionen, die symmetrisch unter Bitinversion sind), ist der Startpunkt in der ersten Generation ein kritischer Punkt mit verschwindendem Gradienten.*

Wie bereits bei Definition 3.9 bemerkt, fallen viele interessante Funktionen in diese Kategorie, wie die Graph-Partitionierung. Es ist schon länger bekannt, das diese Klasse von Problemen den genetischen Algorithmen Schwierigkeiten machen, zum Beispiel in (Naudts, 1998). Die hier vorgestellte Technik erlaubt aber eine genauere Analyse als etwa die Epistasis-Untersuchungen in (Davidor, 1991). In der Epistasis-Untersuchung wird fälschlicherweise $f(x)$ statt der durchschnittlichen Fitness untersucht. Dies entspricht der Analyse der durchschnittlichen Fitness für den Fall, dass die Verteilung die Gleichverteilung ist. Daher kann dort nur die Startgeneration betrachtet werden, und auch nur, wenn die Startverteilung die Gleichverteilung ist.

Falls die Situation des Spezialfalls aus Korollar 3.15 gegeben ist, so ist in der Notation der Schemata aus zum Beispiel (Goldberg, 1989a)

$$f(*,\dots,*,1,*,\dots,*) = f(*,\dots,*,0,*,\dots,*). \tag{3.49}$$

Diese Notation beschreibt die durchschnittliche Fitness bei festgehaltenen Bits. Es gilt also zum Beispiel $f(1,*,\dots,*) = \bar{f}_1(1,p_0)$ wobei p_0 die Gleichverteilung ist. Es gibt also keine Differenzierung in den Schemata erster Ordnung, also Schemata mit einem festgehaltenen Bit. Nach Korollar 3.15 ist dies ein kritischer Punkt, unabhängig von Schemata höherer Ordnung. Aber die Schema- und Epistasis-Theorie untersucht allein die Funktion unabhängig von der Population und kann daher nur Aussagen zu dem *Beginn* der Suche machen, während die Theorie der durchschnittlichen Fitness für den gesamten Prozess geeignet ist.

3.4 Effiziente Berechnung der durchschnittlichen Fitness

Die durchschnittliche Fitness und deren Ableitung beschreiben also zumindest für proportionale Selektion das Verhalten des *UMDA*. Daher ist es wichtig, diese Größe effizient ausrechnen zu können. Diese Rechnungen werden zunächst nur für binäre Fitnessfunktionen durchgeführt. Es gilt:

Lemma 3.16 (Möbius-Inversion). *Es seien Φ und Ψ Funktionen aller Teilmengen einer Grundmenge $\mathcal{S}$. Dann gilt für $c \subseteq \mathcal{S}$:*

$$\Phi(c) = \sum_{b \subseteq c} \Psi(b) \qquad \Leftrightarrow \qquad \Psi(c) = \sum_{b \subseteq c} (-1)^{|c \setminus b|} \Phi(b) \tag{3.50}$$

$$\Phi(c) = \sum_{b \supseteq c} \Psi(b) \qquad \Leftrightarrow \qquad \Psi(c) = \sum_{b \supseteq c} (-1)^{|b \setminus c|} \Phi(b) \tag{3.51}$$

Für den Beweis siehe zum Beispiel (Lauritzen, 1996). Durch zweimaliges Einsetzen ergibt sich

Korollar 3.17. *Es sei Φ wie oben. Dann gilt*

$$\Phi(b) = \sum_{c \subseteq b} \sum_{d \subseteq c} (-1)^{c \setminus d} \Phi(d) \tag{3.52}$$

$$\Phi(b) = \sum_{c \supseteq b} \sum_{d \supseteq c} (-1)^{d \setminus c} \Phi(d) \tag{3.53}$$

Damit folgt der erste Darstellungssatz:

Lemma 3.18. *Jede Fitnessfunktion $f : \{0,1\}^n \to \mathrm{I\!R}$ lässt sich (eindeutig) darstellen als*

$$f(x) = \sum_{s \subseteq I} a_s x^s \tag{3.54}$$

Dabei gilt

$$f(x) = \sum_{s \subseteq \check{x}} a_s \qquad \textit{und} \tag{3.55}$$

$$a_c = \sum_{b \subseteq c} (-1)^{c \setminus b} f(\hat{b}), \tag{3.56}$$

wobei die dualen Mengen $\check{x}$ in Anhang A.1.2 definiert werden.

Beweis: Da x^s nach (A.7) genau dann 1 ist, wenn $s \subseteq \check{x}$, sind (3.54) und (3.55) äquivalent. Mit gegebener Funktion definiert (3.56) die Koeffizienten, die dann nach der Möbius-Inversion (3.55) ergeben. Die Koeffizienten bestimmen nun ebenfalls nach der Möbius-Inversion eindeutig die Funktion. (Lemma 3.18) ∎

Diese Darstellung ist ähnlich der polynomiellen Darstellung binärer Funktionen (Wegener, 1987). Der entscheidende Unterschied ist aber, dass hier reelle Koeffizienten den Monomen zugeordnet sind. In der Theorie der *Markov Random Fields* (Winkler, 1995) wird mit Hilfe der Möbius-Inversion gezeigt, dass sich *random fields* immer als Gibb's-Felder für ein Potential darstellen lassen. Auf die Unterschiede und Ähnlichkeiten zu der Theorie der *Markov Random Fields* wird in Abschnitt 4.2.3 eingegangen.

Die Darstellung wird auch in der Walsh-Transformation (Bethke, 1981) benutzt. Für eine polynomielle Berechenbarkeit dürfen nur polynomiell viele Koeffizienten ungleich Null sein.

Beispiel 3.4. Für lineare Funktionen $Linear(x) = \alpha_0 + \alpha_i x_i$ ergeben sich die Koeffizienten wegen der Eindeutigkeit zu

$$a_c = \begin{cases} \alpha_i & c = \{i\} \\ \alpha_0 & c = \emptyset \\ 0 & \text{sonst} \end{cases} \tag{3.57}$$

Analog würde die Funktion $f_1(x) = x_1x_2 + x_3x_4$ nur die Koeffizienten $a_{1,2}$ und $a_{3,4}$ ungleich Null haben.

Betrachte schließlich $f_2(x)$ mit $f_2(x) = 1$ für $x = (1, \ldots, 1)$ und 0 sonst. Dann sind alle Koeffizienten gleich Null bis auf $a_{1,\ldots,n}$.

3.4.1 Bezug zur additiven Zerlegung

Die Zerlegung in Parameter ist kompatibel zu einer additiven Zerlegung der Funktion:

Satz 3.19. *Die binäre Funktion f habe die folgende additive Zerlegung:*

$$f(x) = \sum_{i=1}^{m} f_{s_i}(x), \tag{3.58}$$

wobei die f_{s_i} nur von den Variablen mit Index $i \in s_i$ abhängen, siehe Abschnitt A.1.3 im Anhang. Dann gilt für alle $s \subseteq I$:

$$\nexists i\ : s \subseteq s_i \implies a_s = 0, \tag{3.59}$$

also

$$f(x) = \sum_{s \subseteq s_i, i \in I} a_s x^s \tag{3.60}$$

Beweis: Betrachte eine Teilfunktion f_c, die nur von den Variablen in c abhängt. Nach Lemma 3.18 gilt

$$f_c(x) = \sum_{s \subseteq I} a^c_s x^s \tag{3.61}$$

wobei die a_s^c für festes c die Koeffizienten von f_c aus dem Lemma sind. Dann gilt für alle b mit $b \not\subseteq c$: $a_b^c = 0$. Denn sei b eine kleinste Menge mit $b \not\subseteq c$ und $a_b^c \neq 0$. Es gilt $b = c \cup d$. Wegen der Minimalität von b gilt nun

$$f_c(\hat{b}) = f_c(\bar{P}_c \hat{b}) + a_b \tag{3.62}$$

$\hat{b}$ und $\bar{P}_c \hat{b}$ unterscheiden sich aber nur in Variablen, die nicht in c liegen im Widerspruch zu der Definition von f_c.

Da f die Summe der f_{s_i} ist, folgt damit der Satz. (Satz 3.19) ■

Definition 3.20. *Für $s \subseteq I$ definiere*

$$p_s := p(X_i = 1 | i \in s), \qquad p_\emptyset = 1 \tag{3.63}$$

Für $s = \{x_i\}$ ist diese Notation eine Verallgemeinerung der Abkürzung $p_i := p(X_i = 1)$ für die univariaten Randverteilungen.

Die Randverteilung p_c berechnet sich durch Summieren über alle $p(x)$, wobei die Variablen x_i mit $i \in c$ gleich Eins sein müssen. Ausgedrückt über die assoziierten Mengen ist dies eine Summation über alle Obermengen von $\hat{c}$. Mit der Möbius-Inversion gilt nun

Lemma 3.21.

$$p_c = \sum_{b \supseteq c} p(\hat{b}) \implies \qquad p(x) = \sum_{s \supseteq \check{x}} (-1)^{|s \setminus \check{x}|} p_s \tag{3.64}$$

Anmerkung 3.22. Diese Darstellung ist eine parametrische Verteilung. Mit $\mathcal{R}_i = \{0, 1\}$ und $Q \subset [0, 1]^{2^n}$ lässt sich jede Dichte p darstellen durch

$$p(x) = \tilde{p}^{uniA}(x, q) := \sum_{s \supseteq \check{x}} (-1)^{|s \setminus \check{x}|} q_s \tag{3.65}$$

wobei $q_c = \sum_{b \supseteq c} p(\hat{b})$ der (selbsterhaltende) Parameterschätzer ist. Für den Parameterraum Q gilt dadurch die Einschränkung

$$\forall b: \quad 0 \leq \sum_{c \supseteq b} (-1)^{|c \setminus b|} q_c \leq 1, \qquad q_\emptyset = 1 \tag{3.66}$$

Jede solche Parameterwahl liefert eine gültige Verteilung. Zu zeigen ist dafür lediglich, dass die Summe der Dichten gleich 1 ist. Mit Korollar 3.17 gilt

$$\sum_{c \supseteq \emptyset} \sum_{d \supseteq c} (-1)^{|d \setminus c|} q_d = q_\emptyset = 1 \tag{3.67}$$

Beispiel 3.5. Für $n = 2$ gibt es die vier verallgemeinerten univariaten Randverteilungen $p_\emptyset = 1$, $p_1 = p(X_1 = 1)$, $p_2 = p(X_2 = 1)$ und $p_{12} = p(X_1 = X_2 = 1)$. Es gilt

$$\begin{aligned} p(0,0) &= p_\emptyset - p_1 - p_2 + p_{12} & \qquad p(1,0) &= p_1 - p_\emptyset \\ p(0,1) &= p_2 - p_\emptyset & p(1,1) &= p_{12} \end{aligned} \tag{3.68}$$

Weiterhin gilt der folgende Darstellungssatz:

Satz 3.23. *Wenn die Fitnessfunktion $f(x)$ für binäres x_i gegeben ist durch*

$$f(x) = \sum_{s \subseteq I} a_s x^s, \tag{3.69}$$

dann ist die durchschnittliche Fitness $W_f(p)$ gleich

$$W_f(p) = \sum_{s \subseteq I} a_s p_s \tag{3.70}$$

Beweis:

$$W_f(p) = \sum_x f(x)p(x) = \sum_x f(x) \sum_{c \supseteq \check{x}} (-1)^{|c \setminus \check{x}|} p_c \tag{3.71}$$

$$= \sum_{c \subseteq I} \sum_{b \subseteq c} f(\hat{b})(-1)^{|c \setminus b|} p_c = \sum_{c \subseteq I} a_c p_c \tag{3.72}$$

(Satz 3.23) ■

Wenn man als Verteilung die Gleichverteilung ansetzt, ergibt sich als Spezialfall eine ähnliche Darstellung wie bei der Walsh-Transformation von Bethke (1981), die Goldberg (1989a) zur Analyse der Deception verwendet hat. Dieser Ausdruck vereinfacht sich, wenn die Wahrscheinlichkeitsverteilung faktorisiert. Mit der einfachen Produktverteilung ergibt sich zum Beispiel

$$p_c = \prod_{i \in c} p_i, \tag{3.73}$$

damit ist $W_f(p)$ nur noch eine Funktion von n Parametern und daher

Korollar 3.24. *Für die einfache Produktverteilung ergibt sich die durchschnittliche Fitness zu*

$$W_f(p) = \sum_{s \subseteq I} a_s \prod_{i \in s} p_i \tag{3.74}$$

Dies ist effizient berechenbar, wenn die Anzahl an nicht-verschwindenden Koeffizienten der Funktion in der Darstellung aus Lemma 3.18 polynomiell ist.

Ebenso führt eine Fitnessfunktion, die keine Interaktionen höherer Ordnung hat, zu einem einfachen Ausdruck für die durchschnittliche Fitness. In der Wahrscheinlichkeitsverteilung tauchen dann Korrelationen höherer Ordnung auch nicht mehr auf. Es gilt zum Beispiel

Korollar 3.25. *Für lineare Fitnessfunktion gilt*

$$W_{Linear}(p) = a_\emptyset + \sum_{i=1}^{n} a_i p_i. \tag{3.75}$$

Damit gilt unabhängig von der Verteilung p, *dass die durchschnittliche Fitness nur von den univariaten Randverteilungen abhängt.*

Die Funktion $f_2(x)$ aus Beispiel 3.4 hat schließlich die durchschnittliche Fitness

$$W_{f_2}(p) = p_{1,\dots,n} \tag{3.76}$$

3.4.2 Zerlegungssatz für allgemeinen diskreten Definitionsbereich

Für beliebigen diskreten Definitionsbereich $\mathcal{R} = \prod_{i=1}^{n} \mathcal{R}_i$ gilt der Zerlegungssatz:

Satz 3.26. *Es sei $s_1, \dots, s_m$ eine disjunkte Zerlegung von I und es seien f_{s_i} Funktionen, die nur von den Variablen in s_i abhängen. Dann gilt für beliebige Verteilungen p:*

$$W_f(p) = \sum_{i=1}^{m} \sum_{x_{s_i} \in \mathcal{R}_{s_i}} f_{s_i}(x_{s_i}) p_{s_i}(x_{s_i}) = \sum_{i=1}^{m} W_{f_{s_i}}(p_{s_i}) \tag{3.77}$$

Insbesondere gilt für $s_i = \{i\}$:

$$W_f(p) = \sum_{i=1}^{n} \sum_{x_i \in \mathcal{R}_i} f_i(x_i) p_i(x_i) \tag{3.78}$$

Der Spezialfall entspricht Korollar 3.25.

Beweis:

$$W_f(p) = \sum_{x \in \mathcal{R}} f(x)p(x) = \sum_{x_{s_1} \in \mathcal{R}_{s_1}} \cdots \sum_{x_{s_m} \in \mathcal{R}_{s_m}} \big(f_{s_1}(x_{s_1}) + \dots + f_{s_m}(x_{s_m})\big)p(x) \tag{3.79}$$

$$= \sum_{i=1}^{m} \sum_{x_{s_1} \in \mathcal{R}_{s_1}} \cdots \sum_{x_{s_m} \in \mathcal{R}_{s_m}} f_{s_i}(x_{s_i}) p(x) \tag{3.80}$$

$$= \sum_{i=1}^{m} f_{s_i}(x_{s_i}) p_{s_i}(x_{s_i}) \tag{3.81}$$

(Satz 3.26) ■

Als Beispiel für eine konkrete Darstellung folgt

Lemma 3.27. *Für Funktionen mit höchstens quadratischer Interaktion gibt es die Darstellung*

$$f(x) = \sum_{i,j=1}^{n} \sum_{y_i,y_j} a_{i,y_i,j,y_j} \delta(x_i = y_i, x_j = y_j) \tag{3.82}$$

Dann gilt für die durchschnittliche Fitness

$$W_f(p) = \sum_{i,j=1}^{n} \sum_{y_i,y_j} a_{i,y_i,j,y_j} p(X_i = y_i, X_j = y_j) \tag{3.83}$$

Beweis: Es gilt

$$W_f(p) = \sum_x f(x)p(x) = \sum_x \sum_{i,j} \sum_{y_i,y_j} a_{i,y_i,j,y_j} \delta(x_i = y_i, x_j = y_j)p(x) \tag{3.84}$$

$$= \sum_{i,j} \sum_{y_i,y_j} a_{i,y_i,j,y_j} \sum_x \delta(x_i = y_i, x_j = y_j)p(x) \tag{3.85}$$

$$= \sum_{i,j} \sum_{y_i,y_j} a_{i,y_i,j,y_j} p(X_i = y_i, X_j = y_j) \tag{3.86}$$

Daraus folgt die Behauptung. (Lemma 3.27) ■

Diese Darstellung lässt sich analog erweitern für Korrelationen höherer Ordnung.

3.4.3 Analyse der dynamischen Verteilungen

Beispiel 3.6. Die Differenzengleichung für proportionale Selektion aus Lemma 3.12 ergibt für die lineare Funktion

$$\mathit{Linear}(x) = \alpha_0 + \sum_{i=1}^{n} \alpha_i x_i \tag{3.87}$$

das System

$$q_i(t+1) = q_i(t) + q_i(t)\frac{(1 - q_i(t))\alpha_i}{W_f(t)}, \tag{3.88}$$

weil die Verteilung hier die einfache Produktverteilung ist. $W_f(t)$ ergibt sich deshalb durch Gleichung (3.12) auf Seite 17.

Die Lösung zu diesem System kann numerisch leicht bestimmt werden, es gibt für den allgemeinen Fall aber keine analytische geschlossene Lösung. Für die *OneMax*-Funktion gilt (siehe auch (Mühlenbein, 1998))

$$q_i(t) = 1 - \tfrac{1}{2}(1 - {}^1\!/_n)^t \tag{3.89}$$

Ansonsten wurde in (Mühlenbein and Mahnig, 1999a) folgende approximative Lösung für einen Spezialfall gezeigt:

Lemma 3.28. *Für $\alpha_0 = 0, \alpha_i > 0$ und $q_i(0) = 1/2$ gilt als Dynamik für lineare Funktionen approximativ*

$$q_i(t) \simeq 1 - r_0 a_i^t \qquad a_i = \frac{\alpha - \alpha_i\left(1 + \frac{\alpha^2-\alpha_i^2}{\alpha^2+\beta}\right)}{\alpha - \alpha_i} \qquad r_i = \frac{1 - \frac{\alpha_i}{\alpha}}{2a_i} \tag{3.90}$$

mit

$$\alpha := \sum_j \alpha_j \qquad \beta := \sum_j \alpha_j^2 \tag{3.91}$$

Beweis: Aus Gleichung (3.88) folgt

$$q_i(t+1) = q_i(t)\left(1 + \alpha_i \frac{1 - q_i(t)}{W(t)}\right) \tag{3.92}$$

Für die ersten beiden Generationen ergibt sich

$$q_i(1) = \tfrac{1}{2}\left[1 + \alpha_i \frac{1}{\alpha}\right] \tag{3.93}$$

$$= \tfrac{1}{2}\left(1 + \tfrac{\alpha_i}{\alpha}\right) = 1 - \tfrac{1}{2}\left(1 - \tfrac{\alpha_i}{\alpha}\right) \tag{3.94}$$

$$q_i(2) = \tfrac{1}{2}\left(1 + \tfrac{\alpha_i}{\alpha}\right)\left[1 + \alpha_i \frac{\frac{1}{2}(1 - \frac{\alpha_i}{\alpha})}{\frac{1}{2}\sum_j \alpha_j(1 + \frac{\alpha_j}{\alpha})}\right] = \tfrac{1}{2}\left(1 + \tfrac{\alpha_i}{\alpha}\right)\left[1 + \alpha_i \frac{\alpha - \alpha_i}{\alpha^2 + \beta}\right] \tag{3.95}$$

$$= \tfrac{1}{2} + \tfrac{1}{2}\tfrac{\alpha_i}{\alpha} + \tfrac{1}{2}\tfrac{\alpha_i}{\alpha}\frac{\alpha^2 - \alpha_i^2}{\alpha^2 + \beta} = 1 - \tfrac{1}{2}\left[1 - \tfrac{\alpha_i}{\alpha}\left(1 + \frac{\alpha^2 - \alpha_i^2}{\alpha^2 + \beta}\right)\right] \tag{3.96}$$

Die Lösung für *OneMax* legt den Ansatz $q_i(t) = 1 - r_i a_i^t$ nahe. Es folgt

$$r_i a_i = \tfrac{1}{2}\left(1 - \tfrac{\alpha_i}{\alpha}\right) \tag{3.97}$$

$$r_i a_i^2 = \tfrac{1}{2}\left[1 - \tfrac{\alpha_i}{\alpha}\left(1 + \frac{\alpha^2 - \alpha_i^2}{\alpha^2 + \beta}\right)\right] \tag{3.98}$$

Durch Dividieren der zweiten Gleichung durch die erste bekommt man a_i und damit dann r_i. (Lemma 3.28) ∎

Für *OneMax* ist die durch das Lemma erhaltene approximative Lösung exakt. Dies kann man als Rechtfertigung für den gewählten Ansatz sehen.

Als weiteres Beispiel betrachte $n = 10, \alpha_0 = 0, \alpha_i = 2^{i-1}$. Es gilt also

$$\mathit{Int}(x_1, \ldots, x_5) = \sum_i 2^{i-1} x_i \qquad \text{oder} \qquad \mathit{Int}(x) = x \tag{3.99}$$

Aus Lemma 3.28 ergeben sich folgende Werte für $a_8, \ldots, a_{10}$ und $q_8, \ldots, q_{10}$:

$a_{10} = 0.437$	$q_{10} = 0.571$
$a_9 = 0.765$	$q_9 = 0.490$
$a_8 = 0.894$	$q_8 = 0.489$

(3.100)

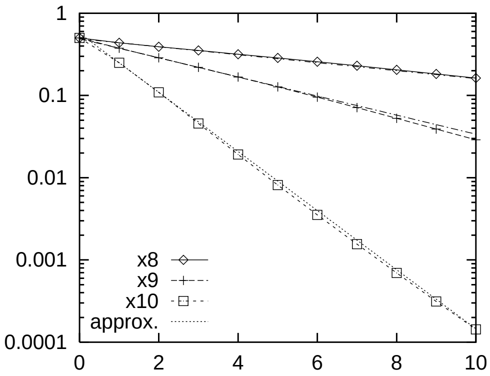

Abbildung 3.1: Approximative und exakte Bithäufigkeiten für die Funktion *Int*, Gleichung (3.99)

Ein Vergleich der Approximation mit den exakten Werten aus einem Lauf sind in Abbildung 3.1 gezeigt. Die Häufigkeiten $(1-q_i(t))$ sind logarithmisch aufgetragen und ergeben einen nahezu linearen Graphen wie vorhergesagt.

3.5 Methoden der Populationsgenetik

Die Bestimmung der dynamischen Gleichung für *UMDA* ist für andere Selektionsverfahren als proportionale Selektion außerordentlich schwierig. In der Populationsgenetik sind eine Reihe von approximativen Verfahren entwickelt worden. Zentrale Variable ist dabei ebenfalls die **durchschnittliche Fitness** einer Population, die dort nach S. Wright mit W bezeichnet wird. Die Generationen entsprechen dabei unterschiedlichen Populationen $\mathfrak{X}(t)$, die wiederum äquivalent zu Verteilungen sind.

$$W_f(\mathfrak{X}(t)) := \sum_{i=1}^{N} \frac{f(\mathfrak{X}^i(t))}{N} \tag{3.101}$$

Als nächstes wird nach (Falconer, 1981) das **Selektionsdifferenzial** benötigt. Es misst die Differenz zwischen der durchschnittlichen Fitness der selektierten Population $\hat{\mathfrak{X}}(t)$ zu der durchschnittlichen Fitness der gesamten Population $\mathfrak{X}(t)$.

$$S_f(t) := W_f(\hat{\mathfrak{X}}(t)) - W_f(\mathfrak{X}(t)) \tag{3.102}$$

Der **Selektionserfolg** $R_f(t)$ ist dann

$$R_f(t) = W_f(\mathfrak{X}(t+1)) - W_f(\mathfrak{X}(t)) \tag{3.103}$$

Als nächstes wird die folgende Regression gemacht, die sogenannte **Züchter-Gleichung**:

$$R_f(t) = b(t)S_f(t) \tag{3.104}$$

Dabei bezeichnet $b(t)$ die **realisierte Heritabilität**. Sie quantifiziert, wieviel von dem Gewinn an Fitness durch Selektion in der nächsten Generation übrig bleibt. Bei Züchtern beschreibt dies, wieviel an Fitnessgewinn vererbt werden kann. Für den *UMDA* entsteht durch das Erzeugen von neuen Punkten nach der einfachen Produktverteilung hier ein Verlust. Näherungsweise gilt dabei unter bestimmten Voraussetzungen

$$S_f(t) \approx I_\tau \sqrt{\sigma_f^2(t)} \tag{3.105}$$

wobei I_τ die **Selektionsintensität** ist und σ_f^2 die Varianz der Fitness-Verteilung. Die Beziehung zwischen τ und I_τ hängt von der Fitness-Verteilung ab und kann zum Beispiel für die Normalverteilung berechnet werden (Mühlenbein, 1998). In Tabelle 3.1 sind einige Werte dargestellt.

τ	0.05	0.125	0.3	0.38	0.5	0.6
I_τ	2.063	1.647	1.159	1.002	0.798	0.664

Tabelle 3.1: Zusammenhang zwischen τ und I_τ für die Normalverteilung

Zusammen ergibt sich die **Gleichung für den Selektionserfolg:**

$$R_f(t) \approx b(t) I_\tau \sqrt{\sigma_f^2(t)} \tag{3.106}$$

Die Schwierigkeit besteht nun darin, $b(t)$ zu schätzen. Züchter schätzen

$$b(t) \approx \frac{V_A(t)}{\sigma_f^2(t)}. \tag{3.107}$$

$V_A(t)$ ist dabei die additive genetische Varianz.

$$V_A(t) = \sum_{i=1}^{n} \sum_{x_i=0}^{l_i} p(x_i,t) F_i(x_i,t)^2, \tag{3.108}$$

wobei

$$F_i(x_i,t) = \left(\sum_{x|X_i=x_i} f(x) \prod_{j\neq i} p_j(x_j) \right) - W_f(t) \tag{3.109}$$

in Lemma 3.7 definiert wurde. Der Satz von Fisher (1958) aus der Populationsgenetik sagt:

Satz 3.29 (Fundamentalsatz der natürlichen Selektion). *Der Selektionserfolg für proportionale Selektion ist näherungsweise gegeben durch*

$$R_f(t) \approx \frac{V_A(t)}{W_f(t)} \tag{3.110}$$

Bestimme $R_f(t)$ für proportionale Selektion exakt. Für den Beweis sind einige Lemmata notwendig.

Lemma 3.30. *Mit $\Delta q_{i,y_i}(t) := q_{i,y_i}(t+1) - q_{i,y_i}(t)$ gilt für den Selektionserfolg bei proportionaler Selektion*

$$R_f(t) = \sum_{k=1}^{n} \frac{1}{k!} \sum_{\substack{J \subseteq I \\ |J|=k}} \sum_{y_J \in \mathcal{R}_J^u} \left(\prod_{j \in J} \Delta q_{j,y_j}(t) \right) \left(\prod_{j \in J} \frac{\partial}{\partial q_{j,y_j}} \right) W_f^u(q(t)) \tag{3.111}$$

Beweis: Die multidimensionale Taylorformel für R liefert zunächst

$$R_f(t) = \sum_{k \geq 1} \frac{1}{k!} \left(\sum_{i \in I} \sum_{y_i=1}^{l_i} \Delta q_{i,y_i}(t) \frac{\partial}{\partial q_{i,y_i}} \right)^k W_f^u(q(t)) \tag{3.112}$$

Weil $\partial^2 W / \partial q_{i,y_i} \partial q_{i,z_i} = 0$ kommt in der mehrdimensionalen Ableitung immer nur je ein Index i vor. In der k-ten Potenz müssen also k Indizes aus I ausgewählt werden, die dazu passenden y_i-Werte sind aber beliebig. Das entspricht der Summation aus Lemma 3.30 und beendet den Beweis. (Lemma 3.30) ■

Für proportionale Selektion ergibt sich nach (3.45) ein spezieller Ausdruck für $\Delta q_{i,y_i}$. Für binäre Variablen hat Mühlenbein (1998) die Dynamik untersucht. In (Mühlenbein and Mahnig, 1999a) erfolgte die folgende Erweiterung auf beliebige diskrete Wertebereiche.

Es gilt folgender Satz:

Satz 3.31. *Für UMDA mit proportionaler Selektion ist der Selektionserfolg $R_f(t) = W_f^u(q(t+1)) - W_f^u(q(t))$ gegeben durch*

$$\begin{aligned} R_f(t) = {} & \frac{V_A(t)}{W_f^u(t)} + \frac{1}{2} \sum_{i,j \in I} \sum_{\substack{y_i \in \mathcal{R}_i^u \\ y_j \in \mathcal{R}_j^u}} \frac{q_{i,y_i}(t) F(y_i,t) q_{j,y_j}(t) F(y_j,t)}{W_f^u(t)^2} \frac{\partial^2 W_f^u(t)}{\partial q_{i,y_i} \partial q_{j,y_j}} \\ & + \sum_{k=3}^{n} \frac{1}{k! \cdot W_f^u(t)^k} \sum_{|J|=k} \sum_{y_J \in \mathcal{R}_J^u} \left(\prod_{j \in J} q_{j,y_j}(t) F_j(y_j,t) \right) \prod_{j \in J} \left(\frac{\partial}{\partial q_{j,y_j}} \right) W_f^u(t) \end{aligned} \tag{3.113}$$

Beweis: Aus (3.45) folgt, dass

$$\Delta q_{i,y_i}(t) = q_{i,y_i}(t) \cdot \frac{F_i(y_i,t)}{W_f(t)} \tag{3.114}$$

Mit Lemma 3.30 ist der Beweis für die Ableitungen von Grad 2 und höher damit bereits fertig. Nach Lemma 3.7 gilt

$$\frac{\partial W_f^u}{\partial q_{i,y_i}} = \bar{f}_i(y_i,t) - \bar{f}_i(0,t) = F_i(y_i,t) - F_i(0,t) \tag{3.115}$$

Der erste Term von $R_f(t)$ ist deshalb

$$\sum_{i\in I}\sum_{y_i\in\mathcal{R}_i^u}\Delta q_{i,y_i}(t)\frac{\partial W_f^u(t)}{\partial q_{i,y_i}} = \frac{1}{W_f^u(t)}\sum_{i\in I}\sum_{y_i\in\mathcal{R}_i^u} q_{i,y_i}(t)F_i(y_i,t)\left[F_i(y_i,t)-F_i(0,t)\right] \tag{3.116}$$

Aus (3.30) folgt

$$W_f^u(t) = \sum_{y_i\in\mathcal{R}_i^u} q_{i,y_i}(t)\Big(F_i(y_i,t)-F_i(0,t)\Big) + F_i(0,t) + W_f^u(t) \tag{3.117}$$

$$\Longrightarrow \sum_{y_i\in\mathcal{R}_i^u} q_{i,y_i}(t)F_i(y_i,t) = -\Big(1-\sum_{y_i\in\mathcal{R}_i^u} q_{i,y_i}(t)\Big)F_i(0,t) = -p_i(0,t)F_i(0,t) \tag{3.118}$$

Deshalb ergibt sich

$$\sum_{i\in I}\sum_{y_i\in\mathcal{R}_i^u}\Delta q_{i,y_i}(t)\frac{\partial W_f^u(t)}{\partial q_{i,y_i}} = \sum_{i\in I}\sum_{y_i\in\mathcal{R}_i^u} q_{i,y_i}(t)F_i(y_i,t)^2 - \sum_{i\in I}F_i(0,t)\sum_{y_i\in\mathcal{R}_i^u} q_{i,y_i}(t)F_i(y_i,t) \tag{3.119}$$

$$= \sum_{i\in I}\sum_{y_i\in\mathcal{R}_i} p_i(y_i,t)F_i(y_i,t)^2 \tag{3.120}$$

(Satz 3.31) ■

Korollar 3.32. *Für eine generalisierte lineare Funktion*

$$GLin(x) = \sum_{i=1}^{n} f_i(x_i) \tag{3.121}$$

ist der Selektionserfolg für UMDA mit proportionaler Selektion gegeben durch

$$R_{GLin}(t) = \frac{V_A(t)}{W_f(t)} \tag{3.122}$$

Beweis: Es gilt für $y_i \in \mathcal{R}_i^u$: $\frac{\partial W(t)}{\partial q_{i,y_i}} = \bar{f}_i(y_i,t) - \bar{f}_i(0,t)$. Es folgt für $y_i \in \mathcal{R}_i$

$$\bar{f}_i(y_i,t) = \sum_{i=1}^{n}\sum_{x|x_i=y_i} f_i(x_i)\bar{q}_i(x,t) \tag{3.123}$$

$$= f_i(y_i) + \sum_{k\neq i}\sum_{x|x_i=y_i} f_k(x_k)\bar{q}_i(x,t) \tag{3.124}$$

Für $y_i = 0$ folgt

$$\bar{f}_i(0,t) = f_i(0) + \sum_{k\neq i}\sum_{x|x_i=0} f_k(x_k)\bar{q}_i(x,t) \tag{3.125}$$

Zusammen ergibt sich

$$\frac{\partial W(q)}{\partial q_{i,y_i}} = f_i(y_i,t) - f_i(0,t) \tag{3.126}$$

Damit sind die ersten Ableitungen von $W(q)$ unabhängig von q und somit verschwinden alle höheren Ableitungen. (Korollar 3.32) ■

Weiterhin ist in der Populationsgenetik bekannt:

Lemma 3.33. *Für proportionale Selektion ist das Selektionsdifferenzial $S_f(t)$ gegeben durch*

$$S_f(t) = \frac{\sigma_f^2(t)}{W_f(t)} \tag{3.127}$$

Beweis: Es gilt

$$S_f(t) = \left(\sum_{x\in\mathcal{R}} p(x,t)\frac{f(x)^2}{W_f(t)}\right) - W_f(t) \tag{3.128}$$

$$= \sum_{x\in\mathcal{R}} p(x,t)\frac{f(x)^2 - W_f(t)^2}{W_f(t)} \tag{3.129}$$

$$= \frac{\sigma_f^2(t)}{W_f(t)} \tag{3.130}$$

(Lemma 3.33) ■

Beim *UMDA* muss man unterscheiden zwischen den Verteilungen der neu generierten Punkte und der Verteilung der selektierten Punkte. Die Verteilung der selektierten Punkte ist im Allgemeinen keine einfache Produktverteilung. Wenn die selektierten Punkte aber eine einfache Produktverteilung darstellen, so zeigt das Lemma, dass die durchschnittliche Fitness in diesem Fall für proportionale Selektion nicht abnimmt.

Eine ausführlichere empirische Betrachtung der additiven Varianz und Heritabilität findet sich in (Mühlenbein and Mahnig, 2000a).

3.5.1 Eine schwache Variante des Satzes von Fisher

Satz 3.31 zeigt, dass der Satz von Fisher (Satz 3.29) der erste Term einer Reihenentwicklung ist, wie schon in (Mühlenbein, 1998) für den binären Fall bewiesen wurde. Wenn die Terme höherer Ordnung aber nicht klein werden, ist die Approximation sehr schlecht.

Für beliebige Funktionen kann man aber zeigen, dass der Selektionserfolg immer größer oder gleich Null ist, wenn die dynamischen Gleichungen (3.45) benutzt werden. Der Beweis aus (Mühlenbein and Mahnig, 1999a) basiert auf einem allgemeineren Resultat von Baum und Eagon (1967), umgeschrieben in die Notation dieser Arbeit.

Satz 3.34 (Baum, Eagon). *Es sei $W^{u0}(q) = W^{u0}(\{q_{ij}\})$ ein Polynom mit nichtnegativen Koeffizienten, homogen vom Grad n in den Parametern q_{ij} für $i \in I$, $j \in \mathcal{R}_i$. Es sei $q = \{q_{ij}\}$ ein Punkt mit $q_{ij} \geq 0$, $\sum_j q_{ij} = 1$. Es sei q' der Punkt mit den Koordinaten*

$$q'_{ij} = \frac{q_{ij} \frac{\partial W^{u0}(q)}{\partial q_{ij}}}{\sum_{k \in \mathcal{R}_i} q_{ik} \frac{\partial W^{u0}(q)}{\partial q_{ik}}} \tag{3.131}$$

Dann gilt $W^{u0}(q') > W^{u0}(q)$ falls $q' \neq q$.

Satz 3.35. *Für UMDA und proportionale Selektion gilt $R_f(t) > 0$, es sei denn, alle univariaten Randverteilungen bleiben gleich.*

Beweis: Der Satz folgt direkt aus dem Satz von Baum und Eagon, wenn man $q_{ij} = q_{i,x_i}$ setzt. Gleichung (3.48) (Seite 22) ist dabei identisch zu (3.131). (Satz 3.35) ■

UMDA hat also die Eigenschaft, dass die durchschnittliche Fitness mit proportionaler Selektion niemals abnimmt. Dies lässt sich auch als Abbruchkriterium benutzen.

3.5.2 Die Analyse von *Truncation Selection*

Truncation Selection ist ein robusteres Selektionsverfahren als proportionale Selektion, da es sich um die absoluten Werte der Fitness-Funktion nicht kümmert, sondern nur um die Ordnungsrelation auf ihnen. Die mathematische Analyse ist leider viel schwieriger. In (Mühlenbein and Schlierkamp-Voosen, 1993b,a; Mühlenbein, 1998) wurde bereits eine approximative Untersuchung von *OneMax* gemacht. Es ergab sich

Satz 3.36. *Für* Truncation Selection *mit Selektions-Intensität I_τ ist die Dynamik der Parameter $q_i(t)$ des UMDA für alle i identisch falls $q_i(0) = q_j(0)$. Es gilt*

$$q(t+1) = q(t) + \frac{I_\tau}{n}\sqrt{n \cdot q(t)(1-q(t))}. \tag{3.132}$$

Es ergibt sich näherungsweise mit $q(0) = 1/2$

$$q(t) = \frac{1}{2}\left(1 + \sin\left(\frac{I_\tau}{\sqrt{n}} \cdot t\right)\right) \qquad \textit{mit } t \leq \frac{\pi\sqrt{n}}{2I_\tau} \tag{3.133}$$

und somit für die Zahl an Generationen zur Konvergenz

$$Gen(n, I_\tau) = \frac{\pi\sqrt{n}}{2I_\tau} \tag{3.134}$$

In Abschnitt 4.4.1 ist eine empirische Kurve dargestellt, die das gleiche Resultat liefert.

Eine einfache analytische Formel lässt sich für *Truncation Selection* leider nicht mehr aufstellen. Darum lässt sich auch nicht mehr sagen, dass die Parameter dem Gradienten

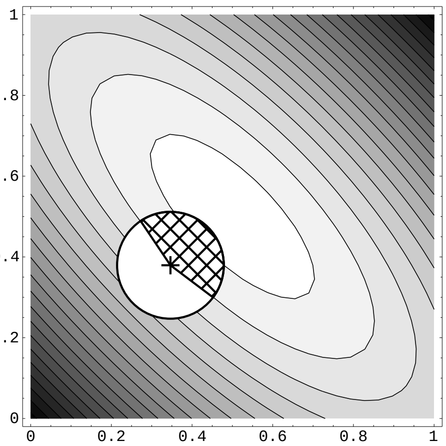

Abbildung 3.2: Veranschaulichung von *Truncation Selection*: Der Kreis stellt die Population dar, der schraffierte Teil die selektierten Individuen.

von W_f^u folgen. Es zeigt sich auch in der Praxis, dass die durchschnittliche Fitness nicht notwendigerweise wächst.

Anschaulich gesehen (siehe Abbildung 3.2) wird durch die Rekombination eine Menge Punkte um den Mittelpunkt $(q_1, \ldots, q_n)$ erzeugt, wobei die Verteilung einfach die Binomialverteilung in jedem Parameter ist. Es entsteht also näherungsweise ein Gauß-Blob um diesen Punkt (weißer Kreis). Jedes Selektionsverfahren wählt jetzt bestimmte Punkte aus und bestimmt einen neuen, gewichteten Mittelpunkt. *Truncation Selection* nimmt dabei einfach einen Anteil τ und gibt allen gewählten Punkten das gleiche Gewicht (schraffierte Fläche). Der neue Parameter ist einfach der Schwerpunkt dieser Fläche.

Dadurch folgt *Truncation Selection* ebenfalls ungefähr dem Gradienten. Wenn also der Gradient zu einem lokalen Maximum weist, so wird auch *Truncation Selection* Schwierigkeiten haben, zu dem globalen Optimum zu finden.

3.6 Analyse der Fitness-Landschaft

Die Idee, die abstrakte Fitness durch eine adaptive „Landschaft“ zu repräsentieren, ist schon von Wright. Es ist dabei ein wesentlicher Schritt, nicht mehr die statischen Eigenschaften der Fitness-Funktion zu betrachten, wie dies immer noch in der Theorie der genetischen Algorithmen üblich ist (zum Beispiel (Heckendorn, 2000; Hu et al., 2000)), sondern die Dynamik, die durch die Population *in* dieser Landschaft entsteht. Die statische Analyse entspricht dem Betrachten der dynamischen Landschaft mit einer Population in der Gleichverteilung. Diese Situation ist aber bestenfalls in der Startgeneration gegeben.

Die parametrisierte einfache Produktverteilung erlaubt eine kanonische Einbettung des Definitionsraums $\mathcal{R}$ in den Parameterraum Q:

Definition 3.37. *Für die parametrisierte einfache Produktverteilung $\tilde{p}^u$ kann $\mathcal{R}$ in den Parameterraum Q^u durch $\bar{Q}$ folgendermaßen eingebettet werden:*

$$\bar{Q} : \mathcal{R} \to Q^u \qquad \bar{Q}(x) = q \text{ mit } q_{i,y_i} = \delta(y_i = x_i) \tag{3.135}$$

wobei $\delta(x=y)$ die Kronecker-Funktion ist mit $\delta(x=y) = 1$ für $x=y$ und 0 sonst.

Beispiel 3.7. Für binäres $\mathcal{R}_i$ vereinfacht sich die Einbettung zu

$$x \in \mathcal{R},\ \bar{Q}(x) = q \implies q_i = x_i \tag{3.136}$$

Zunächst gilt folgendes Lemma:

Lemma 3.38. *Für die Verteilung $\tilde{p}^u$ und die Einbettung $\bar{Q}$ gilt:*

$$f(x) = W_f^u(\bar{Q}(x)), \tag{3.137}$$

die Funktion $W_f^u(q)$ ist also eine Erweiterung der Funktion $f(x)$ auf einen vergrößerten Definitionsbereich.

Beweis: Es gilt

$$\tilde{p}^u(x, \bar{Q}(y)) = \prod_{i=1}^{n} \left[\prod_{z_i=1}^{l_i} \delta(z_i = y_i)^{\delta(x_i = z_i)} \right] \left(1 - \sum_{z_i=1}^{l_i} \delta(z_i = y_i) \right)^{\delta(x_i = 0)} \tag{3.138}$$

$$= \prod_{i=1}^{n} \delta(x_i = y_i) \quad = \delta(x = y) \tag{3.139}$$

und somit

$$W_f^u(\bar{Q}(y)) = \sum_{x \in \mathcal{R}} f(x) \tilde{p}^u(x, \bar{Q}(y)) = \sum_{x \in \mathcal{R}} f(x) \delta(x = y) = f(y) \tag{3.140}$$

(Lemma 3.38) ■

Somit kann man das *diskrete* Optimierungsproblem $\max_{x \in \mathcal{R}} f(x)$ auch als *kontinuierliches* Problem $\max_{q \in Q} W_f^u(q)$ auffassen. Eine Lösung im Bildbereich von $\bar{Q}$ entspricht nach diesem Lemma dann einer Lösung in $\mathcal{R}$. Der Vorteil ist unter anderem der, dass $W_f^u(q)$ eine stetig differenzierbare Funktion darstellt und somit diverse Techniken wie Gradientensuche denkbar sind.

Es stellt sich aber die Frage, was passiert, wenn die Maximierung von $W_f^u(q)$ eine Lösung im Inneren von Q^u liefert. Wie sich zeigt, ist so ein Fall aber nicht möglich. Zur einfacheren Darstellung sei zunächst der binären Spezialfall diskutiert.

3.6.1 Die durchschnittliche Fitness für binäre Variablen

Im binären Fall lässt sich die Darstellung aus Lemma 3.18 verwenden.

$$W_f^u(q) = \sum_{s \subseteq I} a_s \prod_{i \in s} q_i \tag{3.141}$$

$W_f^u(q)$ ist linear in den q_i, also hängen die partiellen Ableitungen von $W_f^u(q)$ nach q_i nicht mehr von q_i ab. Es gilt daher $\frac{\partial^2}{\partial q_i \partial q_i} W_f^u(q) = 0$.

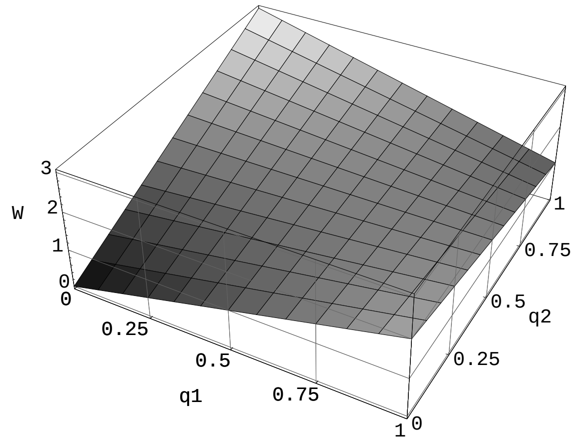

Abbildung 3.3: Die durchschnittliche Fitness $W_f^u(q_1, q_2) = 2 \cdot q_1(1 - q_2) + 3 \cdot q_2(1 - q_1) + 1 \cdot q_1 q_2$

Die Fitnesslandschaft besteht also aus Geradenstücken, die sich eventuell gegeneinander drehen. Betrachte als Beispiel Abbildung 3.3. Die Fitnesswerte sind durch Graustufen markiert. Es gilt

$$W_f^u(q_1, q_2) = 2q_1(1 - q_2) + 3q_2(1 - q_1) + 1q_1q_2 = 2q_1 + 3q_2 - 4q_1q_2 \tag{3.142}$$

Ein kritischer Punkt (mit verschwindendem Gradienten) ist bei $\hat{q} = (3/4, 1/2)$, erkennbar an der gleichbleibenden Graustufe parallel zu den Achsen. Um den kritischen Punkt entwickelt ist die Fitness gleich

$$W_f^u(3/4 + h_1, 1/2 + h_2) = \frac{3}{2} - 4 \cdot h_1 h_2 \tag{3.143}$$

Man sieht daran sofort, dass es sich um einen Sattelpunkt handeln muss. Daher ist das globale Maximum der *diskreten* Fitnessfunktion ein lokales Extremum dieser Landschaft.

Es gibt allerdings noch den Fall, dass das Maximum nicht eindeutig ist und zwei Maxima sich nur in einem Bit unterscheiden. Betrachte zur Illustration Abbildung 3.4. Das globale Maximum hat Fitnesswert 4, ist aber nicht isoliert. Da die Verbindungsstrecke für festes q_2 ein Geradenabschnitt ist, muss die Fitness entweder bei beiden Eckpunkten verschieden

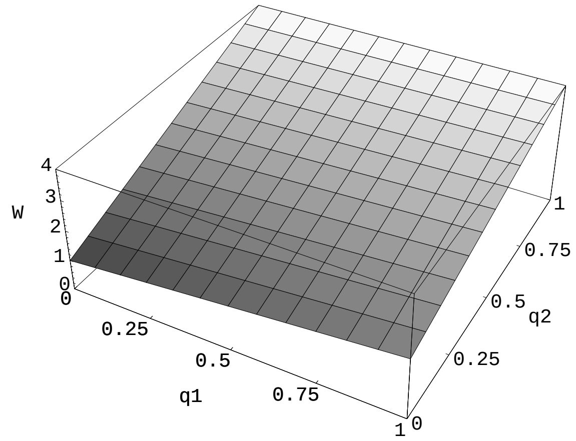

Abbildung 3.4: $W_f^u(q_1, q_2, 1) = 1 + q_1 + 3q_2 - q_1 q_2$

sein (wie in Bild 3.3), oder sie ist auf der ganzen Strecke gleich. Deshalb gibt es kein lokales Extremum in der Fitness-Landschaft für diesen Fall. Dies ist aber auch nicht so wichtig, weil jede Wahl von q_1 (für $q_2 = 1$) den Maximalwert liefert. Wenn man also die Fitnesslandschaft einschränkt auf die relevante Variable q_2 (für ein beliebiges, festes q_1), so ist $q_2 = 1$ lokales Maximum der neuen Landschaft.

Beispiel 3.8. Betrachte $W_f^u(q_1, q_2, q_3) = q_1 + q_2 + q_3 + 2q_2q_3 - q_1q_2q_3$. Die Maxima von $f(x)$ sind $f(0,1,1) = f(1,1,1) = 4$. Es gilt $W_f^u(q_1, q_2, 0) \leq W_f^u(q_1, q_2, 1)$, also gilt für ein Maximum $\hat{q}$: $\hat{q}_3 = 1$. In Abbildung 3.4 ist $W_f^u(q_1, q_2, 1)$ dargestellt. Es gilt $W_f^u(q_1, 1, 1) = 4$ unabhängig von q_1, da ja auch das Maximum nicht eindeutig war. Beachte, dass $f(x)$ und $W_f^u(q)$ nicht global unabhängig von x_1 beziehungsweise q_1 sind, nur an der Stelle des Maximums.

Nach diesen Vorüberlegungen gilt folgender Satz.

Satz 3.39. *Mit der Einbettung $\bar{Q}$ ist $W_f^u(q)$ für binäres x_i eine Erweiterung von $f(x)$ auf den Einheitswürfel $[0,1]^n$. Im Innern von Q^u gibt es keine lokalen Extrema der Funktion W_f^u, daher gilt*

$$\forall q \in Q^u : W_f^u(q) \leq \max_{x \in \mathcal{R}} f(x) \tag{3.144}$$

Für jedes Maximum $x_{\max}$ gibt es eine Menge von Variablen X_c mit $c \subseteq I$ (die auch leer sein kann), so dass die Fitness an der Stelle $x_{\max}$ nicht von den Variablen in X_c abhängt. In allen anderen Variablen ist $x_{\max}$ ein lokales Optimum.

Beweis: Betrachte ein lokales Extremum $\hat{q}$ im Inneren. Eine notwendige Bedingung dafür ist, dass der Gradient verschwindet. Mache nun eine Taylor-Entwicklung um diesen Punkt $\hat{q}$. Die Koeffizienten erster Ordnung verschwinden, weil der Gradient Null ist; mehrfache partielle Ableitungen nach der gleichen Variablen verschwinden ebenfalls. Mit q und h vektoriell bleibt

$$W_f^u(\hat{q}+h) = W_f^u(\hat{q}) + \sum_{i<j} c_{i,j} h_i h_j + \sum_{i<j<k} c_{i,j,k} h_i h_j h_k + \cdots + c_{1,2,\ldots,n} h_1 h_2 \cdots h_n \tag{3.145}$$

Weil $W_f^u(q)$ ein Polynom ist, ist dieser Ausdruck exakt. Angenommen, mindestens ein $c_{i,j}$ ist ungleich Null, zum Beispiel $c_{1,2}$. Mit $\hat{h} = (h_1, h_2, 0, \ldots, 0)$ folgt

$$W_f^u(\hat{q}+\hat{h}) = W_f^u(\hat{q}) + c_{1,2} h_1 h_2 \tag{3.146}$$

Weil $c_{i,j} h_1 h_2$ beim Ursprung das Vorzeichen wechselt, kann $\hat{q}$ kein Extremum sein, also ist es ein Sattelpunkt. Dies passiert, weil in der Taylorentwicklung keine echten quadratischen Terme auftauchen.

Falls alle $c_{i,j}$ Null sind, aber eines der $c_{i,j,k}$ nicht, kann man das Argument wiederholen mit $W_f^u(\hat{q}+\hat{h}) = W_f^u(\hat{q}) + c_{i,j,k} h_i h_j h_k$, was ebenfalls das Vorzeichen bei Null wechselt. Ansonsten wiederhole diesen Prozess analog. Wenn sogar der letzte Koeffizient $c_{1,\ldots,n}$ verschwindet, gilt $W_f^u(\hat{q}+\hat{h}) = W_f^u(\hat{q})$; dann ist $W_f^u(q)$ aber konstant und daher auch $f(x)$ eine konstante Funktion ohne lokales Extremum.

Es ist jetzt also klar, dass es im Inneren kein Extremum gibt. Betrachte nun ein Extremum $\hat{q}$ am Rand, das heißt, einige der $\hat{q}_i$ sind 0 oder 1. Ohne Einschränkung sind die m Parameter q_1 bis q_m ungleich 0 und 1. Falls $m = 0$, handelt es sich um einen Eckpunkt und der Beweis ist fertig. Setze $V(q_1, \ldots, q_m) := W_f^u(q_1, \ldots, q_m, \hat{q}_{m+1}, \ldots, \hat{q}_n)$. Mit obiger Argumentation für V anstelle von W^u folgt, dass $(\hat{q}_1, \ldots, \hat{q}_m)$ ein Sattelpunkt von V sein muss und daher $\hat{q}$ auch Sattelpunkt von $W_f^u(q)$ sein muss im Widersprich zur Annahme; oder die Fitness hängt nicht von $(q_1, \ldots, q_m)$ ab bei $\hat{q}$. Damit zerfällt X in die zwei Mengen wie im Satz beschrieben. (Satz 3.39) ■

Beispiel 3.9. Wenn sich die Fitness beim Kippen eines Bits nicht ändert, so ergibt sich in der Fitnesslandschaft eine „waagerechte“ Gerade mit gleicher Höhe, die die Eckpunkte verbindet. Wenn mehrere Bits diese Eigenschaft haben, so entsteht eine achsenparallele Ebene. Das heisst aber nicht, dass immer zwei Punkte mit gleicher Fitness durch einen Ebenenabschnitt verbunden werden, wie Abbildung 3.5 illustriert. Entscheidend für die Ausbildung eines Plateaus ist es, dass die jeweiligen Punkte direkt benachbart sind.

3.7 Diskussion und weitere Beispiele

Die Tatsache, dass die Fitnesslandschaft aus Geradenstücken besteht, bedeutet nicht, dass die Optimierung einfach ist. Abbildung 3.3 zeigt, dass ein Gradientenverfahren bei $(q_1 = 0.8, q_2 = 0.2)$ unweigerlich zum lokalen Optimum $(1, 0)$ führt und nicht zu $(0, 1)$ (das Beispiel lässt sich leicht anpassen, so dass vom Startpunkt $(0.5, 0.5)$ aus das globale Optimum nicht erreicht wird).

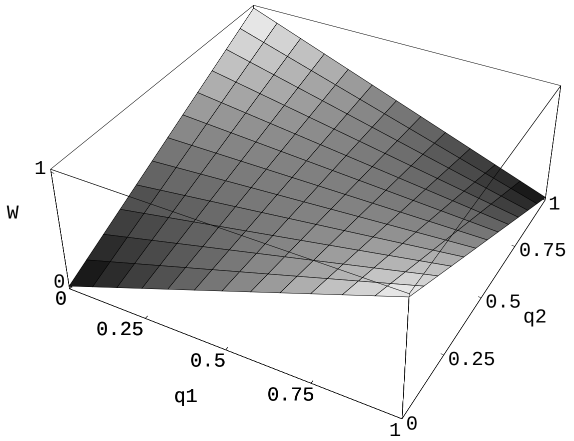

Abbildung 3.5: $W_f^u(q_1, q_2, 1) = q_1 + q_2 - 2 \cdot q_1 q_2$

Ein Maximum stellt aber für seine Umgebung einen Attraktor dar. Aus Satz 3.39 folgt nämlich

Korollar 3.40. *Es sei $\hat{q}$ ein lokales Maximum in der Fitnesslandschaft $W_f^u(q)$. Dann liegt $\hat{q}$ in einer Ecke des Einheitswürfels, entspricht also einem Punkt $\hat{x}$ des Optimierungsproblems $f(x)$. Dieser Punkt ist auch ein lokales Maximum in dem Raum $\mathcal{R}$, einzelne Bitänderungen verschlechtern also die Fitness.*

Man muss unterscheiden zwischen Attraktoren der Dynamik aus Lemma 3.12 und lokalen Maxima. Die Sätze dieses Abschnitts geben Auskunft über die Landschaft von W_f^u und damit über das Verhalten eines Gradientenverfahrens für die Parameter q_i. Bei proportionaler Selektion wird der Gradient aber noch gewichtet, um im Intervall $[0, 1]$ zu bleiben, wie Gleichung (3.47) auf Seite 22 zeigt.

3.7.1 *Unitations*-Funktionen

Für Funktionen, die symmetrisch in den Parametern sind, ist das Gleichsetzen der Variablen q_i eine wertvolle Hilfe. Weil nämlich die dynamischen Gleichungen alle identisch sind und alle Parameter an der gleichen Stelle $1/2$ starten, kann man so tun, als ob es nur *einen* Parameter gäbe. Dies gilt insbesondere für *Unitations*-Funktionen, also Funktionen, bei denen die Fitness nur von der *Anzahl* an gesetzten Bits abhängt:

$$f(x) = g(u) \qquad \text{mit } u = \sum_{i=1}^{n} x_i \tag{3.147}$$

Die Anzahl der Parameter ist hier drastisch von 2^n auf $n + 1$ reduziert. Dennoch sind auch schwierige Optimierungsaufgaben dabei, als Extremfall die Funktion, die überall Null ist und nur an einer Stelle Eins.

In der Regel sind die *Unitations*-Funktionen so konstruiert, dass das globale Optimum bei $(1,\ldots,1)$, also $u{=}n$, liegt. Dann ist es nämlich eindeutig, ansonsten gibt es $\binom{n}{u}$ gleich gute Lösungen. Dabei ist die genaue Bitsequenz aber egal, man kann die Familie leicht erweitern, in dem man für beliebiges $\hat{x}$ setzt:

$$\hat{f}(x) = f((x_1 \Leftrightarrow \hat{x}_1),\ldots,(x_n \Leftrightarrow \hat{x}_n)) \tag{3.148}$$

wobei $(x \Leftrightarrow y)$ gleich 1 ist, wenn $x{=}y$, sonst 0. Diese Funktion $\hat{f}(x)$ hat dann das eindeutige globale Maximum bei $\hat{x}$, sofern $f(x)$ ein eindeutiges Maximum bei $(1,\ldots,1)$ hat.

Bei dem Gleichsetzen der Parameter q_i muss man allerdings aufpassen, weil sich Sattelpunkte hier in lokale Maxima verwandeln können. Die Fitnesslandschaft aus Abbildung 3.5 ist ein solches Beispiel. Die resultierende Fitnesslandschaft für $q_1{=}q_2$ ist in Abbildung 3.6 dargestellt.

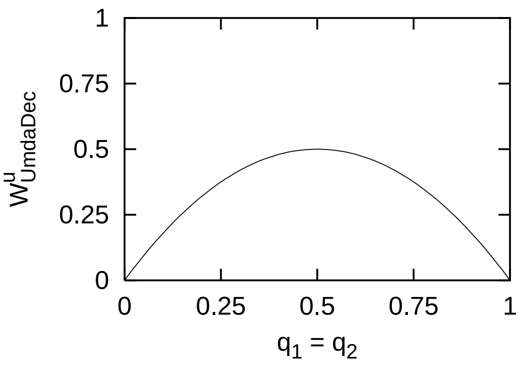

Abbildung 3.6: Fitnessfunktion $W^u_f(q) = 2q - 2q^2$ für $q := q_1 = q_2$

Man sieht, dass entlang dieser Kurve bei $q_1{=}q_2{=}\,{}^1\!/\!_2$ ein lokales Maximum entsteht. In der gesamten Fitnesslandschaft ist dies natürlich nur ein Sattelpunkt, wie Abbildung 3.5 zeigt. Es handelt sich um einen instabilen Punkt, eine winzige Schwankung von der Diagonale weg führt direkt zum globalen Optimum. So hat der *UMDA* in der Praxis auch kein Problem mit dieser Funktion, er bleibt nicht an dem Sattelpunkt stehen.

3.7.2 Die *Saw*-Funktion

OneMax ist eine *Unitations*-Funktion, ein anderer Vertreter ist die *Saw*-Funktion aus Abbildung 3.7.

Dargestellt ist hier die *Unitations*-Komponente $g(u)$. Die Fitness geht linear von 0 bis zur Peak-Höhe. Die Höhe vom letzten Gipfel ist $36 \cdot 0.85$, die vom nächsten ist $28 \cdot 0.85^2$ und so weiter. Der Abstand von Gipfel zu Tal ist in diesem Beispiel 4.

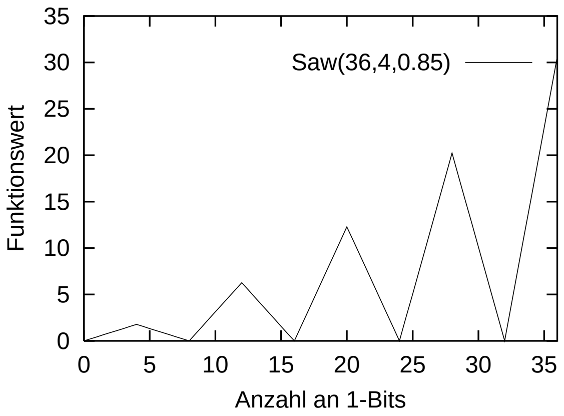

Abbildung 3.7: Definition der *Saw*-Funktion

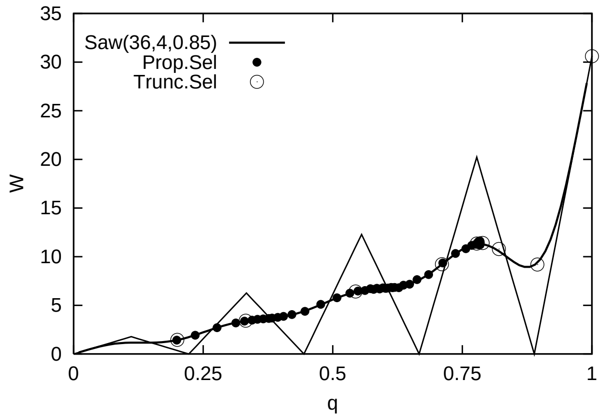

Abbildung 3.8: Fitness-Landschaft der Funktion *Saw* mit zwei *UMDA*-Läufen, siehe Abschnitt 3.7.2.

In Abbildung 3.8 ist die durchschnittliche Fitness dargestellt, wenn alle Parameter q_i gleich einem q sind. Zur Illustration ist die originale Funktion dahinter noch mal dargestellt. Man sieht deutlich, dass die durchschnittliche Fitness die Funktion *glättet*: wo vorher tiefe Täler waren, sind nun kleine Einbuchtungen.

Ein Simulationslauf (mit Populationsgröße 7000) mit proportionaler Selektion zeigt, dass sich der *UMDA* wirklich gemäß dieser Fitnesskurve verhält. Zur Veranschaulichung war die Startpopulation nicht gleichverteilt erzeugt, sondern mit $q = 0.2$ (also sind alle Bits mit 20% Wahrscheinlichkeit 1). Deutlich wird auch, dass der Algorithmus bei $q \approx 0.78$ an dem lokalen Maximum stehenbleibt. Hierbei handelt es sich um einen stabilen Attraktor der Dynamik. Im Gegensatz zu dem vorherigen Beispiel in Abbildung 3.6 gibt es hier keinen Weg zum Optimum, bei dem die durchschnittliche Fitness nicht sinkt. Bei proportionaler Fitness muss die durchschnittliche Fitness nach Lemma 3.33 aber steigen, deshalb kann das Tal nicht durchquert werden.

Gleichzeitig ist auch dargestellt, dass *Truncation Selection* diese Probleme nicht in dem gleichen Maße hat. Es ist in der Lage, das Tal zu durchqueren, wobei die durchschnittliche Fitness zwischendurch sogar kleiner wird.

3.7.3 Die *Deceptive*-Funktion

Die *Deceptive*-Funktion ist eine in der Literatur bekannte Funktion, weil sie ein Beispiel ist für eine Funktion, mit der genetische Algorithmen große Schwierigkeiten haben. Ein einfacher Repräsentant in der *Unitations*-Notation ist in Tabelle 3.2 dargestellt.

u	0	1	2	3	4
$Deceptive_4(u)$	3	2	1	0	4

Tabelle 3.2: Definition der *Deceptive*-Funktion

Die Funktion hat ihr Maximum, wenn alle Bits auf 1 sind, während die lokale Information immer in Richtung 0-Bits zeigt. Probleme mit mehr als 4 Bits erzeugt man hier, indem man mehrere Kopien der Funktion aneinanderhängt (ansonsten wird das Problem exponentiell schwieriger).

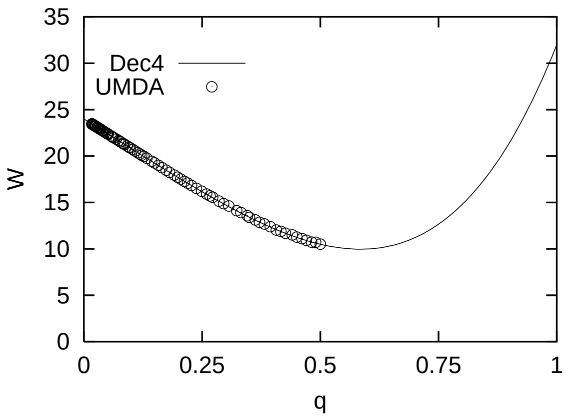

Abbildung 3.9: Die *Deceptive*-Funktion mit 32 Bits und ein *UMDA*-Lauf

Man kann das Verhalten des *UMDA* mit Hilfe der Schema-Theorie (Goldberg, 1989b) analysieren, die hier vorgestellte Technik macht die Problematik aber auf einfache Weise sehr deutlich. Betrachte eine *Deceptive*-Funktion mit 32 Bits, die also aus 8 Kopien der einfachen Funktion besteht:

$$Deceptive(x_1, \ldots, x_{32}) = \sum_{i=1}^{8} Deceptive_4(x_{4i-3}, x_{4i-2}, x_{4i-1}, x_{4i}) \tag{3.149}$$

Wie im vorigen Abschnitt kann die Gleichung der durchschnittlichen Fitness auf einen Parameter reduziert werden. Die resultierende Kurve sowie ein Lauf des *UMDA* ist in Abbildung 3.9 dargestellt, wobei hier wiederum proportionale Selektion verwandt wurde und eine Populationsgröße von 2000 Individuen.

Die Graphik zeigt deutlich, dass der *UMDA* überhaupt keine Chance hat, zu dem globalen Optimum zu finden, da er zunächst ein Tal mit niedrigerer Fitness durchqueren müsste. Das lokale Minimum liegt dabei bei $q = \sqrt[3]{1/5} \approx 0.58$.

3.7.4 Die Funktion *Dyna*

Betrachte als weiteres Beispiel einer genaueren Analyse in Abbildung 3.10 die Fitness-Funktion

$$Dyna(x_1, x_2) = 9 - 8x_1 - x_2 + 10x_1x_2. \tag{3.150}$$

Das Maximum dieser Funktion ist $f(1,1)=10$, aber $f(0,0)=9$ und $f(0,1)=8$ haben ebenfalls hohe Fitness, während $f(1,0)=1$ gilt.

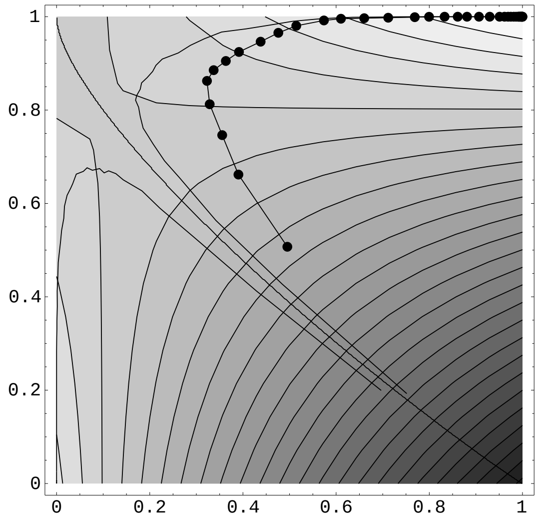

Abbildung 3.10: Dynamik und Landschaft von $Dyna(x_1, x_2) = 9 - 8x_1 - x_2 + 10x_1x_2$, diskutiert in 3.7.4.

In der Abbildung sind mehrere unterschiedliche Informationen zusammengeführt:

1. Zunächst einmal ein Konturenplot der durchschnittlichen Fitness $W^u_{Dyna}(q_1, q_2)$. Hellere Farben bedeuten höhere Fitness, man erkennt das Optimum bei $(1,1)$. Darüber sind die Isolinien eingezeichnet.

2. Die Linie, die die Punkte $(1, 0)$ und $(0, 1)$ verbindet, ist folgendermassen entstanden: Für proportionale Selektion ist die Dynamik der Parameter gegeben durch die Gleichungen aus Lemma 3.12. Ein einfaches Programm simulierte diese Dynamik für ein Gitter von Startpunkten aus $[0, 1] \times [0, 1]$. Die Parameter konvergierten dabei entweder gegen $(1, 1)$ oder $(0, 0)$. Die Linie gibt die Grenzkurve dieses Verhaltens an. Solch eine Grenzkurve nennt man *Separatrix* der Dynamik.

   Man sieht sehr schön, dass die Kurve die Isolinien ungefähr senkrecht schneidet. Der Winkel ist nicht genau 90°, weil die Dynamik nach Korollar 3.14 die Parameter vom Rand fernhält.

3. Schließlich sind noch drei Simulationsläufe eingezeichnet mit proportionaler Selektion, Populationsgröße 6000 und verschiedenen Startwerten. Einmal wurden die Startpopulation mit den Bitwahrscheinlichkeiten $(0.5, 0.5)$ initialisiert, einmal mit den Werten $(0.75, 0.2)$ und einmal mit $(0.7, 0.2)$. Der Startpunkt $(0.5, 0.5)$ entspricht der herkömmlichen Gleichverteilung, das Optimum wird hier wie erwartet erreicht. Für diesen Lauf sind auch die eigentlichen Datenpunkte eingezeichnet, die ansonsten nur durch Linien verbunden sind. Die relativ große Schrittweite am Anfang bewirkt eine Abweichung von der Gradientenrichtung, viel ausgeprägter ist die Diskrepanz aber am Ende, wo der Gradient aus dem Quader hinauszeigt. Dort bleiben die Werte natürlich am Rande stehen.

   Die anderen beiden Kurven demonstrieren, wie genau die Vorhersage ist, obwohl die Startwahrscheinlichkeiten nahe der Separatrix liegen.

Verfahren	Bitlänge	Startpunkt	Resultat
UMDA	10	$(.70, .2)$	$(0, 0)$
UMDA	10	$(.75, .2)$	$(1, 1)$
SGA	10	$(.65, .2)$	$(0, 0)$
SGA	10	$(.75, .2)$	$(1, 1)$
SGA + Muta.	10	$(.65, .2)$	$\approx (.5, .5)$
UMDA,trunc	30	$(.65, .2)$	$(0, 0)$
UMDA,trunc	30	$(.75, .2)$	$(1, 1)$

Tabelle 3.3: Verhalten der Verfahren für die Funktion *Dyna*

Die Situation in der Simulation ist sehr untypisch, weil hier mit einer sehr großen Populationsgröße (6000) ein Problem mit einer sehr kleinen Bitzahl (2) bearbeitet wurde. Damit ist bereits in der Startpopulation jedes Individuum sehr oft vertreten. Wenn sich das Verhältnis umkehrt, treten neue Effekte auf, da die Individuen nur noch vereinzelte Stichproben aus der Verteilung sind. Die Population beschreibt die *Verteilung* immer ungenauer.

Dass sich die Analyse dann immer noch zur Vorhersage eignet, zeigt Tabelle 3.3. Hier wurden mehrere Kopien der *Dyna*-Funktion aneinandergehängt. Die Startpunkte wurden dabei entsprechend aneinandergehängt.

Die ersten beiden Zeilen zeigen, dass sich der *UMDA* auch bei fünf Kopien (also Bitlänge 10) genauso verhält wie bei einer Kopie. In mehreren Läufen konvergierte der Algorithmus zum jeweils erwarteten Punkt und fand das Optimum vom Startpunkt $(.65, .2)$ auch nicht.

Die nächsten Zeilen dienen als Beleg dafür, dass sich der genetische Algorithmus auch nicht anders verhält. Ohne Mutation ist das Verhalten identisch zum *UMDA*. Die Mutation führt allerdings zu einem ganz anderen Verhalten. Mutation bewegt die Population nämlich immer zum Startpunkt hin. Dies führt hier offenbar dazu, dass der Algorithmus sich nicht mehr für eine Seite entscheiden kann.

Für *Truncation Selection* gelten die Differenzengleichungen nicht. Die Simulation zeigt aber, dass der *UMDA* für 30 Bits mit Populationsgröße 10000 von dem Startpunkt $(0.65, 0.2)$ immer gegen das lokale Maximum konvergiert, während der Algorithmus in 10 Läufen vom Startpunkt $(0.75, 0.2)$ zum Optimum konvergierte.

3.7.5 Die Funktion *XorKette*

Als weiteres Beispiel für eine schwierige Funktion folgt nun die Funktion *XorKette*. Sie besteht aus mehreren Kopien der Funktion aus Abbildung 3.5, die jetzt aber *überlappend* angeordnet sind:

$$XorKette(x) := \sum_{i=1}^{n-1}(1 + x_i + x_{i+1} - 2x_i x_{i+1}) = \sum_{i=1}^{n-1}(1 + xor(x_i, x_{i+1})) \qquad (3.151)$$

Es wird jeweils eine 1 addiert, um den Fitnesswert Null zu vermeiden, der bei proportionaler Selektion in der Regel ausgeschlossen wird. Das Modell ist in der Physik als anti-ferromagnetische Isingkette bekannt.

Die Optima sind die alternierenden Bitketten $(1, 0, 1, 0, \dots)$ und $(0, 1, 0, 1, \dots)$. Dies Funktion ist aus zwei Gründen sehr schwierig zu optimieren. Zum einen ist sie symmetrisch in den Variablen und daher verschwindet nach Korollar 3.15 der Gradient in der Startgeneration.

Zum anderen gibt es einen Konflikt zwischen *lokaler* und *globaler* Optimierung. Lokal gesehen erhöht es die Fitness, wenn ein Bit anders ist als die beiden Nachbarbits. Der Gradient der durchschnittlichen Fitness für ein $1 < i < n$ ist

$$\frac{\partial W^u(q)}{\partial q_i} = 2 - 2(q_{i-1} + q_{i+1}), \quad \frac{\partial W^u(q)}{\partial q_1} = 1 - 2q_2, \quad \frac{\partial W^u(q)}{\partial q_n} = 1 - 2q_{n-1} \qquad (3.152)$$

Sind also sowohl q_{i-1} als auch q_{i+1} gleich $1/2$, so verschwindet der Gradient. Sind beide größer als 0.5, so ist der Gradient negativ; wenn beide kleiner als 0.5 sind, ist er positiv.

Von einer zufälligen Konfiguration aus werden sich die Bitfrequenzen daher entsprechend verschieben. Wenn sie weit genug von 0.5 weg sind, werden sie ferner ihre Nachbarn in die entgegengesetzte Position verschieben. In einem *UMDA* mit unendlicher Populationsgröße würde folgendes passieren, wenn man alle q_i mit $1/2$ startet und nur q_1 zum Beispiel auf 0.6 setzt: q_1 wandert in Richtung 1 und schiebt dadurch q_2 in Richtung Null.

Die anderen Frequenzen bleiben zunächst fest. Sobald q_2 nicht mehr gleich 0.5 ist, wird q_3 in Richtung 1 wandern und so weiter. Die Lösung wandert also durch die Bitkette, bis das globale Optimum gefunden ist.

In einem realen *UMDA* sind die q_i aber nie genau 0.5, daher verändern sich direkt von Beginn an alle Parameter in eine zufällige Richtung. Was passiert nun, wenn sich nicht kompatible Lösungen aufeinander zu bewegen? Betrachte die hypothetische Situation aus Tabelle 3.4.

i	1	2	3	4	5	6	...
q_i	0.3	0.7	0.5	0.3	0.7	0.3	
$\partial W^u/\partial q_i$	-0.4	0.4	0	-0.4	0.4		

Tabelle 3.4: Dynamik von *XorKette*

Bei X_3 treffen die beiden Lösungen aufeinander. Von rechts gesehen, müsste q_3 wachsen; von links gesehen, müsste es fallen. Man sieht, dass die anderen Variablen weiter zu den Randpositionen drängen, während der Gradient für q_3 klein bleibt. Aus dieser Situation findet der Algorithmus nicht hinaus.

Man könnte meinen, dass ein Hillclimber dieses Problem leicht lösen kann. Wenn man die Variablen nämlich der Reihe nach durchgeht, kann man sie in einem Durchgang jeweils auf den richtigen Wert setzen. Dies ist aber nur möglich, wenn man genau die Reihenfolge der Verkettung einhält. Permutation der Indizes macht es auch für solch ein Verfahren sehr unwahrscheinlich, das globale Optimum zu finden.

i	...	25	26	27	28	29	30
q_i	...	0.02	0.99	0.03	0.52	0.96	0.02

Tabelle 3.5: *UMDA*-Lauf für *XorKette*, Stand nach 50 Generationen

In Tabelle 3.5 ist ein aktueller Lauf des *UMDA* dargestellt. Gezeigt sind die Bitfrequenzen nach 50 Generation für $n = 30$ Bits, Populationsgröße 300, *Truncation Selection* und Bayes'schem Schätzer 2 (siehe 3.8). *Truncation Selection* wurde gewählt, um zu zeigen, dass die Analyse auch für diese Selektionsform gilt, obwohl die Formeln ursprünglich für proportionale Selektion bestimmt wurden.

Wegen des Priors werden die Extremwerte 0 und 1 nicht erreicht. Man sieht aber deutlich, dass sich die in Tabelle 3.4 vorhergesagte Situation eingestellt hat und der Algorithmus von diesem lokalen Maximum nicht mehr wegfindet.

Weitere Beispiele für diese Art der Analyse finden sich in (Mühlenbein and Mahnig, 2000b,a).

3.7.6 Beliebiger Wertebereich in Satz 3.39

Es lässt sich auch für beliebigen diskreten Wertebereich eine Darstellung analog zu Lemma 3.18 herleiten. Sie wird aber für den folgenden Satz nicht benötigt und daher ausgelassen.

Satz 3.41. *Mit der Einbettung $\bar{Q}$ ist $W_f^u(q)$ eine Erweiterung von $f(x)$ in den Raum Q^u. Die lokalen Extrema $\hat{q}$ von $W_f^u(q)$ sind im Bildbereich der Einbettung, also in den Ecken: für jedes i ist* höchstens *ein $\hat{q}_{i,y_i}$ gleich Eins, alle anderen sind gleich Null.*

Beweis: Betrachte folgende Darstellung von $W_f^u(q)$ nach (3.7):

$$W_f^u(q) = \sum_{x \in \mathcal{R}} f(x) \prod_{i=1}^{n} \left[\prod_{y_i=1}^{l_i} q_{i,y_i}^{\delta(x_i=y_i)} \right] \left(1 - \sum_{y_i=1}^{l_i} q_{i,y_i} \right)^{\delta(x_i=0)} \tag{3.153}$$

Es handelt sich offensichtlich wieder um eine lineare Funktion in jedem Parameter q_{i,y_i}, auch wenn die Koeffizienten nicht so gut bekannt sind. Es gibt hier aber im Gegensatz zu Satz 3.39 die Schwierigkeit, dass die Parameter durch Nebenbedingungen gebunden sind.

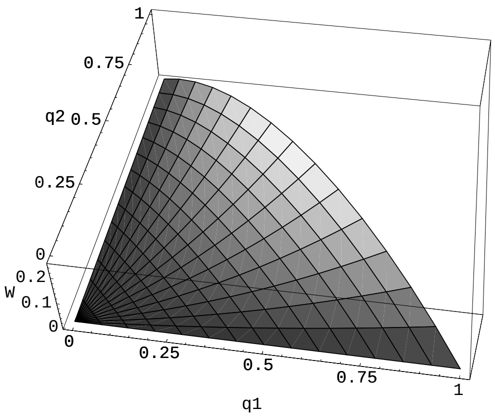

Abbildung 3.11: $W_f^u(q_1, q_2) = q_1 q_2$ mit $q_1 + q_2 \leq 1$

Betrachte Abbildung 3.11. Die Fitnessfunktion $W_f^u(q_1, q_2)$ ist offenbar wieder linear in den Argumenten, deshalb wird das Maximum auch am Rand des Definitionsbereichs erreicht. Dieser Rand wird aber durch die Bedingung $q_1 + q_2 \leq 1$ so eingeschränkt, dass das Maximum bei $(1/2, 1/2)$ liegt.

In Gleichung (3.153) kommt aber in jedem Faktor für festes i jeweils nur *ein* q_{i,y_i} vor. Damit sind die Parameter in einem Faktor nicht durch Nebenbedingungen eingeschränkt und deshalb lässt sich der Beweis zu Satz 3.39 übernehmen. Eine Fitnesslandschaft wie in Abbildung 3.11 ist also nicht möglich. (Satz 3.41) ∎

3.7.7 Umparametrisierung der einfachen Produktverteilung

Um aus den formalen Parametern q_{i,x_i} mit $x_i \in 0, \ldots, l_i$ eine Parametrisierung zu erhalten, muss ein Parameter eliminiert werden. Bisher wurde immer $q_{i,0}$ eliminiert durch

$$q_{i,0} = 1 - \sum_{x_i=1}^{l_i} q_{i,x_i} \tag{3.154}$$

Die so gewonnene Parametrisierung von W_f^u erlaubte es, partielle Ableitungen von W_f^u nach den q_{i,x_i} zu betrachten.

Nun macht der Algorithmus *UMDA* aber diese Unterscheidung gar nicht, er zählt einfach die Häufigkeit jedes Allels x_i. Da das Verhalten des Algorithmus aber unter anderem über die partiellen Ableitungen von W_f^u bestimmt wurde, stellt sich die Frage, was mit diesen Ableitungen bei einer Umparametrisierung passiert.

Beispiel 3.10. Die Funktion aus Abbildung 3.3 ist $W_f^u(q_1, q_2) = 2q_1 + 3q_2 - 4q_1 q_2$ und hat den kritischen Punkt $(q_1 = 3/4, q_2 = 1/2)$. Man könnte aber genauso gut anders herum parametrisieren, also mit (r_1, r_2), wobei r_i den Anteil an 0-Bits angibt. Dann ergibt sich $W^r(r_1, r_2) = 2(1 - r_1) + 3(1 - r_2) - 4(1 - r_1)(1 - r_2) = 1 + 2r_1 + r_2 - 4r_1 r_2$. Hier ist der kritische Punkt $(r_1 = 1/4, r_2 = 1/2)$. Beide Koordinaten beschreiben nun aber den selben Punkt im Raum Q^{u0}, das heißt, die andere Parametrisierung hat die Lage der kritischen Punkte nicht verändert. Es zeigt sich, dass dies immer gilt:

Definition 3.42. *Der natürliche Definitionsraum der durchschnittlichen Fitness W_f^{u0} für den UMDA ist der Raum Q^{u0} aus Definition 3.5. Diese Hyperfläche muss wegen der Nebenbedingung $\sum_{x_i=0}^{l_i} q_{i,x_i} = 1$ parametrisiert werden, bevor die Ableitungen von W_f^{u0} definiert werden können. Eine* ***Umparametrisierung*** *ist eine Menge (auf den Raum Q^{u0}) bijektiver stetig differenzierbarer Funktionen $H_1, \ldots, H_n$ mit*

$$H_i(r_i) = H_i(r_{i,1}, \ldots, r_{i,l_i}) = (q_{i,0}(r_i), \ldots, q_{i,l_i}(r_i)) \quad \textit{mit} \sum_{x_i=0}^{l_i} q_{i,x_i}(r_i) = 1 \tag{3.155}$$

Assoziiert ist die Funktion der durchschnittlichen Fitness nach diesen Parametern

$$W_f^H(r_1, \ldots, r_n) = W_f^{u0}(H(r)) \tag{3.156}$$

Für die bisherige Parametrisierung U durch W_f^u gilt also

$$U_i(r_{i,1}, \ldots, r_{i,l_i}) = \big((1 - r_{i,1} - \cdots - r_{i,l_i}), r_{i,1}, \ldots, r_{i,l_i}\big) \tag{3.157}$$

Für den Definitionsraum von H ist fast nichts gefordert. Wegen der Bijektivität und der Stetigkeit muss dieser Raum aber diffeomorph zu einer Teilmenge des $\prod_{i=1}^n \mathbb{R}^{l_i}$ sein.

Lemma 3.43. *Es sei M eine Hyperfläche im $\mathbb{R}^m$, also eine $(m-1)$-dimensionale Untermannigfaltigkeit. Φ und Ψ seien zwei Parametrisierungen, also bijektive Abbildungen von Teilmengen $\mathbb{D}_\Phi$ bzw. $\mathbb{D}_\Psi$ des $\mathbb{R}^{m-1}$ nach M, so dass $\Psi^{-1} \circ \Phi$ ein Homöomorphismus ist (also $\mathcal{C}^\infty$ und die Umkehrabbildung ebenfalls $\mathcal{C}^\infty$). Wenn nun W eine Funktion auf M ist, so entsprechen kritische Punkte von $W \circ \Phi$ eineindeutig kritischen Punkten von $W \circ \Psi$:*

$$\nabla\big(W(\Phi)\big)\big|_{[s]} = 0 \;\Leftrightarrow\; \nabla\big(W(\Psi)\big)\big|_{[t]} = 0 \qquad \textit{mit } \Phi(s) = \Psi(t) \tag{3.158}$$

Beweis: Das Lemma folgt aus Standardsätzen der Differentialgeometrie, lässt sich aber auch einfach direkt zeigen. Bezeichne die partiellen Ableitungen nach Parameter Nummer i mit ∂_i. Wenn Φ_i die i-te Koordinate von Φ bezeichnet, gilt nach der Kettenregel

$$\partial_k W(\Phi)\big|_{[s]} = \sum_{i=1}^{m} \partial_i W\big|_{[\Phi(s)]} \cdot \partial_k \Phi_i\big|_{[s]} \tag{3.159}$$

und somit, wenn $D\Phi$ die Matrix der totalen Ableitung von Φ ist:

$$\nabla(W(\Phi))\big|_{[s]} = (D\Phi\big|_{[s]}) \cdot (\nabla W\big|_{[\Phi(s)]}) \tag{3.160}$$

als Multiplikation einer Matrix und eines Vektors. Setze nun $T := \Psi^{-1} \circ \Phi$, das ist nach Voraussetzung ein Homöomorphismus von $\mathbb{D}_\Phi$ auf $\mathbb{D}_\Psi$. Es gilt: $W(\Phi(s)) = W(\Psi(T(s))$ und damit

$$\partial_k W(\Phi)\big|_{[s]} = \sum_{i,j=1}^{m} \partial_i W\big|_{[\Psi(T(s))]} \cdot \partial_j \Psi_i\big|_{[T(s)]} \cdot \partial_k T_j\big|_{[s]} \tag{3.161}$$

und damit

$$\nabla(W(\Phi))\big|_{[s]} = (DT\big|_{[s]}) \cdot (D\Psi\big|_{[T(s)]}) \cdot (\nabla W\big|_{[\Psi(T(s))]}) \tag{3.162}$$

Nach Voraussetzung hat DT vollen Rang. Wenn jetzt also an einer Stelle $\Phi(\tilde{s})$ der Gradient $\nabla(W(\Phi))|_{[\tilde{s}]}$ verschwindet, ist (3.160) gleich dem Nullvektor, also auch (3.162). Die Multiplikation mit DT ändert daran nun nichts, also gilt auch

$$0 = (D\Psi\big|_{[T(\tilde{s})]}) \cdot (\nabla W\big|_{[\Psi(T(\tilde{s}))]}) \quad = \nabla(W(\Psi))\big|_{[T(\tilde{s})]} \tag{3.163}$$

(Lemma 3.43) ■

Daraus folgt direkt der

Satz 3.44. *Es seien $q_{i,1}$ bis q_{i,l_i} die Standard-Parameter von W_f^u und H eine Umparametrisierung. Wenn $(\hat{q}_{i,x_i})$ ein kritischer Punkt von W_f^u ist und $H^u(\hat{q}) = H(\hat{r})$, so ist $\hat{r}$ ein kritischer Punkt von $W_f^H(r)$. Dies sind auch die einzigen kritischen Punkte von $W_f^H(r)$.*

Damit sind die kritischen Punkte der durchschnittlichen Fitness *unabhängig* von der Parametrisierung. Dies gilt sogar, wenn noch kompliziertere Parametrisierungen gewählt werden als einfach nur die, eine der Koordinaten zur Summe 1 zu ergänzen.

3.8 Bayes'sche Schätzer und die Mutation

In Anhang A.4 werden zwei Möglichkeiten vorgestellt, um die Wahrscheinlichkeit von „Kopf" einer gezinkten Münze zu schätzen. Der Maximum-Likelihood-Ansatz zählt die Häufigkeiten und kommt bei m Mal „Kopf" auf N Würfe auf $p = {}^{m}\!/_{N}$. Der Bayes'sche Ansatz geht von einer A-Priori-Verteilung aus und modifiziert die Wahrscheinlichkeit mittels der Bayes'schen Regel. In der Praxis heißt das, dass die Wahrscheinlichkeit $p = \frac{m+N'}{N+2N'}$ wird und man den sogenannten Hyperparameter N' vorgeben muss. Dies Vorgehen verallgemeinert sich auf multinomiale Verteilungen wie die Verteilung im *UMDA*.

Es blieb aber die Frage, wie beim Bayes'schen Ansatz die Hyperparameter N' zu wählen waren. Dazu nun folgende Überlegung: Der Bayes'sche Ansatz bewirkt, dass jede Allelfolge mit Wahrscheinlichkeiten größer 0 generiert werden kann, selbst wenn alle entsprechenden Gene in der Population gleich sind. Diese Rolle spielt in dem klassischen *SGA* die Mutation.

Es erfolgt wiederum die Beschränkung auf binäre Variablen.

Satz 3.45. *Bei binären Variablen entsprechen Bayes'sche Wahrscheinlichkeiten mit dem Priorparameter N' gezählten Häufigkeiten mit Mutationsrate $N'/(N+2N')$.*

Beweis: Mit Wahrscheinlichkeit μ werde ein Bit gekippt. m sei die Häufigkeit der Eins-Bits der Population mit Größe N. Betrachte nur ein einziges Bit. Es gilt dann klassisch:

$$p(X=1) = \frac{m}{N}(1-\mu) + \frac{N-m}{m} \cdot \mu \tag{3.164}$$

$$= \mu\frac{N-2m}{N} + \frac{m}{N} \tag{3.165}$$

Mittels Bayes'scher Wahrscheinlichkeiten ergibt sich

$$\tilde{p}(X=1) = \frac{m+N'}{N+2N'} \tag{3.166}$$

Gleichsetzen dieser Wahrscheinlichkeiten liefert

$$\mu = \left[\frac{m+N'}{N+2N'} - \frac{m}{N}\right]\frac{N}{N-2m} \tag{3.167}$$

$$= \frac{NN' - 2mN'}{N(N+2N')} \cdot \frac{N}{N-2m} = \frac{N'}{N+2N'} \tag{3.168}$$

(Satz 3.45) ■

Bemerkenswert ist es, dass die beiden Verfahren unabhängig von m die gleichen Wahrscheinlichkeiten liefern.

Für *Truncation Selection* hat sich eine Mutationsrate von ${}^{1}\!/_{n}$ als günstig erwiesen (Mühlenbein, 1991, 1992). Damit ergibt sich $N' = N/(n-2)$.

Nun gibt es aber noch eine andere Randbedingung. Ohne Einschränkung sei das Optimum gleich $(1,\ldots,1)$. Wenn die Population nur noch aus diesen Optima besteht, dann werden neue Individuen nach der Bayes'schen Methode mit Wahrscheinlichkeit

$$p(X_1\!=\!1,\ldots,X_n\!=\!1) = \left(\frac{N+N'}{N+2N'}\right)^n \tag{3.169}$$

generiert. Wenn das N' zu groß wird, so wird die Wahrscheinlichkeit dafür immer kleiner! Für $N' = {}^N\!/_n$ gilt aber

$$\left(\frac{N+N'}{N+2N'}\right)^n = \left(\frac{1+{}^1\!/_n}{1+{}^2\!/_n}\right)^n = \left(1-\frac{1}{n+2}\right)^n \geq 0.3 \tag{3.170}$$

weil der Ausdruck rechts monoton fallend gegen e^{-1} konvergiert. Damit ist für diese Wahl gewährleistet, dass die Population nahe genug an das Optimum herankommen kann. Es gilt also die

Faustregel. *Bei binären Variablen sollte der Bayes'sche Schätzer mit Prior-Parameter $N' \propto N/n$ gewählt werden.*

Ein exakter Wert lässt sich nicht so leicht bestimmen, die Faustregel liefert aber einen Anhaltspunkt, wie man bei Skalieren von N oder n vorgehen sollte.

3.8.1 Proportionale Selektion und die Mutationsrate

Mit Mutation sind die dynamischen Gleichungen für proportionale Selektion nicht mehr gültig. Es ergibt sich vielmehr für binäre Parameter

$$q_i(t+1) = \frac{q_i^s(t)N+N'}{N+2N'}, \tag{3.171}$$

wobei $q_i^s(t)$ durch Gleichung (3.47) gegeben ist, und somit:

$$q_i(t+1) = \left[\left(q_i(t) + q_i(t)\frac{\big(1-q_i(t)\big)}{W_f^u(t)}\frac{\partial W_f^u}{\partial q_i}\right)N + N'\right]\cdot\frac{1}{N+2N'} \tag{3.172}$$

Es ergeben sich also durch die Mutation veränderte Gleichgewichtspunkte. Mutation verschiebt die Parameter zu $p_i = {}^1\!/_2$ hin. Für *OneMax* ergibt sich als Gleichgewichtspunkt der Dynamik $\mu = \frac{N'}{N+2N'}$

$$q_i^* = \frac{N+nN'}{N+2nN'} = \frac{1+\mu(n-2)}{1+2\mu(n-1)} \tag{3.173}$$

Mit $\mu\!=\!{}^1\!/_n$ beziehungsweise $N' = \frac{N}{n-2}$ folgt daraus

$$q_i^* = \frac{2(n-1)}{3n-2} \approx \frac{2}{3} \tag{3.174}$$

In einem Simulationslauf des *UMDA* ergab sich für $n=10$, $N=1000$ und $N' = 100$ als durchschnittlicher Wert in der letzten Generation $\bar{q}_i = 0.6692$, also eine sehr gute Bestätigung des theoretischen Modells.

Wright (1935) hat ein etwas anderes Modell benutzt und folgende Gleichung postuliert:

$$q_i(t+1) = q_i(t) + q_i(t)\frac{\big(1-q_i(t)\big)}{\tilde{W}}\frac{\partial \tilde{W}}{\partial q_i} - \mu\big(2q_i(t)-1\big) \tag{3.175}$$

Für $\mu = 0$ bleibt es bei dem Gleichgewichtspunkt $q_i^* = 1$, für $\mu = {}^1\!/\!_n$ ergibt sich auch $q_i^* = {}^2\!/\!_3$. Das heißt aber, dass das Optimum im besten Fall nur mit Wahrscheinlichkeit $({}^2\!/\!_3)^n$ generiert wird. Für proportionale Selektion ist diese Mutationsrate also zu hoch, nicht einmal die einfache *OneMax*-Funktion kann damit optimiert werden!

Durch Mutation kann die Population nicht mehr in die Ecken der Fitnesslandschaft gelangen. Alle Punkte werden zum Zentrum des Würfels hin verschoben. Das ist deshalb ein Vorteil, weil Variablen im *UMDA*, die einmal einen Extremwert erreicht haben, diesen immer behalten werden. Dieses Phänomen nennt man *Fixation*. Wenn eine Variable in der gesamten Population den gleichen Wert hat, so wird sie diesen ohne Mutation immer behalten. In Abschnitt 4.4.3 folgt ein Beispiel, wie mit Hilfe der Mutation die Populationsgröße von 20000 auf 200 reduziert werden konnte.

3.8.2 Beispielhafte Analyse der Fixpunkte

Anhand von drei einfachen Funktionen soll nun analysiert werden, wie die Fixpunkte der Gleichung (3.172) aussehen.

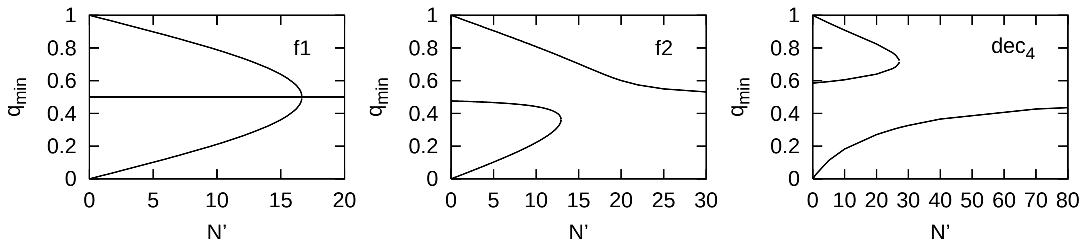

Abbildung 3.12: Lage der Fixpunkte der Gleichung (3.175) für verschiedene Funktionen mit $N = 100$, siehe Abschnitt 3.8.2

Betrachte Abbildung 3.12. Links sind die Fixpunkte der Gleichung (3.175) für die Funktion $f_1(x_1, x_2) = 2 - x_1 - x_2 + 2x_1x_2$ dargestellt mit $N=100$. Diese Funktion hat ihre Maxima bei $(0,0)$ und $(1,1)$ und ist symmetrisch. Ohne Mutation gibt es hier die drei Fixpunkte 0, 0.5 und 1. Für $N' \to \infty$ bleibt für jede Funktion nur 0.5 als Fixpunkt übrig. Mutation ist dann so stark, dass die Selektion keine Rolle mehr spielt. Bei $N' \approx 17$ spaltet sich der Fixpunkt in drei Fixpunkte auf.

Die Funktion f_2 ist definiert durch $f_2(x_1, x_2) = 2 - x_1 - x_2 + 2.1x_1x_2$. Das Optimum liegt also bei $(1, 1)$. Die Funktionen f_1 und f_2 unterscheiden sich kaum. Für die Fixpunkte hat dies aber eine dramatische Auswirkung. Von großem N' kommend, werden die Werte von q_i in Richtung 1 gedrängt. Für $N' > 13$ gibt es nur einen Fixpunkt der Dynamik, und der liegt in der Nähe des gesuchten globalen Optimums.

Die Fixpunkte verschwinden, weil es sich um Nullstellen von Polynomen von (in diesem Fall) drittem Grad handelt. Diese Nullstellen werden bei $N' \approx 13$ komplex.

Das dritte Bild zeigt die Fixpunkte *Deceptive*$_4$-Funktion. Wie erwartet, gibt es drei Fixpunkte bei $N' = 0$, nämlich 0, 1 und $\sqrt[3]{1/5}$. Leider führt die Kurve von großem N' aus zum lokalen Optimum. Das heißt aber auch, dass die Erhöhung der Mutationsrate für diese Funktion keine Verbesserung bringt. Für diese Funktion kann ein Algorithmus, der auf der einfachen Produktverteilung basiert, vermutlich nicht das globale Optimum gefunden werden, weil das lokale Optimum einen zu starken Attraktor darstellt.

3.9 Zusammenfassung

Die durchschnittliche Fitness W_f^u ist für den *UMDA* die zentrale Größe zur Bestimmung des Verhaltens. Sie transformiert das *diskrete Optimierungsproblem* in ein *kontinuierliches Optimierungsproblem*. Die Variablen des kontinuierlichen Raumes sind die univariaten Randverteilungen $p_i(x_i)$.

Für proportionale Selektion zeigt sich, dass der *UMDA* durch ein System von dynamischen Gleichungen beschrieben werden kann, welches gewichteten Gradientenaufstieg in diesem Raum durchführt. Der erwartete Zuwachs der durchschnittlichen Fitness kann durch eine Reihenentwicklung dargestellt werden. Es stellt sich heraus, dass der Satz von Fisher der erste Term in dieser Reihenentwicklung ist.

Truncation Selection folgt zwar auch annähernd dem Gradienten, die Dynamik ist aber eine andere. Leider ist das Verhalten so kompliziert, dass sich im Allgemeinen keine dynamischen Gleichungen mehr für die Parameter aufstellen lassen. Ein Resultat für *OneMax* zeigt aber die Unterschiede zu proportionaler Selektion auf. Die Analyse der Fitnesslandschaft zeigt auch hier die Stellen auf, wo der Algorithmus Schwierigkeiten hat. Im Gegensatz zu proportionaler Selektion kann hier aber auch zwischendurch die durchschnittliche Fitness absinken.

Die diskrete Fitness-Funktion ist definiert in den Ecken eines Hyperwürfels. Die kontinuierliche Fitness-Landschaft erweitert den Definitionsbereich um das Innere des Würfels. In diesem Raum optimiert der *UMDA*. Da die Fitness-Landschaft in jeder Variablen linear ist, liegen die Extrema in den Ecken, korrespondieren also zu Punkten im ursprünglichen (diskreten) Raum. Deshalb liefert ein gradientenbasiertes Verfahren eine Lösung für das ursprüngliche Problem.

Weil die durchschnittliche Fitness bei proportionaler Selektion immer steigt, kann der *UMDA* keine Täler durchqueren, um zu dem globalen Optimum zu finden. Dieses theoretische Resultat bestätigte sich in den Testläufen.

Der *UMDA* entspricht in seiner einfachen Variante einem genetischen Algorithmus ohne Mutation. Durch Wahl des Bayes'schen Schätzers für die Parameter kann ein der Mutation äquivalenter Prozess benutzt werden. Dadurch wird indirekt die Fitness-Landschaft so verändert, dass die Parameter zu Gleichverteilung hin verschoben werden. So kann in vielen Fällen eine verfrühte Fixation der Variablen verhindert werden.

4 Die Boltzmann-Verteilung und der *FDA*

4.1 Optimierung durch Such-Verteilungen

4.1.1 Allgemeine Verteilungen

Der *UMDA* ist ein sehr erfolgreicher Optimierungsalgorithmus, der mit relativ wenig (Programmier-)Aufwand viele Probleme lösen kann. Die Struktur der zugrunde liegenden Verteilung zeigt aber sofort die Schwäche auf: Sobald es starke Interaktionen zwischen den einzelnen Variablen gibt, versagt er, wie bei der *Deceptive*-Funktion aus Abschnitt 3.7.3.

Im Rahmen der Theorie der verteilungsbasierten Optimierung kann man als Ausweg nun anstelle der einfachen Produktverteilung andere Verteilungen zulassen, die diese Schwäche nicht haben. Ein geeigneter Kandidat dafür ist die von *Simulated Annealing* (*SA*) (Kirkpatrick et al., 1983) bekannte Boltzmann-Verteilung, weil sie die nötige Fokussierung um die Menge der Optima herstellt. Dabei stellt sich das Problem, dass man aus beliebigen Verteilungen nicht so leicht Werte ziehen kann wie bei der Binomial-Verteilung. Während dies bei *SA* über den Gleichgewichtszustand einer Markovkette gelöst wird, wird beim in diesem Kapitel vorgestellten *FDA* die Faktorisierung der Verteilung genutzt.

Etwas allgemeiner lässt sich der *PDA (Parametric Distribution Algorithm)* definieren. Es sei $\tilde{p}$ eine parametrische Verteilung, bei der q_0 die Gleichverteilung definiert. Außerdem besitze $\tilde{p}$ einen Parameterschätzer $\tilde{P}$. Dann beschreibt Algorithmus 4.1 einen allgemeinen probabilistischen Suchalgorithmus, der mit beliebigen Verteilungen arbeitet.

Algorithmus 4.1: PDA – Parametric Distribution Algorithm

Erzeuge eine zufällige Ausgangspopulation mit N Individuen $\mathfrak{X}^i$ aus $\tilde{p}(x, q(0))$ (Gleichverteilung), $t \Leftarrow 1$.
do {
    Selektiere $\hat{N} \leq N$ Punkte $\hat{\mathfrak{X}}^i$.
    Bestimme $q(t) = \tilde{P}(\hat{\mathfrak{X}}, t)$ aus den selektierten Punkten.
    Erzeuge neue Punkte gemäß $\tilde{p}(x, q(t))$.
    $t \Leftarrow t + 1$.
} **until** (Abbruchkriterium erreicht)

4.1.2 Die Boltzmann-Verteilung

Definition 4.1. *Für $\beta \geq 0$ ist die Gibbs- oder* ***Boltzmann-Verteilung*** *einer Funktion $f(x)$ definiert durch*

$$p_{\beta,f}(x) := \frac{e^{\beta f(x)}}{\sum_y e^{\beta f(y)}} \qquad =: \frac{e^{\beta f(x)}}{Z_f(\beta)} \tag{4.1}$$

$Z_f(\beta)$ *ist die Zustandssumme* (partition function). *Zur Vereinfachung der Notation entfällt β und/oder f, wenn sie sich aus dem Kontext ergeben.*

In Anlehnung an die Thermodynamik wird die Boltzmann-Verteilung oft auch $p_{\beta,f}(x) = \exp\left(-\frac{1}{T}(g(x)-E)\right)$ notiert, wobei T als Temperatur interpretiert wird und $E = -T \ln Z_T$ die freie Energie ist. Die Boltzmann-Verteilung zu gegebenem f ist ein Beispiel einer parametrischen Verteilung gemäß Definition 3.3 mit Parameterraum $Q = \mathbb{R}^+$ und

$$\tilde{p}^f(x,q) = \frac{e^{qf(x)}}{\sum_y e^{qf(y)}} \tag{4.2}$$

Für die Boltzmann-Verteilung gibt es keinen effizienten selbsterhaltenden Parameterschätzer: Selbst wenn man weiß, dass gegebene Daten von einer Boltzmann-Verteilung mit bekanntem f stammen, ist die Bestimmung der inversen Temperatur nicht einfach.

Lemma 4.2 (Eigenschaften der Boltzmannverteilung). *Sei x_m ein globales Optimum der Funktion $f(x)$, x_l ein Punkt mit $f(x_l) < f(x_m)$. Dann gilt*

1. *$p_{\beta=0,f}$ ist die Gleichverteilung für jedes f.*
2. *$p_\beta(x_m) \geq p_\beta(x_l)$, für $\beta > 0$ ist die Ungleichung strikt.*
3. *Sei $g(x) := f(x) + c$. Dann gilt $p_{\beta,f}(x) = p_{\beta,g}(x)$.*
4. *Sei $g(x) := c \cdot f(x)$. Dann gilt $p_{\beta,g}(x) = p_{c\beta,f}(x)$.*
5. *$p_\beta(x)/p_\beta(y) = e^{\beta(f(x)-f(y))}$*

4.1.3 Boltzmann-Selektion

Eng verwandt mit der Boltzmannverteilung ist die Boltzmann-Selektion (de la Maza and Tidor, 1993).

Definition 4.3. *Die Boltzmann-Selektion ist ein Operator in dem Raum der Verteilungen über $\mathcal{R}$. Gegeben sei eine Verteilung p sowie eine Selektionsstärke γ, dann liefert Boltzmann-Selektion die Verteilung der selektierten Punkte*

$$p^s(x) = \frac{p(x)e^{\gamma f(x)}}{\sum_y p(y)e^{\gamma f(y)}} \tag{4.3}$$

Boltzmann-Selektion kann auch als proportionale Selektion mit exponentiell skalierter Fitness aufgefasst werden.

In der Theorie der genetischen Algorithmen wird die Boltzmannverteilung normalerweise durch ein Ablehnungsschema ähnlich dem von *SA* erzeugt (zum Beispiel (Lozano et al., 1999)). Dies ist deshalb nötig, weil dort neue Individuen durch Rekombination und Mutation erzeugt werden. Über die Verteilung der neu generierten Punkte gibt es daher keine Kontrolle, deshalb muss das Ablehnungsverfahren die neuen Punkte zur Boltzmannverteilung bringen. Hier soll aber ein anderer Weg gegangen werden.

Satz 4.4. *Es sei $p_0(x)$ eine Verteilung über $\mathcal{R}$. Wenn nacheinander Boltzmann-Selektion mit Parametern β und γ angewandt wird, ist die resultierende Verteilung*

$$p^{ss}(x) = \frac{p_0(x)e^{(\beta+\gamma)f(x)}}{\sum_y p_0(y)e^{(\beta+\gamma)f(y)}} \tag{4.4}$$

Beweis: Setze $Z_0(\beta) := \sum_y p_0(y)e^{\beta f(x)}$. Dann gilt

$$\begin{aligned} p^{ss}(x) &= \frac{p_0(x)e^{\beta f(x)}}{Z_0(\beta)} \cdot \frac{e^{\gamma f(x)}}{\sum_y \frac{p_0(x)e^{\beta f(y)}}{Z_0(\beta)} e^{\gamma f(y)}} && (4.5) \\ &= \frac{p_0(x)e^{(\beta+\gamma)f(x)}}{\sum_y p_0(x)e^{(\beta+\gamma)f(y)}}. && (4.6) \end{aligned}$$

(Satz 4.4) ■

Insbesondere gilt dies für $p_0(x)$ die Gleichverteilung auf $\mathcal{R}$. In diesem Fall kürzt sich $p_0(x)$ heraus und es gilt

Korollar 4.5. *Wenn $p_0(x)$ die Gleichverteilung ist, so ergibt sich durch iteriertes Anwenden von Boltzmann-Selektion mit Parameter β_τ eine Folge von Boltzmann-Verteilungen $p_{\tilde{\beta}_t}(x)$ mit*

$$\tilde{\beta}_t = \sum_{\tau=1}^{t} \beta_\tau \tag{4.7}$$

4.1.4 *BEDA* - Boltzmann Estimated Distribution Algorithm

Mit Hilfe von Satz 4.4 kann nun der *BEDA*-Algorithmus 4.2 definiert werden. Da die Berechnung der Boltzmann-Verteilung exponentiellen Aufwand erfordert und die Generierung neuer Punkte nach dieser Verteilung ebenfalls nicht trivial ist, handelt es sich hierbei nur um einen konzeptionellen Algorithmus.

4.1.5 Die Fokussierungseigenschaft des *BEDA*

Der folgende Fokussierungssatz für den *BEDA* muss mit Vorsicht betrachtet werden, weil er voraussetzt, dass die Optima bereits in der Startverteilung enthalten sind. Wenn

Algorithmus 4.2: *BEDA* – Boltzmann Estimated Distribution Algorithm

$t \Leftarrow 0$. Generiere N Punkte mit der Gleichverteilung $p(x,0)$ für $\beta_0 = 0$.
do {
  Berechne für gegebenes $\beta_t > 0$

$$p^s(x,t) = \frac{p(x,t)e^{\beta_t f(x)}}{\sum_y p(y,t)e^{\beta_t f(y)}}. \tag{4.8}$$

  Generiere N neue Punkte nach der Verteilung $p(x,t+1) = p^s(x,t)$.
  $t \Leftarrow t+1$.
} **until** (Abbruchkriterium erfüllt)

man den *BEDA* also direkt verwenden will, benötigt man eine Populationsgröße, die mindestens so groß wie der Suchraum ist. Später wird mit dem *FDA* vorgestellt, wie man durch Faktorisierung den Parameterraum polynomiell halten kann. Es ist bisher aber nicht gelungen, die Fehler (die durch die stochastischen Prozesse Sampling und Boltzmann-Selektion entstehen) so abzuschätzen, dass man eine Aussage über die dann benötigte Populationsgröße erhält.

Satz 4.6 (Fokussierungseigenschaft). *Gegeben sei ein Abkühlungsschema, also für jeden Zeitpunkt t eine Erhöhung der inversen Temperatur β um β_τ. Mit Initialverteilung $p_0(x)$ ist für BEDA zum Zeitpunkt t die Verteilung gegeben durch*

$$p(x,t) = \frac{p_0(x)e^{\tilde{\beta}_t f(x)}}{\sum_y p_0(y)e^{\tilde{\beta}_t f(y)}} \tag{4.9}$$

mit der inversen Temperatur $\tilde{\beta}_t = \sum_{\tau=1}^{t} \beta_\tau$. Ferner sei $\mathcal{M}$ die Menge der Optima. Wenn $\tilde{\beta}_t \to \infty$ und für mindestens ein $x^m \in \mathcal{M}$ gilt $p_0(x^m) > 0$, dann gilt

$$\forall x \notin \mathcal{M} : \lim_{t\to\infty} p(x,t) = 0 \tag{4.10}$$

Gilt außerdem $\forall x^m, y^m \in \mathcal{M} : p_0(x^m) = p_0(y^m) > 0$, dann folgt auch

$$\lim_{t\to\infty} p(x,t) = \begin{cases} 1/|\mathcal{M}| & x \in \mathcal{M} \\ 0 & \textit{sonst} \end{cases} \tag{4.11}$$

Beweis: Es sei $x^m \in \mathcal{M}$ ein Punkt mit maximaler Fitness und $p_0(x_m) > 0$ sowie $x \notin \mathcal{M}$ ein Punkt mit $f(x) < f(x^m)$. Dann gilt

$$p(x,t) = \frac{p_0(x)e^{\tilde{\beta} f(x)}}{\sum_y p_0(y)e^{\tilde{\beta} f(y)}} \leq \frac{p_0(x)e^{\tilde{\beta} f(x)}}{p_0(x^m)\cdot e^{\tilde{\beta} f(x^m)}} \leq \frac{p_0(x)}{p_0(x^m)e^{\tilde{\beta}[f(x^m)-f(x)]}} \tag{4.12}$$

Da $\tilde{\beta} \to \infty$, konvergiert $p(x,t)$ (exponentiell) gegen Null. Weil ferner für alle $x^m, y^m \in \mathcal{M}$ mit $p_0(x^m) = p_0(y^m)$ auch $p(x^m,t) = p(y^m,t)$ gilt, ist die Grenzverteilung in diesem Fall die Gleichverteilung auf der Menge der Optima. (Satz 4.6) ■

Anmerkung 4.7. Dieser Satz entspricht den fundamentalen Konvergenzsätzen von *Simulated Annealing*. Dort unterscheidet man zwischen zwei Fällen: den Fall der homogenen und den der inhomogenen Markovkette. Die Unterschiede sind wie folgt:

- Bei der *homogenen* Markovkette wird die Temperatur festgehalten und die (dadurch homogene, das heißt zeitunabhängige) Markovkette (unendlich lange) iteriert. Klassische Sätze aus der Theorie der Markovketten zeigen dann, dass die Markovkette gegen die Boltzmann-Verteilung konvergiert. Wenn man nun nach der Konvergenz die Temperatur herunterregelt (also $\tilde{\beta}$ gegen unendlich gehen lässt), so folgt der Konvergenzsatz ähnlich einfach wie der Fokussierungssatz (van Laarhoven, 1987). Ein echter Algorithmus muss aber natürlich bei fester Temperatur nach einer endlichen Anzahl von Schritten abbrechen. Dann ist der Beweis aber nicht mehr gültig.
- Bei der *inhomogenen* Markovkette wird der Prozess des Abkühlens in die Analyse integriert. Es ergibt sich für jeden Schritt mit wechselnder Temperatur eine andere Tabelle von Übergangswahrscheinlichkeiten. Für den diskreten Fall gelang 1988 der Beweis (Hajek, 1988), unter welchen Bedingungen der Algorithmus gegen die gewünschte Zielverteilung konvergiert. Es ergab sich dabei der bekannte logarithmische Schedule, der in der Praxis kaum Bedeutung hat, weil ein logarithmischer Schedule exponentielle Laufzeit zur Folge hat.

Der Beweis geht davon aus, dass in jedem Schritt die Boltzmann-Verteilung *exakt* erzeugt wird. Es lässt sich also der einfache Konvergenzbeweis für den homogenen Fall übernehmen, ohne dass eine unendliche Zahl an Schritten für den Gleichgewichtszustand nötig ist. Insbesondere ist der Beweis für eine beliebige Abkühlungsfolge gültig, solange die inverse Temperatur gegen unendlich konvergiert und auch der Selektionsprozess *exakt* arbeitet.

Gleichung (4.12) zeigt nicht nur, dass die Verteilung überall jenseits der Optima gegen Null konvergiert, sie macht auch eine Aussage über die Geschwindigkeit. Obwohl die Abschätzung sehr grob ist (eine Summe von exponentiell vielen Termen wird gegen einige Summanden abgeschätzt), zeigt sich, dass die Dichte von jedem solchen Punkt *exponentiell* gegen Null geht. Mit bekanntem Schedule der β_t lässt sich die Zeit bis zur Annäherung an die Grenzverteilung abschätzen (die Abschätzung wird immer genauer, je größer $\tilde{\beta}_t$ wird).

Lemma 4.8. *Es gelte $|\mathcal{R}_i| \leq K$ und für jeden nicht optimalen Punkt $x \notin \mathcal{M}$ gelte mit $x^m \in \mathcal{M}$*

$$f(x) \leq f(x^m) - \delta \tag{4.13}$$

Dann gilt mit $p_0(x)$ der Gleichverteilung

$$\beta \geq \frac{n \cdot \ln K}{\delta} \quad \Longrightarrow \quad p_\beta(\mathcal{M}) \geq 0.5, \tag{4.14}$$

ab dieser inversen Temperatur wird also durch Ziehen aus p_β mit Wahrscheinlichkeit größer $1/2$ ein Optimum generiert.

Beweis: $|\mathcal{M}|$ ist die Zahl der Maxima. Die Anzahl der Summanden in der Zustandssumme kann man gegen K^n abschätzen. Für ein $x^m \in \mathcal{M}$ gilt mit $M := f(x^m)$

$$p_\beta(x^m) = \frac{e^{\beta M}}{\sum_y e^{\beta f(y)}} \geq \frac{e^{\beta M}}{K^n \cdot e^{\beta(M-\delta)} + |\mathcal{M}| \cdot e^{\beta M}} = \frac{1}{e^{n \ln K - \beta\delta} + |\mathcal{M}|} \quad \stackrel{!}{\geq} \frac{1}{2|\mathcal{M}|}, \tag{4.15}$$

damit die Menge der Maxima Maß $1/2$ hat. Es ergibt sich also, um $p_\beta(\mathcal{M}) \geq 1/2$ zu erreichen:

$$e^{n \ln K - \beta\delta} \leq 2|\mathcal{M}| \qquad \Leftrightarrow \qquad \beta \geq \frac{n \cdot K - \ln(2|\mathcal{M}|)}{\delta} \tag{4.16}$$

oder als hinreichende Bedingung (4.14). (Lemma 4.8) ■

Korollar 4.9. *Für eine binäre Funktion mit ganzzahligen Werten hat für $\beta \geq 0.7n$ die Hälfte der gesampelten Punkte maximale Fitness, wenn man mit der Gleichverteilung beginnt. Dies gilt unabhängig von der Struktur der Funktion oder dem Wert der Optima.*

Ohne einen Schedule sagt dies natürlich noch nichts aus, da dieses β ja in beliebig vielen Schritten erreicht werden kann. Neben der Analyse fester Schedules gibt das Lemma aber auch ein einfaches Abbruchkriterium vor.

4.2 *FDA* und die Faktorisierung

4.2.1 Die Faktorisierung der Verteilung und die *running intersection property*

Um einen praktikablen Algorithmus zu erhalten, muss die Zahl der Parameter für die Boltzmann-Verteilung drastisch reduziert werden. Hier soll nun gezeigt werden, wie eine additive Zerlegung der Fitness-Funktion eine Faktorisierung der Verteilung induziert.

Zentral ist die additive Zerlegungen in Teilfunktionen, siehe Definition A.10 aus dem Anhang:

$$f(x) = \sum_{i=1}^{m} f_{s_i}(x_{s_i}) \tag{4.17}$$

Definition 4.10. *Gegeben sei $S = \{s_1, \dots, s_m\}$, insbesondere eine Sortierung der s_i. Definiere*

$$\begin{aligned} d_i &:= \textstyle\bigcup_{j=1}^{i} s_j && \textit{(histories)} \\ b_i &:= s_i \setminus d_{i-1} && \textit{(residuals)} \\ c_i &:= s_i \cap d_{i-1} && \textit{(separators)} \end{aligned} \tag{4.18}$$

mit $d_0 := \emptyset$. S ist ein ***Faktorisierungssystem****, wenn*

- $b_i \neq \emptyset \quad \forall 1 \leq i \leq m$, *und*
- $d_m = I$, *d.h. alle Variablen werden erfasst.*

Nach Konstruktion gilt immer $c_i \subseteq d_{i-1}$.

Die Bezeichnungen für die Mengen b_i, c_i und d_i sind aus (Lauritzen, 1996) übernommen. Ein Faktorisierungssystem ist Voraussetzung für eine Faktorisierung:

Lemma 4.11. *Es sei* $s_1, \ldots, s_m$ *ein Faktorisierungssystem und* p *ein Wahrscheinlichkeitsmaß auf* $\mathcal{R}$. *Dann ist auch*

$$\tilde{p}(X) = \prod_{i=1}^{m} p(X_{b_i}|X_{c_i}) \tag{4.19}$$

ein Wahrscheinlichkeitsmaß auf $\mathcal{R}$ *und es lassen sich in der Reihenfolge der* X_{b_i} *auf einfache Art Punkte nach dieser Verteilung generieren.*

Beweis: Zu zeigen ist zunächst, dass $\tilde{p}$ ein Wahrscheinlichkeitsmaß ist. Dazu muss $\tilde{p}(x) \geq 0$ gelten für alle x, was offenbar erfüllt ist, und $\sum_x \tilde{p}(x) = 1$. Man kann die Faktoren von hinten nach vorne bearbeiten. Weil in b_i immer nur neue Variablen vorkommen und alle Variablen abgedeckt werden, gilt

$$\begin{aligned} \sum_x \tilde{p}(x) &= \sum_x p(x_{b_1})p(x_{b_2}|x_{c_2}) \cdots p(x_{b_m}|x_{c_m}) && (4.20)\\ &= \sum_{x_{d_{m-1}}} p(x_{b_1})p(x_{b_2}|x_{c_2}) \cdots \underbrace{\sum_{x_{b_m}} p(x_{b_m}|x_{c_m})}_{=1} && (4.21)\\ &= \cdots = 1 && (4.22) \end{aligned}$$

Um Punkte zu generieren, kann man folgendermaßen iterativ vorgehen. Nach Voraussetzung ist $c_1 = \emptyset$. Wähle daher x_{b_1} aus der Verteilung $p(X_{b_1})$. Als Induktionsvoraussetzung seien für die Variablen X_{b_i} alle Variablen $X_{d_{i-1}}$ bereits gesetzt. Nach Voraussetzung an ein Faktorisierungssystem ist $c_i \subseteq d_{i-1}$, daher gilt $X_{c_i} = x_{c_i}$. Somit lässt sich eine Belegung für X_{b_i} aus der Verteilung $p(X_{b_i}|x_{c_i})$ bestimmen und die Induktionsvoraussetzung bleibt erhalten. (Lemma 4.11) ■

Anmerkung 4.12. Die Voraussetzungen an ein Faktorisierungssystem werden alle benötigt. Für $b_i = \emptyset$ ist bereits die Verteilung nicht richtig definiert. Wenn $d_m \neq I$, so fehlen Variablen in der Faktorisierung. Mit $\tilde{p}(X_1, X_2) = p(X_1)$ etwa gilt dann $\sum_x \tilde{p}(x_1, x_2) = 2 \neq 1$ (für binäres X_1).

Zyklen sind schon nach Konstruktion der b_i und c_i in (4.18) ausgeschlossen. So kann man $\tilde{p}(X_1, X_2) = p(X_1|X_2)p(X_2|X_1)$ betrachten. Das stellt aber keine gültige Verteilung dar, und man kann danach auch keine Punkte generieren. Es gibt aber kein System s_1, s_2, welches diese b_i und c_i hervorbringen könnte.

Ein Faktorisierungssystem erzeugt also eine Faktorisierung $\tilde{p}$ der Verteilung p. Im Allgemeinen gilt jedoch $\tilde{p} \neq p$. Für die Gleichheit ist noch folgende Bedingung nötig:

Definition 4.13. *Ein Faktorisierungssystem erfüllt die* ***running intersection property****, falls gilt:*

$$\forall i \geq 2 \quad \exists j < i : \quad c_i \subseteq s_j \tag{4.23}$$

Diese Bedingung ist anschaulich nicht leicht verständlich. Sie wird in dieser Arbeit noch mehrmals auftauchen (Beispiel 4.2, Abschnitte 5.9.1 und 10.2.1). Sie soll im Folgenden durch verschiedene Sätze und Beispiele veranschaulicht werden, die zeigen, was passiert, wenn sie (nicht) erfüllt ist.

Für den Faktorisierungssatz wird noch das folgende Lemma benötigt:

Lemma 4.14. *Sei $p(X_\alpha, X_\beta, X_\gamma)$ eine (Rand-)Verteilung mit*

$$p(X_\alpha, X_\beta, X_\gamma) = g(X_\alpha, X_\gamma) h(X_\beta, X_\gamma), \tag{4.24}$$

dann folgt

$$p(X_\alpha, X_\beta, X_\gamma) = p(X_\alpha, X_\gamma) p(X_\beta | X_\gamma) \tag{4.25}$$

$$p(X_\alpha, X_\gamma) = g(X_\alpha, X_\gamma) \cdot \tilde{h}_\gamma(X_\gamma) \tag{4.26}$$

$$\textit{und} \qquad p(X_\alpha, X_\beta | X_\gamma) = p(X_\alpha | X_\gamma) p(X_\beta | X_\gamma), \tag{4.27}$$

Beweis: Bestimme folgende Randverteilungen:

$$p(X_\alpha, X_\gamma) = g(X_\alpha, X_\gamma) \sum_{y_\beta} h(y_\beta, X_\gamma) \quad p(X_\beta, X_\gamma) = h(X_\beta, X_\gamma) \sum_{y_\alpha} g(y_\alpha, X_\gamma) \tag{4.28}$$

$$p(X_\beta | X_\gamma) = \frac{h(X_\beta, X_\gamma)}{\sum_{y_\beta} h(y_\beta, X_\gamma)} \tag{4.29}$$

Daraus folgt Gleichung (4.25), Gleichung (4.27) folgt durch Division durch $p(X_\gamma)$. Mit $\tilde{h}_\gamma(X_\gamma) = \sum_{y_\beta} h(y_\beta, X_\gamma)$ ist der Beweis vollständig. (Lemma 4.14) ■

Mit diesem Hilfslemma und der Bedingung aus Definition 4.13 kann man nun den zentralen Faktorisierungssatz beweisen. Er erlaubt es, eine Brücke zu schlagen zwischen additiven Zerlegungen von Funktionen und in polynomieller Zeit berechenbaren Faktorisierungen.

Satz 4.15 (Faktorisierungssatz). *Sei $p(x)$ eine Boltzmannverteilung zu f auf X, also*

$$p(x) = \frac{e^{\beta f(x)}}{Z_f(\beta)} \tag{4.30}$$

mit $f(x) = \sum_{i=1}^m f_{s_i}(x_{s_i})$. Wenn die s_i die running intersection property *erfüllen, so gilt*

$$p(X) = \prod_{i=1}^m p(X_{b_i} | X_{c_i}) \tag{4.31}$$

Beweis: Der Beweis findet sich in abgewandelter Form in der Literatur (Whittaker, 1990), er lässt sich aber auch leicht direkt führen (vgl. (Mühlenbein et al., 1999)). Führe den Beweis durch inverse Induktion über i. Die Induktionsbehauptung ist wie folgt. Falls

$$p(X) = p(X_{d_i})p(X_{b_{i+1}}|X_{c_{i+1}})\cdots p(X_{b_m}|X_{c_m}), \tag{4.32}$$
$$\text{und}\quad p(X_{d_i}) = F_{s_1}^{(i)}(X)\cdots F_{s_i}^{(i)}(X) \tag{4.33}$$

dann existieren Funktionen $F_{s_j}^{(i-1)}(X)$ mit $j = 1,\ldots,i-1$, die nur von den Variablen in s_j abhängen, so dass

$$p(X) = p(X_{d_{i-1}})p(X_{b_i}|X_{c_i})\cdots p(X_{b_m}|X_{c_m}) \tag{4.34}$$
$$\text{und}\quad p(X_{d_{i-1}}) = F_{s_1}^{(i-1)}(X)\cdots F_{s_{i-1}}^{(i-1)}(X) \tag{4.35}$$

Mit $i = m$ setze $F_{s_j}^{(m)}(X) := \exp\big(\beta f_{s_j}(X)\big)$ mit $j > 1$ und $F_{s_1}^{(m)}(X) := \exp\big(\beta f_{s_1}(X)\big)/Z_u$. Dann sind die Voraussetzungen erfüllt, weil $p(X_{d_m}) = p(X)$, und damit die Induktionsverankerung geschafft.

Nun sei $1 < i \leq m$. Mit (4.33) folgt

$$p(X_{d_i}) = \underbrace{F_{s_1}^{(i)}(X)\cdots F_{s_{i-1}}^{(i)}(X)}_{g(X)}\cdot\underbrace{F_{s_i}^{(i)}(X)}_{h(X)} \tag{4.36}$$

Mit $\alpha := d_{i-1}\setminus c_i$, $\beta = b_i$ und $\gamma = c_i$, weil $s_i = b_i \cup c_i$ und $b_i \cap d_{i-1} = \emptyset$, folgt mit Lemma 4.14

$$p(X_{d_i}) = p(X_{d_{i-1}})p(X_{b_i}|X_{c_i}) \tag{4.37}$$
$$p(X_{d_{i-1}}) = F_{s_1}^{(i)}(X)\cdots F_{s_{i-1}}^{(i)}(X)\tilde{h}_{c_i}(X) \tag{4.38}$$

Wegen der Annahme $c_m \subseteq s_j$ für ein $j < i$ setze

$$F_{s_k}^{(i-1)}(X) := \begin{cases} F_{s_k}^{(i)}(X) & ,k \neq j \\ F_{s_k}^{(i)}(X)\cdot\tilde{h}_{c_i}(X) & ,k = j \end{cases} \tag{4.39}$$

und daher

$$p(X_{d_{i-1}}) = F_{s_1}^{(i-1)}(X)\cdots F_{s_{i-1}}^{(i-1)}(X) \tag{4.40}$$

Durch Einfügen von (4.37) in (4.32) erhält man

$$p(X) = p(X_{d_i})p(X_{b_{i+1}}|X_{c_{i+1}})\cdots p(X_{b_m}|X_{c_m}) \tag{4.41}$$
$$= p(X_{d_{i-1}})p(X_{b_i}|X_{c_i})p(X_{b_{i+1}}|X_{c_{i+1}})\cdots p(X_{b_m}|X_{c_m}) \tag{4.42}$$

Dies beweist den Induktionsschritt. (Faktorisierungssatz) ∎

Anmerkung 4.16. Um von einem Faktorisierungssystem auf eine Faktorisierung nach (4.31) zu kommen, muss man praktisch durch die Liste an Teilfunktionen gehen, alle *neuen* Variablen links von der Konditionierung schreiben und alle *alten* Variablen rechts. Also:

$$f(x) = f_{1,2}(x_1, x_2) + f_{2,3}(x_2, x_3) + f_{3,4}(x_3, x_4) \tag{4.43}$$

$$\Longrightarrow p(X) = p(X_1, X_2)p(X_3|X_2)p(X_4|X_3) \tag{4.44}$$

Die *running intersection property* fordert nun, dass alle Variablen, auf die konditioniert wird, schon *gemeinsam* in einem Faktor aufgetaucht sind. Deshalb erfüllt

$$p(X) = p(X_1)p(X_2|X_1)p(X_3|X_1)p(X_4|X_2, X_3) \tag{4.45}$$

diese Eigenschaft *nicht*, denn X_2 und X_3 treten niemals gemeinsam in den ersten drei Faktoren auf. Es ist übrigens egal, auf welcher Seite der Konditionierung das passiert, $p(X) = p(X_1)p(X_2|X_1)p(X_3|X_1, X_2)p(X_4|X_2, X_3)$ ist zulässig.

Die Zerlegung ist nicht eindeutig. Man kann natürlich (ohne Veränderung der Funktion an sich) eine einzige Teilfunktion nehmen, die alle Variablen umfasst. Wünschenswert ist dabei eine *minimale* Faktorisierung (mit minimaler Anzahl an Parametern für die Verteilung), die dennoch gültig ist. Dies wird in Kapitel 5 weiter diskutiert.

4.2.2 Der *FDA*: Factorized Distribution Algorithm

Mit dieser Zerlegung kann der *BEDA* in einen praktikablen Algorithmus, Algorithmus 4.3, überführt werden, den *FDA Factorized Distribution Algorithm* (Mühlenbein et al., 1999).

Algorithmus 4.3: *FDA* – Factorized Distribution Algorithm

```
Bestimme aus der Zerlegung der Funktion ein Faktorisierungssystem.
Erzeuge eine zufällige Ausgangspopulation mit N Individuen 𝔛^i.
do {
    Selektiere N̂ ≤ N Individuen 𝔛̂^i.
    Berechne die konditionalen Wahrscheinlichkeiten p(x_{b_i}|x_{c_i}, t) aus
    den selektierten Punkten.
    Erzeuge neue Punkte gemäß p(x, t+1) = ∏_{i=1}^{l} p(x_{b_i}|x_{c_i}, t).
    t ⇐ t + 1.
} until (Abbruchkriterium erreicht)
```

Der *FDA* berücksichtigt jede Interaktion zwischen den Variablen. Die Frage ist nun, ob die Fokussierungseigenschaft des *BEDA* auch für eine reale Implementierung des *FDA* gültig ist. Es gibt zwei stochastische Prozesse im *FDA*: Die Selektion (im Fall der Boltzmann-Selektion) und die Erzeugung neuer Punkte. Durch diese Prozesse wird sich die durch

Gleichung (4.31) erzeugte Verteilung von der gewünschten Boltzmann-Verteilung unterscheiden. Dem kann man nach den Gesetzen der großen Zahlen durch eine große Population entgegenwirken. Leider ist es noch nicht gelungen, dies für den gesamten Prozess zu quantifizieren. Es ist also nicht klar, wie groß die Population tatsächlich sein muss, um Konvergenz zu dem globalen Optimum zu erhalten.

Der zweite Punkt betrifft die Existenz des globalen Optimums in der Startpopulation. Diese Voraussetzung war im Beweis des Fokussierungssatzes nötig, um eine positive Wahrscheinlichkeit für jeden Punkt zu haben. Die Faktorisierung entschärft aber diese Problematik. Man muss sich vor Augen führen, dass die abgebildete Verteilung eben nicht die Verteilung der selektierten Punkte ist, sonder die Verteilung aus (4.31).

Betrachte als Extremfall eine lineare binäre Funktion mit $n=100$ Variablen. Ohne Kenntnisname der Linearität hat der Suchraum 2^{100} Punkte. Man braucht also ein Vielfaches davon als Populationsgröße, um die Verteilung zu beschreiben. Die Boltzmann-Verteilung faktorisiert in diesem Fall aber zu

$$p(X) = \prod_{i=1}^{100} p_i(X_i) \tag{4.46}$$

Die einzelnen Randverteilungen $p_i(X_i)$ lassen sich aber schon mit wenigen Punkten sehr genau schätzen. In (Vapnik, 1998) wird vorgeschlagen, 20mal so viele Sample wie freie Parameter der Verteilung zu nehmen. Dies führt zu einer linearen Populationsgröße für den *UMDA*. Beachte auch, dass jeder Punkt positive Wahrscheinlichkeit hat, solange alle $p_i(X_i)$ nicht die Extremwerte 0 oder 1 annehmen.

Die Frage der Konvergenz hängt also für den *FDA* essentiell von der Struktur des Problems ab: Eine einfache Struktur benötigt weniger Parameter für die Verteilung und damit auch eine niedriger Populationsgröße.

Ob Konvergenz zum globalen Optimum erzielt wird, hängt sicherlich auch von der Geschwindigkeit der Abkühlung ab. Der Konvergenzsatz aus *Simulated Annealing* für die inhomogene Markovkette lässt sich leider nicht ohne weiteres auf den *FDA* übertragen. Bei *Simulated Annealing* ist das Prinzip wie folgt (Abbildung 4.1): Für festes β erzeugt die Markov-Kette Sample aus der (exakten) Boltzmann-Verteilung, wenn man sie lange genug iteriert. Wenn nun β gegen unendlich geht, fokussiert die Verteilung um die globalen Maxima. Der logarithmische Schedule entsteht nun aus der Frage, unter welcher Bedingung eine *Diagonalfolge* dieser Folge von Folgen konvergiert.

Weil der *FDA* zu gegebenem β Punkte aus der Vorgängergeneration generiert, wird die Boltzmann-Verteilung mit steigender Populationsgröße *nicht* immer genauer angenähert, weil sich Fehler, die in den Vorgängerpopulationen gemacht wurden, fortpflanzen. Steigende Populationsgröße kann nur bewirken, dass die Verteilung, die durch die selektierten Punkte beschrieben wird, besser approximiert wird. Um also eine *insgesamt* bessere Approximation der echten Boltzmann-Verteilung zu haben, muss bereits in der ersten Generation die Populationsgröße angehoben werden. Und an dieser Stelle versagt das Diagonalverfahren: Wenn bereits in der ersten Generation die Populationsgröße gegen unendlich gehen muss, entsteht kein implementierbarer Algorithmus.

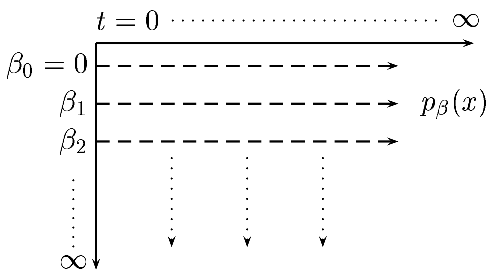

Abbildung 4.1: Konvergenz der inhomogenen Markovkette

Natürlich könnte man versuchen, die Approximationsfehler abzuschätzen und so Aussagen über Abkühlungsgeschwindigkeit und benötigte Populationsgrößen erhalten, wie oben erwähnt. Dies ist äquivalent zu der Fragestellung, wie gut die Boltzmann-Verteilung durch eine endliche Zahl von Sampeln approximiert wird. Die Antwort darauf ist aber erheblich schwieriger zu finden als die auf die Frage nach dem optimalen Schedule bei unendlicher Rechenzeit.

Auch für *Simulated Annealing* gibt es noch keine Lösung auf die Frage, wie man am Besten abkühlt, wenn man nur endliche Rechenzeit zur Verfügung hat: *„...it is unclear how to efficiently find an optimal temperature sequence ... for a problem instance. It may be that computing such a sequence may be far more difficult than to solve the problem instance“* (Hajek and Sasaki, 1989).

Beachte, dass der *FDA* sich auch von einem N-fach parallelen *Simulated Annealing* wesentlich unterscheidet, weil es auf die *Abhängigkeiten* zwischen den einzelnen Punkten ankommt, die dann zu empirischen Häufigkeiten der Variablen destilliert werden. Der *FDA* lässt sich zwar auch als Markov-Kette modellieren. Der Zustandsraum besteht hier aber nicht aus den einzelnen Punkten des Suchraumes, sondern aus den Parametern der Verteilung, die die selektierten Punkte beschreiben. Allein die (zeitinvarianten) Übergangswahrscheinlichkeiten zu beschreiben, ist eine sehr aufwändige Aufgabe, von der Analyse der Grenzverteilung des Prozesses ganz zu schweigen.

Da eine theoretische Behandlung nicht gelungen ist, werde ich mich auf eine empirische Untersuchung stützen. Anhand von Testfunktionen mit bekanntem Optimum kann so eine empirische Schätzung der benötigten Populationsgröße erfolgen.

Für den *FDA* stellt sich ferner das Problem von *Struktur* versus *Instanz*. Da der Algorithmus die aktuelle Belegung der Teilfunktionen gar nicht berücksichtigt (der Aufwand pro Generation ist unabhängig davon), können bestimmte Vereinfachungen nicht gemacht werden. So kann man die *OneMax*-Funktion auch als *eine* Teilfunktion mit n Variablen kodieren und hat dann Aufwand 2^n. Im nächsten Kapitel werden Methoden vorgestellt, wie die Abhängigkeiten nicht durch die vorher bekannte Struktur ermittelt werden, sondern aus den generierten Punkten geschätzt werden.

4.2.3 Vergleich mit *Markov Random Fields*

Es gibt einige Ähnlichkeiten zwischen dem *FDA* mit Boltzmann-Selektion und den *Markov Random Fields* aus zum Beispiel (Winkler, 1995). Die Fitness-Funktion entspricht dort genau den Potentialen, die auf die gleiche Weise nur Funktionen von Teilmengen der Variablenmenge sind. Die zugrunde liegende Verteilung ist auch die Boltzmann-Verteilung. Es ergeben sich analog lokale Ausdrücke für die konditionalen Wahrscheinlichkeiten von Variablen, gegeben ihrer Nachbarn.

Der wesentliche Unterschied ist es, dass bei dem *FDA* gefordert ist, dass sich die gemeinsame Verteilung als das Produkt dieser lokalen Ausdrücke schreiben lässt. Dadurch kann man direkt die Boltzmann-Verteilung bestimmen und ist nicht auf einen Prozess angewiesen, der die gewünschte Verteilung als Grenzverteilung erzeugt.

Die Kehrseite ist es, dass die Faktorisierung zu exponentiellem Aufwand führen kann wie zum Beispiel bei dem Gitter, Abschnitt 4.2.6.

4.2.4 Erzeugung der Faktorisierung

Die Wahl der Faktorisierung im ersten Schritt des *FDA* ist ein schwieriges Problem. Unter alleiniger Ausnutzung der s_i aus der additiven Zerlegung wird man selten die *running intersection property* erfüllen können. So dürfen zum Beispiel höchstens n Teilfunktionen benutzt werden.

Der *FDA* benötigt lediglich ein Faktorisierungssystem. Algorithmus 4.4 (siehe (Mühlenbein and Mahnig, 1999b)) ist ein einfaches Verfahren, welches aus einer beliebigen Aufteilung ein Faktorisierungssystem erzeugt. Dann gilt der Faktorisierungsatz natürlich nicht mehr, die Boltzmann-Verteilung ist nicht notwendigerweise identisch zu dieser berechneten Faktorisierung.

Algorithmus 4.4: *FDA*–Faktorisierung

$i \Leftarrow 1$. Wähle eine Teilfunktion $\tilde{s}_1$ aus den s_j.
do {
    Bestimme $\tilde{d}_i \Leftarrow \bigcup_{j=1}^{i} \tilde{s}_j$.
    Wähle s_k mit maximaler Schnittmenge zu $\tilde{d}_i$, $s_k \cap \tilde{d}_i \neq \emptyset$.
    $\tilde{s}_{i+1} \Leftarrow s_k$, $i \Leftarrow i + 1$.
} **until** $i = l$ oder keine geeignete Menge gefunden wurde.
Bestimme die Faktorisierung nach Satz 4.15 mit den Mengen $\tilde{s}_i$.

Beispiel 4.1. Betrachte eine Zerlegung der Funktion durch die Mengen $s_1 = \{1, 2\}$, $s_2 = \{2, 3\}$, $s_3 = \{3, 4\}$ und $s_4 = \{1, 4\}$. Keine Reihenfolge erzeugt ein Faktorisierungssystem. Algorithmus 4.4 würde zum Beispiel die Faktorisierung $p(X_1, X_2)p(X_3|X_2)p(X_4|X_3)$ generieren und somit die Teilfunktion $f_{1,4}$ ignorieren.

Im ersten und im vierten Schritt gibt es die Wahl zwischen mehreren möglichen Kandidaten. In (Mühlenbein and Mahnig, 1999b) wurde hierfür folgendes Kriterium herangezogen: *Wähle die Teilfunktion, die maximal nicht-linear ist.* Nicht-Linearität ist definiert als der quadratische Abstand zur linearen Regressionskurve. Dieses Kriterium wurde gewählt, weil es leicht zu berechnen ist, nur lokale Information braucht (also nur die Werte der entsprechenden Teilfunktion), und es folgendes Kriterium erfüllt: Wenn die Teilfunktion $f_{i,j}$ sich als Summe von zwei Funktionen f_i und f_j schreiben lässt, so bekommt sie den Koeffizienten Null; ansonsten ist der Koeffizient größer Null.

Schöner wäre es natürlich, ein sauber definiertes Maß für den Raum der Faktorisierungen zur Verfügung zu haben und danach die Wahl zu treffen. Sich aber hier auf die additiven Zerlegungen und dazu passende Faktorisierungen einzuschränken, führt in eine Sackgasse. In Kapitel 5 wird allgemeiner eine Modellierung von Faktorisierungen durch Graphen besprochen, als Nebenergebnis dieser Theorie fällt dann ein theoretisch besser fundiertes Verfahren als Algorithmus 4.4 dabei ab.

Für einfache Fälle (zum Beispiel Ketten) liefert der Algorithmus die exakte Faktorisierung.

4.2.5 Verschmelzen von Teilfunktionen

Die Faktorisierung benötigt die s_i, die eine Zerlegung der Funktion in Teilfunktionen beschreibt. Algorithmus 4.4 trifft dabei einfach eine Auswahl, um zu einem Faktorisierungssystem zu kommen. Es gilt allerdings noch die folgende Möglichkeit, zwei oder mehr Funktionen zu verschmelzen:

Definition 4.17. *Zwei Teilfunktionen f_{s_i} und f_{s_j} lassen sich zu einer neuen Teilfunktion f_s verschmelzen, mit*

$$s := s_i \cup s_j \qquad\qquad f_s(x_s) := f_{s_i}(x_{s_i}) + f_{s_j}(x_{s_j}) \tag{4.47}$$

Die Teilfunktion f_s lässt sich dann anstelle von den zwei Ursprungsfunktionen für die Faktorisierung verwenden. Analog kann man auch mehr als zwei Funktionen verschmelzen.

Theoretisch lässt sich mit dieser Operation auch ein Faktorisierungssystem erzeugen, welches sogar die *running intersection property* erfüllt, indem bei Bedarf die gewählte Teilfunktion mit einer anderen verschmolzen wird.

Beispiel 4.2. Betrachte noch einmal Beispiel 4.1 von Seite 71. Es gibt (abgesehen von Symmetrien) folgende mögliche Verschmelzungen:

- Alle Teilfunktionen zusammen liefern die Faktorisierung $p(X_1, X_2, X_3, X_4)$.
- s_1 und s_2 sowie s_3 und s_4 liefert $p(X_1, X_2, X_3)p(X_4|X_1, X_3)$.

- Verschmelzen von s_2 und s_3 erzeugt keine eindeutige Faktorisierung. Je nachdem, in welcher Reihenfolge die Teilfunktionen in Algorithmus 4.4 gewählt werden, entsteht die Faktorisierung $p(X_1, X_2)p(X_3, X_4|X_2)$ oder $p(X_2, X_3, X_4)p(X_1|X_2)$ (wobei jeweils eine Teilfunktion ignoriert werden musste), oder aber die Zerlegung $p(X_1, X_2)p(X_4|X_1)p(X_3|X_2, X_4)$, bei der die *running intersection property* nicht erfüllt ist.

Die nicht erfüllte *running intersection property* führt zu folgendem Problem. Angenommen, die Parameter werden aus folgenden Datenpunkten geschätzt:

$$\mathfrak{X} = \{1001, 0011, 0110, 1110\} \tag{4.48}$$

Dann gilt $p(X_1, X_2) = 1/4$ und $p(X_4|X_1) = 1/2$. Wenn nun nach dem Samplingverfahren Punkte generiert werden, so wird mit Wahrscheinlichkeit $1/4$ die Startbelegung $X_2 = X_4 = 0$ gewählt. Die konditionale Wahrscheinlichkeit $p(X_3|X_2{=}0, X_4{=}0)$ ist aber nicht definiert, da nach den Daten $p(X_2{=}0, X_4{=}0) = 0$ gilt!

Mit einer solchen Faktorisierung kann es also passieren, dass der Nenner in der konditionalen Wahrscheinlichkeit in (4.31) Null wird. In diesem Fall wird behelfsweise gleichverteilt gezogen. Die Gründe dazu werden in Abschnitt 5.9.1 diskutiert.

Lemma 4.18. *Wenn die Teilmengen s_i die* running intersection property *erfüllen und die Parameter der nach dem Faktorisierungssatz zugehörigen Verteilung p als gezählte Häufigkeiten geschätzt werden, so entstehen beim Sampling keine undefinierten Wahrscheinlichkeiten.*

Beweis: Betrachte die Faktorisierung $\prod_{i=1}^{m} p(X_{b_i}|X_{c_i})$ und beim Sampeln den Schritt i. Es muss also eine Belegung für die Variablen X_{b_i} erzeugt werden nach der Verteilung $p(X_{b_i}|x_{c_i})$, wobei die Variablen X_{c_i} in den früheren Schritten belegt wurden.

Eine undefinierte Situation gibt es nur für den Fall, dass $p(x_{c_i}) = 0$ ist. Nun gilt aber nach Voraussetzung $c_i \subseteq s_j$ für ein $j < i$, damit gilt auch $p(x_{s_j}) = 0$, ganz egal wie die restlichen Variablen aus X_{s_j} belegt sind. Das heißt aber, dass in Schritt j die Belegung $X_{s_j} = x_{s_j}$ gar nicht zustande kommen konnte, da die Häufigkeit Null wahr. (Lemma 4.18) ■

4.2.6 Beispiel eines schwierigen Falls: Das Gitter

Als Beispiel für eine Problemklasse, bei der die *running intersection property* zu Problemen führt, sei hier das Gitter genannt. In Abbildung 4.2 ist die Interaktion dargestellt. Die Variablen sind der Anschaulichkeit halber durch zwei Indizes gekennzeichet, die Teilfunktionen hängen je von zwei Variablen ab, die in der Abbildung durch eine Linie verbunden sind.

$$
\begin{array}{ccccccc}
X_{1,1} & — & X_{1,2} & — & X_{1,3} & \cdots\cdots & X_{1,n} \\
| & & | & & | & & | \\
X_{2,1} & — & X_{2,2} & — & X_{2,3} & \cdots\cdots & X_{2,n} \\
| & & | & & | & & | \\
X_{3,1} & — & X_{3,2} & — & X_{3,3} & \cdots\cdots & X_{3,n} \\
\vdots & & \vdots & & \vdots & \ddots & \vdots \\
X_{n,1} & — & X_{n,2} & — & X_{n,3} & \cdots\cdots & X_{n,n}
\end{array}
$$

Abbildung 4.2: Ein Gitter als Zerlegung einer Funktion

Der einfache Faktorisierungsalgorithmus 4.4 lässt hier etwa die Hälfte aller Kanten weg. Eine Beispiellösung berücksichtigt nur die waagerechten Kanten sowie die erste senkrechte Spalte:

$$\tilde{p}(X) = p(X_{1,1}) \left(\prod_{i=2}^{n} p(X_{1,i}|X_{1,i-1}) \right) \prod_{j=2}^{n} \left[p(X_{j,1}|X_{j-1,1}) \prod_{i=2}^{n} p(X_{j,i}|X_{j,i-1}) \right] \tag{4.49}$$

Ein Faktorisierungsystem, welches auch noch die *running intersection property* erfüllt, erzeugt sehr große Teilfunktionen. Wenn man die Variablen in Viererblöcken zusammenfasst und entsprechend faktorisiert, wird zwar keine Kante ausgelassen, aber die *running intersection property* ist immer noch nicht erfüllt:

$$\begin{aligned} \tilde{p}(X) = p(X_{1,1}, X_{1,2}, X_{2,1}, X_{2,2}) p(X_{1,3}, X_{2,3}|X_{1,2}, X_{2,2}) \cdots p(X_{1,n}, X_{2,n}|X_{1,n-1}, X_{2,n-1}) \\ \cdot p(X_{3,1}, X_{3,2}|X_{2,1}, X_{2,2}) p(X_{3,3}|X_{2,2}, X_{2,3}, X_{3,2}) \cdots p(X_{3,n}|X_{2,n-1}, X_{2,n}, X_{3,n-1}) \cdots \end{aligned} \tag{4.50}$$

Hier verletzt als erstes $p(X_{3,3}|X_{2,2}, X_{2,3}, X_{3,2})$ die *running intersection property*. Um dies auch noch zu erreichen, kann man (zum Beispiel) zeilenweise vorgehen: Die erste Zeile als Block verschmelzen und dann jede Zeile auf die vorgehende konditionieren:

$$\tilde{p}(X) = p(X_{1,1}, \ldots, X_{1,n}) \prod_{j=2}^{n} p(X_{j,1}, \ldots, X_{j,n}|X_{j-1,1}, \ldots, X_{j-1,n}) \tag{4.51}$$

Damit haben die Faktoren $2n$ Parameter und der Aufwand ist exponentiell.

Das zweidimensionale binäre magnetische Ising-Modell lässt sich durch diese Struktur beschreiben.

4.2.7 Zusammenhang zwischen Faktorisierungen und vergrößerter Zahl an Allelen

So, wie man Zahlen in beliebigen Zahlensystemen darstellen kann, kann man den Übergang auf eine größere Zahl von Allelen (also möglichen Werten für die Variablen) auch durch eine Faktorisierung mit einer größeren Zahl von Variablen mit dafür kleinerem Wertebereich erzielen.

So kann man eine Variable Y mit Wertebereich $\{0, \ldots, 7\}$ auch durch drei Variablen X_1, X_2 und X_3 mit binärem Wertebereich repräsentieren (und umgekehrt). Dabei gilt dann zum Beispiel

$$p(Y) = p(X_1, X_2, X_3) \tag{4.52}$$

Die Zahl der (echten) Parameter ist für beide Verteilungen 7. Daher ist die Einschränkung auf binären Wertebereich keine Reduktion in der Darstellungsmöglichkeit. Im Extremfall kann man die gesamte Variablenmenge zu einer Hyper-Variablen zusammenfassen, die dann aber eine exponentiell große Wertemenge hat.

Diese Art der Transformation kann man umgekehrt auch nutzen, um den *UMDA* auf *separable* Faktorisierungen zu erweitern, indem man die Blöcke zusammenfasst.

Beispiel 4.3. Betrachte die (binäre) Funktion $f_{1,2}(x_1, x_2) + f_{3,4,5}(x_3, x_4, x_5)$. Mit den neuen Variablen Y_1 und Y_2 mit Wertebereich $\{0, \ldots, 3\}$ bzw. $\{0, \ldots, 7\}$ ergibt sich eine neue Funktion $f_1(y_1) + f_2(y_2)$, die der (Multi-Allel)-*UMDA* wegen der Linearität problemlos optimiert werden kann. Der *FDA* mit der dazu passenden Faktorisierung

$$p(X) = p(X_1, X_2)p(X_3, X_4, X_5) \tag{4.53}$$

hat die gleiche Zahl an Parametern und verhält sich auch identisch. Deshalb wird der der Multi-Allel-*UMDA* in der Praxis nicht benötigt.

Beachten muss man dabei, dass das Zusammenfassen von mehreren Variablen, wenn die zugrunde liegende Funktion keine hohe Interaktion zwischen diesen hat, die benötigte Populationsgröße ansteigen lässt, weil eine Variable mit größerem Wertebereich auch mehr Datenpunkte braucht, um die Verteilung zu schätzen. Dies wird in Kapitel 8 weiter diskutiert.

Im Extremfall kann man die gesamte Variablenmenge zu einer Hyper-Variablen zusammenfassen, die dann aber eine exponentiell große Wertemenge hat. Weitere Einzelheiten dieser Technik sind in (Mühlenbein and Mahnig, 1999a) diskutiert.

4.3 Ein Abkühlungsverfahren für die Boltzmann-Verteilung

Für Boltzmann-Selektion stellt sich immer noch die Frage nach dem Abkühlungsverfahren. Lemma 4.8 hat gezeigt, wie schnell man *mindestens* abkühlen sollte, um Konvergenz in einem bestimmten Zeitrahmen zu erzielen.

Wenn man aber zu schnell abkühlt, wird die Approximation der Boltzmann-Verteilung mit fester Populationsgröße immer schlechter. Wenn man etwa so schnell abkühlt, dass bereits nach einem Schritt das Optimum mit 30% Wahrscheinlichkeit in der Verteilung vorliegt, so muss die erste zufällig gezogene Population bereits das Optimum enthalten, also exponentielle Größe haben.

Um Kontrolle über die Abkühlung zu gewinne, wird eine Reihenentwicklung für die durchschnittliche Fitness der Boltzmann-Verteilung gezeigt.

4.3.1 Reihenentwicklung der durchschnittlichen Fitness

Die durchschnittliche Fitness wird hier entwickelt über Ableitungen der Zustandssumme. Dabei bezeichnen $Z'_f(\beta)$ die erste und $Z_f^{(k)}(\beta)$ die k-te Ableitung der Zustandssumme nach β.

Satz 4.19 (Reihenentwicklung der Boltzmann-Verteilung). *Es gilt für die durchschnittliche Fitness der Boltzmann-Verteilung $W_f(\beta)$ folgende Reihenentwicklung in β:*

$$W_f(\tilde{\beta}) = W_f(\beta) + \sum_{i \geq 1} \frac{(\tilde{\beta} - \beta)^i}{i\,!} M^z_{i+1}(\beta) \tag{4.54}$$

Dabei lassen sich die zentrierten Momente über die Ableitungen der Zustandssumme bestimmen:

$$M^z_{i+1}(\beta) = \left(\frac{Z'_f(\beta)}{Z_f(\beta)} \right)^{(i)} \qquad \text{für } i \geq 1, \quad M^z_1 = 0 \tag{4.55}$$

Beweis: Für die k-te Ableitung der Zustandssumme gilt für $k \geq 0$:

$$Z_f^{(k)}(\beta) = \sum_{x \in \mathcal{R}} f(x)^k e^{\beta f(x)} \tag{4.56}$$

und somit für die Momente für $k \geq 1$

$$M_k(\beta) = \frac{Z_f^{(k)}(\beta)}{Z_f(\beta)} \tag{4.57}$$

also

$$W_f(\beta) = M_1(\beta) = Z'_f(\beta)/Z_f(\beta). \tag{4.58}$$

Direktes Ausrechnen der Ableitungen von W führt wegen des Nenners zu länglichen Ausdrücken. Betrachte daher die verschobene Fitness-Funktion $\tilde{f}(x) := f(x) + r.\ \tilde{p}_\beta(x)$

bezeichne die dazugehörige Boltzmannverteilung, $\tilde{Z}_f$ und $\tilde{M}_k^z$ entsprechend Zustandssumme und Momente. Sei $C := e^{\beta r}$. Dann gilt für $k \geq 1$:

$$\tilde{Z}_f(\beta) = \sum_x e^{\beta(f(x)+r)} = C \cdot Z_f(\beta) \tag{4.59}$$

$$\tilde{p}_\beta(x) = \frac{e^{\beta(f(x)+r)}}{\tilde{Z}_f} = \frac{C \cdot e^{\beta f(x)}}{C \cdot Z_f} = p_\beta(x) \tag{4.60}$$

$$\tilde{Z}'_f(\beta) = \sum_x \tilde{f}(x) e^{\beta(f(x)+r)} = \sum_x f(x) C e^{\beta f(x)} + r \sum_x C e^{\beta f(x)} \tag{4.61}$$

$$= C \cdot Z'_f(\beta) + rC \cdot Z_f(\beta) \tag{4.62}$$

$$\tilde{W}_f(\beta) = \tilde{M}_1(\beta) = \frac{\tilde{Z}'_f(\beta)}{\tilde{Z}_f(\beta)} = \frac{Z'_f(\beta)}{Z_f(\beta)} + r = M_1(\beta) + r \tag{4.63}$$

$$\tilde{W}_f^{(k)}(\beta) = \left(\frac{Z'_f(\beta)}{Z_f(\beta)} + r\right)^{(k)} = \left(\frac{Z'_f(\beta)}{Z_f(\beta)}\right)^{(k)} = W_f^{(k)}(\beta) \tag{4.64}$$

$$\tilde{M}_k^z(\beta) = \sum_x \left[(f(x)+r) - (M_1(\beta)+r)\right]^k \tilde{p}_\beta(x) = M_k^z(\beta) \tag{4.65}$$

Daraus folgt also, dass die Ableitungen von W sowie die zentrierten Momente gleich bleiben, wenn die Fitnessfunktion verschoben wird. Benutze nun das folgende Hilfslemma:

Lemma 4.20. *Für ein β^* gelte $\tilde{Z}'_f(\beta^*) = 0$. Dann gilt für $k \geq 1$:*

$$\left(\frac{\tilde{Z}'_f(\beta^*)}{\tilde{Z}_f(\beta^*)}\right)^{(k-1)} = \frac{\tilde{Z}_f^{(k)}(\beta^*)}{\tilde{Z}_f(\beta^*)} \tag{4.66}$$

Beweis: Der Beweis erfolgt über vollständige Induktion. Für $k = 1$ ist das Lemma offensichtlich wahr. Als Induktionsschritt folgt

$$\left(\frac{\tilde{Z}'_f(\beta^*)}{\tilde{Z}_f(\beta^*)}\right)^{(k)} = \left[\left(\frac{\tilde{Z}'_f(\beta^*)}{\tilde{Z}_f(\beta^*)}\right)^{(k-1)}\right]' \overset{\text{I.V.}}{=} \left(\frac{\tilde{Z}_f^{(k)}(\beta^*)}{\tilde{Z}_f(\beta^*)}\right)' \tag{4.67}$$

$$= \frac{\tilde{Z}_f^{(k+1)}(\beta^*)}{\tilde{Z}_f(\beta^*)} - \underbrace{\frac{\tilde{Z}_f^{(k+1)}(\beta^*) \cdot \tilde{Z}'_f(\beta^*)}{\tilde{Z}_f(\beta^*)^2}}_{=0} \tag{4.68}$$

(Lemma 4.20) ■

Nun müssen nur noch die Teile zusammengefügt werden. Sei β beliebig, aber fest. Mit $r := -Z'_f(\beta)/Z_f(\beta) = -M_1(\beta)$ folgt $\tilde{Z}'_f(\beta) = 0$. Dann gilt $\tilde{M}_1(\beta) = M_1(\beta) + r = 0$ und somit

$$\tilde{M}_k^z(\beta) = \tilde{M}_k(\beta) \tag{4.69}$$

Weiter gilt für $k \geq 1$:

$$W_f^{(k)}(\beta) \overset{(4.64)}{=} \tilde{W}_f^{(k)}(\beta) = \left(\frac{\tilde{Z}'_f(\beta)}{\tilde{Z}_f(\beta)}\right)^{(k)} \overset{(4.66)}{=} \frac{\tilde{Z}_f^{(k+1)}(\beta)}{\tilde{Z}_f(\beta)} \overset{(4.57)}{=} \tilde{M}_{k+1}(\beta) \tag{4.70}$$

$$\overset{(4.69)}{=} \tilde{M}_{k+1}^z(\beta) \overset{(4.65)}{=} M_{k+1}^z(\beta) \tag{4.71}$$

Der Beweis war für ein bestimmtes r geführt. Da Start- und Endausdrücke aber nicht mehr von r abhängen, gilt die Gleichung universell. Mit der Taylor-Entwicklung von $W_f(\beta)$ ist der Beweis fertig. (Satz 4.19) ■

Korollar 4.21. *Es gilt in erster Näherung:*

$$W_f(\tilde{\beta}) \approx W_f(\beta) + (\tilde{\beta} - \beta) \cdot \sigma_f^2(\beta) \tag{4.72}$$

Diese Näherung findet sich auch schon in (Kirkpatrick et al., 1983).

Lemma 4.22. *Für die Varianz der Boltzmann-Verteilung gilt:*

$$f(x) \neq konst. \implies \sigma_f^2(\beta) > 0 \tag{4.73}$$

Beweis: Es gilt $\forall x : p_\beta(x) > 0$. Damit $\sigma_f^2(\beta) = \sum_x \left[f(x) - W_f(\beta)\right]^2 p_\beta(x)$ gleich Null wird, muss somit für alle x gelten: $f(x) = W_f$ im Widerspruch zur Voraussetzung. (Lemma 4.22) ■

Korollar 4.23. *Es gilt, wenn $f(x) \neq konst.$:*

$$\tilde{\beta} > \beta \implies W_f(\tilde{\beta}) > W_f(\beta) \tag{4.74}$$

Dieses wichtige Korollar sagt, dass die durchschnittliche Fitness auch für Boltzmann-Selektion niemals abnimmt.

4.3.2 Ein neues Abkühlungsverfahren für Populationen

Aus (4.72) kann ein Abkühlungsschema hergeleitet werden. Die Varianz (wie natürlich auch die höheren Momente) kann aus den generierten Punkten einfach geschätzt werden. Solange die Näherung gültig ist, kann man sich einen Zuwachs in der durchschnittlichen Fitness „wünschen“ und dann einfach das zugehörige $\beta(t+1)$ wählen gemäß

$$\Delta\beta(t) := \beta(t+1) - \beta(t) = \frac{W_f^{\text{neu}}(t) - W_f(\beta(t))}{\sigma_f^2(\beta(t))} \tag{4.75}$$

Aus Gleichung (4.72) sieht man sofort, dass die Wahl von $\Delta\beta$ proportional zum Inversen der Varianz in erster Näherung zu einem konstanten Zuwachs in der durchschnittlichen

Fitness führt. Dies ist sicherlich insbesondere in der Nähe des Optimums zu schnell. In einer Fülle von Anwendungen hat sich *Truncation Selection* als robustes und effizientes Selektionsverfahren gezeigt. Versuche nun, die Boltzmann-Selektion ähnlich zu *Truncation Selection* mit

$$R_f(t) = I_\tau b\sqrt{\sigma_f^2} \tag{4.76}$$

anzunähern.

Faustregel. $\Delta\beta(t) = c/\sqrt{\sigma_f^2(\beta(t))}$ *liefert ein Abkühlungsverfahren, welches in erster Näherung die Fitness proportional zur Standardabweichung wachsen lässt:*

$$R_f(t) \approx c \cdot \sqrt{\sigma_f^2(\beta(t))} \tag{4.77}$$

Dieses Abkühlungsverfahren heißt SDS (standard deviation schedule).

Es ist schon bekannt, dass bei Boltzmann-Selektion die Addition einer Konstanten zur Fitnessfunktion an der Optimierung nichts ändert, weil die Verteilungen gleich bleiben. Jetzt gilt aber auch folgendes

Lemma 4.24. *Für Boltzmann-Selektion mit dem SDS, also*

$$\Delta\beta(t) = \frac{c}{\sqrt{\sigma_f^2(\beta(t))}} \tag{4.78}$$

ist BEDA invariant unter linearer Transformation der Fitnessfunktion mit positivem Faktor.

Beweis: Dieses Lemma gilt deshalb, weil die Standardabweichung unter linearen Transformationen ebenfalls linear skaliert. Es sei $f(x)$ also eine Fitnessfunktion, betrachte $\hat{f}(x) = \hat{c} \cdot f(x)$. Die Behauptung ist, dass $\hat{\beta}(t) = \beta(t)/\tilde{c}$, dann sind die Verteilungen für jedes t gleich. Für $t = 0$ sind β und $\hat{\beta}$ gleich Null, die Behauptung ist also wahr. Sei nun t und $\beta{=}\beta(t)$ gegeben und nach Induktionsvoraussetzung $\hat{\beta} = \beta/\hat{c}$.

Nach Lemma 4.2 gilt dann $p_{\beta,f}(x) = p_{\hat{\beta},\hat{f}}(x)$. Ferner gilt $\sigma_f^2(\beta) = \hat{c}^2 \cdot \sigma_{\hat{f}}^2(\hat{\beta})$. Damit folgt $\Delta\hat{\beta}(t) = \Delta\beta(t)/\hat{c}$ und somit die Behauptung. (Lemma 4.24) ∎

Satz 4.25. *Es sei* $\sigma(t) := \sqrt{\sigma_f^2(\beta(t))}$. *Dann ist der Selektionserfolg für Boltzmann-Selektion mit dem SDS gegeben durch*

$$R_f(t) = \sum_{i\geq 1} \frac{c^i}{i!\,\sigma(t)^i} M_{i+1}^z \tag{4.79}$$

$$= c \cdot \sigma(t) + \frac{c^2 M_3^z}{2\,\sigma(t)^2} + \frac{c^3 M_4^z}{6\,\sigma(t)^3} + \dots \tag{4.80}$$

4.4 Beispiele

4.4.1 Die Funktion *OneMax*

Für *OneMax* lässt sich die Boltzmannverteilung bestimmen. Es gilt mit $f = \mathit{OneMax}$

$$Z_f(\beta) = \sum_{x \in \mathcal{R}} e^{\beta|x|} = \sum_{i=0}^{n} \binom{n}{i} \left(e^{\beta}\right)^i = (1+e^{\beta})^n \tag{4.81}$$

Alle Randverteilungen bleiben gleich, wenn sie gleich starten. Es ergibt sich

$$Z_f(\beta) \cdot p_\beta(X_1 = 1) = \sum_{x, X_1 = 1} e^{\beta|x|} = \sum_{i=0}^{n-1} \binom{n-1}{i} e^{\beta(i+1)} = e^{\beta} \cdot (1+e^{\beta})^{n-1} \tag{4.82}$$

und daher

$$p_\beta(X_1{=}1) = \frac{e^{\beta}}{1+e^{\beta}} =: \hat{p}_\beta \tag{4.83}$$

Nach Satz 4.19 lässt sich die durchschnittliche Fitness $W_f(\beta)$ einfach ausrechnen durch $W_f(\beta) = Z_f'(\beta)/Z_f(\beta)$:

$$W_f(\beta) = \frac{n(1+e^{\beta})^{n-1}e^{\beta}}{(1+e^{\beta})^n} = \frac{ne^{\beta}}{1+e^{\beta}} \qquad \left(= n \cdot \hat{p}_\beta\right) \tag{4.84}$$

Dies folgt auch aus Korollar 3.25. Die Varianz ergibt sich nach (4.55) durch

$$\begin{aligned} \sigma_f^2(\beta) &= \tfrac{Z_f''(\beta)}{Z_f(\beta)} - \left(\tfrac{Z_f'(\beta)}{Z_f(\beta)}\right)^2 \\ &= \frac{n(n-1)(1+e^{\beta})^{n-2}e^{2\beta} + n(1+e^{\beta})^{n-1}e^{\beta}}{(1+e^{\beta})^n} - \frac{n^2e^{2\beta}}{(1+e^{\beta})^2} \\ &= \frac{ne^{\beta}}{(1+e^{\beta})^2} = n \cdot \hat{p}_\beta(1-\hat{p}_\beta) \end{aligned} \tag{4.85}$$

Damit folgt die Differenzengleichung für die inverse Temperatur

$$\beta(t+1) = \beta(t) + c \cdot \frac{1+e^{\beta(t)}}{\sqrt{n} \cdot e^{\beta(t)/2}} = \beta(t) + \frac{2c}{\sqrt{n}} \cosh\left({}^{\beta(t)}\!/_2\right) \tag{4.86}$$

Die höheren Momente lassen sich nun ebenfalls leicht ausrechnen, es ergibt sich für den Selektionserfolg nach (4.79)

$$R_f(t) = c\frac{e^{\beta/2}\sqrt{n}}{1+e^{\beta}} - c^2\frac{(e^{\beta}-1)}{2(e^{\beta}+1)} + c^3\frac{e^{-\beta/2}(1-4e^{\beta}+e^{2\beta})}{6(e^{\beta}+1)\sqrt{n}} + \ldots \tag{4.87}$$

Das Abkühlungsverfahren benutzt als Faustregel nur das erste Glied, also

$$R_f(t) \approx c \frac{e^{\beta/2}\sqrt{n}}{1+e^{\beta}} \tag{4.88}$$

Für eine geschlossene approximative Lösung kann man die Differenzengleichung für β in eine Differenzialgleichung umwandelt und erhält

$$\frac{d\beta}{dt} = \frac{c}{\sigma_{\beta(t)}} = \frac{2c}{\sqrt{n}} \cosh\bigl(\beta(t)/2\bigr) \tag{4.89}$$

Die Lösung dieser Differenzialgleichung liefert mit $\beta(0) = 0$:

$$\beta(t) = 4 \operatorname{argtanh}\left[\tan\left(\frac{c \cdot t}{2\sqrt{n}}\right)\right] \qquad \text{für } t \leq \frac{\pi\sqrt{n}}{2c} \tag{4.90}$$

Wenn man das wieder in die Gleichung (4.83) einsetzt, erhält man

$$\hat{p}_{\beta}(t) = \frac{1}{2}\left(1 + \sin\left(\frac{c \cdot t}{\sqrt{n}}\right)\right) \tag{4.91}$$

Vergleicht man diese Gleichung mit der Dynamik für *Truncation Selection* (3.133) von Seite 36), so zeigt sich, dass die beiden Gleichungen identisch sind. Das c spielt dabei die Rolle von I_τ. Somit lässt sich aus der Theorie von *Truncation Selection* der Wertebereich von c, nämlich $c \in [0.8, 1.3]$, übernehmen.

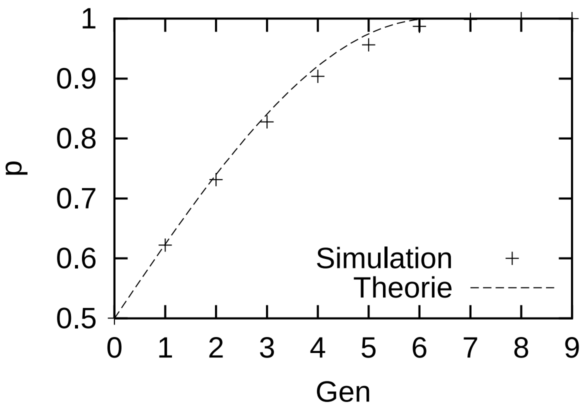

Abbildung 4.3: Univariate Randverteilungen für *OneMax* mit Boltzmann-Selektion und $n = 16$

In Abbildung 4.3 ist diese theoretische Lösung mit einem Simulationslauf verglichen. In der Simulation wurde mit einer Populationsgröße von 30000 Individuen, $c{=}1$, $n{=}16$ die Standardabweichung jeweils aus der Population geschätzt. Man sieht, dass der Simulationslauf etwas konservativer ist als die Theorie: nach (4.90) erreicht die Kurve bei $t = 2\pi$

die Achse $p=1$, in der Simulation nähert sie sich ihr noch bis $t=9$ an. Aus der zweiten und dritten Spalte aus Tabelle 4.1 kann man ablesen, dass am Ende die Lösungen der Differenzialgleichung und die der Differenzengleichung auseinanderlaugen. Dies beeinflusst natürlich auch die Schätzung von p.

Gen	β-Diff	β-DGL	$W_f(\beta)$	ΔW	$R_f(t-1)$
0	0.00	0.00	32.00	0.00	0.00
1	0.25	0.25	35.98	3.98	4.00
2	0.50	0.51	39.87	3.89	3.97
3	0.76	0.77	43.61	3.74	3.88
4	1.03	1.04	47.14	3.53	3.73
5	1.31	1.34	50.42	3.28	3.52
6	1.62	1.66	53.41	2.98	3.27
7	1.95	2.03	56.01	2.65	2.97
8	2.33	2.45	58.34	2.28	2.64
9	2.77	2.97	60.24	1.90	2.27
10	3.30	3.64	61.73	1.50	1.88
11	3.98	4.64	62.83	1.09	1.48
12	4.91	6.68	63.53	0.71	1.07
13	6.38	∞	63.89	0.36	0.68
14	9.43	∞	63.99	0.10	0.33
15	23.4	∞	64.00	0.01	0.07

Tabelle 4.1: Selektionserfolg und inverse Temperatur für *OneMax* (64), $c=1$. β-Diff ist durch (4.86) berechnet, β-DGL durch (4.89). ΔW ist die Differenz aufeinanderfolgender Zeilen von $W_f(\beta)$ und $R_f(t)$ die Schätzung nach (4.88).

Hier werden die Auswirkungen der verschiedenen Approximationen für 64 Bits und $c=1$ untersucht. In der zweiten und dritten Spalte sind die Werte für β nach der Differenzengleichung und der Differenzialgleichung verglichen. Man sieht deutlich, dass die Näherung am Anfang noch plausibel ist. Am Ende unterscheiden sich die Werte aber deutlich.

Die Faustregel nähert den *erwarteten* Zuwachs in der durchschnittlichen Fitness durch das erste Glied der Taylorreihe an. Die fünfte Spalte zeigt den tatsächlichen Zuwachs, also die Differenz zweier aufeinanderfolgender Werte der durchschnittlichen Fitness $W_f(\beta)$, wobei für β die Werte der Differenzengleichung benutzt wurden. In der letzten Spalte ist schließlich die Näherung nach (4.88) dargestellt.

In Abbildung 4.4 ist das durch diese Gleichungen entstandene Abkühlungsverfahren für 16 Bits dargestellt. Gezeigt sind sowohl Werte für die inverse Temperatur β als auch die Temperatur $1/\beta$. Die theoretischen Kurven nach (4.86) sind zusammen mit einem echten Lauf des *FDA* geplottet. Zum Vergleich ist auch die Lösung der Differenzialgleichung (4.90) dargestellt, man sieht wieder die Ungenauigkeit am Ende des Laufes.

Man sieht, dass am Anfang die inverse Temperatur β fast linear ansteigt, während sich am Ende die Temperatur linear der Null nähert. Nach Lemma 4.8 besteht bereits bei $\beta \geq 11.2$ die Hälfte der Population nur noch aus Optima. Man sieht, dass der Algorithmus terminiert, nach dem er diesen Wert erreicht hat.

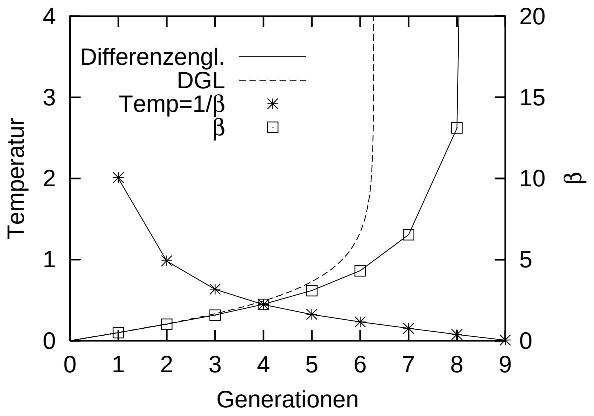

Abbildung 4.4: Werte von $\beta(t)$ für *OneMax*(16) in einem *FDA*-Lauf im Vergleich zu theoretischen Kurven nach (4.86) und (4.90).

4.4.2 Lineare Funktionen

Für lineare (binäre) Funktionen

$$Linear(x) = \sum_{i=1}^{n} \alpha_i x_i \tag{4.92}$$

zerfällt die Boltzmann-Verteilung nach dem Faktorisierungssatz in das Produkt der univariaten Randverteilungen, wie bei *OneMax*. Man kann leicht ausrechnen, dass dann

$$Z_f(\beta) = \prod_{i=1}^{n} (1 + e^{\beta\alpha_i}) \tag{4.93}$$

und

$$p_i(\beta) := p_\beta(X_i = 1) = \frac{e^{\beta\alpha_i}}{1 + e^{\beta\alpha_i}} \tag{4.94}$$

gilt. Ferner folgt aus der Tatsache, dass es eine einfache Produktverteilung ist,

$$\sigma_f^2(\beta) = \sum_{i=1}^{n} \frac{\alpha_i^2 e^{\beta\alpha_i}}{(1 + e^{\beta\alpha_i})^2} = \sum_{i=1}^{n} \alpha_i^2 p_i(\beta)\big(1 - p_i(\beta)\big) \tag{4.95}$$

und damit

$$\beta(t+1) = \beta(t) + \frac{c}{\sqrt{\sum_i \alpha_i^2 p_i(\beta)\big(1 - p_i(\beta)\big)}} \tag{4.96}$$

Durch Ableiten von (4.94) erhält man

$$\begin{aligned}\frac{dp_i(\beta)}{dt} &= \frac{\alpha_i e^{\beta\alpha_i}(1+e^{\beta\alpha_i})\beta' - e^{\beta\alpha_i}\alpha_i e^{\beta\alpha_i}\beta'}{(1+e^{\beta\alpha_i})^2} \\ &= p_i(\beta)\big(1-p_i(\beta)\big)\alpha_i \tfrac{d\beta}{dt}\end{aligned} \tag{4.97}$$

Daher ergibt sich die approximative Differenzialgleichung

$$\frac{dp_i(\beta)}{dt} = c \cdot \frac{p_i(\beta)\big(1-p_i(\beta)\big)\alpha_i}{\sqrt{\sum_i \alpha_i^2 p_i(\beta)\big(1-p_i(\beta)\big)}} \tag{4.98}$$

Die Lösungen dieser Differenzialgleichung bleiben invariant, wenn alle α_i mit einer Konstante multipliziert werden, wie vorhergesagt.

Für *OneMax* gilt $\alpha_i = 1$. Dann sind alle Randverteilungen gleich p_β. Es ergibt sich die Differenzialgleichung

$$\frac{dp_\beta}{dt} = c\sqrt{p_\beta(1-p_\beta)/n} \tag{4.99}$$

Die Lösung dieser Gleichung ist (4.91).

4.4.3 Die *Deceptive*-Funktion

Als einfaches Beispiel dafür, wieviel die Kenntnis der Struktur hilft, soll die Funktion *Deceptive* aus Abschnitt 3.7.3 (Seite 45) dienen. In Abbildung 4.5 ist der *UMDA*-Lauf einem Lauf des *FDA* gegenübergestellt.

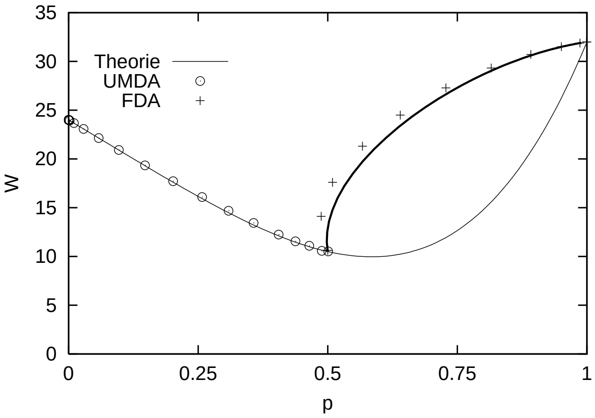

Abbildung 4.5: *FDA*-Lauf für *Deceptive*$_4$(32)

Es wurde jeweils Boltzmann-Selektion verwendet. Die theoretische Kurve für den *FDA* ergab sich einfach durch Bestimmen der univariaten Randverteilungen und durchschnittlichen Fitness der Boltzmann-Verteilung für verschiedene β-Werte.

Nach dem Fokussierungssatz für den *BEDA* sollte der Algorithmus auch dann noch zum globalen Optimum finden, wenn er weit weg davon startet, solange die Wahrscheinlichkeit des Optimums nie Null wird. In Abschnitt 8.2.5 wird gezeigt, wie ein nahe des lokalen Optimums gestarteter *FDA* dennoch zum globalen Optimum finden kann. Dabei kann der Prior eine entscheidende Rolle spielen.

In Abschnitt 6.2 werden weitere Läufe des *FDA* vorgestellt, auch mit nicht separablen Funktion, die die Stärken des Algorithmus weiter verdeutlichen. In Abschnitt 8.1 folgt weiterhin ein Vergleich von Boltzmann-Selektion und *Truncation Selection* anhand von 5 verschiedenen Funktionen und verschiedenen Bitlängen. Es wird sich zeigen, dass sich beide Selektionsverfahren auch in diesen Fällen ähnlich verhalten.

4.5 Zusammenfassung

Der *FDA* verwendet als Such-Verteilung die Boltzmann-Verteilung. Im Gegensatz zu *Simulated Annealing* wird dies nicht über den Gleichgewichtszustand einer Markovkette erreicht, sondern direkt. Dazu muss die Verteilung faktorisiert werden.

Für diese faktorisierten Verteilungen gibt es einen prinzipiellen Fokussierungssatz. Der *FDA* ist damit theoretisch in der Lage, Funktionen mit beliebiger Interaktion zu optimieren. Um keinen exponentiellen Aufwand zu haben, muss die Größe der Faktoren allerdings beschränkt bleiben. Auch dann gibt es leider noch keine theoretischen Resultate über die benötigten Populationsgrößen.

Für die durchschnittliche Fitness der Boltzmann-Verteilung existiert eine Reihenentwicklung in der inversen Temperatur. Diese Entwicklung erlaubte es mir, ein Abkühlungsschema für Boltzmann-Selektion zu entwickeln. Dieses Abkühlungsschema macht die Verwendung von Boltzmann-Selektion erst praktikabel, mit den oben genannten theoretischen Vorzügen. Denn in diesem Fall ist wenigstens eine Grenzverteilung bekannt, die bei genügend großer Populationsgröße gilt. Für andere Selektionsverfahren ist im Allgemeinen nicht bekannt, was die Grenzverteilung ist.

Mit diesem Schema ist das Verfahren invariant unter positiven linearen Transformationen der Fitness-Funktion.

Die Frage nach der Wahl der Faktorisierung ist noch nicht befriedigend gelöst. Ein neuer Ansatz, die Theorie der *graphischen Modelle*, wird daher im nächsten Kapitel vorgestellt.

5 Graphische Modelle

5.1 Einleitung

Die Frage, wie man mit Hilfe der Transformationen aus Abschnitt 4.2.4 aus einer additiven Zerlegung eine Faktorisierung gewinnen kann, ist noch nicht zufriedenstellend beantwortet. Insbesondere ist nicht geklärt, welche Abhängigkeiten man weglassen soll, wenn es mehr Teilfunktionen gibt als in einem Faktorisierungssystem erlaubt. Interessant wäre es ferner, eine Faktorisierung zu bestimmen, wenn die Zerlegung nicht von vorneherein bekannt ist.

Diese Fragen werden von der Theorie der graphischen Modelle beantwortet. Diese Theorie ist sehr beliebt, wenn es darum geht, unsicheres Wissen in Expertensystemen darzustellen (Howard and Matheson, 1981; Pearl, 1988).

Im Kern geht es dabei darum, Abhängigkeiten zwischen Variablen durch einen Graphen darzustellen. In unserer Anwendung liefert dieser Graph eine Faktorisierung, mit deren Hilfe dann neue Punkte erzeugt werden können.

Hier wird nicht auf die ganze Theorie in ihrer Breite eingegangen. Für diese Arbeit ist nur der Spezialfall der Bayes'schen Netze interessant. Dafür werden alle nötigen Beweise direkt geführt.

5.2 Abhängigkeiten von Variablen

Eine der Anwendungsmöglichkeiten von graphischen Modellen ist es, Wahrscheinlichkeiten mittels einer Faktorisierung möglichst effizient ausrechnen zu können. Dafür sollen noch zu definierende *Unabhängigkeiten* genutzt werden.

Definition 5.1. *Zwei Zufallsvariablen A, B heißen* unabhängig*, geschrieben $A \perp\!\!\!\perp B$, wenn*

$$p(A, B) = p(A)p(B) \tag{5.1}$$

Das ist eine symmetrische, nicht reflexive, nicht transitive Relation.

Zur Erinnerung: In dieser Arbeit kann man sich bei Zufallsvariablen immer die Koordinatenprojektionen aus dem Raum $\mathcal{R}$ vorstellen. Die Dichten sind dann einfach die entsprechenden Randverteilungen. Dadurch lässt sich die obige Definition einfach auf Mengen von Zufallsvariablen erweitern. Es gilt per Definitionem immer $A \perp\!\!\!\perp \emptyset$.

Lemma 5.2. *Zwei (Mengen von) Zufallsvariablen A und B sind genau dann unabhängig, wenn zwei Funktionen g und h existieren, so dass*

$$p(A,B) = g(A)h(B) \tag{5.2}$$

Außerdem gilt mit $p(B) > 0$

$$A \perp\!\!\!\perp B \implies p(A|B) = \frac{p(A,B)}{p(B)} = p(A) \tag{5.3}$$

Beweis: Aus der Unabhängigkeit folgt die Existenz durch $g(A) := p(A)$ und $h(B) := p(B)$, das heißt die Funktionen sind einfach die Randverteilungen. Falls $p(A,B) = g(A)h(B)$, so folgt durch Aufsummieren über alle möglichen Werte von B direkt $p(A) = g(A)$ und entsprechend $p(B) = h(B)$. (Lemma 5.2) ■

Diese Definition scheint ein aussichtsreicher Kandidat für ein Unabhängigkeitsmaß zu sein. Leider ist dies aber nicht der Fall, denn

$$A \perp\!\!\!\perp B, A \perp\!\!\!\perp C \not\Rightarrow A \perp\!\!\!\perp (B,C) \tag{5.4}$$

$$A \perp\!\!\!\perp B, A \perp\!\!\!\perp C, B \perp\!\!\!\perp C \not\Rightarrow p(a,b,c) = p(a)p(b)p(c) \tag{5.5}$$

Insbesondere die Implikation (5.5) ist aber wesentlich.

Beispiel 5.1. Die Zufallsvariablen A,B und C seien binär und haben folgende gemeinsame Dichte:

abc	000	100	010	110	001	101	011	111
p	$1/4$	0	0	$1/4$	$1/4$	0	0	$1/4$

Die Zufallsvariablen sind zwar paarweise unabhängig, es gilt aber $p(a,b,c) \neq p(a)p(b)p(c)$ sowie $p(a,b,c) \neq p(a)p(b,c)$.

Um solche Zerlegungen durchführen zu können, braucht man also ein anderes Maß von Unabhängigkeit:

Definition 5.3. *Die (Mengen von) Zufallsvariablen A und B heißen* konditional unabhängig gegeben C*, geschrieben $A \perp\!\!\!\perp B|C$, wenn gilt*

$$p(A,B|C) = p(A|C)p(B|C) \tag{5.6}$$

für $p(C) > 0$. Das ist eine in den ersten beiden Argumenten symmetrische Relation. Es gilt immer $A \perp\!\!\!\perp \emptyset|C$, ferner gilt $A \perp\!\!\!\perp B|\emptyset \Leftrightarrow A \perp\!\!\!\perp B$.

Lemma 5.4. *Es sind äquivalent (wenn alle Dichten größer Null sind):*

$$\begin{aligned} & A \perp\!\!\!\perp B|C \\ \Leftrightarrow\ & p(a|b,c) = p(a|c) \\ \Leftrightarrow\ & p(b|a,c) = p(b|c) \\ \Leftrightarrow\ & p(a,b,c) = p(a,c)p(b|c) = p(b,c)p(a|c) \\ \Leftrightarrow\ & \exists g,h : p(a,b,c) = g(a,c)h(b,c) \end{aligned} \tag{5.7}$$

Die meisten Implikationen sind schon in Lemma 4.14 bewiesen, der Rest folgt aus den Definitionen.

Anmerkung 5.5. Weder impliziert Unabhängigkeit die konditionale Unabhängigkeit noch umgekehrt. Es gilt aber:

Lemma 5.6. *(Lauritzen, 1996) Falls $A \perp\!\!\!\perp B|C$ und $A \perp\!\!\!\perp C|B$ und $p(B) > 0$, $p(C) > 0$, so folgt $A \perp\!\!\!\perp (B,C)$.*

Beweis: Wegen der Unabhängigkeiten existieren also Funktionen g, h, k, l, so dass

$$p(a,b,c) = g(a,c)h(b,c) = k(a,b)l(b,c) \tag{5.8}$$

$$\Longrightarrow \quad k(a,b) = \frac{g(a,c)h(b,c)}{l(b,c)} \tag{5.9}$$

$$= \frac{g(a,c_0)h(b,c_0)}{l(b,c_0)} \quad \text{für ein beliebiges, festes } c_0 \tag{5.10}$$

Also gilt

$$p(a,b,c) = \underbrace{g(a,c_0)}_{\tilde{g}(a)} \cdot \underbrace{\frac{h(b,c_0)l(b,c)}{l(b,c_0)}}_{\tilde{h}(b,c)} \tag{5.11}$$

und somit nach Lemma 5.2 $A \perp\!\!\!\perp (B,C)$. (Lemma 5.6) ■

Beispiel 5.2. Es sei $A \perp\!\!\!\perp B|C$, $B \perp\!\!\!\perp C|A$ und $A \perp\!\!\!\perp C|B$ und alle Randverteilungen positiv. Dann gilt nach Lemma 5.6, dass $A \perp\!\!\!\perp (B,C)$, also

$$p(a,b,c) = p(a)p(b,c) \qquad A \perp\!\!\!\perp (B,C) \tag{5.12}$$

$$\Longrightarrow \quad p(a,b) = p(a)p(b) \qquad \text{Summieren über } C \tag{5.13}$$

Weiter folgt

$$p(a,b,c) = p(a)p(b|a)p(c|a,b) \qquad \text{nach (5.23)} \tag{5.14}$$

$$= p(a)p(b|a)p(c) \qquad \text{Lemma 5.6}, C \perp\!\!\!\perp A|B, C \perp\!\!\!\perp B|A \tag{5.15}$$

$$= p(a)p(b)p(c) \qquad \text{nach (5.13)} \tag{5.16}$$

Man sieht an diesem Beispiel zwei Dinge: zum einen leistet die konditionale Unabhängigkeit in diesem Fall also, was man von ihr erwartet, zum anderen gestaltet sich die Rechnung ohne weitere Hilfsmittel ausgesprochen aufwändig. Die graphischen Modelle schaffen einen Rahmen, mit dem man den konditionalen Unabhängigkeiten direkt die Faktorisierung ablesen kann.

Lemma 5.7. *Es gilt:*

$$A \perp\!\!\!\perp (B,C)|D \implies A \perp\!\!\!\perp B|D \tag{5.17}$$

$$A \perp\!\!\!\perp (B,C)|D \implies A \perp\!\!\!\perp B|(C,D) \tag{5.18}$$

$$A \perp\!\!\!\perp (B,C) \implies A \perp\!\!\!\perp B \tag{5.19}$$

Beweis: Aus $A \perp\!\!\!\perp (B,C)|D$ folgt $p(A,B,C,D)=p(A|D)p(B,C,D)$. Summation über C liefert $p(A,B,D)=p(A|D)p(B,D)$ und somit $A \perp\!\!\!\perp B|D$. Für die zweite Aussage setze $g(A,C,D) = p(A|D)$ und $h(B,C,D) = p(B,C,D)$. Mit Lemma 5.4 ist der Beweis fertig. Die letzte Aussage ist nur ein Spezialfall. (Lemma 5.7) ■

5.3 Bayes'sche Netzwerke

Eine für diese Anwendung besonders geeignete Darstellung sind die Bayes'schen Netzwerke.

5.3.1 Einführung in Graphen

Zunächst eine paar Definitionen aus der Graphentheorie, siehe zum Beispiel (Lauritzen, 1996). Ein **gerichteter Graph** $\mathcal{G}$ besteht aus einer Menge V von **Knoten** *(Vertices)* (in der Anwendung sind dies Variablennamen), also $\{X_1, \ldots, X_n\}$, sowie einer Menge E von **Kanten** *(Edges)*, das sind (geordnete) Paare von Knoten. Eine Kante $(X_i, X_j) \in E$ stellt eine Kante von X_i zu X_j da und wird im folgenden $(X_i \to X_j)$ notiert. Die Variablen in der Menge $\Pi_i := \{X_j | (X_j \to X_i)\}$ heißen die **Eltern** von X_i, genauso gibt es die **Kinder** $\{X_j | (X_i \to X_j)\}$ von X_i.

Ein **Pfad** von X_a nach X_b ist eine Liste von Knoten $\{X_{i_j} | j \in \{1, \ldots, l\}\}$, so dass gilt: $X_{i_1} = X_a$, $X_{i_l} = X_b$ und für alle $1 \leq j < l$ gilt: $(X_{i_j} \to X_{i_{j+1}})$. Die **Länge** des Pfades ist dabei l. Ein **Zyklus** ist ein Pfad von X_i zu sich selber. Ein gerichteter azyklischer Graph (GaG) ist ein gerichteter Graph, der keine Zyklen enthält. Insbesondere kann in einem GaG niemals $(X_i \to X_j)$ und $(X_j \to X_i)$ gelten, da die beiden sonst einen Zyklus bilden würden.

Die Eltern einer Variablen X_i sind also genau die Knoten, die X_i durch einen Pfad der Länge 1 erreichen. Definiere die **Vorfahren** von X_i als die Menge der Knoten, die X_i durch einen Pfad (beliebiger Länge) erreichen, sowie die **Nachkommen** von X_i als die Knoten, die von X_i durch einen Pfad erreicht werden können. Besonderes Interesse gilt der Menge der **Nicht-Nachkommen** nN_i. In einem GaG sind die Mengen der Vorfahren und der Nachkommen disjunkt, und es gilt $\Pi_i \subseteq nN_i$.

In Abbildung 5.1 ist ein Beispiel für einen ayzklischen Graphen abgebildet. Es gilt $\Pi_3 = \{X_1, X_2\}$, $nN_3 = \{X_1, X_2, X_7\}$.

Ein **ungerichteter Graph** kann als ein Spezialfall eines gerichteten Graphen aufgefasst werden, für den gilt: $(X_i \to X_j) \implies (X_j \to X_i)$. Die beiden Kanten werden dann zu einer ungerichteten Kante zusammengefasst. Aus einem gerichteten Graphen kann trivialerweise ein ungerichteter erzeugt werden. Ein Zyklus habe eine **Sehne** *(Chord)*, wenn es in dem Zyklus $\{X_{i_j} | j \in \{1, \ldots, l\}\}$ zwei Knoten X_{i_c} und X_{i_d} gibt, die nicht aufeinanderfolgen (also $|c - d| > 1$), aber trotzdem gilt: $(X_{i_c} \to X_{i_d})$. Ein ungerichteter Graph heißt **trianguliert**, wenn alle Zyklen der Länge vier oder mehr eine Sehne besitzen.

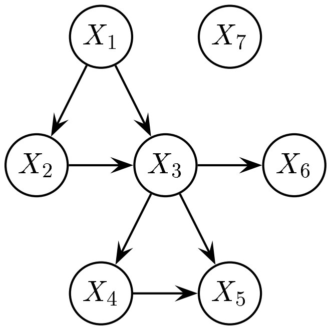

Abbildung 5.1: Beispiel für den Graphen eines Bayes'schen Netzwerkes

Eine Teilmenge C der Knoten **trennt** zwei Knoten X_i und X_j, wenn jeder Pfad von X_i nach X_j durch mindestens einen Knoten aus C führen muss. Weiterhin trennt C die Teilmengen A und B, wenn C alle Paare von Knoten X_i und X_j trennt, wobei $X_i \in A$ und $X_j \in B$ gilt.

Eine **Clique** in einem Graphen ist eine maximale Menge von Knoten, in der je zwei Elemente durch eine Kante verbunden sind.

Ein **Baum** ist eine ungerichteter azyklischer Graph. Ein einfaches Argument zeigt, dass ein Baum höchstens $n-1$ Kanten haben kann (manchmal fordert man für einen Baum, dass er genau $n-1$ Kanten hat, und nennt „meine" Bäume Wälder). Ein **Poly-Baum** *(Polytree)* ist ein gerichteter Graph, dessen ungerichtete Version ein Baum ist.

5.3.2 Definition der Bayes'schen Netzwerke

Bayes'sche Netzwerke (Pearl, 1988; Lauritzen, 1996; Jordan, 1999) sind ein Werkzeug der Wahrscheinlichkeitstheorie, um Wissen über (Un-)abhängigkeiten von Variablen in einer Wahrscheinlichkeitsverteilung p zu repräsentieren.

Definition 5.8. *Ein* ***Bayes'sches Netzwerk*** *ist ein Paar $M = (\mathcal{G}, \Theta)$. $\mathcal{G}$ ist ein gerichteter, azyklischer Graph und repräsentiert konditionale (Un-)abhängigkeiten zwischen den Variablen. $\Theta = \theta^{ijk}$ sind die konditionalen Wahrscheinlichkeiten, die nötig sind, um die Verteilung p mit Hilfe des Graphen zu beschreiben, wobei θ^{ijk} die Wahrscheinlichkeit ausdrückt, dass X_i den Zustand k hat und die Eltern den Zustand j haben.*

Der Graph muss azyklisch sein, damit er ein gültiges Modell darstellt. Aus dem Graph aus Abbildung 5.1 ergibt sich die Faktorisierung

$$p_M(X) = p(X_1)p(X_2|X_1)p(X_3|X_1, X_2)p(X_4|X_3)p(X_5|X_3, X_4)p(X_6|X_3)p(X_7) \quad (5.20)$$

Die θ^{ijk} beschreiben dann die Parameter der einzelnen Wahrscheinlichkeitstabellen. So sind für den Faktor $p(X_3|X_1, X_2)$ bei binärem Wertebereich die 8 Parameter θ^{3jk} mit

$k \in \{0,1\}$ und $j \in \{0,1,2,3\}$ nötig. Beachte, dass die Zustände der Eltern einfach durchnummeriert werden.

Weil r_i die Größe des Wertebereichs $\{0,\dots,l_i\}$ von X_i ist, definiere q_i als Produkt der r_k, wobei k die Elternmenge Π_i durchläuft (bei leerer Elternmenge ist $q_i := 1$). Dann läuft j von 0 bis $q_i - 1$. Es gilt $\sum_{k=0}^{l_i} \theta^{ijk} = 1$ für alle $1 \le i \le n$ und $0 \le j < q_i$.

Der Graph induziert also eine Faktorisierung der Verteilung. Beim Lernen des Modells unterscheidet man dabei zwischen der „wahren" Verteilung $\hat{p}(x)$ und der Verteilung $p_M(x)$, die zu dem Modell assoziiert ist. Wenn es der Kontext erlaubt, entfallen die Verzierungen.

Man kann den Graphen auch so interpretieren, dass er Kausalitäten ausdrückt. Der Graph in Abbildung 5.1 beschreibt die Annahme, dass Variable X_2 vom Wert von X_1 abhängt, während Variable X_3 sowohl von X_1 als auch von X_2 in noch zu spezifizierender Form abhängig ist. Variable X_7 ist in diesem Modell völlig unabhängig von allen anderen Variablen. Aus dem Graphen lassen sich Aussagen über konditionale Unabhängigkeiten der Variablen untereinander ableiten.

Die Darstellung dieser Unabhängigkeitsbeziehungen in einem Graphen ermöglicht es, fortschrittliche Algorithmen der Graphentheorie in diesem Kontext zu verwenden. Diese Tatsache hat zur Popularität der graphischen Modelle in den letzten Jahren beigetragen.

5.3.3 Gerichtete azyklische Graphen und konditionale Unabhängigkeit

Ein GaG $\mathcal{G}$ beschreibt bestimmte konditionale Unabhängigkeiten:

Definition 5.9. *Ein GaG $\mathcal{G}$ über Variablen X_1 bis X_n heißt **kompatibel** zu der Verteilung $p(X_1,\dots,X_n)$, wenn für jeden Knoten X_i gilt:*

$$X_i \perp\!\!\!\perp (nN_i \setminus \Pi_i) \mid \Pi_i \tag{5.21}$$

beziehungsweise

$$X_i \perp\!\!\!\perp nN_i \qquad \textit{falls } \Pi_i = \emptyset \tag{5.22}$$

Als Beispiel betrachte Abbildung 5.2.

Man kann aus den konditionalen Unabhängigkeiten auf verschiedene Arten eine Faktorisierung ablesen. Eine Möglichkeit fängt mit der trivialen Faktorisierung an:

Satz 5.10 (Triviale Faktorisierung). *Mit positiven Randverteilungen gilt immer*

$$p(X_1,\dots,X_n) = p(X_1)p(X_2|X_1)p(X_3|X_1,X_2)\cdots p(X_n|X_1,\dots,X_{n-1}), \tag{5.23}$$

Dies gilt, weil sich die Nenner wegkürzen. Natürlich gilt das Lemma auch für jede Permutation der Variablen. Mit Lemma 5.4 kann man dann die konditionalen Wahrscheinlichkeiten vereinfachen. Betrachte Abbildung 5.2. Mit Lemma 5.7 folgt aus $X_3 \perp\!\!\!\perp X_2, X_4|X_1$ auch $X_3 \perp\!\!\!\perp X_2|X_1$. Damit gilt

$$\begin{aligned} p(X) &= p(X_1)p(X_2|X_1)p(X_3|X_1,X_2)p(X_4|X_1,X_2,X_3) && (5.24)\\ &= p(X_1)p(X_2|X_1)p(X_3|X_2)p(X_4) && (5.25) \end{aligned}$$

Diese Vorgehensweise ist allerdings abhängig von der Reihenfolge der Variablen in der trivialen Faktorisierung. Daher wird hier anders vorgegangen.

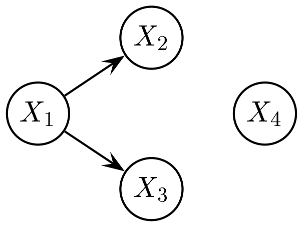

Abbildung 5.2: Ein Beispiel zur Faktorisierung. Die durch den Graphen kodierten Unabhängigkeitsbedingungen sind nach Definition 5.9 $X_1 \perp\!\!\!\perp X_4$, weiterhin $X_2 \perp\!\!\!\perp (X_3, X_4)|X_1$, $X_3 \perp\!\!\!\perp (X_2, X_4)|X_1$ sowie $X_4 \perp\!\!\!\perp (X_1, X_2, X_3)$.

Satz 5.11 (Faktorisierungsatz für Bayes'sche Netze). *Der GaG $\mathcal{G}$ sei kompatibel mit $p(X_1, \ldots, X_n)$. Dann gilt:*

$$p(X) = \prod_{i=1}^{n} p(X_i|\Pi_i) \tag{5.26}$$

Dieser Satz folgt aus den Sätzen, die in den Lehrbüchern zu graphischen Modellen gezeigt werden. Da ein eigenständiger Beweis aber nicht sehr lang ist, wird er im Folgenden ausgeführt.

Beweis: Betrachte zur besseren Anschaulichkeit das Beispiel aus Bild 5.2.

Der Beweis ist konstruktiv. Wähle zunächst einen Knoten ohne Nachkommen, ohne Einschränkung X_1 (die Existenz ist garantiert durch die Tatsache, dass $\mathcal{G}$ nicht zyklisch ist). Nach Voraussetzung gilt $X_1 \perp\!\!\!\perp (nN_1 \setminus \Pi_1) \mid \Pi_1$. Weil X_1 keine Nachkommen hat, gilt $nN_1 = (X_2, \ldots, X_n)$. Daher folgt

$$p(X) = p(X_1, \Pi_1, (nN_1 \setminus \Pi_1)) = p(X_1|\Pi_1)p(\Pi_1, (nN_1 \setminus \Pi_1)) \tag{5.27}$$

Im dem Beispiel kann als Startknoten X_2, X_3 oder X_4 gewählt werden, also zum Beispiel X_2. Mit $X_2 \perp\!\!\!\perp (X_3, X_4)|X_1$ folgt $p(X) = p(X_2|X_1)p(X_1, X_3, X_4)$. Im folgenden sei $\mathcal{R}_i$ die bereits bearbeiteten Menge an Variablen, also $\mathcal{R}_1 = \emptyset$ und ohne Einschränkung $\mathcal{R}_2 = \{X_1\}$. Es gelte in jedem Schritt:

1. Für alle $X_j \in X \setminus \mathcal{R}_i$ gilt $\Pi_j \cap \mathcal{R}_i = \emptyset$.
2. $p(X) = \left(\prod_{X_j \in \mathcal{R}_i} p(X_j|\Pi_j)\right) p(\{X \setminus \mathcal{R}_i\})$

Per Konstruktion ist dies für $\mathcal{R}_1$ und $\mathcal{R}_2$ erfüllt.

Betrachte nun Schritt i. Wähle eine Variable X_j aus $X \setminus \mathcal{R}_i$, so dass gilt: alle Nachkommen von X_j (wenn vorhanden) liegen in $\mathcal{R}_i$. Weil $\mathcal{G}$ azyklisch ist, gibt es immer solch eine

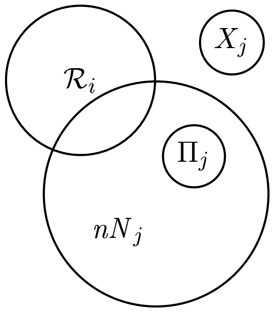

Abbildung 5.3: Zerlegung der Grundmenge X im Beweis zu Satz 5.11

Variable. Bild 5.3 veranschaulicht die Mengen, wobei X die Vereinigung der abgebildeten Mengen ist. Es erfüllt $\mathcal{R}_{i+1} = \mathcal{R}_i \cup \{X_j\}$ Punkt 1 obiger Bedingungen.

Wieder in dem Beispiel sind X_3 und X_4 mögliche Kandidaten für den nächsten Schritt. Wähle also X_3. Da $X_3 \perp\!\!\!\perp (X_2, X_4)|X_1$ gilt mit Lemma 5.7 auch $X_3 \perp\!\!\!\perp X_4|X_1$. Somit lässt sich der rechte Faktor $p(X_1, X_3, X_4)$ weiter zerlegen in $p(X_3|X_1)p(X_1, X_4)$. Aus $X_4 \perp\!\!\!\perp (X_1, X_2, X_3)$ folgt mit dem gleichen Lemma $X_4 \perp\!\!\!\perp X_1$ und somit zerfällt $p(X_1, X_4)$ in $p(X_1)p(X_4)$. Zusammen ergibt sich also $p(X_1)p(X_2|X_1)p(X_3|X_1)p(X_4)$ wie behauptet.

Weiter im Beweis gilt nach Voraussetzung

$$X_j \perp\!\!\!\perp (nN_j \setminus \Pi_j) \mid \Pi_j \tag{5.28}$$

Nach Konstruktion gilt $\mathcal{R}_i \cap \Pi_j = \emptyset$, also folgt auch mit Lemma 5.7

$$X_j \perp\!\!\!\perp (nN_j \setminus \Pi_j) \setminus \mathcal{R}_i \mid \Pi_j \tag{5.29}$$

und daher

$$\begin{aligned} p(\{X \setminus \mathcal{R}_i\}) &= p(X_j, (nN_j \setminus \Pi_j) \setminus \mathcal{R}_i, \Pi_j) && (5.30)\\ &= p(X_j|\Pi_j)p(nN_j \setminus \mathcal{R}_i) && (5.31) \end{aligned}$$

und damit

$$p(X) = \left(\prod_{X_j \in \mathcal{R}_{i+1}} p(X_j|\Pi_j) \right) p(\{X \setminus \mathcal{R}_i\}) \tag{5.32}$$

(Satz 5.11) ■

5.3.4 Einfacher Faktorisierungsalgorithmus

Ein einfacher Algorithmus, der aus dem Graphen in $O(n^2)$ eine Faktorisierung liefert, ist Algorithmus 5.1. Dieses Problem ist identisch zu der sogenannten *topologischen Sortierung*. Die hier gefundene Lösung ist zwar nicht eindeutig, aber jede so gefundene Faktorisierung erzeugt die *gleiche* Verteilung.

Algorithmus 5.1: Faktorisierungsalgorithmus

Markiere alle Knoten als nicht bearbeitet.
do {
  Suche einen unmarkierten Knoten X_i, der keine Eltern hat oder dessen Eltern alle schon markiert sind.
  Keine Eltern: füge den Term $p(X_i)$ der Faktorisierung hinzu, sonst den Term $p(X_i|\Pi_i)$.
  Markiere den Knoten X_i als gelesen.
} **while** (noch unmarkierte Knoten übrig sind)

Jede Wahrscheinlichkeitsverteilung kann durch eine Bayes'sches Netzwerk beschrieben werden. Die Zuordnungen sind allerdings nicht eindeutig, wie das Beispiel in Abbildung 5.4 verdeutlicht:

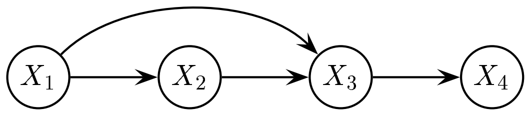

Abbildung 5.4: Dieser Graph ist zu den Verteilungen (5.33), (5.34), (5.35) und (5.36) kompatibel

Aus dem Graphen kann man die folgende Faktorisierung ablesen:

$$p(X) = p(X_1)p(X_2|X_1)p(X_3|X_1, X_2)p(X_4|X_3) \tag{5.33}$$

Aber: die folgenden Faktorisierungen haben die *gleiche* Wahrscheinlichkeitsverteilung:

$$\begin{aligned} p(X) &= p(X_3)p(X_4|X_3)p(X_2|X_3)p(X_1|X_2, X_3) && (5.34)\\ &= p(X_4)p(X_3|X_4)p(X_2|X_3)p(X_1|X_2, X_3) && (5.35) \end{aligned}$$

Es folgt, dass *verschiedene Netzwerke die gleiche zugrunde liegende Wahrscheinlichkeitsstruktur kodieren können*. Problematisch ist das, wenn man die Pfeile als Kausalitäten interpretieren will. Es lassen sich nämlich alle Pfeile in Abbildung 5.4 umdrehen, ohne die Faktorisierung zu ändern. Kausalität drückt sich also in der Verteilung nicht aus.

Weiterhin gilt natürlich auch die triviale Faktorisierung

$$p(X) = p(X_1)p(X_2|X_1)p(X_3|X_1, X_2)p(X_4|X_1, X_2, X_3) \tag{5.36}$$

Diese kanonische Faktorisierung kann ja, wie oben erläutert, mit Hilfe der Unabhängigkeiten, die man aus dem Graphen ablesen kann, vereinfacht werden zu einer der obigen Formen. Die triviale Faktorisierung ist aber weiterhin gültig, d.h., die Verteilung kann so dargestellt werden. Sie hat nur unnötig viele Parameter. In der Sprache der graphischen Modelle heißt das, dass man in einem Graphen Kanten hinzufügen kann (also Abhängigkeiten hinzufügen kann), ohne zu einer falschen Verteilung zu kommen:

Lemma 5.12. *Der GaG $\mathcal{G}$ sei kompatibel zu der Verteilung $p(X)$. Wenn sich die Kante $(X_j \to X_i)$ zu $\mathcal{G}$ hinzufügen lässt, so dass $\mathcal{G}^* = \mathcal{G} \cup (X_j \to X_i)$ azyklisch ist, so ist $\mathcal{G}^*$ auch kompatibel zu $p(X)$.*

Beweis: Durch das Hinzufügen einer Kante $(X_j \to X_i)$ wird die Elternmenge Π_k nur bei $k = i$ geändert. Es gilt hier nach Voraussetzung $X_i \perp\!\!\!\perp (nN_i \setminus \Pi_i) \mid \Pi_i$ und $X_j \in nN_i$, $X_j \notin \Pi_i$ aber $X_j \in \Pi_i^* \subseteq nN_i^*$. Also soll aus $X_i \perp\!\!\!\perp (A, X_j) \mid \Pi_i$ folgen $X_i \perp\!\!\!\perp A \mid (\Pi_i, X_j)$. Dies gilt aber wegen (5.18).

Die Mengen nN_k können durch Hinzufügen einer Kante nur *kleiner* werden, wobei die Elternmengen Π_k gleich bleiben müssen. Sei also k so, dass $nN_k \subset nN_k^*$. Nach Voraussetzung gilt $X_k \perp\!\!\!\perp (nN_k^* \setminus \Pi_k, A) | \Pi_k$. Daher folgt mit der Gleichung (5.17) auch die Beziehung $X_k \perp\!\!\!\perp (nN_k^* \setminus \Pi_k) \mid \Pi_k$. (Lemma 5.12) ■

Zusammenfassend lässt sich sagen, dass eine gegebene Verteilung $p(X)$ auf verschiedene Weise in einem Graphen dargestellt werden kann. Jeder Graph entspricht eindeutig einer Faktorisierung, es gibt allerdings mehrere Graphen, die die gleiche Verteilung beschreiben können. Es gibt aber eine Menge von *minimalen* Graphen, die alle konditionalen Unabhängigkeiten in p ausdrücken. Minimal heißt in diesem Zusammenhang, dass die dazugehörende Verteilung die gleiche Zahl an Parametern haben. So hat die Verteilung aus (5.33) $1+2+4+2=9$ Parameter, die Verteilungen aus (5.34) und (5.35) ebenfalls, während die Verteilung aus (5.36) 15 Parameter hat.

5.4 Zusammenhang mit der Boltzmann-Verteilung

In Abschnitt 4.2.1 tauchte auch schon die Faktorisierung der Wahrscheinlichkeitsverteilung auf, dort allerdings ausgehend von einer additiven Zerlegung der Funktion, die die Boltzmann-Verteilung bestimmte:

$$p_{\beta,f}(x) = \frac{e^{\beta f(x)}}{Z_\beta} \tag{5.37}$$

Es gilt

Lemma 5.13. *Es sei $f(x)$ eine additiv zerlegte Funktion (Abschnitt A.1.3) und es seien X_a, X_b zwei Variablen, die in keiner Teilfunktion gemeinsam vorkommen:*

$$\nexists k : \{a, b\} \subseteq s_k \tag{5.38}$$

Dann gilt mit $r := I \setminus \{a, b\}$ für die Boltzmann-Verteilung $p_{\beta,f}(x)$:

$$p_{\beta,f}(x) = p_{\beta,f}(X_a, X_r) \cdot p_{\beta,f}(X_b | X_r) \qquad \textit{bzw. } X_a \perp\!\!\!\perp X_b \mid X_r \tag{5.39}$$

Beweis: Aufgrund der Voraussetzungen gilt $f(x) = f_1(X_a, X_r) + f_2(X_b, X_r)$. Die Boltzmann-Verteilung wandelt diese Summe in ein Produkt um, und mit Lemma 5.4 folgt die Behauptung. (Lemma 5.13) ■

Wegen der konditionalen Unabhängigkeit ergibt sich das wichtige

Korollar 5.14. *Eine additive Zerlegung einer Funktion $f(x)$ definiert eine Obermenge für die minimalen zu $p_{\beta,f}$ kompatiblen Bayes'schen Netze in dem Sinne, dass Variablen, die in keiner Teilfunktion gemeinsam auftreten, in keinem solchen Netz durch eine Kante verbunden sind.*

Die beiden Faktorisierungen haben auf den ersten Blick einen großen Unterschied: Während bei der Zerlegung nach Abschnitt 4.2.4 mehrere Variablen auf der linken Seite der Konditionierung stehen können, gibt es hier immer nur eine. Mit Hilfe der trivialen Faktorisierung (5.23) lässt sich aber jede Faktorisierung in eine überführen, die nur eine Variable auf der linken Seite hat, zum Beispiel:

$$p(X_1, X_2, X_3|X_4) = p(X_1|X_4)p(X_2|X_3, X_4)p(X_3|X_1, X_2, X_4) \tag{5.40}$$

Die Beispiel-Faktorisierungen aus Kapitel 4 lassen sich alle auch mit Hilfe der Bayes'schen Netze gewinnen.

5.5 Graphische Modelle mit ungerichteten Graphen

Neben den Bayes'schen Netzen gibt es noch andere Möglichkeiten, Verteilungen und insbesondere Unabhängigkeiten in den Verteilungen durch Graphen darzustellen. Neben Erweiterungen, die verdeckte Variablen, kontinuierliche Verteilungen und gemischte diskrete / kontinuierliche Verteilungen berücksichtigen, ist die Verwendung von ungerichteten Graphen eine weitere beliebte Alternative.

Ohne auf die Beweise einzugehen, seien im Folgenden die wichtigsten Eigenschaften dieser Modellierung genannt. Konditionale Unabhängigkeiten werden hier durch Trennungseigenschaften des Graphen modelliert: Es seien a, b und c Teilmengen der Indexmenge $\{1, \dots, n\}$. Genau dann, wenn X_a und X_b von X_c getrennt werden, muss auch $X_a \perp\!\!\!\perp X_b|X_c$ gelten. Eine Reihe von Sätzen zu diesen sogenannten Markov-Eigenschaften stellen sicher, dass diese Eigenschaft (die globale Markov-Eigenschaft) für positive Dichten äquivalent ist zu der lokalen Markov-Eigenschaft: Wenn X_i und X_j nicht durch eine Kante verbunden sind, gilt $X_i \perp\!\!\!\perp X_j|X \setminus \{X_i, X_j\}$ (Pearl, 1988; Lauritzen, 1996; Whittaker, 1990).

Die Faktorisierung wird durch das Verfahren in Algorithmus 5.2 gewonnen (Spiegelhalter and Lauritzen, 1990; Jensen and Jensen, 1994). Die Suche nach der minimalen Triangulierung ist NP-vollständig. Eine minimale Triangulierung ist zwar nicht notwendig, aber je größer die Cliquen, desto aufwändiger sind die folgenden Schritte.

Die Zeit- und Platzkomplexität für die Berechnung der Verteilung nach obiger Formel ist exponentiell in der Cliquengröße, daher ist das Verfahren in der Praxis nur eingeschränkt verwendbar.

Beispiel 5.3. Die ungerichtete Version des Graphen aus Bild 5.4 ist bereits trianguliert. Die Cliquen sind $\{X_1, X_2, X_3\}$ sowie $\{X_3, X_4\}$. Damit ist $\{X_3\}$ Separator und die Faktorisierung lautet $p(X) = p(X_1, X_2, X_3)p(X_3, X_4)/p(X_3)$ oder auch $p(X) = p(X_1, X_2, X_3)p(X_4|X_3)$ wie im Falle des Bayes'schen Netzwerkes.

Algorithmus 5.2: Verzweigungsbaum-Faktorisierung

1. Trianguliere den Graphen, das heißt, füge solange Kanten hinzu, bis er trianguliert ist.
2. Identifiziere die Cliquen C_i in dem Graphen.
3. Erzeuge den Verzweigungsgraph: Knoten sind die Cliquen C_i, Kanten werden zwischen Knoten mit nicht leerem Schnitt gesetzt. Die Kanten werden mit der Größe der Schnittmenge bezeichnet.
4. Erzeuge einen maximalen Überdeckungsbaum *(Junction Tree)* aus dem Graphen: Bestimme den Teilbaum, der maximales Gewicht bezogen auf die Summe der Verzierungen hat (polynomieller Algorithmus).
5. Die „Separatoren" S_i sind die Schnittmengen von Cliquen, die in dem Verzweigungsbaum benachbart sind.
6. Die Faktorisierung ist dann

$$p(X) = \frac{\prod_{C_i} p(X_{C_i})}{\prod_{S_i} p(X_{S_i})}. \tag{5.41}$$

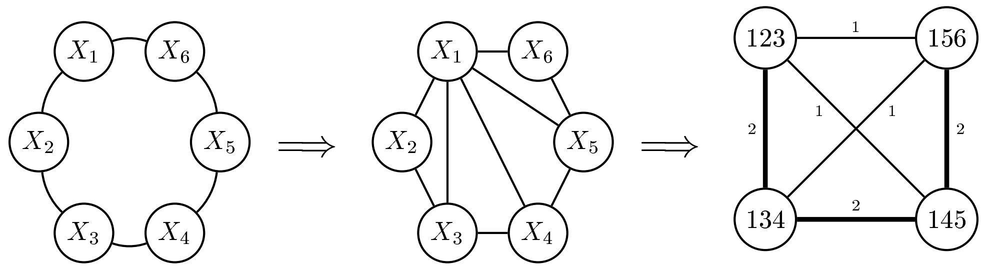

Abbildung 5.5: Triangulierung eines Zyklus nach Algorithmus 5.2, Beispiel 5.4

Beispiel 5.4 (Zyklus). Abbildung 5.5 zeigt einen Zyklus und eine mögliche Triangulierung sowie den zugehörigen Verzweigungsgraph. Die Kanten sind mit den Größen der Schnittmengen bezeichnet. Die fettgedruckten Kanten sind ausgewählt, um den Verzweigungsbaum zu bilden. Es ergibt sich als Faktorisierung

$$\begin{aligned} p(X) &= \frac{p(X_1, X_2, X_3)p(X_3, X_4, X_1)p(X_4, X_5, X_1)p(X_5, X_6, X_1)}{p(X_1, X_3)p(X_4, X_1)p(X_5, X_1)} && (5.42) \\ &= p(X_1, X_2, X_3)p(X_4|X_1, X_3)p(X_5|X_1, X_4)p(X_6|X_1, X_5) && (5.43) \end{aligned}$$

Beispiel 5.5 (Gitter). Das Gitter aus Abschnitt 4.2.6 zeigt, wieso die Triangulierung so schwierig ist. Abbildung 5.6 sieht auf den ersten Blick wie eine gültige Triangulierung aus. Die nicht gestrichelten Linien bilden aber einen Zyklus, der keine Sehne hat. Daher führt das Verfahren auch hier zu exponentiellem Aufwand.

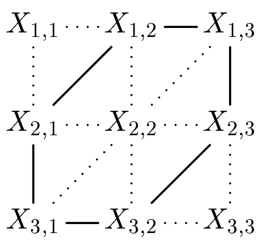

Abbildung 5.6: Wie man das Gitter *nicht* triangulieren kann, Illustration zu Beispiel 5.5

5.6 Lernen von Bayes'schen Netzwerken

Die Ermittlung des Bayes'schen Netzwerkes aus gegebenen Daten nennt man lernen. Es werden dabei folgende Fragen beantwortet:

- Gegeben eine Menge von Datenpunkten, welcher Graph beschreibt die konditionalen (Un-)abhängigkeiten der Daten?
- Wie sind die Werte der Parameter Θ zu berechnen?

Gesucht ist zunächst eine Maß, um die Qualität von Netzwerken bezogen auf einen festen Datensatz zu beschreiben. Je nach Ansatz ergeben sich dabei auch unterschiedliche Methoden, um die Parameter zu berechnen.

In diesem Abschnitt werden einige Definitionen aus der Informationstheorie benötigt, die ich in Anhang A.3 zusammengestellt habe.

5.6.1 Die Log-Likelihood

Ein einfacher Ansatz für ein Gütemaß bestimmt einfach die *Log-Likelihood* des Netzwerkes bei gegebenen Daten. Der Graph $\mathcal{G}$ des Bayes'schen Netzwerkes $M = (\mathcal{G}, \Theta)$ bestimmt eine Faktorisierung von $\hat{p}(x)$ gemäß dem Faktorisierungssatz auf Seite 93 und damit eine neue Verteilung $p_M(x)$.

Definition 5.15. *Die **Log-Likelihood** der Verteilung $p_M(x)$ bezogen auf die „echte" Verteilung $p(x)$ ist definiert durch*

$$\mathcal{L}(M|p) = \sum_x p(x) \log p_M(x) \tag{5.44}$$

Sie ist negativ und nach Lemma A.8 genau dann maximal, wenn $p(x) = p_M(x)$ ist. Ferner gilt $\mathcal{L}(M|p) = H_p(X) - D^{KL}(p \| p_M)$.

Es gilt:

Lemma 5.16. *Die Log-Likelihood berechnet sich durch*

$$\mathcal{L}(M,p) = -\sum_{i=1}^{n} H(X_i|\Pi_i) \tag{5.45}$$

Beweis: Es sei für jedes i die Menge der Variablen $\bar{X}$ disjunkt zerlegt in X_i, Π_i und Q_i.

$$\sum_{i=1}^{n} H(X_i|\Pi_i) = -\sum_{i=1}^{n} \sum_{x_i,\pi_i} p(x_i,\pi_i) \log p(x_i|\pi_i) \tag{5.46}$$

$$= -\sum_{i=1}^{n} \sum_{x_i,\pi_i,q_i} p(x) \log p(x_i|\pi_i) = -\sum_{x} p(x) \log p_M(x) \tag{5.47}$$

(Lemma 5.16) ■

Wie sich in Lemma 5.12 zeigte, lassen sich zu dem Graphen eines Bayes'schen Netzes Kanten hinzufügen (solange er azyklisch bleibt), ohne die Kompatibilität zu verletzen. Im Extremfall lässt sich jede Verteilung natürlich durch einen vollständigen Graphen beschreiben.

Die Wahl der Log-Likelihood als Güte-Kriterium hat dieselbe unangenehme Eigenschaft der Überanpassung: Die maximale Struktur hat den größten (also besten) Wert. Die Log-Likelihood beschreibt nämlich, wie gut das Modell die konditionalen Unabhängigkeiten erfasst. Der folgende Satz macht klar, dass sie als Metrik bei Verwendung von gesampelten Daten immer die maximale Struktur bevorzugen wird, weil in der Praxis die wechselseite Information nie genau Null wird.

Satz 5.17. *Sei $\mathcal{G}$ ein GaG und $\mathcal{G}^*$ der um die Kante $(X_j \to X_i)$ erweiterte Graph. Definiere die negative Log-Likelihood*

$$H(\mathcal{G}|D) = \sum_{i=1}^{n} H(X_i|\Pi_i), \tag{5.48}$$

wobei die konditionalen Entropien aus der aus den Daten geschätzten Verteilung stammen. Dann gilt

$$H(\mathcal{G}|D) \geq H(\mathcal{G}^*|D) \tag{5.49}$$

Für die Differenz gilt:

$$H(\mathcal{G}|D) - H(\mathcal{G}^*|D) = \begin{cases} \mathrm{Inf}(X_i;X_j) & \textit{falls } \Pi_i = \emptyset \\ \mathrm{Inf}(X_i;X_j|\Pi_i) & \textit{sonst} \end{cases} \tag{5.50}$$

Weiterhin folgt:

$$H(\mathcal{G}|D) - H(\mathcal{G}^*|D) = 0 \;\Leftrightarrow\; \begin{cases} X_i \perp\!\!\!\perp X_j & \textit{falls } \Pi_i = \emptyset \\ X_i \perp\!\!\!\perp X_j|\Pi_i & \textit{sonst} \end{cases} \tag{5.51}$$

Beweis: Da $H(X|Y) = H(X) - \mathrm{Inf}(X;Y)$, folgt für $\Pi_i \neq \emptyset$:

$$\begin{aligned} H(\mathcal{G}|D) - H(\mathcal{G}^*|D) &= H(X_i|\Pi_i) - H(X_i|\Pi_i, X_j) && (5.52)\\ &= -\mathrm{Inf}(X_i;\Pi_i) + \mathrm{Inf}(X_i;\Pi_i, X_j) && (5.53)\\ &= \mathrm{Inf}(X_i;X_j|\Pi_i) && (5.54)\end{aligned}$$

Aus Lemma A.11 folgen damit die Behauptungen. Für $\Pi_i = \emptyset$ kann man den Beweis analog führen. (Satz 5.17) ∎

Benötigt wird also ein Maß, welches einfachere Strukturen bevorzugt. Eine geeignete Wahl ist das BIC-Maß. Es lässt sich auf verschiedene Art und Weise herleiten. Eine Möglichkeit ist inspiriert durch das Prinzip der minimalen Beschreibungslänge (MDL, *minimal description length*), d.h. nach der minimalen Anzahl von Bits, um die gegebenen Daten D zu beschreiben. Hierzu benötigt man ein geeignetes *Modell* sowie eine Darstellung der Daten im Kontext dieses Modells.

5.6.2 Das MDL-Maß

Für die Herleitung der folgenden Formeln siehe (Friedman and Goldzmidt, 1999). Das MDL-Prinzip von Rissanen (1987) hat seine Wurzeln in der Kodierungstheorie. Ziel ist es, gegebene Daten $\mathfrak{X}$ möglichst kompakt zu übertragen. Eine Möglichkeit ist es, zunächst ein Modell zu wählen, welches die Daten beschreibt, dieses Modell zu übertragen und dann die Daten in diesem Modell zu senden. Die *vollständige Beschreibungslänge* ist dann die Summe der komprimierten Daten $\mathfrak{X}$ und die Länge der Beschreibung des Modells. Nach dem MDL-Prinzip ist das optimale Modell dasjenige, welches die kürzeste vollständige Beschreibungslänge hat.

Eine konsequente Anwendung des MDL-Prinzipes lässt den gesamten möglichen Modellraum zu. Dies ist aber nicht praktikabel. In dem Kontext dieser Arbeit sind die zulässigen Modelle die Bayes'schen Netze $(\mathcal{G}, \Theta)$, die Daten selbst werden also durch eine Wahrscheinlichkeitsverteilung p repräsentiert. Die vollständige Beschreibungslänge mit dem Modell $(\mathcal{G}, \Theta)$ setzt sich zusammen aus den Längen der Beschreibungen der Variablen X, des Graphen $\mathcal{G}$ und der Tabellen in Θ.

Die Variablen $\bar{X} = (X_1, \ldots, X_n)$ können beschrieben werden durch die Anzahl an Variablen sowie die Kardinalitäten r_i der Wertemengen $\mathcal{R}_i$. Da diese Information aber für alle Netze gleich ist und man nur untereinander vergleichen will, kann man diesen Teil weglassen.

Der Graph $\mathcal{G}$ kann folgendermaßen kodiert werden: für jede Variable i speichere die Anzahl $|\Pi_i|$ an Eltern ($\log n$ Bits) und dann die Liste der Elternknoten. Da es $\binom{n}{|\Pi_i|}$ Möglichkeiten für die Eltern gibt, folgt

$$\text{Länge}_G(\mathcal{G}) = \sum_{i=1}^{n} \left(\log n + \log \binom{n}{|\Pi_i|} \right) \tag{5.55}$$

Eine andere Repräsentation speichert einfach die äquivalente Matrix der Eltern und braucht deshalb n^2 Bits. Eigentlich sollte man ja die minimale Beschreibung wählen, und diese Darstellung ist aufwändiger, aber aus praktischen Gründen wird auf diese Darstellung noch zurückgekommen.

Mit $q_i := \prod_{X_k \in \Pi_i} r_k$ gibt es $\prod_i (r_i - 1) q_i$ Parameter in Θ (einer der Parameter ist redundant, weil die Summe der Wahrscheinlichkeiten 1 geben muss). Mit $\log(N)$ Bits kann man die Parameter mit hinreichender Genauigkeit speichern. Unter der Annahme, dass die Daten normalverteilt sind mit Varianz $1/\sqrt{N}$, reichen nach (Friedman and Yakhini, 1996) sogar die unteren $\frac{1}{2}\log(N)$ Bits zur Darstellung aus. Setze zunächst also

$$\text{Länge}_T(\Theta|\mathcal{G}) = \prod_{i=1}^{n} (r_i - 1) q_i, \tag{5.56}$$

wobei die vollständige Länge mit obiger Annahme also $\frac{1}{2}\log(N)\,\text{Länge}_T(\Theta|\mathcal{G})$ ist. Man kann nun die Daten $\mathfrak{X}$ mit Hilfe der Wahrscheinlichkeitsfunktion p beschreiben, indem man einen Huffmann-Code konstruiert, der Worten mit niedriger Wahrscheinlichkeit kürzere Codelänge zuweist. Es sei N_{ijk} die Anzahl von Fällen, bei denen Variable X_i in Zustand k ist und die Eltern Π_i in Zustand j. In dieser Arbeit sind dabei die Zustände aus Gründen der Konsistenz von 0 ab durchnummeriert, in der Literatur (Heckerman et al., 1995; Cooper and Herskovits, 1992) fangen die Zustände bei 1 an. Nach (Cover and Thomas, 1991) kann man die Codelänge durch den Logarithmus der Wahrscheinlichkeit approximieren und erhält

$$\begin{aligned}
\text{Länge}_{\mathfrak{X}}(\mathfrak{X}|\Theta,\mathcal{G}) &= -\sum_{l=1}^{N} \log p(\mathfrak{X}^l) && (5.57)\\
&= -\sum_{x} N \cdot p(x) \cdot \log p(x) && (5.58)\\
&= -\sum_{i=1}^{n} \sum_{j=0}^{q_i-1} \sum_{k=0}^{l_i} N_{ijk} \underbrace{\log p(X_i = j|\Pi_i = k)}_{\theta^{ijk}} && (5.59)
\end{aligned}$$

Mit Hilfe der Shannon-Gibbs-Ungleichung (Lemma A.8) folgt, dass die Beschreibungslänge der Daten genau dann minimal ist, wenn $\theta^{ijk} = N_{ijk}/N$. Mit gewähltem Graphen $\mathcal{G}$ zu Daten $\mathfrak{X}$ wird die vollständige Beschreibungslänge also minimiert, indem man für Θ die empirischen Häufigkeiten wählt.

Mit dieser Wahl für Θ (und somit p_M) kann man nun $\text{Länge}_{\mathfrak{X}}$ auch schreiben als

$$\text{Länge}_{\mathfrak{X}}(\mathfrak{X}|\Theta,\mathcal{G}) = N \sum_{i=1}^{n} H(X_i|\Pi_i) = N \cdot H(\mathcal{G}|\mathfrak{X}) \tag{5.60}$$

Die vollständige Beschreibungslänge ist damit

$$\text{Länge}(\mathcal{G}|\mathfrak{X}) = \text{Länge}_G(\mathcal{G}) + N \cdot H(\mathcal{G}|\mathfrak{X}) + \tfrac{1}{2}\log N \cdot \text{Länge}_T(\Theta|\mathcal{G}) \tag{5.61}$$

Der erste Term in Länge($\mathcal{G}|\mathfrak{X}$) ist unabhängig von N. Aus diesem Grund und weil er auch mit n^2 abgeschätzt werden kann (siehe vorige Anmerkung) wird er bei der folgenden Schätzung einfach weggelassen:

Definition 5.18. *Das* BIC-*Maß („Bayesian Information Criterion") (Schwarz, 1978) ist definiert durch*

$$\mathrm{BIC}(\mathcal{G}|\mathfrak{X}) := -N \cdot H(\mathcal{G}|\mathfrak{X}) - (\tfrac{1}{2}\log N) \prod_{i=1}^{n} (r_i - 1) q_i \tag{5.62}$$

Das BIC-Maß ist von Schwartz mit einer anderen Methode abgeleitet worden.

Führe nun einen Parameter α ein, um die Möglichkeit zu haben, zwischen der Genauigkeit der Approximation und der Größe des Modells zu wählen.

Definition 5.19. *Das* MDL-*Maß („Minimal Description Length") ist definiert durch*

$$\begin{aligned} \mathrm{MDL}_\alpha(\mathcal{G}|\mathfrak{X}) &:= N \cdot H(\mathcal{G}|\mathfrak{X}) + \alpha \log(N)\, \mathrm{Länge}_T(\Theta|\mathcal{G}) && (5.63)\\ &= -\sum_{i=1}^{n} \sum_{j=0}^{q_i-1} \sum_{k=0}^{l_i} N_{ijk} \log \tfrac{N_{ijk}}{N_{ij}} + \alpha \log(N) \prod_{i=1}^{n} (r_i - 1) q_i && (5.64) \end{aligned}$$

Im MDL repräsentiert die konditionale Entropie also, wie gut die Daten durch das Modell beschrieben werden (je kleiner die Entropie, desto besser), während der zweite Term eine Strafe für zu komplexe Modelle darstellt. Das MDL-Maß ist positiv und beschreibt, wieviele Bits zur Kodierung des Modells und der Daten (in dem gewählten Rahmen) nötig sind.

Satz 5.17 schafft nun eine Verbindung zwischen MDL und den Bayes'schen Netzen. Zusammenfassend lässt sich feststellen:

- Da das Hinzufügen von Kanten die konditionale Entropie *verkleinert*, würde das BIC-Maß automatisch steigen, wenn es nicht als Korrektiv den Straftterm der Komplexität des Modells gäbe. Ein Maß ohne einen solchen Term erzeugt also maximal komplexe Modelle.

- Abgesehen von der Komplexität entscheidet die Größe der konditionalen wechselseitigen Information $\mathrm{Inf}(X_i; X_j|\Pi_i)$ darüber, ob die Kante $(X_j \to X_i)$ in das Modell aufgenommen werden kann. Aus der Sicht der Informationstheorie ist dies ein Maß dafür, wie sehr X_i von X_j beeinflusst ist gegeben die Information in Π_i. Dies liefert eine plausible Begründung für die Wahl dieses Maßes.

- Insbesondere werden konditionale Unabhängigkeiten durch das Maß direkt berücksichtigt. Falls $X_i \perp\!\!\!\perp X_j|\Pi_i$, so wird das BIC-Maß durch Hinzufügen der Kante $(X_j \to X_i)$ kleiner (also schlechter) werden, weil der Entropieterm gleich bleibt, aber die Komplexität des Modells steigt. Dies ist im Rahmen dieser Theorie eine notwendige Forderung an ein Komplexitätsmaß, weil die additive Zerlegung sonst nicht berücksichtigt würde.

Beispiel 5.6. Betrachte die Population $\mathfrak{X} = \{00, 01, 10, 11\}$ mit den Variablen X_1 und X_2 und den leeren Graphen $\mathcal{G}_0$ mit Knoten X_1 und X_2 und leerer Kantenmenge sowie den Graphen $\mathcal{G}_1$ mit Kante $(X_1 \to X_2)$.

Für $\mathcal{G}_0$ gilt $r_1 = r_2 = 2$, $q_1 = q_2 = 1$ und die Zählwerte ergeben

$$N_{100} = N_{101} = N_{200} = N_{201} = 2 \qquad N_{10} = N_{20} = 4 \tag{5.65}$$

Damit gilt

$$\mathrm{MDL}_\alpha(\mathcal{G}_0, \mathfrak{X}) = -8 \log \tfrac{2}{4} + \alpha \log(4)1 = 8 + 2\alpha \tag{5.66}$$

Für $\mathcal{G}_1$ hingegen ist $r_1 = r_2 = 2$, $q_1 = 1$, aber $q_2 = 2$ und damit

$$N_{100} = N_{101} = 2 \qquad N_{200} = N_{201} = N_{210} = N_{211} = 1 \tag{5.67}$$

$$N_{10} = 4 \qquad N_{20} = N_{21} = 2 \tag{5.68}$$

Daher folgt

$$\mathrm{MDL}_\alpha(\mathcal{G}_1, \mathfrak{X}) = -4 \log \tfrac{2}{4} - 4 \log \tfrac{1}{2} + \alpha \log(4)2 = 8 + 4\alpha \tag{5.69}$$

Unabhängig von der Wahl von α wird also immer das einfachere (richtige) Netzwerk bevorzugt.

5.7 Bayes'sche Modellinferenz

Der Bayes'sche Ansatz zur Modellfindung ist etwas anders. Grundlage ist eine Verteilung im Raum der Modelle. Ausgehend von einer A-Priori-Verteilung im Raum der Modelle wird mittels der Bayes'schen Regel eine A-Posteriori-Verteilung ermittelt, mit der das Modell gewählt wird. Siehe hierzu auch Abschnitt A.4.

Wenn $\mathcal{G} \in \mathcal{M}$ die möglichen Strukturen repräsentiert, so ergibt sich die Wahrscheinlichkeitsverteilung gegeben der Daten als

$$p(X|\mathfrak{X}) = \sum_{\mathcal{G} \in \mathcal{M}} p(X, \mathcal{G}|\mathfrak{X}) = \sum_{\mathcal{G} \in \mathcal{M}} p(X|\mathcal{G}, \mathfrak{X}) p(\mathcal{G}|\mathfrak{X}), \tag{5.70}$$

also als Summe über alle möglichen Modelle. Da der Raum der möglichen Strukturen riesig ist, wird in der Praxis eine Approximation benutzt. Eine Möglichkeit ist es, nur eine Teilmenge an Strukturen zu berücksichtigen, die in der Summe fast Gewicht Eins haben (also $\sum_{\tilde{\mathcal{G}}} p(\tilde{\mathcal{G}}|\mathfrak{X}) \approx 1$). Meist wird aber ein Schritt weiter gegangen und nur *ein* Modell ausgesucht *(model selection)*, also

$$p(x|\mathfrak{X}) \approx p(x|\tilde{\mathcal{G}}, \mathfrak{X}) \qquad \text{mit } \tilde{\mathcal{G}} = \operatorname*{argmax}_{\mathcal{G} \in \mathcal{M}} p(\mathcal{G}|\mathfrak{X}) \tag{5.71}$$

Nun wird noch ein Maß benötigt, um das Modell auszuwählen. Hierfür wählt man den Logarithmus der Wahrscheinlichkeit Daten, gegeben das Modell. Man berechnet also nicht wirklich die A-Posteriori-Wahrscheinlichkeit des Modells (was wegen der Normalisierung sehr aufwändig wäre), sondern nur einen Term, der proportional dazu ist. Nach Bayes gilt

$$p(\mathcal{G}|\mathfrak{X}) \propto p(\mathfrak{X}|\mathcal{G})p(\mathcal{G}) = p(\mathfrak{X}, \mathcal{G}) \tag{5.72}$$

wobei $p(\mathcal{G})$ die Wahrscheinlichkeit für Modell $\mathcal{G}$ darstellt, bevor die Daten bekannt sind. Die folgenden Berechnungen stützen sich auf folgende Voraussetzungen: (Heckerman et al., 1995; Cooper and Herskovits, 1992):

1. Diskrete Variablen (hier erfüllt),
2. Vollständige Daten, also keine Variablen ohne Messungen (hier ebenfalls klar),
3. A-Priori-Verteilung von $p(\theta_{ijk}|\mathcal{G})$ ist die Gleichverteilung,
4. Datenpunkte aus $\mathfrak{X}$ sind unabhängig erzeugt, gegeben das Modell.

Die ersten beiden Punkte sind in unserer Anwendung also immer gegeben und der dritte drückt aus, dass man keine Annahmen über die Parameter hat (und wird in Abschnitt 5.7.1 aufgeweicht). Der letzte Punkt ist problematischer, weil er für den *FDA* nicht unbedingt gegeben ist: Wenn die Punkte durch ein Selektionsverfahren entstehen, dann sind sie nicht mehr unabhängig erzeugt, gegeben ein Bayes'sches Modell. Da aber versucht wird, diese Datenpunkte durch ein solches Modell zu beschreiben, ist diese Vereinfachung hier zulässig.

In (Cooper and Herskovits, 1992) wird gezeigt, dass unter diesen Voraussetzungen gilt:

$$p(\mathfrak{X}, \mathcal{G}) = p(\mathcal{G}) \prod_{i=1}^{n} \prod_{j=0}^{q_i-1} \frac{(r_i - 1)!}{(N_{ij} + r_i - 1)!} \prod_{k=0}^{l_i} N_{ijk}! \tag{5.73}$$

Der Logarithmus dieser Verteilung wird nun als Maß für die Güte angesehen.

Definition 5.20. *Das* BD*-Maß (Bayes-Dirichlet) ist definiert durch*

$$\mathrm{BD}(\mathcal{G}|\mathfrak{X}) := \log \left(p(\mathcal{G}) \prod_{i=1}^{n} \prod_{j=0}^{q_i-1} \frac{(r_i - 1)!}{N_{ij} + r_i - 1)!} \prod_{k=0}^{l_i} N_{ijk}! \right) \tag{5.74}$$

Oft wird im Raum der Modelle als A-Priori-Verteilung eine Gleichverteilung angesetzt. Dann kann der Faktor $p(\mathcal{G})$ entfallen, da er für alle Modelle gleich ist.

Wie werden nun in diesem Modell Wahrscheinlichkeiten ausgerechnet? Wenn man, wie oben, nur ein einziges Modell heranzieht, gilt nach (Heckerman et al., 1995)

$$p(x|\mathcal{G}, \mathfrak{X}) = \prod_{i} E[\theta^{ijk}|\mathcal{G}, \mathfrak{X}] \tag{5.75}$$

Es ist dabei das θ^{ijk} zu wählen, welches zu der Belegung x passt. Der Erwartungswert ist nach (Cooper and Herskovits, 1992) bei gleichverteilter A-Priori-Verteilung gegeben durch

$$E[\theta^{ijk}|\mathcal{G},\mathfrak{X}] = \frac{N_{ijk}+1}{N_{ij}+r_i} \tag{5.76}$$

Zusammen gibt das

$$p(x|\mathfrak{X}) \approx p(x|\tilde{\mathcal{G}},\mathfrak{X}) = \prod_{i=1}^{n} \frac{N_{ijk}+1}{N_{ij}+r_i} = \prod_{i=1}^{n} p(x_i|\pi_i) \tag{5.77}$$

In dieser Form ist das BD-Maß hier nicht geeignet, da es in der Praxis größere Strukturen bevorzugt. Theoretisch ist es zwar möglich, die A-Priori-Verteilung über $\mathcal{G}$ so zu wählen, dass größere Netze kleinere Wahrscheinlichkeit in der A-Priori- (und damit auch in der A-Posteriori-) Verteilung haben. Diese Wahl ist jedoch nicht so einfach.

Beispiel 5.7. Betrachte noch einmal die Daten aus Beispiel 5.6. Das Maß ist bestimmt durch den Logarithmus von $p(\mathcal{G})p(\mathfrak{X}|\mathcal{G})$. Es gilt für $\mathcal{G}_0$:

$$p(\mathfrak{X}|\mathcal{G}_0) = \left(\frac{2!2!}{4!}\right)\left(\frac{2!2!}{4!}\right) = \frac{1}{36} \tag{5.78}$$

und für $\mathcal{G}_1$:

$$p(\mathfrak{X}|\mathcal{G}_1) = \left(\frac{2!2!}{4!}\right)\left(\frac{1!1!}{2!2!}\right) = \frac{1}{24} \tag{5.79}$$

Wenn jetzt $p(\mathcal{G}_0) = p(\mathcal{G}_1)$, also beide Modelle a priori gleiche Wahrscheinlichkeit haben, so wird hier das *komplexere* Modell bevorzugt.

Asymptotisch (also für großes N) sind die beiden Maße äquivalent. Man kann für $N \to \infty$ das BIC-Maß auch als Approximation des BD-Maßes herleiten (Bouckaert, 1993). Für die hier verwendeten kleinen Datenmengen sind diese Überlegungen aber nicht relevant, da das BD-Maß hier nicht geeignet ist, siehe auch (Bouckaert, 1994).

5.7.1 Dirichlet-Prior für die Parameter

In (Cooper and Herskovits, 1992) wird auch der Fall behandelt, dass der Prior für die Parameter θ nicht mehr die Gleichverteilung ist. So wie in Anhang A.4 für den Münzwurf die *Beta*-Verteilung als Prior geeignet ist, ergibt sich hier die Dirichlet-Familie, die eine Erweiterung der *Beta*-Verteilung auf mehr als zwei Werte darstellt.

Mit den Hyperparametern N'_{ijk} für den Prior, das heißt

$$p(\theta_{ij1},\ldots,\theta_{ijl_i}|\mathcal{G}) \propto \prod_{k=0}^{l_i} \theta_{ijk}^{N'_{ijk}-1} \quad \forall 1 \le i \le n, \quad 0 \le j \le q_i - 1 \tag{5.80}$$

($\theta_{ij0} = 1 - \sum_{k=1}^{l_i} \theta_{ijk}$ ist kein Parameter) ergibt sich mit $N'_{ij} = \sum_{k=0}^{l_i} N'_{ijk}$ nach (Cooper and Herskovits, 1992)

$$E[\theta^{ijk}] = \frac{N_{ijk} + N'_{ijk}}{N_{ij} + N'_{ij}} \tag{5.81}$$

Es gibt hier wiederum keinen Grund, einen bestimmten *Wert* zu bevorzugen, daher kann man die N'_{ijk} für festes i gleich wählen und hat nur n Parameter N'_i, also $N'_{ijk} = N'_i$ und $N'_{ij} = r_i \cdot N'_i$.

5.8 Suche nach dem besten Modell

Die Suche nach einem Modell mit maximaler Bewertung (sei es BIC, MDL oder BD) ist leider noch aufwändiger als die Suche nach dem Maximum der Funktion. Dies wird plausibel, wenn man sich die Zahl der GaGs vergegenwärtigt:

Wenn G_n die Anzahl an gerichteten azyklischen Graphen bezeichnet, so ist klar, dass durch Hinzufügen eines Knotens X_{n+1} mindestens $2^n \cdot G_n$ neue Graphen entstehen, indem nämlich der neue Knoten Vater der bisherigen Knoten wird. Also folgt: $G_{n+1} \geq 2^n G_n$ oder $\log G_{n+1} \geq n + \log G_n$ und daher $G_n \geq 2^{n(n-1)/2} \approx 2^{n^2}$ (in (Robinson, 1977) wird eine exakte Formel hergeleitet). Aufzählen der Belegungen der Variablen hat aber nur einen Aufwand von 2^n. Man kann sich eine erschöpfende Suche also gar nicht leisten.

Nun könnte es aber sein, dass die Zerlegbarkeit des Maßes benutzt werden kann, einen effizienten Suchalgorithmus zu finden. In (Chickering et al., 1994) wird aber gezeigt, dass das Lernen eines Netzwerkes mit dem BD-Maß im schlimmsten Fall NP-hart ist. Daher wird ein Algorithmus benötigt, der eine approximative Lösung in polynomieller Zeit findet. Dafür wird hier ein einfacher *greedy* Algorithmus verwandt, der sukzessiv die Kante hinzufügt, die MDL am stärksten verringert.

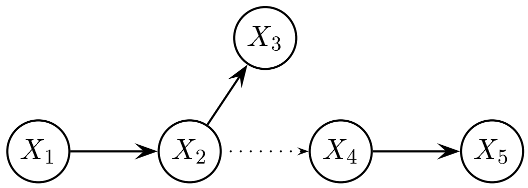

Abbildung 5.7: Beispiel zum Hinzufügen einer Kante aus Abschnitt 5.8

Um den Graphen azyklisch zu halten, wird für jeden Knoten die Liste der Vorgänger gespeichert. Betrachte Abbildung 5.7. Wenn die Kante ($X_2 \rightarrow X_4$) hinzugefügt wird, dann darf es keine Kanten mehr von X_4 oder X_5 zu X_1 oder X_2 geben. X_3 ist nicht beeinflusst von dieser Veränderung. Weiterhin muss die Liste der Vorgänger für X_4 und X_5 aktualisiert werden. Die genauere Beschreibung findet sich in Algorithmus 5.3. Dabei

Algorithmus 5.3: Lernen des Graphen

1 $F \Leftarrow \{(X_i \to X_j) | 1 \leq i \neq j \leq n\}$, die Liste der zulässigen Kanten.
2 $E \Leftarrow \emptyset$, $A_i \Leftarrow \emptyset$ (Liste der Vorgänger).
3 **do** {
4 Wähle $(X_i \to X_j) \in F$ aus, so dass MDL_α maximal verkleinert wird.
5 $E \Leftarrow E \cup (X_i \to X_j)$.
6 Entferne $(X_i \to X_j)$ und $(X_j \to X_i)$ aus F sowie rekursiv alle $(X_a \to X_b)$ mit X_a Nachkomme von X_j und $X_b \in A_i \cup \{X_i\}$.
7 $A_j \Leftarrow A_j \cup A_i$; Für alle Nachkommen X_a von X_j: $A_a \Leftarrow A_a \cup A_i$.
8 Falls X_j genau $k_{\max}$ Eltern hat: entferne für alle k $(X_k \to X_j)$ aus F.
9 } **while** (es noch eine Kante in F gibt, die MDL_α verkleinert)

wird außerdem noch eine maximale Anzahl an Elternknoten vorgegeben, um die Größe des gelernten Graphen zu beschränken.

Der Aufwand von Algorithmus 5.3 bestimmt sich wie folgt: Die äußere Schleife wird höchstens $n \cdot k_{\max}$ mal durchlaufen. Die Bestimmung der Kante habe Aufwand $A(N, n)$. Schritt 6 und 7 haben insgesamt höchstens Aufwand n^2, weil es nur so viele Kanten gibt. Zusammen ergibt sich

$$\textbf{Aufwand für das Lernen:} \qquad O(n^2 + n \cdot k_{\max} \cdot A(N, n)) \tag{5.82}$$

Anmerkung 5.21. Unter Berücksichtung von Korollar 5.14 und der Überlegungen vom Ende von Abschnitt 5.6.2 kann man für additiv zerlegte Funktionen die anfängliche Liste der zulässigen Kanten einschränken auf die Paare von Variablen, die gemeinsam in einer Teilfunktion auftauchen.

5.9 Inferenz in und Sampling aus Netzwerken

Sowohl für den Bayes'schen Ansatz zur Modellfindung als auch für das MDL-Maß kann man nun die Parameter θ_{ijk} aus der Maximum-Likelihood-Schätzung bestimmen oder durch den Bayes'schen Schätzer wie in Abschnitt 3.8. Es ergibt sich also

$$\theta_{ijk} = \frac{N_{ijk} + N'_i}{N_{ij} + r_i N'_i}, \tag{5.83}$$

wobei $N'_i = 0$ für Maximum Likelihood.

Das **Sampling** von neuen Punkten aus dem Netzwerk $(\mathcal{G}, \Theta)$ ist in beiden Fällen gleich. Die Wahrscheinlichkeitsdichte für jeden Punkt ist einfach

$$p(X) = \prod_i p(X_i | \Pi_i) \tag{5.84}$$

Der Faktorisierungsalgorithmus (Algorithmus 5.1 auf Seite 95) erzeugt eine Reihenfolge der Faktoren $p(X_i|\Pi_i)$, so dass alle Variablen aus Π_i schon vorkamen. Jetzt kann einfach in dieser Reihenfolge binomielles Sampling durchgeführt werden, da alle Parameter, auf die konditioniert wird, schon bekannt sind.

Beispiel 5.8. Betrachte die Faktorisierung

$$p(X) = p(X_1)p(X_2, X_3|X_1)p(X_4|X_3) \tag{5.85}$$

Erzeuge nun einen zufälligen Wert x_1 anhand der (bekannten) Verteilung $p(X_1)$. Da dieser Wert nun gesetzt ist, kann man aus $p(X_2, X_3|x_1)$ sampeln. Der Wert x_3 von X_3 erlaubt es nun, $p(X_4|x_3)$ auszuwerten.

Wahl des Bayes'schen Schätzers

In Abschnitt 3.8 wurde der Schätzer über die Nebenbedingung bestimmt, dass bei einer Population, die nur aus Optima besteht, die Wahrscheinlichkeit zur Generierung dieses Optimums mindestens 30% beträgt. Dies lässt sich auch auf die Bayes'schen Netze erweitern.

Satz 5.22. *Es seien N_i' die Parameter des Priors. Für große N und n gilt: Wenn $N_i' = \frac{N}{n \cdot r_i}$, so erzeugt eine identische Population dieses Individuum mit Wahrscheinlichkeit größer* 30%.

Beweis: Die Parameter berechnen sich zu

$$\theta^{ijk} = \frac{N_{ijk} + N_i'}{N_{ij} + r_i N_i'} \tag{5.86}$$

Wenn alle Individuen gleich sind, sind alle Zählwerte für diesen Fall gleich N. Abzuschätzen ist also

$$\prod_{i=1}^{n} \frac{N + N_i'}{N + r_i N_i'} \geq e^{-1} \tag{5.87}$$

Wie in 3.8 reicht es, wenn jeder Faktor größer als $(1 - {}^1\!/\!{}_n)$ ist. Dazu muss gelten:

$$\frac{N + N_i'}{N + r_i N_i'} = 1 - \frac{(r_i - 1)N_i'}{N + r_i N_i'} \geq 1 - \frac{1}{n} \tag{5.88}$$

$$\Leftrightarrow N_i' \left[(r_i - 1) - \tfrac{r_i}{n}\right] \geq \frac{N}{n} \tag{5.89}$$

$$\Leftrightarrow N_i' \geq \frac{N}{n(r_i - 1) - r_i} \tag{5.90}$$

(Satz 5.22) ■

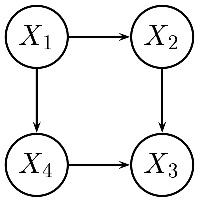

Abbildung 5.8: Dieser Graph erfüllt nicht die *running intersection property* (siehe Abschnitt 5.9.1)

5.9.1 Die Running Intersection Property

Die durch Bayes'sche Netze generierten Wahrscheinlichkeits-Verteilungen respektieren die *running intersection property* nicht notwendigerweise, wie Abbildung 5.8 verdeutlicht.

Die Faktorisierung zu diesem Graphen ist

$$p(X) = p(X_1)p(X_2|X_1)p(X_4|X_1)p(X_3|X_2, X_4) \tag{5.91}$$

Dies ist ein gültiges Bayes'sches Netzwerk. Da die Faktorisierung identisch ist zu Beispiel 4.2 auf Seite 72, treten dieselben undefinierten Wahrscheinlichkeiten auf, wenn nicht der Bayes'sche Schätzer für die Parameter θ_{ijk} verwendet wird. Zur Erinnerung: Es kann etwa passieren, dass beim Generieren von neuen Punkten X_2 und X_4 auf Null gesetzt werden (weil dies in den Ursprungsdaten auch vorkam), dass aber beide nie *gemeinsam* Null waren, daher $p(X_2{=}0, X_4{=}0) = 0$ ist und somit $p(X_3|X_2{=}0, X_4 = 0)$ nicht definiert.

Bei Wahl von Bayes'schen Schätzern entsteht in dieser Situation nach (5.86) als Wahrscheinlichkeit $\frac{0+N_i'}{0+r_i N_i'} = \frac{1}{r_i}$, also die Gleichverteilung. Aus der Sicht der Bayesianer ist dies folgerichtig, weil über den Fall $p(X_2{=}0, X_4{=}0)$ keine Evidenzen vorliegen und daher eine Verschiebung der Wahrscheinlichkeiten von der Gleichverteilung weg (die durch den Prior gegeben ist) nicht begründet werden kann.

Die Tatsache, dass die *running intersection property* für Bayes'sche Netze nicht unbedingt erfüllt sein muss, erlaubt es, das Gitter zu faktorisieren, wie Abbildung 5.9 zeigt. Diese Faktorisierung ist zwar nicht exakt, aber sie erlaubt, fast alle Abhängigkeiten zu erfassen und dennoch polynomiellen Aufwand zu haben.

5.10 Zusammenfassung

Die Theorie der graphischen Modelle und insbesondere der Bayes'schen Netzwerke schafft eine theoretische Grundlage für die Frage, wie die Faktorisierung für den *FDA* gewählt werden kann.

$$
\begin{tikzcd}
X_{1,1} \arrow[r] \arrow[d] & X_{1,2} \arrow[r] \arrow[d] & X_{1,3} \arrow[r, dotted] \arrow[d] & X_{1,n} \arrow[d] \\
X_{2,1} \arrow[r] \arrow[d] & X_{2,2} \arrow[r] \arrow[d] & X_{2,3} \arrow[r, dotted] \arrow[d] & X_{2,n} \arrow[d] \\
X_{3,1} \arrow[r] \arrow[d, dotted] & X_{3,2} \arrow[r] \arrow[d, dotted] & X_{3,3} \arrow[r, dotted] \arrow[d, dotted] \arrow[dr, dotted, no head] & X_{3,n} \arrow[d, dotted] \\
X_{n,1} \arrow[r] & X_{n,2} \arrow[r] & X_{n,3} \arrow[r, dotted] & X_{n,n}
\end{tikzcd}
$$

Abbildung 5.9: Das Gitter als Bayes'sches Netzwerk

Eigentlich drücken die (fehlenden) Kanten in dem Graphen konditionale Unabhängigkeiten aus. Es zeigt sich aber, dass die additive Zerlegung der Funktion über die Boltzmann-Verteilung ebenso konditionale Unabhängigkeiten induziert. Die beiden Arten der Zerlegung liefern also das gleiche Ergebnis.

In der Theorie der Bayes'schen Netze gibt es mehrere Ansätze, die Modelle aus den Daten zu lernen. Für diese Arbeit hat sich die *Minimum Description Length* als am günstigsten herausgestellt. Da der Suchraum so groß ist, wird in dieser Arbeit ein einfaches Verfahren benutzt, um ein Modell mit kleinem MDL-Maß auszuwählen.

Im nächsten Kapitel wird der *FDA* mit diesem Lernalgorithmus zusammengeführt, um den *LFDA* zu erhalten.

6 Der *LFDA* und erste Resultate

In dem *FDA* musste die Faktorisierung noch vorgegeben werden oder aus der additiven Zerlegung der Funktion gewonnen werden. In Kapitel 5 wurde die Theorie erklärt, wie man ein Bayes'sches Netz und damit eine Faktorisierung aus Datenpunkten gewinnen kann. In diesem Kapitel werden die zwei Elemente zu einem Algorithmus zusammengefügt, dem *LFDA*, der die Faktorisierung in jedem Schritt neu lernt.

6.1 Definition des *LFDA*

Bild 6.1 veranschaulicht den Zyklus, den der *LFDA* durchläuft.

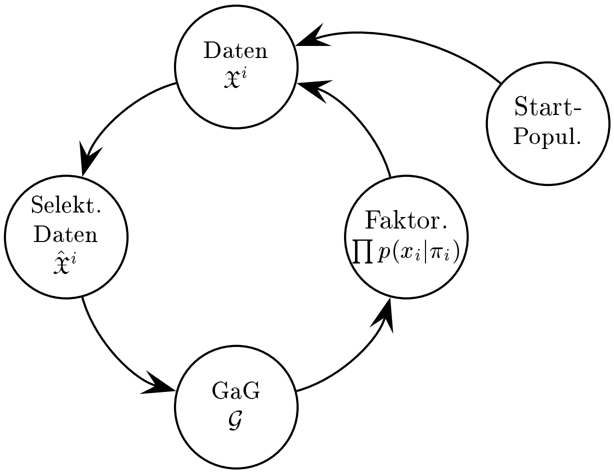

Abbildung 6.1: Der *LFDA*-Zyklus

Zunächst werden die Individuen $\mathfrak{X}^i$ zufällig initialisiert. Dann setzt der Selektionsprozess ein, der aus diesen Individuen selektiert. Die selektierten Individuen werden nun als Datenpunkte interpretiert. Mit Hilfe des MDL-Maßes wird ein Bayes'sches Netzwerk konstruiert und daraus eine Faktorisierung inklusive Wahrscheinlichkeitstabelle.

Mit dieser Faktorisierung werden dann neue Daten generiert und somit der Zyklus fortgesetzt, bis das Abbruchkriterium greift.

Algorithmus 6.1: *LFDA* – Learning *FDA*

Erzeuge eine zufällige Ausgangspopulation mit N Individuen.
do {
  Selektiere $\hat{N} \leq N$ Punkte.
  Bestimme ein Bayes'sches Netzwerk durch Minimieren des MDL-Maßes (Algorithmus 5.3).
  Bestimme eine Faktorisierung aus dem Graphen (Algorithmus 5.1).
  Berechne die konditionalen Wahrscheinlichkeiten $p(x_{b_i}|x_{c_i}, t)$ aus den selektierten Punkten.
  Erzeuge neue Punkte gemäß $p(x, t+1) = \prod_{i=1}^{l} p(x_{b_i}|x_{c_i}, t)$.
  $t \Leftarrow t + 1$.
} **until** (Abbruchkriterium erreicht)

6.2 Beispielläufe

Die folgenden Läufe wurden mit Boltzmann-Selektion mit Konstante $c = 1$ durchgeführt. Nach 30 Generationen oder wenn das globale Optimum erreicht war wurde der Algorithmus abgebrochen. Für den *FDA* wurde empirisch ein günstiger Wert für den Prior N' ermittelt. Dieser Wert wurde dann auch für die anderen Algorithmen benutzt.

Es geht in den folgenden Läufen nur darum, die Leistungsfähigkeit des *FDA* und des *LFDA* unter Beweis zu stellen. Die behandelten Probleme haben alle eine relativ kleine Größe von nur 30 Bits. Größere Probleme und Fragen der Skalierung werden in Kapitel 8 behandelt. Dort zeigt sich auch, dass die Erfolgsrate sehr stark von der Populationsgröße N abhängt. In der Regel führt bereits eine Reduktion um ein Drittel dazu, diese von über 90% auf fast Null zu reduzieren.

6.2.1 *OneMax*

Das Lernen der Faktorisierung hat einen Preis. Wenn die gelernte Struktur nicht mit der tatsächlichen übereinstimmt, kann der Algorithmus in die Irre geführt werden. Weiterhin steigt die benötigte Populationsgröße mit der Zahl der Eltern.

Als erstes betrachte daher in Tabelle 6.1 die Funktion *OneMax*. Gegenübergestellt sind der *UMDA* (der in diesem Fall dem *FDA* entspricht, weil die einfache Produktverteilung für *OneMax* die richtige Verteilung ist), der *LFDA*, der die Struktur lernt, und eine Variante des *FDA*, der mit *UMDA*-4 bezeichnet ist und der je vier benachbarte Variablen zusammenfasst:

$$p(X) = p(X_1, X_2, X_3, X_4)p(X_5, X_6, X_7, X_8) \cdots \tag{6.1}$$

Es zeigt sich, dass der *LFDA* nicht nur deutlich höheren Aufwand hat als der *UMDA*, sondern für diese Funktion auch schlechtere Ergebnisse liefert. Er braucht eine hohe Populationsgröße, um die Struktur richtig zu schätzen und um die Parameter der Struktur

	Ohne Prior			Prior .25		
	N	Erfolg	Gen	N	Erfolg	Gen
UMDA	40	91/100	7.7	20	96/100	15.3
$UMDA-4$	200	91/100	5.5	80	99/100	11.7
LFDA	300	90/100	6.6	80	95/100	17.5

Tabelle 6.1: Erfolgsraten für *OneMax*, 32 Bits

zu bestimmen. Dies wird durch die Ergebnisse des *UMDA*-4 deutlich. Alleine die Tatsache, dass hier je vier Variablen zu einer Meta-Variablen mit Wertebereich $\{0, \ldots, 31\}$ zusammengefasst wurden, erhöhte die benötigte Zahl an Individuen deutlich.

Für diese einfache Funktion kommen alle Algorithmen auch bei sehr kleiner Populationsgröße dem Optimum schon sehr nahe. Deshalb reicht etwas Mutation aus, um das Optimum dann noch zu finden. Mit Prior $1/4$ kann man sehr kleine Populationsgrößen verwenden. Die Anzahl an Generationen (bis das Optimum zum ersten Mal generiert wurde) steigt dabei aber deutlich an, weil der Algorithmus dann nur noch eine Zufallssuche macht. Die Zahl der Funktionsevaluierungen (Produkt aus N und Zahl der Generationen) verändert sich durch die Mutation daher für dieses Beispiel nicht sehr stark.

Weiterhin kann man beobachten, dass die Zahl der Generationen für *UMDA*-4 *kleiner* ist als für *UMDA*. Dies liegt ebenfalls daran, dass *UMDA*-4 das Problem als Funktion von 8 Meta-Variablen auffasst und für weniger Variablen eben weniger Generationen benötigt.

6.2.2 *Deceptive*

Die *Deceptive*-Funktion aus Abschnitt 3.7.3 (Seite 45) soll nun die Leistungsfähigkeit des *FDA* und des *LFDA* demonstrieren.

	Ohne Prior			Prior .25		
	N	Erfolg	Gen	N	Erfolg	Gen
UMDA	3000	0/30	–	3000	0/30	–
FDA	200	94/100	5.0	80	97/100	11.0
$UMDA-2$	3000	5/100	13.4	1000	5/100	12.6
LFDA	900	86/100	5.7	800	91/100	6.9

Tabelle 6.2: Erfolgsraten für $Deceptive_4$, 32 Bits

Wie schon die Analyse der Fitness-Landschaft gezeigt hat, kann der *UMDA* diese Funktion nicht optimieren, selbst mit einer sehr großen Populationsgröße. Da auch jeweils vier Variablen neu zu setzen sind, um von einem Maximum zum nächsten zu springen, nützt die Mutation hier auch nicht.

Für den *FDA* ist diese Funktion nicht sehr schwierig. Er fasst die Blöcke wieder als Meta-Variablen auf. Dies zeigt sich dadurch, dass Erfolgsrate und Zahl der Generationen den Ergebnissen von *UMDA*-4 (Tabelle 6.1) sehr ähnlich sind. Beide benutzen dieselbe Verteilung (Gleichung 6.1).

Der *UMDA*-2 fasst je zwei Variablen zusammen und kommt damit der wahren Faktorisierung einen Schritt näher. Es zeigt sich aber, dass dies immer noch nicht ausreicht, um das Problem zu lösen. Ein erfolgreicher Algorithmus muss alle Blöcke aus vier Variablen erkennen und danach die Verteilung bestimmen. Es zeigt sich, dass der *LFDA* mit 900 Individuen dazu in der Lage ist.

Gen	1	2	3	4	5	6	...	14
UMDA	1.18	3.21	5.18	6.15	5.90	4.54		0.80
LFDA	2.35	1.11	0.46	0.31	0.29	0.30		0.60

Tabelle 6.3: Kullback-Leibler-Divergenz für *Deceptive*, 16 Bits

In Tabelle 6.3 ist dargestellt, wie die Kullback-Leibler-Divergenz zwischen der einfachen Produktverteilung, der gelernten Verteilung und der echten Verteilung aussieht. Dazu wurden für 16 Bits mit Populationsgröße 200 und Prior 0.25 30 Läufe mit dem *FDA* durchgeführt. Die Struktur zur Erzeugung neuer Punkte war daher immer die echte Struktur

$$p(X) = p(X_1, X_2, X_3, X_4)p(X_5, X_6, X_7, X_8) \cdots p(X_{13}, X_{14}, X_{15}, X_{16}). \tag{6.2}$$

Außerdem wurden für jede selektierte Population die Parameter der einfachen Produktverteilung (entsprechend dem *UMDA*) bestimmt und zwischen diesen beiden Verteilungen die Kullback-Leibler-Divergenz durch aufsummieren über alle 2^{16} möglichen Individuen bestimmt. Ferner wurde für die selektierte Population noch ein Struktur gelernt und dann ebenfalls die Divergenz zwischen dieser Verteilung und der aus (6.2) berechnet.

Auf Grund des Priors ist die gelernte Struktur nicht notwendigerweise besser als die einfache Produktverteilung, wie sich auch in der ersten Generation zeigt. In den weiteren Generationen zeigt sich aber, dass sich die schlechte Leistung des *UMDA* auch in einer sehr viel größeren Abweichung von der erwarteten Verteilung widerspiegelt. Am Ende des Laufes sind beide Divergenzen relativ klein. Die Population besteht dann fast nur noch aus dem Optimum. Die beiden Divergenzen unterscheiden sich, weil sich der feste Prior für die verschiedenen Faktorisierungen unterschiedlich auswirkt.

6.2.3 *XorKette*

Als nächstes Beispiel dient die Funktion *XorKette* aus Abschnitt 3.7.5. Es handelt sich dabei um lauter *xor*-Funktionen, die als Kette angeordnet sind. Das globale Optimum sind die beiden Individuen mit alternierenden 0- und 1-Bits. Die Ergebnisse sind in Tabelle 6.4 dargestellt.

Es ist für den *UMDA* nicht unmöglich, das Optimum zu finden. An vielen Stellen gleichzeitig stellen sich Ketten von alternierenden Bits ein. Mit einer geringen Wahrscheinlichkeit (je größer n, desto unwahrscheinlicher) passen diese Ketten zueinander, dann wird

	Ohne Prior			Mit Prior 2		
	N	Erfolg	Gen	N	Erfolg	Gen
UMDA	3000	14/100	13.8	3000	16/100	13.1
UMDA	500	9/100	12.4	500	6/100	12.0
FDA	90	87/100	5.7	90	96/100	7.4
LFDA	400	82/100	6.0	400	90/100	6.9

Tabelle 6.4: Ergebnisse für *XorKette* mit 30 Bits

das globale Optimum gefunden. Wenn sie nicht passen, kann der *UMDA* dies nicht mehr reparieren. Natürlich fällt die Wahrscheinlichkeit für das Zusammenpassen drastisch mit der Problemlänge ab.

Diese Vermutung wird dadurch bestätigt, dass die Erfolgsrate bei der deutlich kleineren Populationsgröße 500 nicht wesentlich abnimmt.

Der *FDA* kann das Problem, wie erwartet, lösen. Mutation bringt eine leichte Verbesserung der Erfolgsrate.

Gen	1	2	3	4	5	6	...	20
UMDA	0.71	2.31	4.31	6.43	8.03	8.85		1.84
LFDA	0.52	0.29	0.17	0.13	0.19	0.28		0.80

Tabelle 6.5: Kullback-Leibler-Divergenz für *XorKette*, 16 Bits

Aber auch der *LFDA* ist in der Lage, die Struktur zu lernen, wie nicht nur die Läufe zeigen, sondern auch Tabelle 6.5. Für 16 Bits wurde $N = 200$ gewählt. Wieder ist die Kullback-Leibler-Divergenz (zur „echten“ Verteilung) deutlich niedriger für den *LFDA* als für den *UMDA*.

6.2.4 *IsoC*

Eine schwieriges Beispiel ist die Funktion *IsoC*. Sie ist wie folgt definiert:

$$\mathit{IsoC}(x) = \sum_{i=1}^{m-1} \mathit{IsoC}_1(x_{2i-1}, x_{2i}) + \mathit{IsoC}_2(x_{2m-1}, x_{2m}) \tag{6.3}$$

x	00	01	10	11
IsoC_1	m	0	0	$m-1$
IsoC_2	0	0	0	m

mit $n = m + 1$. Das globale Optimum ist $(1, 1, \dots, 1)$ mit Wert $m \cdot (m-1) + 1$. Dieses Optimum ist durch die Belegung von IsoC_2 ausgelöst. Es ist stark isoliert. Der zweitbeste Wert kommt mehrmals vor, aber nur bei Individuen mit führenden Nullen, zum Beispiel

	Ohne Prior		
	N	Erfolg	Gen
UMDA	3000	0/30	–
FDA	250	97/100	5.1
LFDA	750	91/100	5.4

Tabelle 6.6: Erfolgsraten für *IsoC*, 30 Bits

für $(0,\ldots,0)$ mit Wert $m \cdot (m-1)$. Alle diese Punkte sind in der Fitness-Landschaft sehr weit von dem Optimum entfernt.

Auch für diese Funktion findet der *UMDA* das globale Optimum nicht. Der *FDA* hat wieder keine Schwierigkeiten mit einer ähnlichen Zahl an Individuen wie für *Deceptive*. Auch dem *LFDA* gelang es, die Kettenstruktur zu lernen und so das Optimum zu finden.

Mutation brachte für keines der Verfahren eine spürbare Verbesserung, wie schon bei *XorKette*. Dies liegt wohl daran, dass alle Bits gemeinsam gekippt werden müssen, um vom lokalen zum globalen Optimum zu kommen. Wenn diese Struktur vom Algorithmus nicht erfasst wird, hilft auch die Mutation nicht weiter.

6.2.5 Kaufmans $n-k$-Funktionen

Ein weiterer Repräsentant sind die Funktionen aus Kauffmans $n-k$-Modell (Kauffman and Levin, 1987). Sie bestehen aus n binären Variablen und n Subfunktionen, wobei die i-te Subfunktion von X_i und k weiteren Variablen abhängt. Diese anderen Variablen bilden die *Nachbarschaft* von X_i. Die Funktionswerte der Subfunktionen werden zufällig aus einer gegebenen Verteilung bestimmt, in der Regel die Gleichverteilung auf dem Intervall $[0,1]$. Anschließend werden die Werte noch durch die Zahl an Teilfunktionen dividiert, um für die gesamte Funktion Wertebereich $[0,1]$ zu erhalten:

$$\mathit{Kauffman}_{n,k}(x) = \frac{1}{n}\sum_{i=1}^{n} f_{s_i}(x) \qquad |s_i| = k+1,\ i \in s_i,\ f_{s_i}(x) \in [0,1] \tag{6.4}$$

Die Nachbarschaft kann unterschiedlich definiert sein. Im einfachsten Fall sind es einfach die angrenzenden Variablen X_{i-1}, X_{i+1} etc. Oft werden die Indizes aber ebenfalls zufällig gezogen. Die Wahl von k erlaubt es, unterschiedlich schwierige Funktionen zu generieren. Für $k=0$ handelt es sich um eine lineare Funktion, während $k=n-1$ einer vollkommen zufälligen Funktion entspricht. Für angrenzende Variablen gibt es einen polynomiellen Lösungsalgorithmus. Viele Suchverfahren können diese Eigenschaft aber nicht ausnutzen und verhalten sich für zufällige Nachbarn und angrenzende Nachbarn gleich. Eine ganze Reihe von Eigenschaften der Fitness-Landschaften sind zudem auch identisch (wie Höhe der lokalen Optima, Länge von Hillclimber-Wegen) (Kauffman and Weinberger, 1989). Dies kann man ausnutzen, um Probleminstanzen zu generieren, bei denen das Optimum bekannt ist.

Für zufällige Nachbarn ist das Entscheidungsproblem zur Optimierung ab $k=2$ NP-vollständig (Weinberger, 1996; Thompson and Wright, 1996).

			Ohne Prior			Mit Prior N'			
Algo	Nachbarn	N	$\bar{f}$	σ	Erfolg	N'	$\bar{f}$	σ	Erfolg
DynProg	Kette	–	0.7543	0.026	100/100	–	0.7543	0.026	100/100
UMDA	Kette	200	0.7463	0.029	27/100	1	0.7476	0.029	31/100
LFDA	Kette	200	0.7489	0.028	35/100	.5	0.7513	0.027	57/100
LFDA sub	Kette	200	0.7526	0.027	64/100	.5	0.7519	0.027	63/100
UMDA	Kette	400	0.7484	0.029	28/100	1	0.7489	0.028	37/100
LFDA	Kette	400	0.7537	0.027	82/100	.5	0.7537	0.027	84/100
LFDA sub	Kette	400	0.7540	0.026	89/100	.5	0.7538	0.027	87/100
UMDA	Zufall	200	0.7654	0.027		1	0.7739	0.026	
LFDA	Zufall	200	0.7624	0.025		.5	0.7778	0.023	
LFDA sub	Zufall	200	0.7730	0.024		.5	0.7824	0.023	
UMDA	Zufall	400	0.7683	0.025		1	0.7710	0.024	
LFDA	Zufall	400	0.7814	0.024		.5	0.7859	0.026	
LFDA sub	Zufall	400	0.7904	0.022		.5	0.7884	0.021	

Tabelle 6.7: Ergebnisse für *Kauffman* für die verschiedenen Algorithmen

In Tabelle 6.7 sind Ergebnisse für 30 Bits, $k=2$, angrenzende und zufällige Nachbarschaft sowie Populationsgröße 200 und 400 dargestellt. Bei „*LFDA* sub“ wurden nur die aus der additiven Zerlegung bekannten Kanten zugelassen. Für angrenzende Nachbarschaft (Kette) wurde mittels dynamischem Programmieren das globale Maximum ermittelt und so eine Erfolgsrate bestimmt. In beiden Fällen wurden 100 verschiedene Probleminstanzen generiert und je ein Lauf pro Instanz durchgeführt. Die Maxima dieser verschiedenen Instanzen erzeugen eine Verteilung. Da dynamisches Programmieren diese zuverlässig ermittelt für die Kette, wurden Mittelwert und Standardabweichung dieser Verteilung in der Zeile *DynProg* angegeben.

Für die anderen Algorithmen sind der durchschnittliche erreichte Fitnesswert $\bar{f}$ und die Standardabweichung dieser Werte angegeben. Für eine hohe Erfolgsrate müssen sich diese Werte denen des dynamischen Programmierens annäheren. Die beiden Blöcke mit angrenzender und zufälliger Nachbarschaft sind nicht direkt vergleichbar.

Der *UMDA* ist deutlich abgeschlagen gegenüber den beiden Lernalgorithmen. Man sieht auch, dass er von der Vergrößerung der Populationsgröße eigentlich nicht profitieren kann. Alle Methoden profitieren deutlich von dem Prior gegenüber der *Maximum-Likelihood*-Schätzung, nur der *LFDA*-sub bei großer Populationsgröße nicht.

Der *LFDA* schneidet deutlich besser ab als der *UMDA* und kann diesen Vorsprung durch eine größere Populationsgröße (oder Prior) noch weiter ausbauen. Eine weitere starke Verbesserung erreicht der *LFDA*-sub.

6.3 Andere Ansätze

Es gibt in der Literatur eine Reihe von Ansätzen, um Optimierungen anhand von Wahrscheinlichkeitsverteilungen durchzuführen. Die ersten Modelle verwendeten die einfache Produktverteilung und können daher Abhängigkeiten zwischen Variablen nicht aufdecken. Zu dieser Kategorie gehören der *BSC (Bit-Based Simulated Crossover)* (Syswerda, 1993) und der *PBIL (Population-Bases Incremental Learning)* (Baluja and Caruana, 1995). Diese benutzen alle verschiedene Heuristiken, um die Parameter der Verteilung zu modifizieren. Der *UMDA* (Mühlenbein and Paaß, 1996) gehört auch in diese Kategorie. Hier werden zum ersten Mal die Parameter ausschließlich aus den selektierten Punkten geschätzt.

Die nächste Kategorie versucht Abhängigkeiten von je zwei Variablen zu entdecken. *MIMIC (Mutual Information Maximization for Input Clustering)* (Bonet et al., 1997) bildet als Faktorisierungen grundsätzlich Ketten, also

$$p(x) = p(X_{i_1})p(X_{i_2}|X_{i_1})p(X_{i_3}|X_{i_2})\cdots p(X_{i_{n-1}}|X_{i_n}) \tag{6.5}$$

und minimiert die Kullback-Leibler-Divergenz zwischen diesen Faktorisierungen und den selektierten Punkten. Dazu nehmen die Autoren eine einfache *greedy* Suche, da es $(n!)$ verschiedene Faktorisierungen gibt.

Der *FDA* war der erste Algorithmus, der versuchte, die Verteilung der Punkte exakt abzubilden (Mühlenbein and Mahnig, 1999b, 2000b,a). Außerdem liefert die damit einhergehende Theorie der Boltzmann-Verteilung erstmals einen theoretischen Hintergrund über die Beziehung zwischen Selektion, selektierten Punkten und der generierten Verteilung. Der *LFDA* wurde in (Mühlenbein and Mahnig, 1999b) eingeführt und in (Mühlenbein and Mahnig, 2000b,a) weiter untersucht.

Es gibt inzwischen mehrere Arbeiten, die das Lernen der Struktur einbeziehen (Pelikan et al., 1999; Ortiz et al., 1999). Dabei wird oft die Klasse der betrachteten Strukturen eingeschränkt, um eine geringere Komplexität zu haben. So lassen sich für Baum-Strukturen polynomielle Lernalgorithmen finden.

6.4 Zusammenfassung

Der *LFDA* verbindet die Theorie der graphischen Modelle mit der der populationsbasierten Optimierungsalgorithmen. Die Wirksamkeit dieses Verfahrens wurde an mehrere Beispielfunktionen gezeigt, die von klassischen genetischen Algorithmen nicht optimiert werden können.

In Kapitel 9 wird der Algorithmus auf ein in der Praxis vorkommendes Problem, die Graph Bipartitionierung, angewandt.

7 Implementierungen und Aufwand

7.1 Aufwand der Verfahren

7.1.1 Selektion

Vor der Selektion müssen die Individuen erst sortiert werden. Für manche Verfahren ist dies zwar überflüssig, am Aufwand ändert dies aber nichts, wie sich gleich zeigt. Das Sortieren lässt sich mit Aufwand $O(N \log N)$ erledigen.

Für *Truncation Selection* ist nach der (hier notwendigen) Sortierung nur noch ein linearer Aufwand nötig, um die besten Individuen zu markieren.

Boltzmann-Selektion

Boltzmann-Selektion selektiert gemäß

$$p^s(x) = \frac{p(x)e^{\beta f(x)}}{\sum_y p(y)e^{\beta f(y)}} \tag{7.1}$$

Da β im Laufe der Zeit ständig steigt, werden die zu berechnenden Terme wegen der Exponentialfunktion viel zu groß. Die Boltzmannverteilung ist aber invariant bei Addition einer Konstanten. Es gilt also:

$$p^s(x) = \frac{p(x)e^{\beta(f(x)-K)}}{\sum_y p(y)e^{\beta(f(y)-K)}} \tag{7.2}$$

Bei Selektionsstärke γ verwandelt sich β_{alt} in $\beta = \beta_{alt} + \gamma$, so folgt mit einer endlichen Population $\mathfrak{X}$ Algorithmus 7.1.

Mit diesem Algorithmus bleiben die betrachteten Terme in der Regel so klein, dass die Darstellungsgenauigkeit von *double*-Werten ausreicht. Das Ziehen mit der gegebenen Verteilung ist so implementiert, dass für jedes Element $\hat{\mathfrak{X}}^j$ des sortierten Arrays die kumulative Summe $\sum_{i=1}^{j} e^{\gamma(f(\mathfrak{X}^i)-K)}$ gespeichert wird. Dann wird eine (reelle) Zufallszahl zwischen 0 und S gezogen und mittels binärer Suche der größte Index k gesucht, für den die kumulative Summe noch kleiner ist. Der Aufwand dieses Verfahrens ist $O(\log N)$ (weil $\hat{N} \leq N$), damit ergibt sich zusammen ein Aufwand $O(N \log N)$. Proportionale Selektion lässt sich ebenso implementieren.

Zusammenfassend gilt also, dass für alle betrachteten Selektionsverfahren insgesamt mit Sortierung ein Aufwand von $O(N \log N)$ entsteht.

Algorithmus 7.1: Boltzmann-Selektion

Bestimme $K = \min_i f(\mathfrak{X}^i)$, $\gamma := \beta_{neu} - \beta_{alt}$.
Bestimme $S = \sum_i e^{\gamma(f(\mathfrak{X}^i)-K)}$.
for $j \Leftarrow 1$ **to** $\hat{N}$ **do** {
    Ziehe $\hat{\mathfrak{X}}^j$ aus $\mathfrak{X}$ mit Verteilung $e^{\gamma(f(\mathfrak{X}^i)-K)}/S$.
}

7.1.2 *UMDA*

Für den *UMDA* (Algorithmus 3.1 auf Seite 15) ergibt sich der Aufwand wie folgt. Sei $\hat{\mathcal{R}}$ die maximale Kardinalität der Wertebereiche $\mathcal{R}_i$, für binäre Funktionen also $\hat{\mathcal{R}}=2$. Sortieren und Selektieren erfordert Aufwand $O(N \log N)$. Das Bestimmen der Randverteilungen $p_i(x_i)$ lässt sich in $O(n \cdot (\hat{\mathcal{R}} + N))$ erledigen (Initialisieren und Zählen, siehe Algorithmus 7.2).

Algorithmus 7.2: *UMDA* – Bestimmen der Randverteilungen

for $i \Leftarrow 1$ **to** n
    for $x_i \Leftarrow 0$ **to** l_i
        $N_{i,x_i} \Leftarrow 0.$
for $h \Leftarrow 1$ **to** $\hat{N}$
    for $i \Leftarrow 1$ **to** n
        $N_{i,\hat{\mathfrak{X}}_i^h} \Leftarrow N_{i,\hat{\mathfrak{X}}_i^h} + 1$
for $i \Leftarrow 1$ **to** n
    for $x_i \Leftarrow 0$ **to** l_i
        $p_i(x_i) \Leftarrow \frac{N_{i,x_i}+N_i'}{N+r_i \cdot N_i'}.$

Das Generieren neuer Individuen ist in Algorithmus 7.3 beschrieben und erfordert dort Aufwand $O(n \cdot \hat{\mathcal{R}})$. Für große $\hat{\mathcal{R}}$ kann man die innere Schleife auch durch eine binäre Suche ersetzen. Das Aufsetzen der dafür benötigten kumulativen Summe benötigt dann $O(n \cdot \hat{\mathcal{R}})$ Schritte, das Ziehen geht in $O(n \cdot \log \hat{\mathcal{R}})$, so dass sich $O(n \cdot \log \hat{\mathcal{R}})$ ergibt. In den Anwendungen wurde dies allerdings nicht benötigt.

Im Normallfall kann man $\hat{\mathcal{R}}$ als kleine Konstante ansehen und aus der Abschätzung herauslassen. Es ergibt sich dann

$$\begin{aligned}\textbf{Aufwand für } \textbf{\textit{UMDA}}\textbf{:} \quad & O(\mathit{Selektion}) + O(\mathit{Generieren}) \\ = \quad & O(\mathit{Gen} \cdot N \log N) + O(\mathit{Gen} \cdot N \cdot n)\end{aligned} \tag{7.3}$$

wobei *Gen* die Anzahl an Generationen angibt (siehe Abschnitt 8.2).

Nach Abschnitt 4.2.7 lassen sich aber auch die Variablen zusammenfassen, wobei der Wertebereich exponentiell steigt. Im Extremfall gibt es nur noch eine Variable, deren

Algorithmus 7.3: *UMDA* – Generieren eines neuen Individuums

for $i \Leftarrow 1$ **to** n **do** {
  $r \Leftarrow$ gleichverteilte (Pseudo-)Zufallszahl zwischen 0 und 1
  $x_i \Leftarrow 0$
  while $(r > q_{i,x_i})$ **do** {
    $r \Leftarrow r - q_{i,x_i}$, $x_i \Leftarrow x_i + 1$
  }
}

Wertebereich den gesamten Definitionsbereich abdeckt. Dann ist $\hat{\mathcal{R}}$ viel größer als n oder N und es gibt den Aufwand $O(n \cdot \hat{\mathcal{R}})$: *UMDA* hat dann exponentiellen Aufwand.

7.1.3 *FDA*

Ein wesentlicher Faktor der Komplexität beim *FDA* ist die Größe der Elternmengen. Bei einheitlichen Wertemengen mit $\hat{\mathcal{R}}$ Einträgen ergeben sich für die Wahrscheinlichkeitstabellen für $k_{\max}$ Eltern $\hat{\mathcal{R}}^{k_{\max}+1}$ Parameter. Ohne weitere Einschränkung gilt $k_{\max} = (n-1)$ und es gibt den gleichen exponentiellen Aufwand wie für den *UMDA* mit einer Hyper-Variablen.

Sortieren und Selektieren hat den gleichen Aufwand wie zuvor ($O(N \log N)$)), aber das Schätzen der Parameter aus den Daten hat nun Aufwand $O(n \cdot (\hat{\mathcal{R}}^{k_{\max}+1} + N \cdot k_{\max}))$, siehe Algorithmus 7.4.

Algorithmus 7.4: *FDA* – Bestimmen der Parameter

for $i \Leftarrow 1$ **to** n
  for $j \Leftarrow 0$ **to** $q_i - 1$
    for $k \Leftarrow 0$ **to** l_i
      $N_{ijk} \Leftarrow 0$, $N_{ij} \Leftarrow 0$
for $i \Leftarrow 1$ **to** n **do** {
  for $h \Leftarrow 1$ **to** $\hat{N}$ **do** {
    $k \Leftarrow \hat{\mathfrak{X}}_i^h$, $j \Leftarrow$ Belegung der Eltern in $\hat{\mathfrak{X}}^i$
    $N_{ijk} \Leftarrow N_{ijk} + 1$, $N_{ij} \Leftarrow N_{ij} + 1$
  }
}
for $i \Leftarrow 1$ **to** n
  for $j \Leftarrow 0$ **to** $q_i - 1$
    for $k \Leftarrow 0$ **to** l_i
      $p(X_i = k | \Pi_i = j) \Leftarrow \frac{N_{ijk} + N_i'}{N_{ij} + r_i \cdot N_i'}$

Das Generieren eines neuen Individuum ist in Algorithmus 7.5 dargestellt. Das Identifizieren der richtigen Tabelle $p(x_{b_i}|\Pi_{b_i})$ erfordert $O(k_{\max})$ Schritte, damit kann man *ein* Individuum in $O(n \cdot (\hat{\mathcal{R}} + k_{\max}))$ erzeugen.

Algorithmus 7.5: *FDA* – Generieren eines neuen Individuums

for $i \Leftarrow 1$ **to** n **do** {
    $r \Leftarrow$ gleichverteilte (Pseudo-)Zufallszahl zwischen 0 und 1
    $x_{b_i} \Leftarrow 0$, Π_{b_i} ist schon gesetzt oder leer
    while $(r > p(x_{b_i}|\Pi_{b_i}))$ **do** {
        $r \Leftarrow r - p(x_{b_i}|\Pi_{b_i}), x_{b_i} \Leftarrow x_{b_i} + 1$
    }
}

Zusammen folgt (wieder für kleines $\hat{\mathcal{R}}$):

$$\textbf{Aufwand für } \boldsymbol{FDA}\textbf{:} \qquad O\big(Gen \cdot (N \log N + n\hat{\mathcal{R}}^{k_{\max}+1} + n \cdot N \cdot (2k_{\max} + \hat{\mathcal{R}}))\big) \tag{7.4}$$

Für festes $k_{\max}$ und $\hat{\mathcal{R}}$ ergibt sich also der gleiche Aufwand wie beim *UMDA*.

7.1.4 Lernen des Netzwerkes

Um die Komplexität der Struktursuche zu berechnen, noch einmal die Formel für die Berechnung des MDL-Maßes (5.63):

$$\mathrm{MDL}_\alpha(\mathcal{G}|\mathfrak{X}) := N \cdot H(\mathcal{G}|\mathfrak{X}) + \alpha \log(N) \prod_{i=1}^{n} (r_i - 1) q_i \tag{7.5}$$

wobei

$$H(\mathcal{G}|\mathfrak{X}) = \sum_{i=1}^{n} H(X_i|\Pi_i) = -\sum_{i=1}^{n} \sum_{j=0}^{q_i-1} \sum_{k=0}^{l_i} \frac{N_{ijk}}{N} \log \frac{N_{ijk}}{N_{ij}} \tag{7.6}$$

Der zweite Term im MDL-Maß ist für die Komplexität unerheblich. Wir betrachten ferner nur den Fall, dass $k_{\max}$ und $\hat{\mathcal{R}}$ klein sind gegen n und N. Weil nach Voraussetzung $r_i \le \hat{\mathcal{R}}$ und $q_i \le \hat{\mathcal{R}}^{k_{\max}}$, gibt es die triviale Komplexitätsabschätzung $O(n \cdot \hat{\mathcal{R}}^{k_{\max}+1})$ für die Anzahl der Summanden im MDL-Maß. N_{ijk} ist die Anzahl an Individuen in der Population, bei denen Variable X_i den Wert k hat und die Eltern von X_i (nach dem aktuellen Graph) den Wert j. Es lässt sich mit Aufwand $O(N)$ ermitteln. Damit ist der Aufwand, um einen Term $H(X_i|\Pi_i)$ zu bestimmen, auch $O(N)$. Wenn eine Kante hinzugefügt werden soll, muss diese Rechnung also für $O(n^2)$ mögliche Kanten durchgeführt werden. Da insgesamt höchstens $n \cdot k_{\max}$ Kanten möglich sind, ergibt sich ein Aufwand von $O(n^3 \cdot N)$.

Man kann jetzt eine Reihe von Maßnahmen treffen, um die Effizienz des Algorithmus zu verbessern. Die wesentliche Überlegung ist die, dass Kanten nur inkrementell hinzugefügt werden. Bei Hinzufügen von $(X_i \to X_j)$ ändert sich aber von den Koeffizienten $H(X_k|\Pi_k)$ nur einer, nämlich für $k = i$. Man kann also eine Tabelle erstellen, welchen Gewinn das Hinzufügen jeder möglichen Kante $(X_i \to X_j)$ hat (Aufwand $O(n^2 \cdot N)$), und muss beim Hinzufügen einer Kante in dieser Tabelle höchstens $(n-1)$ Koeffizienten neu berechnen, mit Aufwand $O(n \cdot N)$. Weiterhin kann man für jede Variable X_i speichern, welche Kante $(X_j \to X_i)$ die beste ist und findet so mit Aufwand $O(n)$ die beste Kante zum Hinzufügen. Dies ist in Algorithmus 7.6 ausgeführt.

Algorithmus 7.6: Lernen des Netzwerkes, Bestimmen der Koeffizienten

$F \Leftarrow \{(X_i \to X_j) \big| 1 \leq i, j \leq n\}, E \Leftarrow \emptyset$.
$g_{i,j} \Leftarrow$ Zugewinn von MDL_α bei Hinzufügen von $(X_i \to X_j)$
$g_{*j} \Leftarrow \max_i g_{i,j}$.
do {
    $j^* \Leftarrow \mathrm{argmax}_j\, g_{*j}$; $i^* \Leftarrow \mathrm{argmax}_i\, g_{i,j^*}$. Es folgt: $(X_{i^*} \to X_{j^*})$ verkleinert MDL_α maximal.
    $E \Leftarrow E \cup (X_{i^*} \to X_{j^*})$.
    Berechne $\forall i : g_{i,j^*}$ neu, bestimme $g_{*(j^*)} \Leftarrow \max_i g_{i,j^*}$.
    Entferne $(X_{i^*} \to X_{j^*})$ und $(X_{j^*} \to X_{i^*})$ aus F sowie alle Kanten, die einen Zyklus oder zu viele Eltern bewirken würden.
} **while** (es noch eine Kante in F gibt, die MDL_α verkleinert)

In Abschnitt 5.8 wurde bereits gezeigt, dass sich das Vermeiden von Zyklen mit Aufwand n^2 bewerkstelligen lässt. Zusammen ergibt sich so der Aufwand $O(n^2 \cdot N + (n \cdot k_{\max})[n + n \cdot N])$, also

$$\textbf{Aufwand für das Lernen pro Generation:} \qquad O(n^2 \cdot N) \tag{7.7}$$

Der Aufwand kann erheblich kleiner sein, wenn das Netzwerk von vorneherein durch Kenntnis der additiven Zerlegung eingeschränkt wurde (Anmerkung 5.21).

Eine Reihe von weiteren Optimierungen dient der weiteren Beschleunigung. So können die N_{ijk} in einem Cache zwischengespeichert werden. Damit wird auch die Symmetrie der Berechnung von $(X_i \to X_j)$ bzw. $(X_j \to X_i)$ ausgenutzt.

7.1.5 Aufwand des *LFDA*

Der *LFDA* hat bei beschränkter Zahl der Eltern und beschränktem Wertebereich einen Aufwand von

$$\textbf{Aufwand für } \boldsymbol{LFDA}\textbf{:} \qquad O\big(Gen \cdot (N \log N + n^2 \cdot N)\big) \tag{7.8}$$

In Kapitel 8 sind empirisch ermittelte Werte für die benötigte Populationsgröße und die Anzahl an Generation angegeben.

8 Skalierung der Algorithmen

8.1 *Truncation Selection* oder Boltzmann-Selektion

Die Fokussierung auf die Optima (Satz 4.6) gilt nur für die Boltzmann-Verteilung. Für *Truncation Selection* gibt es noch keinen Beweis, da die Verteilung *nach* der Selektion viel komplizierter ist. Züchter argumentieren hier, dass die Fitness-Verteilung, also die Verteilung der Fitnesswerte, näherungsweise eine Gauß-Verteilung ist.

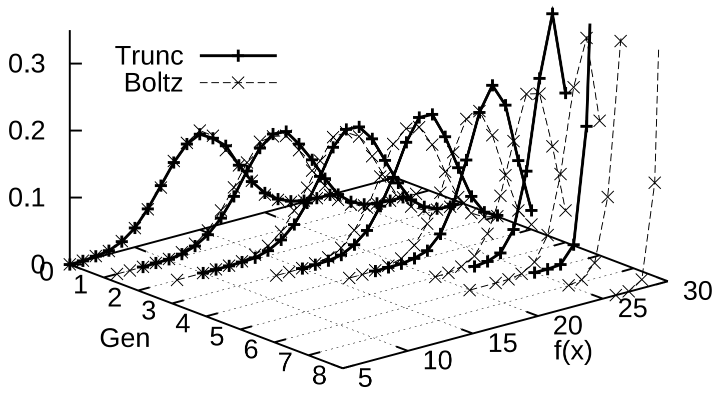

Abbildung 8.1: Fitness-Verteilungen für *OneMax*: Sowohl *Truncation Selection* als auch Boltzmann-Selektion führen zu sehr ähnlichen Verteilungen.

Aus Stabilitätsgründen wurde in den Testläufen oft *Truncation Selection* verwendet. In Abbildung 8.1 sind vergleichend die Fitness-Verteilungen für *Truncation Selection* und Boltzmann-Selektion dargestellt für die Funktion *OneMax* mit 30 Bits und Populationsgröße 10000.

Nach Abschnitt 4.4.1 sollten diese Verteilungen bei gleichem Selektionsdruck gleich sein, was durch den Simulationslauf bestätigt wird. Erst gegen Ende der Simulation unterscheiden sich die Kurven stärker. Dies lässt sich darauf zurückführen, dass die Boltzmann-Verteilung am Ende nicht mehr so gut approximiert wird, was sich in Abschnitt 4.4.1 bereits gezeigt hat. Die z-Achse ist abgeschnitten, eigentlich ist der Wertebereich auf $[0, 1]$ skaliert. Für proportionale Selektion kommt hier übrigens ein ganz anderer Verlauf heraus, da diese Selektionsmethode viel langsamer ist.

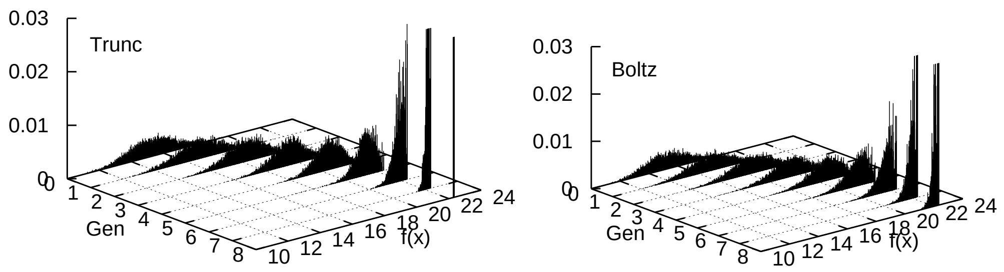

Abbildung 8.2: Fitness-Verteilungen für *Kauffman*

In Abbildung 8.2 ist die gleiche Untersuchung für einen Repräsentanten der $(n-k)$-Funktionen von Kauffman gemacht. Bei 30 Bits wurde $k = 1$ gewählt mit zufälliger Nachbarschaft. Auch für diese Funktion ist die Dynamik bei beiden Selektionsverfahren ähnlich. Welche der beiden Methoden genommen wird, spielt bei diesen Funktionen also kaum eine Rolle. Die Boltzmann-Selektion mit dem *SDS* war ja auch so konstruiert, dass sie *Truncation Selection* möglichst nachbilden sollte. Dies soll hier weiter untermauert werden.

		Trunc, $\tau = 0.3$					Boltz, $c = 1.159$				
Func	n	N	Erfolg	Gen1	Gen	σ	N	Erfolg	Gen1	Gen	σ
OneMax	32	40	89/100	5.2	7.5	0.64	40	89/100	7.0	12.6	1.54
OneMax	64	60	91/100	8.4	10.9	0.65	65	89/100	10.4	16.6	1.87
Saw	30	120	94/100	8.5	23.9	6.40	120	97/100	9.4	23.7	5.01
Deceptive	32	140	90/100	4.6	7.7	0.75	160	87/100	5.0	12.0	2.13
Deceptive	64	280	98/100	7.5	10.6	0.60	320	97/100	8.3	15.4	1.38
Kauffman	32	220	92/100	5.3	9.2	0.93	250	90/100	6.5	28.1	3.80
Kauffman	64	400	96/100	9.3	13.1	0.97	500	92/100	11.4	30.0	0.00
IsoC	32	400	95/100	4.2	8.7	0.89	350	95/100	5.1	14.7	1.48
IsoC	64	1200	94/100	7.3	13.0	1.00	1000	94/100	8.4	19.4	1.62

Tabelle 8.1: Erfolgsraten und Anzahl an Generationen mit Populationsgröße N für verschiedene Probleme mit Bitlänge n. Nach 30 Generationen ohne Konvergenz wurde abgebrochen. Gen1 ist die Anzahl an Generationen (gemittelt über 100 Läufe), bis das Optimum einmal in der Population war, Gen die Zahl der Generationen, bis die Population nur noch aus dem Optimum besteht.

Anstatt die Verteilungen zu vergleichen, kann man auch direkt die Ergebnisse der beiden Methoden vergleichen, also die Erfolgsrate für feste Populationsgröße oder die Anzahl der Generationen. Dies wird in Tabelle 8.1 gemacht. Die Erfolgsrate ist die Anzahl der Läufe aus 100, bei denen das globale Optimum gefunden wurde, Gen1 die dazugehörige Anzahl an Generationen, bis es *einmal* generiert wurde, und Gen die Zahl an Generationen, bis die Population nur noch aus dem Optimum bestand mit Standardabweichung σ.

Die betrachteten Funktionen sind die folgenden. Für *Kauffman* (Abschnitt 6.2.5) wurde $k=2$ und angrenzende Nachbarschaft gewählt sowie 100 verschiedene Probleme.

Die Definition der *Saw*-Funktion ist aus Abbildung 8.3 ersichtlich. Hier wurde nur eine Wert von n genommen, weil eine Verdopplung der Bitzahl das Problem einfacher macht.

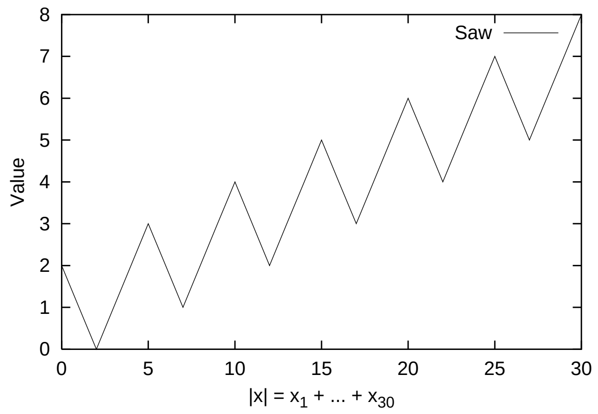

Abbildung 8.3: Definition der *Saw*-Funktion.

Die anderen Funktionen sind in Kapitel 6 definiert. Es wurde jeweils die exakte Faktorisierung gewählt, nur bei der *Saw*-Funktion *UMDA*.

Aus Tabelle 8.1 kann man folgende Schlüsse ziehen. Zunächst ist die benötigte Populationsgröße für eine gegebene Erfolgsrate und die Anzahl an Generationen ähnlich für beide Selektionsmethoden. Die Anzahl an Generationen, bis das Optimum einmal generiert wurde, war etwa 20% höher für Boltzmann-Selektion. Dieser Abstand vergrößert sich noch, wenn die Zeit bis zur Konvergenz betrachtet wird. Dies liegt daran, dass *Truncation Selection* am Ende schneller konvergiert.

Beachte, dass der Algorithmus in manchen Fällen gar nicht konvergierte. Dies zeigt sich an Generationszahlen größer als 20, da nach 30 Generationen abgebrochen wurde. Die Populationsgröße für ungefähr 90% Erfolg war manchmal größer und manchmal kleiner für Boltzmann-Selektion.

8.2 Skalierungseigenschaften

Ohne Einschränkung werden nur binäre Variablen behandelt, also $\hat{\mathcal{R}} = 2$.

Der zur Optimierung nötige Aufwand ist von einer Reihe von Variablen gesteuert. Nach (7.4) ist der Aufwand des *FDA* für festes $\hat{\mathcal{R}}$ und $k_{\max}$ gegeben durch

$$O\big(Gen \cdot N \cdot (n + \log N)\big) \tag{8.1}$$

Weil man in der Regel N, die Populationsgröße, in derselben Größenordnung wie n, die Bitlänge, wählt, ergibt sich ein Aufwand von

$$O(Gen \cdot N \cdot n) \tag{8.2}$$

Es handelt sich natürlich trotzdem nicht um einen Algorithmus, der linearen Aufwand in n hat, weil die erforderliche Zahl der Generationen sicherlich auch von n abhängt. Zusammenfassend gibt es folgende Parameter, die sich auf unterschiedliche Art auf den Algorithmus auswirken.

- Die Problemlänge n.
- Die Populationsgröße N.
- Die Selektionsintensität τ beziehungsweise c für Boltzmann-Selektion.
- Die Mutationsrate oder äquivalent der Bayes'sche Schätzer Prior N'.

Diese Parameter beeinflussen die folgenden Größen:

- Die Anzahl an Generationen *Gen*, nach denen entweder das (bekannte) Optimum erreicht ist oder nach denen die Population konvergiert.
- Die Wahrscheinlichkeit ρ, das Optimum zu finden.
- Die kritische Populationsgröße $N^*(\varepsilon)$, bei der $\rho > 1 - \varepsilon$.

Konvergenz lässt sich zum Beispiel für den *UMDA* dadurch feststellen, dass alle Randverteilungen nahe bei 0 oder 1 sind. Wenn die Faktorisierung exakt ist, so findet der *UMDA* oder *FDA* das Optimum fast sicher, wenn die Population groß genug ist. Deshalb macht es Sinn, die Größe $N^*(\varepsilon)$ zu definieren. Sie stellt eine untere Schranke dar. Ein weiteres Erhöhen der Populationsgröße verschlechtert die Erfolgsrate nicht, treibt aber die Zahl an Funktionsevaluierungen in die Höhe.

Das eigentliche Ziel ist es, mit einer möglichst hohen Wahrscheinlichkeit das Optimum zu finden und dabei möglichst wenig Funktions-Evaluierungen *FE* zu benötigen. Diese Zahl ergibt sich aus dem Produkt der Anzahl an Generationen und der Populationsgröße.

Diese Faktoren sollen nun einzeln diskutiert werden. Es gibt leider für beliebige Funktionen keine analytischen Zusammenhänge zwischen den Parametern und dem Ergebnis. Deshalb werden einerseits die *OneMax*-Funktion betrachtet, bei der man theoretisches Verständnis hat, und eine Sammlung von *Kauffman*-Funktionen (Abschnitt 6.2.5). Hierfür wurde $n=30$ und $k=2$ gewählt mit angrenzender Nachbarschaft, um das Maximum zu kennen. Dafür wurde über 100 verschiedene Instanzen gemittelt.

Die folgenden Läufe sind, wenn nichts anderes angegeben ist, für $n=30$ Bits, Boltzmann-Selektion mit $c=1$ und ohne Mutation durchgeführt.

8.2.1 Die Problemlänge

In (Mühlenbein, 1998) war für *OneMax* die Beziehung

$$Gen = \frac{\pi\sqrt{n}}{2I_\tau} \tag{8.3}$$

hergeleitet worden (siehe auch Abschnitt 3.5.2). In Abbildung 8.4 zeigt sich, dass auch Boltzmann-Selektion (mit Populationsgröße 1400) so skaliert. Man sieht ferner, dass auch für die *Kauffman*-Funktion die Zahl der Generationen nur mit der Wurzel von n wächst. Zu beachten ist dabei, dass die Anzahl an Generationen gezählt wird, bis die Population aus einem identischen Individuum besteht.

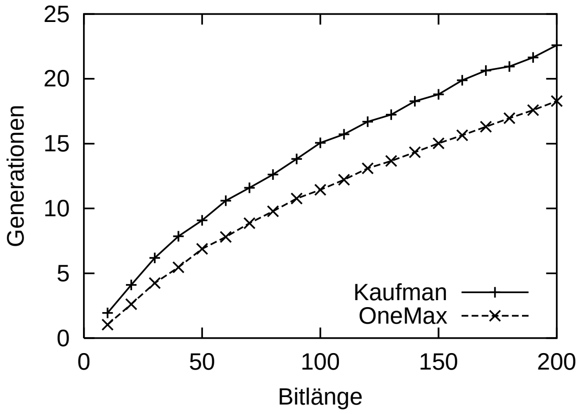

Abbildung 8.4: Skalierung der Anzahl an Generationen mit der Bitlänge. Es ergibt sich das erwartete Resultat $Gen \propto \sqrt{n}$.

Ein anderes Modell für das Verhalten des Algorithmus ist das Folgende: Ausgehend von einer sehr großen Populationsgröße und der Annahme, dass die Verteilungen durch den Algorithmus exakt reproduziert werden, kann man sagen, dass für $\tau = 0.5$ in jedem Schritt die schlechtere Hälfte der Population entfernt wird. Da es $\hat{\mathcal{R}}^n$ verschiedene Individuen gibt und diese Zahl in jedem Schritt um den Faktor τ reduziert wird, wird nach

$$Gen \leq n \cdot \frac{\log \hat{\mathcal{R}}}{\log {}^1\!/\!_\tau} \tag{8.4}$$

Schritten nur noch das Optimum in der Population sein. Dass hier die Skalierung in n linear und nicht wie $\sqrt{n}$ ist, erklärt sich dadurch, dass die Verteilungen eben *nicht* exakt reproduziert werden und dass die Selektion die Verteilung in Richtung des dominanten linearen Anteils verschiebt (siehe auch Abschnitt 3.5.2).

Truncation Selection benötigt in der Praxis weniger Schritte als Boltzmann-Selektion, weil diese gerade am Ende etwas langsamer wird.

Untersuchungen bis zu 500 Bits sind in (Mühlenbein and Mahnig, 1999b) vorgestellt.

8.2.2 Die Populationsgröße

Einer der bemerkenswertesten Zusammenhänge für Boltzmann-Selektion und *Truncation Selection* ist der, dass die Anzahl der Generationen zur Konvergenz *nicht* von der Populationsgröße N abhängt, sofern man nur die Läufe zählt, bei denen das Optimum gefunden wurde. Deshalb ist es legitim, zur Ermittlung der Generationen in Abbildung 8.4 einfach eine sehr große Population zu nehmen, da die Größe keinen Einfluss auf das Resultat hat, solange das Optimum gefunden wird.

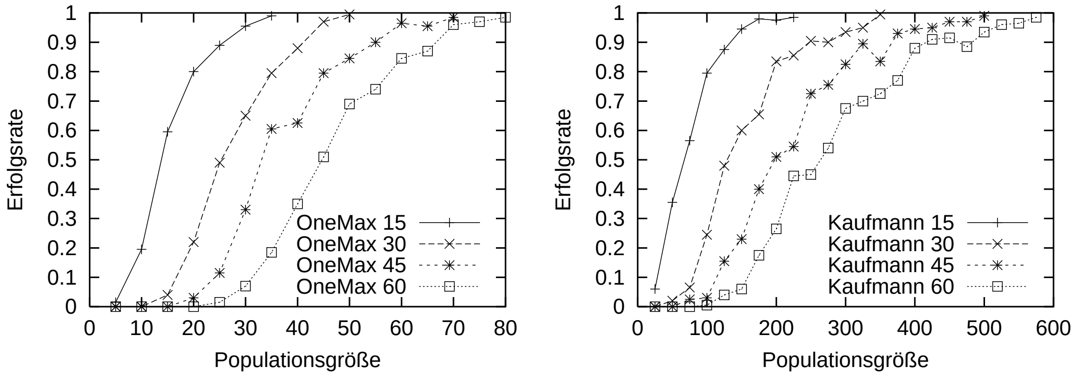

Abbildung 8.5: Skalierung der Erfolgsrate mit der Populationsgröße. Für diese Beispiel skaliert die Erfolgsrate bei fester Erfolgsrate annähernd linear (Abstand zweier Kurven).

Wenn die Populationsgröße zu klein wird, tritt die sogenannte *Fixation* auf. Wenn nämlich in der gesamten Population ein Gen überall den gleichen Wert hat, so wird dies (ohne Mutation) auch in der Nachfolgegeneration so sein.

Zur Erklärung dieser Graphen dient ein einfaches Modell. Betrachte ausschließlich die Fixation in der ersten Generation. Die Wahrscheinlichkeit, dass ein bestimmtes Bit in der ganzen Population den gleichen Wert hat, ist p_0^N. Damit ist die Wahrscheinlichkeit, dass dies für kein Bit passiert, gleich $(1-p_0^N)^n$ und somit

$$\rho \approx (1-p_0^N)^n \tag{8.5}$$

Die so entstehenden Kurven sind in Abbildung 8.6 dargestellt. Man sieht hier einerseits, dass die Kurven in diesem Parameterbereich nur linear in n skalieren. Auflösen der Gleichung (8.5) nach N ergibt mit der Näherung $1-p_0^n \approx \frac{1}{n}\log(p_0)$:

$$N^* \approx \frac{1}{\log(1/\rho)}\big(\log(n) + \log(\log p_0)\big) \tag{8.6}$$

Andererseits ist auch eine Kurve für $p_0 = 0.7$ dargestellt. Dies modelliert den Fall, dass sich die Population bereits sehr weit von dem zu erzielenden Optimum wegbewegt hat. Dadurch verschiebt sich die Kurve deutlich nach rechts, das heißt, die Erfolgswahrscheinlichkeit wird viel kleiner für festes N.

Diese Kurven sind den experimentellen Kurven aus Abbildung 8.5 sehr ähnlich. Die aktuellen Werte sind natürlich ganz anders, weil in dem Modell nur die Fixation in

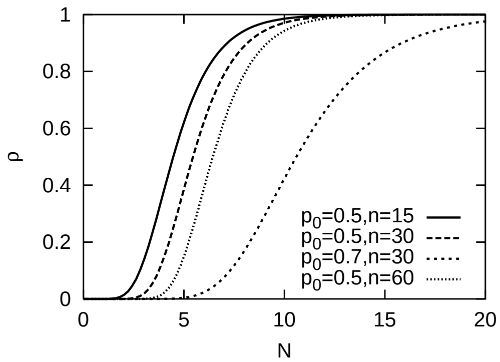

Abbildung 8.6: Nicht-Fixationswahrscheinlichkeit in einer Generation. Das Modell aus Abschnitt 8.2.2 erklärt die Natur der Kurven aus Abbildung 8.5.

der ersten Generation berücksichtigt wird. Selektion spielt dabei überhaupt keine Rolle. Gleichung (8.5) ist also nicht als benutzbare Modellierung gedacht, sondern nur geeignet, die Natur der Graphen aus Abbildung 8.6 besser zu verstehen.

8.2.3 Auswirkung zu kleiner Populationsgröße

Wenn die Populationsgröße zu klein ist, so hat der Algorithmus ohne Mutation in der Regel keine Chance, das Optimum zu finden. Betrachte Tabelle 8.2.

N	N'	Erfolg	Gen	FE
50	2	92/100	23.5	1203
75	2	95/100	14.7	1163
75	0	76/100	6.0	516
120	0	99/100	5.2	733

Tabelle 8.2: Auswirkung der Populationsgröße bei *XorKette*, 30 Bits

Mit Populationsgröße 50 ist für 30 Bits zwar schon eine hohe Erfolgsrate zu verzeichnen. Dies ist aber vor allem der Mutation zuzuschreiben. Wenn nämlich N auf 75 steigt, fällt die Zahl der Generationen (bis das Optimum einmal erreicht wurde) drastisch von 23.5 auf 14.7 ab. Die Suche durch Mutation ist nicht mehr zielgerichtet und benötigt deshalb eine große Zahl an Schritten. Die effektive Zahl an Funktionsevaluierungen ist in beiden Fällen annähernd gleich.

Ganz ohne Mutation sieht man genauer, was passiert. Die Erfolgsrate geht für 75 Individuen auf 76% herunter, dafür wird das Ziel schon in 6 Generationen erreicht. Wenn also mit Prior $N'=2$ das Optimum nicht in den ersten Schritten gefunden wird, setzt eine Suche ein, die hauptsächlich durch die Mutation gesteuert ist und daher nicht sehr effizient arbeitet. Die nötigen Funktionsevaluierungen sind bezogen auf die erfolgreichen Läufe und daher nicht vergleichbar. Mit einer Populationsgröße von 120 wird aber eine extrem hohe Erfolgsquote erreicht mit insgesamt deutlich verringerter Zahl an Evaluierungen. Im Beispiel der *Kauffman*-Funktion in Abschnitt 6.2.5 war der gleiche Effekt bei zu niedriger Populationsgröße zu beobachten.

Das soll nicht heißen, dass Mutation generell schlecht ist. Sie verhindert in vielen Fällen frühzeitige Konvergenz. Für schwierige Funktionen mit gelernter Struktur ist ein erfolgreiches Finden des globalen Optimums auch mit relativ großer Populationsgröße nicht zu garantieren. Dann kann die Mutation einen entscheidenden Beitrag leisten, siehe Abschnitt 8.2.5.

8.2.4 Die Selektionsintensität

Wenn die Selektionsfaktor τ zu klein wird, also zu wenig Individuen pro Generation zur Zucht benutzt werden, dann steigt die benötigte Populationsgröße an, weil die Verteilung mit den wenigen selektierten Punkten nicht mehr reproduziert werden kann. Im Extremfall mit $\tau \leq 1/N$ wird nur noch ein Individuum ausgewählt. Die benötigte Zahl an Funktionsevaluierungen steigt entsprechend.

Aber auch, wenn τ zu groß wird, dauert die Optimierung länger. In diesem Fall ist der Selektionsdruck so klein, dass sich die aufeinanderfolgenden Generationen kaum unterscheiden und so die Anzahl an Generationen steigt.

In (Mühlenbein and Mahnig, 1999b) wird ein vereinfachtes Modell für diesen Parameter hergeleitet. Näherungsweise kann man $I_\tau \propto k$ setzen mit $\tau = 2^{-k}$ (siehe Tabelle 3.1 von Seite 32). Dann liefern sowohl Gleichung (8.3) wie auch (8.4) den Zusammenhang $Gen \propto 1/\log(1/\tau)$.

Wenn stärker selektiert wird, dann wird auch die kritische Populationsgröße N^* steigen, weil dann die Wahrscheinlichkeit, dass ein richtiges Allel verloren geht, steigt. Für eine bestimmte Funktion lieferte die Analyse einer Markovkette, dass Halbierung von τ ein Wachsen von N^* um einen annähernd konstanten Faktor bewirkt (Mühlenbein and Mahnig, 1999b). Daher folgt $N^* \propto 1/\tau^\alpha$ für ein problemabhängiges α. Insgesamt ergibt sich der Zusammenhang

$$FE \propto \frac{1}{\tau^\alpha \log(1/\tau)} \tag{8.7}$$

In Abbildung 8.7 sind zwei Läufe für *OneMax* und *Kauffman* zusammen mit den Schätzkurven dargestellt. Für *OneMax* wurde $\alpha = 0.9$ gewählt, für *Kauffman* $\alpha = 0.8$. Diese Werte habe ich von Hand angepasst, um eine möglichst gute Übereinstimmung zu erhalten.

Dabei wurde zunächst für die verschiedenen τ-Werte empirisch die Populationsgröße bestimmt, bei der $\rho \approx 0.9$ war. Für diesen Wert wurde dann die Zahl der nötigen Funktionsevaluierungen ermittelt. Es zeigt sich, dass das Modell aus (8.7) die Daten in beiden Fällen recht gut beschreibt. Außerdem ergibt sich direkt eine Empfehlung für τ: Im Bereich von $[0.15, 0.5]$ ändert sich die Zahl der Evaluierungen kaum. Dies entspricht einem Intervall für I_τ von $[0.8, 1.6]$. Dies ist empirisch oft bestätigt worden.

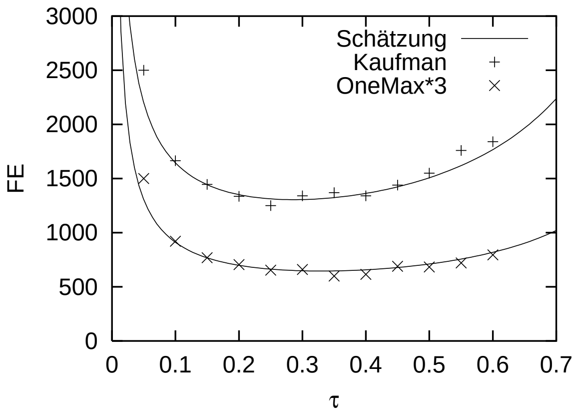

Abbildung 8.7: Abhängigkeit der Zahl an Funktions-Evaluierungen von der Selektionsrate τ mit Fit nach (8.7).

Weil die Konstante c bei der Boltzmann-Selektion dem Wert von I_τ entspricht (Abschnitt 4.4.1), ergibt sich entsprechend $[0.8, 1.6]$ als Wertebereich für diese Konstante.

8.2.5 Wahl der Mutationsrate

Die bisherigen Experimente waren alle ohne Mutation durchgeführt. Durch ein einfaches Terminierungskriterium (alle Individuen gleich) kann man so die Skalierungseigenschaften leichter bestimmen. Die Leistungsfähigkeit des Algorithmus kann durch Mutation aber teilweise deutlich gesteigert werden.

Wie in Abschnitt 3.8 erläutert, entspricht Mutation der Wahl des Bayes'schen Schätzers mit entsprechendem Parameter. Die dortige Empfehlung war es, den Parameter N' proportional zu $^N/_n$ zu wählen.

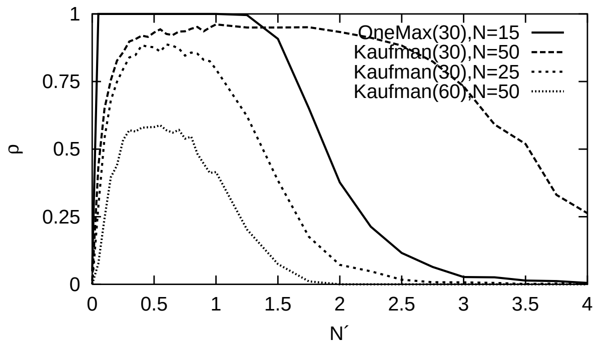

Abbildung 8.8: Auswirkung des Prior-Parameters N' auf die Erfolgsrate, Abschnitt 8.2.5

In Abbildung 8.8 wird der Einfluss des Prior-Parameters vorgestellt. Für *OneMax* mit 30 Bits wurde hier für $N' \in [0.05, 125]$ eine Erfolgsrate über 99% erzielt (in 1000 Läufen). Nach Abbildung 8.5 wird dafür ohne Prior etwa die dreifache Populationsgröße benötigt.

Für die Skalierung in N und n wurden wieder Repräsentanten der Kaufman'schen $n-k$-Funktionen gewählt. Für $n = 30$ und $N = 50$ wurde in einem Intervall $N' \in [0.5, 2.5]$ eine sehr gute Erfolgsrate gemessen (die sich ohne Prior erst bei $N \approx 300$ ergab). Halbieren der Populationsgröße genau so wie Verdoppeln der Bitlänge sollte nach der Empfehlung eine Halbierung von N' nach zu ziehen. Beides lässt sich nach der Abbildung nachvollziehen. Die absoluten Erfolgsraten sind natürlich unterschiedlich, aber der N'-Wert des Maximums hat sich ungefähr halbiert.

Leider lässt sich aus diesen Kurven und der Heuristik keine allgemein gültige Formel für N' ableiten. Er wird sicherlich auch von der Struktur der Funktion und den Funktionswerten abhängen. Mit Hilfe der Approximation

$$N' = const \cdot \frac{N}{n} \tag{8.8}$$

lässt sich aber in vielen Fällen ein günstiger Wert für die Konstante für kleines n oder N ermitteln. Damit lässt sich anschließend der Wert für N' für größere Probleme errechnen.

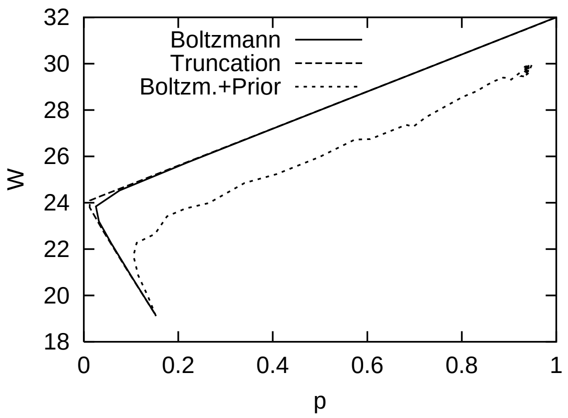

Abbildung 8.9: *FDA*-Lauf für *Deceptive*$_4(32)$ mit Startverteilung $p_i(1) = 0.15$. Die Läufe ohne Prior brauchten eine Populationsgröße von 20000 Individuen, mit Prior 1 reichte 200.

Nach dem Fokussierungssatz für den *BEDA* sollte der Algorithmus auch dann noch zum globalen Optimum finden, wenn er weit weg davon startet, solange die Wahrscheinlichkeit des Optimums nie Null wird. In Abbildung 8.9 sind drei Läufe des *FDA* für die *Deceptive*-Funktion dargestellt, bei denen in der Startgeneration $p_i(X_i\!=\!1) = 0.15$ galt. Mit proportionaler Selektion konvergiert der Algorithmus dann zum lokalen Optimum (nicht dargestellt). Aber mit einer Populationsgröße $N\!=\!20000$ wurde sowohl für Boltzmann-Selektion als auch für *Truncation Selection* das globale Optimum gefunden. Man sieht in der Kurve deutlich die Gefahr der Fixation am Rande. Mit Prior 1.0 reicht daher $N=200$ aus, um zum Optimum zu finden. Der Prior hält dabei die Verteilung von den Rändern fern, wie erwartet (siehe Abschnitt 3.8).

8.2.6 Die Anzahl der Eltern im Bayes'schen Netzwerk

In Abschnitt 4.2.7 wurde bereits auf die Problematik hingewiesen, zu komplexe Strukturen für die Faktorisierung zu verwenden. Das soll hier an einem Beispiel verdeutlicht werden. Für die *OneMax*-Funktion reicht die einfache Produktverteilung aus. Man kann aber auch mehrere (benachbarte) Variablen gruppieren und so eine Faktorisierung höherer Ordnung erhalten. Für unendliche Populationsgröße ist der Unterschied vernachlässigbar, da die Verteilungen eigentlich identisch sind. Bei endlicher fester Zahl an Individuen tritt aber der Effekt auf, dass man die Parameter der Verteilung nicht mehr so gut schätzen kann. So entspricht eine Gruppierung von je 5 Variablen einer Meta-Variablen mit 32 möglichen Werten.

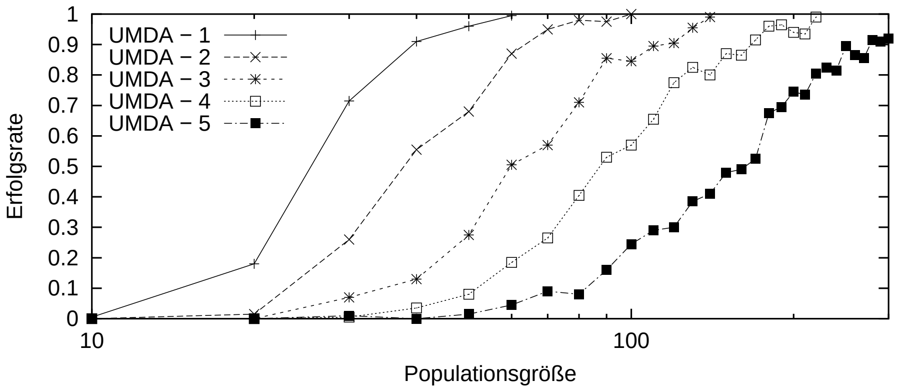

Abbildung 8.10: Erfolgsrate für unterschiedliche Gruppierungen für *OneMax*, siehe Abschnitt 8.2.6

Abbildung 8.10 demonstriert diesen Effekt. Dargestellt ist die Erfolgsrate gegen die Populationsgröße für *OneMax* (30) mit logarithmischer x-Achse, mit unterschiedlichen Faktorisierungen. So entstand die *UMDA*-3 Kurve durch

$$p(X) = p(X_1, X_2, X_3)p(X_4, X_5, X_6) \cdots p(X_{28}, X_{29}, X_{30}) \tag{8.9}$$

Man sieht, dass die nötige Populationsgröße für eine bestimmte Erfolgsrate stark ansteigt. Die einzelnen Kurven sind auf der logarithmischen Skala um einen konstanten Betrag verschoben.

Das Modell nach (8.5) liefert angepasst auf n/q Variablen und Gleichverteilung am Start

$$\rho \approx \left[1 - \left(1 - \frac{1}{2^k}\right)^N\right]^{n/k} \tag{8.10}$$

Daraus ergibt sich für festes ρ

$$N^* \approx \frac{\log\left(1-\rho^{k/n}\right)}{\log(1-2^{-k})} \tag{8.11}$$

Diese Kurve skaliert für $k \in \{1,\ldots,10\}$ praktisch linear in 2^k, der Größe des Wertebereichs. Abbildung 8.11 zeigt, dass diese Schätzung näherungsweise richtig ist: Es ergibt sich empirisch ein fast linearer Zusammenhang für diesen Parameterbereich zwischen N^* und 2^k.

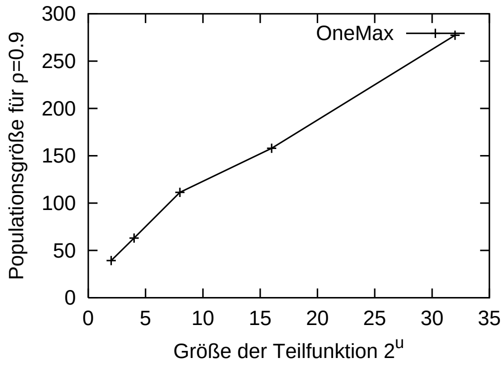

Abbildung 8.11: Zusammenhang zwischen der Blockgröße und der Erfolgsrate aus Abschnitt 8.2.6

8.2.7 Lernen

Um die Skalierung des *LFDA* zu untersuchen, betrachte die Funktion *Kauffman* für $k=1$ mit angrenzender Nachbarschaft. Es gibt zwar einen linearen Algorithmus, der in diesem Fall das Optimum findet (siehe auch Kapitel 10), dafür kann man die Leistung des Verfahrens wegen Kenntnis dieses Optimums besser einschätzen (siehe Abschnitt 6.2.5).

Empirisch zeigte sich, dass eine Populationsgröße proportional zu $n^{1.5}$ ausreicht, um das Optimum mit Wahrscheinlichkeit größer 0.9 zu finden. In Abbildung 8.12 ist der Durchschnitt über 30 Läufe dargestellt.

Da die Zahl der Funktionsevaluierungen in der Abbildung quadratisch mit n skaliert, kann man schließen, dass die Zahl der Generation mit $\sqrt{n}$ wächst wie erwartet (weil $FE = Gen \cdot N$). Nach Abschnitt 7.1.5 sollte der Zeitaufwand des Verfahrens also etwa mit $O(n^{1.5} \cdot n^{0.5} \cdot n^2) = O(n^4)$ skalieren, was durch die Abbildung auch bestätigt wird.

8.3 Zusammenfassung

Ohne Einschränkungen an die zu optimierende Funktion kann man keinen Algorithmus konstruieren, der in polynomieller Zeit läuft und das globale Optimum findet. Die hier

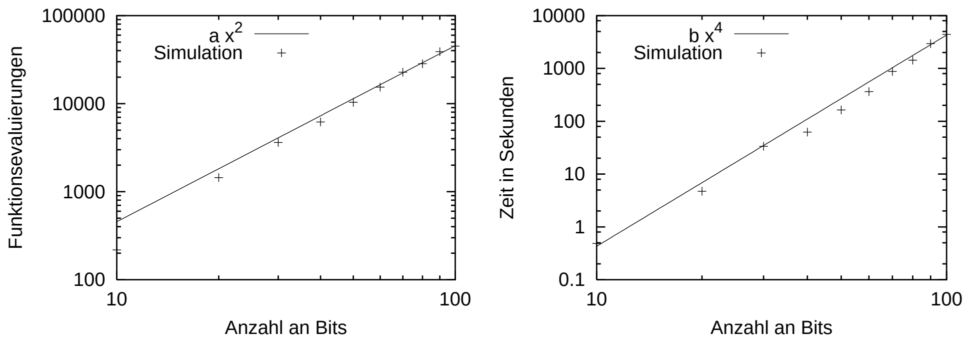

Abbildung 8.12: Populationsgröße und Zeitaufwand für den *LFDA* für *Kauffman* mit $k = 1$ und angrenzender Nachbarschaft, gemittelt über 30 Läufe.

vorgestellten Algorithmen laufen in der Praxis in polynomieller Zeit und können daher das Erreichen des globalen Optimums nicht garantieren.

Der *UMDA* hat Laufzeit $O(Gen \cdot N \cdot (n + \log N))$ (Kapitel 7). Zusammen mit der sublinearen Skalierung der Zahl der Generationen folgt ein Algorithmus, der etwa mit $n\sqrt{n}\cdot N$ skaliert. Für *separable* Probleme skaliert die benötigte Populationsgröße auch nur linear in n, so dass sich als Aufwand $O(n^{2.5})$ ergibt.

Für den *LFDA* ist eine Aussage schwieriger. Man wird dort keine separablen Probleme behandeln. Damit ist eine Abschätzung über benötigte Populationsgröße praktisch unmöglich. Im Allgemeinen wird sogar eine exponentielle Populationsgröße nötig sein.

Für viele Probleme ist die benötigte Populationsgröße aber polynomiell, etwa $N = n^{1+\alpha}$. Nach (7.8) ist der Aufwand für den *LFDA* $O\big(Gen \cdot (N \log N + n^2 \cdot N)\big)$. Dann folgt mit $Gen = O(\sqrt{n})$ zusammen der Aufwand $O(n^{\alpha+3.5})$.

Für die Selektionsintensität und den Bayes'schen Schätzer wurde vorgestellt, wie man diese Werte in der Regel wählen kann. Wenn man die Populationsgröße zu klein wählt, so nimmt die Erfolgsrate stark ab. Durch Mutation kann das zwar kompensiert werden, die Suche durch Mutation ist aber wenig zielgerichtet und führt zu einer großen Zahl an Generationen.

9 Die Graph Bipartitionierung

9.1 Problemstellung

Die Graph Bipartitionierung ist folgendes Optimierungsproblem: Gegeben sei ein ungerichteter Graph (V, E) mit einer geraden Anzahl an Knoten $|V| = n$. Gesucht ist eine Aufteilung der Knoten in zwei gleich große Mengen, so dass die „Schnittgröße" *(cut size)* minimal ist. Die Schnittgröße ist dabei definiert als die Anzahl an Kanten, die *zwischen* den beiden Mengen verläuft. Also formal:

$$\begin{aligned} \min_{A,B \subseteq V} \{cs(A,B) \mid |A| = |B|\} \qquad \text{mit} \\ cs(A,B) := \left|\left\{(v \leftrightarrow w) \in E \,\middle|\, (v \in A \wedge w \in B) \vee (v \in B \wedge w \in A)\right\}\right| \end{aligned} \tag{9.1}$$

(Zur Erinnerung: Ein ungerichteter Graph besteht nach Abschnitt 5.3.1 aus gerichteten Kanten, wobei mit $(v_1 \to v_2)$ auch immer $(v_2 \to v_1)$ gilt. $(v \leftrightarrow w) \in E$ bezeichnet dabei $(v \to w) \in E \wedge (w \to v) \in E$). Erweiterungen dieser Problemklasse sind die Partitionierung in mehr als zwei Klassen sowie die Einbeziehung von Gewichten an den Kanten. Mehr als zwei Klassen werden in der Regel so behandelt, dass rekursiv Graph Bipartitionierung angewendet wird. Für Gewichte muss nur eine einfache Ergänzung der Algorithmen vorgenommen werden. Es wird daher nur das so definierte Graph-Bipartitionierungs-Problem behandelt. Man findet es auch unter dem Namen *Graph Bisection*.

Es taucht in vielen praktischen Anwendungen auf wie Netzwerk-Aufteilung, Flächenplanung, VLSI Schaltkreis-Entwurf etc.

9.1.1 Approximationsergebnisse

Die exakte Bestimmung der minimalen Schnittgröße ist NP-hart (Garey et al., 1976). Für eine Reihe von NP-schwierigen Problemen gibt es Approximations-Algorithmen, die in polynomieller Zeit eine Lösung finden, die sich von dem globalen Optimum c^* um höchstens einen Faktor $f \geq 1$ unterscheiden. Das Problem besitzt einen *PTAS (polynomial time approximation scheme)*, wenn es für jedes feste $f > 1$ einen Algorithmus mit Approximationsfaktor f gibt.

Nach (Bui and Jones, 1992) ist die Suche nach Lösungen, die sich um die *Differenz* $n^{2-\varepsilon}$ unterscheiden, ebenfalls NP-hart. Dies schließt allerdings nicht aus, dass es einen *PTAS* gibt. Zur Zeit ist noch keiner bekannt (Feige and Krauthgamer, 2000).

In (Feige and Krauthgamer, 2000) stellen die Autoren einen Algorithmus vor, der in polynomieller Zeit eine Lösung findet, die Schnittgröße $\leq c^* \cdot (\log n)^2$ hat.

Weiterhin wird in (Arora et al., 1999) gezeigt, dass es für *überall dichte* Graphen einen *PTAS* gibt. Das sind Graphen, die überall Grad $\approx n$ haben.

Die Benchmark-Probleme bestehen alle aus dünn besetzten Graphen, für die es (noch) keinen *PTAS* gibt. Es gibt auch noch keine Veröffentlichung, die einen polynomiellen Algorithmus auf eines dieser Benchmark-Probleme anwendet.

9.1.2 Der Kernighan-Lin-Algorithmus

Der Kernighan-Lin-Algorithmus (Kernighan and Lin, 1970) ist eine effiziente Heuristik, um das Problem der Graph Bipartitionierung zu lösen. KL verwendet mehrere Durchläufe. In jedem Durchlauf wird die bisherige Lösung verbessert, indem paarweise Ecken vertauscht werden, um eine neue Lösung zu erhalten. Dies wird so lange iteriert, bis ein Durchlauf keine Verbesserung mehr bringt.

Es sei (A, B) eine Bipartitionierung von $G = (V, E)$, also $A \cup B = V$, A und B disjunkt und $|A| = n/2$. Der Zuwachs g_v von Knoten v ist die Reduktion der Schnittgröße, wenn v zur anderen Menge wechselt.

$$v \in A: \qquad g_v := \left|\left\{w \in B \middle| (v \leftrightarrow w) \in E\right\}\right| - \left|\left\{w \in A \middle| (v \leftrightarrow w) \in E\right\}\right| \tag{9.2}$$

und analog für $v \in B$. Der Gewinn $g(a, b)$ für $a \in A$ und $b \in B$ ist definiert als der Zuwachs in der Fitness nach Vertauschen von a und b:

$$g(a,b) = g_a + g_b - 2\delta(a,b) \qquad \text{mit } \delta(a,b) = \begin{cases} 1 & (a \leftrightarrow b) \in E \\ 0 & \text{sonst} \end{cases} \tag{9.3}$$

Beispiel 9.1. Betrachte Abbildung 9.1 als Beispiel. Es gilt vor der Vertauschung $g_{a_1} = 1$, $g_{a_3} = -1$ und die anderen Gewinne sind 0. Damit ist der maximale Vertauschungsgewinn $g(a_1, b_2) = +1$. Nach der Vertauschung gilt $g_{a_3} = +1$, $g_{b_1} = -2$ und $g_{b_3} = -2$. Knoten, die nicht mit a_1 oder b_2 verbunden waren, behalten also ihren Zuwachswert, für die anderen verändert er sich um $+2$ und/oder -2.

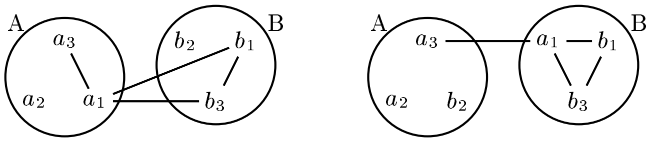

Abbildung 9.1: Beispiel für die Graph Bipartitionierung: Austausch von a_1 und b_2

Zu Beginn jedes Durchlaufs sind alle Knoten *frei*, das heißt, sie können bewegt werden. Nachdem ein Knoten bewegt wurde ist er für den Rest des Durchlaufs *fest*. KL vertauscht nun iterativ die beiden Knoten mit höchstem Gewinn. Dabei darf die Schnittgröße durchaus auch ansteigen. Dies wird solange durchgeführt, bis kein Knoten mehr frei ist. Der Trick ist nun, dass das Ergebnis des Durchlaufs die beste in diesem Lauf gefundene Lösung ist. Die Heuristik nimmt also mehrere die Fitness verschlechternde Vertauschungen in Kauf, wenn am Ende dadurch die Schnittgröße doch noch kleiner wird. Effektiv wird dadurch ein kleiner Teil der *n-opt*-Nachbarschaft gesucht.

Algorithmus 9.1 beschreibt den Vorgang genauer.

Algorithmus 9.1: Kernighan-Lin

```
Bestimme (zufällig) eine Aufteilung A, B mit |A| = |B| = n/2.
do {
    Berechne g_a, g_b für alle a ∈ A,b ∈ B.
    Q_A ⇐ A, Q_B ⇐ B
    for i ⇐ 1 to |V|/2 do {
        Wähle a_i ∈ Q_A und b_i ∈ Q_B mit g_i ⇐ g(a_i, b_i) maximal
        Q_A ⇐ Q_A \ {a_i}; Q_B ⇐ Q_B \ {b_i}
        Für alle a ∈ Q_A: g_a ⇐ g_a + 2δ(a, a_i) − 2δ(a, b_i)
        Für alle b ∈ Q_B: g_b ⇐ g_b + 2δ(b, b_i) − 2δ(b, a_i)
    }
    Suche k ∈ {1, . . . , |V|/2}, so dass ∑_{i=1}^{k} g̃_i maximal.
    Vertausche die Teilmengen {a_1, . . . , a_k} und {b_1, . . . , b_k} in A, B.
} while (etwas vertauscht wurde)
```

Eine einfache Implementierung des Algorithmus benötigt Aufwand $O(|V|^3)$ pro Durchlauf, weil der Aufwand zur Bestimmung des besten Paares (a, b) Aufwand $O(|V|^2)$ hat. Weil höchstens $|E|$ Durchläufe notwendig sind, ist die Komplexität insgesamt $O(|E|\cdot|V|^3)$. In der Praxis zeigt sich aber, dass der Algorithmus fast immer erheblich weniger Durchläufe benötigt.

In (Fiduccia and Mattheyses, 1982) wurde ein ähnliches Verfahren vorgestellt, welches den Aufwand pro Durchlauf auf $O(|E|)$ reduziert. Ein konzeptioneller Unterschied ist der, dass nun in jedem Schritt ein einzelner Knoten die Seite wechselt und nicht mehr ein Paar vertauscht wird. Dadurch kann in Zwischenschritten die Bedingung $|A|{=}n/2$ auch verletzt sein. Im nächsten Schritt wird dann aber wieder ein Element aus der anderen Menge ausgewählt, so dass in jedem zweiten Schritt die Nebenbedingung erfüllt ist.

Der Geschwindigkeitszuwachs ist möglich durch eine zusätzliche Datenstruktur, die es ermöglicht, in konstanter Zeit den Knoten mit maximalem Gewinn zu ermitteln. Dazu wird für jeden möglichen Gewinn eine verkette Liste von entsprechenden Knoten gespeichert. Diese Liste lässt sich in Zeit $O(|E|)$ aktualisieren. Die Details sind in (Fiduccia and Mattheyses, 1982) erläutert.

Dadurch, dass nun einzelne Knoten betrachtet werden und nicht mehr immer nur Paare getauscht werden, ist es möglich, mit ungültigen Aufteilungen zu starten. Dazu wird am Anfang immer nur aus einer Menge gezogen, bis das Gleichgewicht hergestellt ist. Man muss nun nur noch aufpassen, dass am Ende nur solche Zugfolgen berücksichtigt werden, die die Teilmengen gleich groß lassen. Algorithmus 9.2 erläutert das Verfahren.

Algorithmus 9.2: Kernighan-Lin mit Einzeltausch

Start mit einer beliebigen Aufteilung A, B.
do {
    $Q_A \Leftarrow A$, $Q_B \Leftarrow B$, $i \Leftarrow 1$, initialisiere Gewinnlisten g_c.
    do {
        if $|Q_A| > |Q_B|$: $M_i \Leftarrow A$, wähle $c_i \in Q_A$ mit max. g_{c_i}
        if $|Q_A| < |Q_B|$: $M_i \Leftarrow B$, wähle $c_i \in Q_B$ mit max. g_{c_i}
        if $|Q_A| = |Q_B|$: Wähle c_i aus $Q_A \cup Q_B$ mit max. g_{c_i}, setze M_i entsprechend auf A oder B
        $Q_{M_i} \Leftarrow Q_{M_i} \setminus \{c_i\}$; $i \Leftarrow i+1$; $\tilde{g}_i \Leftarrow g_{c_i}$; aktualisiere Gewinnlisten.
        if $|Q_A| = |Q_B|$: $f_i \Leftarrow 1$, **else**: $f_i \Leftarrow 0$
    } **while** $(Q_A \cup Q_B \neq \emptyset)$
    Suche $k \in \{1, \ldots, {}^{|V|}\!/_2\}$, so dass $f_k = 1$ und $\sum_{i=1}^{k} \tilde{g}_i$ maximal.
    Setze die Elemente $\{c_1, \ldots, c_k\}$ in die jeweils andere Menge.
} **while** (etwas vertauscht wurde)

9.1.3 Verknüpfung mit dem *LFDA*

Es gibt eine einfache Zuordnung von binären Vektoren zu Lösungen für das Problem der Graph Bipartitionierung: Der Wert des Bits Nummer i gibt an, in welcher Teilmenge sich der Knoten i befindet. Ein Graph mit n Knoten kann also durch ein Individuum mit n Bits dargestellt werden. Dabei gibt es zwei Probleme:

- Die allermeisten Bitketten sind nicht *gültig*, dafür müssten ja exakt ${}^{n}\!/_2$ Bits den Wert Null haben.
- Die Fitness bleibt gleich, wenn alle Bits invertiert werden. Nach Korollar 3.15 erschwert das den Start für Populationsverfahren.

Da das Kernighan-Lin-Verfahren bereits sehr gute Ergebnisse liefert, soll es nun außerdem mit dem *LFDA* verknüpft werden. Algorithmus 9.2 erlaubt es dann auch, den ersten Punkt zu behandeln.

Für das zweite Problem benutze folgendes Verfahren: Die jeweils beste Lösung in der Population definiert die Referenzmenge. Die grundlegende Idee ist es, dass die anderen Individuen als Varianten davon aufgefasst werden. Betrachte Abbildung 9.2. Bei dem

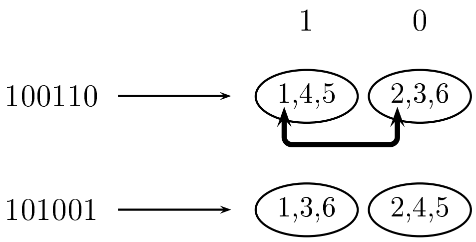

Abbildung 9.2: Zuordnung der binären Vektoren zu Partitionierungen

ersten Vektor sind die Bits 1, 4 und 5 auf Eins und die anderen auf Null. Bei dem zweiten Vektor sind die Bits 1, 3 und 6 auf Eins. Auf den ersten Blick unterscheiden sich die beiden Zeichenketten sehr stark. Auf der Ebene der Aufteilungen betrachtet sieht man aber, dass nur die Elemente 1 und 2 vertauscht wurden und der „Name“ der Menge. Es ist nun aber unerheblich, ob die eine Teilmenge mit 0-Bits oder 1-Bits bezeichnet ist. Wenn der zweite Bitvektor invertiert wird, ergibt sich 010110 und man sieht die Ähnlichkeit auch hier deutlich.

Ziel ist es also, bezogen auf eine Referenzlösung die Bits einer anderen Lösung zu invertieren, wenn sich dadurch die Ähnlichkeit vergrößert. Ein naheliegendes Maß für diese Ähnlichkeit ist die Anzahl an übereinstimmenden Bits:

$$\hat{\delta}(x, y) = |\{i \mid x_i \neq y_i\}| \tag{9.4}$$

Wenn $\neg y$ das Bitinverse bezeichnet, so gilt $\hat{\delta}(x, y) + \hat{\delta}(x, \neg y) = n$. Damit folgt

$$\hat{\delta}(x, \neg y) < \hat{\delta}(x, y) \quad \Leftrightarrow \quad \hat{\delta}(x, y) > {}^{n}\!/_{2} \tag{9.5}$$

In diesem Fall sollte also der invertierte Vektor betrachtet werden.

Algorithmus 9.3: *UMDA* für Graph Bipartitionierung

Erzeuge eine zufällige Ausgangspopulation mit N Individuen. $t \Leftarrow 0$.
do {
  Wende Algorithmus 9.2 auf jedes Individuum an.
  Selektiere $\hat{N} \leq N$ Punkte. Sei $\hat{\mathfrak{X}}^*$ das beste Individuum.
  Für alle $\hat{\mathfrak{X}}^i$: Wenn $\hat{\delta}(\hat{\mathfrak{X}}^*, \hat{\mathfrak{X}}^i) > {}^{n}\!/_{2}$: $\hat{\mathfrak{X}}^i \Leftarrow \neg\hat{\mathfrak{X}}^i$.
  Berechne Bithäufigkeiten $p_i(x_i, t)$ aus den selektierten Punkten.
  Erzeuge neue Punkte gemäß $p(x, t+1) = \prod_{i=1}^{n} p_i(x_i, t)$.
  $t \Leftarrow t + 1$.
} **until** (Abbruchkriterium erreicht)

Zusammen mit dem *UMDA* entsteht also Algorithmus 9.3.

Die Graphstruktur des Problems legt es nahe, die Vorzüge von faktorisierten Verteilungen für dieses Problem zu testen. Der *FDA* scheidet dabei aus, weil eine gültige Faktorisierung anhand der Problemdefinition kaum zu berechnen ist und die Elternmenge dabei im Allgemeinen auch exponentiell ansteigen wird.

Für den *LFDA* kann man dabei noch folgende Modifikation betrachten. Die Berechnung der Schnittgröße (9.1) in der Darstellung der Bitvektoren erfolgt folgendermassen:

$$cs(x) = \sum_{(i \leftrightarrow j) \in E} \delta_{x_i \neq x_j} \tag{9.6}$$

Es handelt sich also um eine additiv zerlegte Funktion. Es folgt daraus, dass Variablen, die nicht durch eine Kante in E verbunden sind, in dem Bayes'schen Netz zur Optimierung auch nicht verbunden sein müssen (siehe Anmerkung 5.21).

Algorithmus 9.3 lässt sich also leicht umwandeln in einen Algorithmus, der *LFDA* statt *UMDA* verwendet. Es wird lediglich beim Lernen des Netzwerkes die Liste an zulässigen Knoten mit E anstatt dem vollen Netz initialisiert. Damit ergibt sich Algorithmus 9.4.

Algorithmus 9.4: *LFDA* für Graph Bipartitionierung

Es sei E^* die Kanten des Graphen in dem Problem. Erzeuge eine zufällige Ausgangspopulation mit N Individuen. $t \Leftarrow 0$.
do {
  Wende Algorithmus 9.2 (Kernighan-Lin mit Einzeltausch) auf jedes Individuum an.
  Selektiere $\hat{N} \leq N$ Punkte. Sei $\hat{\mathfrak{X}}^*$ das beste Individuum.
  Für alle $\hat{\mathfrak{X}}^i$: Wenn $\hat{\delta}(\hat{\mathfrak{X}}^*, \hat{\mathfrak{X}}^i) > n/2$: $\hat{\mathfrak{X}}^i \Leftarrow \neg\hat{\mathfrak{X}}^i$.
  $F \Leftarrow \{(X_i \to X_j) \big| (X_i \leftrightarrow X_j) \in E^*\}$, zulässige Kanten müssen im Originalgraph auch vorhanden sein.
  $E \Leftarrow \emptyset$.
  do {
    Wähle $(X_i \to X_j) \in F$ aus, so dass MDL_α maximal verkleinert wird.
    $E \Leftarrow E \cup (X_i \to X_j)$.
    Entferne $(X_i \to X_j)$ und $(X_j \to X_i)$ aus F sowie alle Kanten, die einen Zyklus oder zu viele Eltern bewirken würden.
  } **while** (es noch eine Kante in F gibt, die MDL_α verkleinert)
  Bestimme eine Faktorisierung aus dem Graphen (Algorithmus 5.1).
  Berechne die konditionalen Wahrscheinlichkeiten $p(x_{b_i}|x_{c_i}, t)$ aus den selektierten Punkten.
  Erzeuge neue Punkte gemäß $p(x, t+1) = \prod_{i=1}^{l} p(x_{b_i}|x_{c_i}, t)$.
  $t \Leftarrow t + 1$.
} **until** (Abbruchkriterium erreicht)

9.2 Benchmark-Ergebnisse

Es gibt in der Literatur eine Reihe von Benchmark-Problemen zur Bipartitionierung. Es gibt auch mehrere Ansätze, diese mit genetischen Algorithmen zu lösen.

In (Merz and Freisleben, 2000) haben die Autoren die bislang besten Resultate zur Bipartitionierung veröffentlicht. Sie vergleichen ihr Verfahren, welches ähnlich wie in dieser Arbeit einen genetischen Algorithmus mit dem Kernighan-Lin-Verfahren verknüpft, mit verschiedenen anderen Algorithmen, unter anderem Multi-Start und iteriertem Kernighan-Lin. Es stellte sich dabei heraus, dass für viele gängige Benchmark-Probleme die Verwendung des genetischen Algorithmus nicht nötig ist, weil bereits ein iteriertes Kernighan-Lin das Optimum findet.

Am schwierigsten war die Klasse der gleichmäßigen zufälligen Graphen G$n.p$. Dabei wurde bei gegebenem n und p ein zufälliger Graph generiert, der n Knoten hat und bei dem Kanten mit Wahrscheinlichkeit p auftauchen. Für verschiedene Werte von n und p wurde dabei eine Instanz generiert und als Benchmark zum herunterladen zur Verfügung gestellt (`ftp://dimacs.rutgers.edu/pub/dsj/partition/`).

Algo.	N	Avg	σ	*FE*	Mx	Algo.	N	Avg	σ	*FE*	Mx
	G500.005 (49)						**G500.01** (218)				
DG		52.0	0.26	15k	0	*DG*		219.3	0.84	9.3k	5
IKL		55.8	2.11	26k	0	*IKL*		229.7	5.21	19k	0
MA-GX	40	**49.1**	0.37	50k	29	*MA-GX*	40	218.1	0.51	38k	28
M50	40	50.9	0.40	28k	1	*M200*	40	**218.0**	0.00	19k	30
Kim	50	50.41		26k		*Kim*	50	218.04		29k	
UMDA	40	**49.43**	0.57	34k	18	*UMDA*	100	**218.0**	0.0	1.8k	30
	G500.02 (626)						**G500.04** (1744)				
DG		627.8	1.45	5.7k	8	*DG*		1747.1	2.12	2.8k	3
IKL		638.8	4.26	13k	0	*IKL*		1763.8	8.67	7.0k	0
MA-GX	40	627.5	1.14	25k	6	*MA-GX*	40	1745.4	1.50	13k	15
M50	40	626.7	0.71	14k	13	*M50*	40	1745.3	1.54	7.8k	12
Kim	50	626.85		29k		*Kim*	50	1745.59		32k	
UMDA	40	**626.03**	0.18	5.5k	29	*UMDA*	40	**1744.0**	0.00	1.6k	30

Tabelle 9.1: Resultate für Graphen mit 500 Variablen. N ist die Populationsgröße, Avg die durchschnittlich erzielte Schnittgröße, σ davon die Standardabweichung, *FE* die benötigten Evaluierungen und Mx, wie oft das Maximum in den 30 Läufen gefunden wurde.

Die verglichenen Verfahren in den Tabellen 9.1 und 9.2 sind ein *Diff-Greedy* und ein *iterated* Kernighan-Lin (DG bzw. IKL), sowie zwei verschiedene *memetische* Verfahren (*MA-GX* und *Mx*), alle Resultate aus (Merz and Freisleben, 2000), daneben Resultate aus (Kim and Moon, 2000). In der Arbeit von Merz und Freisleben werden insgesamt 10 verschiedene memetische Algorithmen vorgestellt. Vier davon waren Verfahren mit starker Mutation, nämlich *M50* bis *M200*. Die Zahl stellt dabei einen Parameter des Verfahrens dar, die Mutationsstärke. Von diesen vier wurde jeweils das beste Verfahren

Algo.	N	Avg	σ	*FE*	Mx	Algo.	N	Avg	σ	*FE*	Mx
	G1000.0025 (93)						**G1000.05** (445)				
DG		101.4	1.45	$11k$	0	*DG*		459.9	2.23	$7.1k$	0
IKL		99.5	2.87	$10k$	0	*IKL*		452.9	4.09	$7.3k$	0
MA-GX	40	**94.5**	1.33	$41k$	10	*MA-GX*	40	**447.7**	0.99	$32k$	2
M100	40	96.3	1.03	$24k$	0	*M150*	40	448.9	1.48	$14k$	0
Kim	50	96.23		$28k$		*Kim*	50	449.49		$35k$	
UMDA	40	**95.3**	0.75	$40k$	1	*UMDA*	40	449.07	1.26	$30k$	0
LFDA	100	**94.83**	0.70	$41k$	1	*LFDA*	100	**447.83**	1.53	$29k$	4
	G1000.01 (1362)						**G1000.02** (3382)				
DG		1378.1	2.62	$3.9k$	0	*DG*		3397.5	5.17	$2.0k$	0
IKL		1370.8	4.66	$4.2k$	2	*IKL*		3399.5	14.73	$2.3k$	1
MA-GX	40	**1363.1**	1.04	$21k$	9	*MA-GX*	40	**3384.0**	0.49	$11k$	0
M150	40	1364.6	2.75	$8.0k$	7	*M150*	40	**3383.2**	0.81	$4.4k$	6
Kim	50	1364.42		$42k$		*Kim*	50	3384.49		$40k$	
UMDA	40	1363.8	1.03	$19k$	1	*UMDA*	100	**3384.1**	0.80	$8.0k$	2
LFDA	100	**1362.7**	0.70	$21k$	13	*LFDA*	100	**3383.57**	0.90	$7.8k$	5

Tabelle 9.2: Resultate für Graphen mit 1000 Variablen, Spaltenbezeichnungen wie in Tabelle 9.1.

ausgewählt. Das jeweils schlechteste war bei fast allen Funktionen deutlich schlechter, die Mutationsstärke ist also ein kritischer Parameter. Ansonsten war MA-GX das beste Verfahren von den dort vorgestellten. *Kim* ist das Verfahren aus (Kim and Moon, 2000). Die Standardabweichung ist dort nicht angegeben.

Dargestellt in den Tabellen sind neben der besten bekannten Lösung in Klammern hinter der Problembezeichnung die Populationsgrößen N für die populationsbasierten Verfahren, die durchschnittliche erzielte Schnittgröße in 30 Läufen (die minimiert werden soll), davon die Standardabweichung über die 30 Läufe, die durchschnittliche Zahl an Funktionsevaluierungen und wie oft das beste bekannte Lösung gefunden wurde. Fettgedruckt sind die Fitness-Werte des jeweils besten pro Problem und die Algorithmen, die sich vom Besten nur um eine Standardabweichung unterscheiden.

Stochastische Algorithmen können ihre Leistung oft dadurch verbessern, in dem sie länger laufen. Um eine Vergleichbarkeit bei sehr unähnlichen Verfahren zu haben, wird hierzu oft die Laufzeit in Sekunden hinzugezogen. Die Algorithmen in (Merz and Freisleben, 2000) hatten dabei 60 Sekunden Zeit für die Optimierung der Probleme mit 500 Bits und 120 Sekunden für die mit 1000 Bits. Dabei stellte sich heraus, dass das *Diff-Greedy* und *iterated* Kernighan-Lin für diese Probleme deutlich schlechter abschnitten bei gleicher Rechenzeit.

Da die Rechenzeiten aber natürlich auch von der verwendeten Hardware abhängt, habe ich zum Vergleich mit dem *UMDA* und dem *LFDA* ein anderes Kriterium verwandt, nämlich die Zahl der Funktionsevaluierungen. Eine Evaluierung entspricht dabei einem kompletten Lauf des Kernighan-Lin-Verfahrens. Dies ist fair, weil die beiden memetischen Algorithmen aus (Merz and Freisleben, 2000) und der Algorithmus aus (Kim and Moon,

2000) nach dem gleichen Prinzip arbeiten, sie verwenden lediglich eine andere Methode der Rekombination. Zum Vergleich: Der *UMDA* benötigte für die 500-Bit-Probleme 100-200 Sekunden, für die 1000-Bit-Probleme etwa 500 Sekunden und der *LFDA* für die 1000-Bit-Probleme etwa 1000 Sekunden. *UMDA* und *LFDA* wurden abgebrochen, wenn sie eine bestimmte Zahl an Funktionsevaluierungen ausgeschöpft hatten. Sie bekamen nie mehr Evaluierungen als *MA-GX*.

Obwohl die einfachen iterierten Verfahren deutlich schlechter abschneiden als die memetischen Verfahren und *FDA* sowie *LFDA*, sind die Probleme insgesamt doch zu einfach, um einen großen Unterschied erkennen zu können.

Bei den Problemen mit 500 Bits erreichen alle populationsbasierten Verfahren vergleichbare Resultate. *UMDA* ist nur einmal schlechter als eines der anderen Verfahren, aber immer noch innerhalb der Standardabweichung. In zwei Fällen liefert *UMDA* bessere Resultate bei gleicher Zahl an Funktionsevaluierungen. Der *LFDA* ist für diese einfachen Probleme nicht notwendig.

Die Probleme mit 1000 Bits sind deutlich schwieriger. Das Verfahren von (Kim and Moon, 2000) fällt deutlich ab, und auch die beiden einfachen Suchalgorithmen werden schlechter. *LFDA* und *MA-GX* erzielen die besten Resultate und unterscheiden sich nicht wesentlich. Aber auch *UMDA* ist in zwei Fällen noch nahe an dem besten Verfahren.

Der *LFDA* hat durchgehend eine sehr gute Leistung bei erhöhtem rechnerischem Aufwand. Merz und Freisleben haben die Bedeutung der Interaktion in ihrer Arbeit bereits erkannt: „*Gene interaction in a given representation can be expressed by a dependency graph … We think that the structure of the dependency graph may have a large impact on the fitness landscape.*“ (Merz and Freisleben, 2000). Sie nutzen diese Eigenschaft in ihren Algorithmen allerdings nicht aus.

Sowohl der *UMDA* als auch der *LFDA* wurden nicht weiter an die Problemstellung angepasst, nur bezüglich der lokalen Optimierung, wie es auch bei den anderen Autoren geschah. Die Algorithmen von (Merz and Freisleben, 2000) und (Kim and Moon, 2000) sind hingegen speziell für dieses Problem geschrieben worden.

Auch die Leistungsfähigkeit des *UMDA* in Verbindung mit der lokalen Optimierung durch Kernighan-Lin ist verblüffend.

10 Andere Ansätze

10.1 Mean Field Approximation

Bei dem *UMDA* werden die Parameter der Verteilung klassisch aus den Zählwerten geschätzt. Die so erhaltene Verteilung maximiert die *Likelihood* über alle möglichen Produktverteilungen. Ein anderer Ansatz ist es, an dieser Stelle ein anderes Maß zu verwenden, welches (hoffentlich) die Kopplung der Variablen berücksichtigt.

Wähle also als approximative Verteilung die parametrische einfache Produktverteilung aus Definition 3.4 von Seite 16:

$$\tilde{p}^u(x,q) = q(x) = \prod_{i=1}^{n} q_i(x_i) \tag{10.1}$$

wobei die $q_{i,x_i} = q_i(x_i)$ für $x_i \neq 0$ die Parameter darstellen. Ziel ist es, die Boltzmann-Verteilung möglichst gut durch ein Mitglied dieser Familie zu approximieren. Als Maß wurde nun die Kullback-Leibler-Divergenz (Definition A.9) verwendet, diesmal aber mit dem natürlichen Logarithmus.

Weil

$$\sum_x q(x) \ln q(x) = \sum_x \prod_{i=1}^{n} q_i(x_i) \sum_{j=1}^{n} \ln q_j(x_j) \tag{10.2}$$

$$= \sum_{x_1} \sum_{x_2,\ldots,x_n} q_1(x_1) \prod_{i=2}^{n} q_i(x_i) \left[\ln q_1(x_1) + \sum_{j=2}^{n} \ln q_j(x_j)\right] \tag{10.3}$$

$$= \sum_{x_1} q_1(x_1) \ln q_1(x_1) \underbrace{\sum_{x_2,\ldots,x_n} \prod_{j=2}^{n} q_j(x_j)}_{=1} \tag{10.4}$$

$$+ \underbrace{\sum_{x_1} q_1(x_1)}_{=1} \sum_{x_2,\ldots,x_n} \prod_{i=2}^{n} q_i(x_i) \sum_{j=2}^{n} \ln q_j(x_j) \tag{10.5}$$

$$= \ldots = \sum_{i=1}^{n} \sum_{x_i=0}^{l_i} q_i(x_i) \ln(q_i(x_i)) \tag{10.6}$$

gilt, ist die resultierende Kullback-Leibler-Divergenz

$$D^{KL}(q\|p) = \sum_x q(x)\big[\ln q(x) - \beta f(x) + \ln Z_\beta\big] \tag{10.7}$$

$$= \ln Z_\beta + \sum_x q(x)\ln q(x) - \beta \sum_x q(x) f(x) \tag{10.8}$$

$$= \ln Z_\beta + \sum_{i=1}^{n}\sum_{x_i=0}^{l_i} q_i(x_i)\ln(q_i(x_i)) - \beta\cdot W_f^u(q_1,\dots,q_n) \tag{10.9}$$

Weil $q_i(0) = 1 - \sum_{x_i=1}^{l_i} q_{i,x_i}$, folgt

$$\frac{\partial \mathrm{KLD}}{\partial q_{i,x_i}} = \ln \frac{q_{i,x_i}}{1-\sum_{y_i=1}^{l_i} q_{i,y_i}} - \beta\big(\bar{f}_i(x_i) - \bar{f}_i(0)\big) \tag{10.10}$$

Ein (lokales) Minimum der Kullback-Leibler-Divergenz kann man ermitteln, in dem man die Ableitungen gleich Null setzt. Dann gilt:

$$\tilde{q}_{i,x_i} = \frac{1-\sum_{y_i=1,y_i\neq x_i}^{l_i} q_{i,y_i}}{1+e^{-\beta[\bar{f}_i(1)-\bar{f}_i(0)]}} = \frac{1-\sum_{y_i=1,y_i\neq x_i}^{l_i} q_{i,y_i}}{1+\exp\left(-\beta\frac{\partial W_f^u}{\partial q_i}\right)} \tag{10.11}$$

Dieses Gleichungssystem sind die sogenannten *Mean-Field-Gleichungen* (Jordan et al., 1999). Auf der rechten Seite der Gleichung kommt die Variable q_{i,x_i} nicht vor. Eine analytische Lösung ist nur für Spezialfälle möglich. In der Regel wird dieses System iterativ gelöst und liefert dann ein *lokales* Minimum der Kullback-Leibler-Divergenz.

Die Gleichung 10.9 lässt sich auch nutzen, um die Parameter der Verteilung über *Reinforcement Learning* zu optimieren (Berny, 2000).

10.1.1 Bestimmung der $\bar{f}_i$ für binäre additiv zerlegte Funktionen

Wenn es eine Darstellung von $f(x)$ als $f(x) = \sum_{s\subseteq I} a_s x^s$ (Lemma 3.18) gibt, so lassen sich die Terme $\bar{f}_i(x_i)$ auch direkt ausrechnen. Es gilt

$$f(1,x_2,\dots,x_n) = \sum_{s\subseteq I\setminus\{1\}} (a_s + a_{s\cup\{1\}}) \prod_{j=2}^{n} x_j^{\hat{s}_j} \tag{10.12}$$

$$f(0,x_2,\dots,x_n) = \sum_{s\subseteq I\setminus\{1\}} a_s \prod_{j=2}^{n} x_j^{\hat{s}_j} \tag{10.13}$$

Weil $\bar{f}_1(x_1)$ einfach die durchschnittliche Fitness von $f(x_1,\cdot)$ bezogen auf die einfache Produktverteilung ist, lässt sich der Darstellungssatz für die durchschnittliche Fitness anwenden (Korollar 3.24) und es ergibt sich

$$\bar{f}_i(1) - \bar{f}_i(0) = \sum_{s\subseteq I\setminus\{i\}} a_{s\cup\{i\}} \prod_{j\in s} p_j \tag{10.14}$$

Lineare Fitness

Für lineare Fitness $f(x) = a_0 + \sum_{i=1}^{n} a_i x_i$ sind in (10.14) alle Koeffizienten Null außer für $s=\emptyset$. Die *Mean-Field*-Gleichungen reduzieren sich dann auf

$$\tilde{q}_i = \frac{1}{1 + e^{-\beta a_i}} = p_i(1) \tag{10.15}$$

In diesem Spezialfall ist die rechte Seite also konstant und es gibt eine eindeutige Lösung des Systems (10.11). Die Lösung ist identisch zu den Randverteilungen der Boltzmann-Verteilung. Weil sich die Boltzmann-Verteilung exakt durch eine einfache Produktverteilung darstellen lässt, ist die Kullback-Leibler-Divergenz Null: Die echte und die angenäherte Verteilung sind gleich.

Quadratische Fitness

Sei $A = (a_{ij})$ eine symmetrische Matrix, betrachte $f(x) = \sum_i a_{ii} x_i + \frac{1}{2} \sum_{i \neq j} a_{ij} x_i x_j$. Es gilt:

$$\tilde{q}_i = \frac{1}{1 + \exp[-\beta(a_{ii} + \sum_{j \neq i} a_{ij} \cdot \tilde{q}_j)]} \tag{10.16}$$

Schon in diesem einfachen Falle kann man keine analytische Lösung mehr bestimmen und muß diese Gleichung vielmehr iteriert lösen.

Deceptive

Betrachte die *Deceptive*-Funktion mit 4 Variablen von Abschnitt 3.7.3. Aus Symmetriegründen setze alle Parameter gleich. Das Gleichungssystem vereinfacht sich dann zu

$$\tilde{q} = \frac{1}{1 + e^{\beta(1-5\tilde{q}^3)}} =: T(\tilde{q}) \tag{10.17}$$

In Abbildung 10.1 ist die Kullback-Leibler-Divergenz für diese Funktion für verschiedene Werte von q aufgetragen. Gesucht ist das (globale) Minimum dieser Funktion.

Die drei Kurven repräsentieren drei verschieden Stadien der Entwicklung für verschiedene Werte von β. Die entsprechenden Werte für β wurden nummerisch bestimmt.

- $\beta < 0.827$: Es gibt genau ein Minimum der Divergenz. Für $\beta = 0$ liegt es bei $\tilde{q} = 0.5$, dann ist die Kullback-Leibler-Divergenz (bis auf eine Konstante) einfach die Entropie der einfachen Produktverteilung. Für größere β verschiebt sich dieses Minimum zur Null hin.
- $0.827 < \beta < 1.11$: Nahe der Eins entsteht ein weiteres lokales Minimum, welches aber noch höhere Divergenz liefert als das (globale) Minimum bei Null.

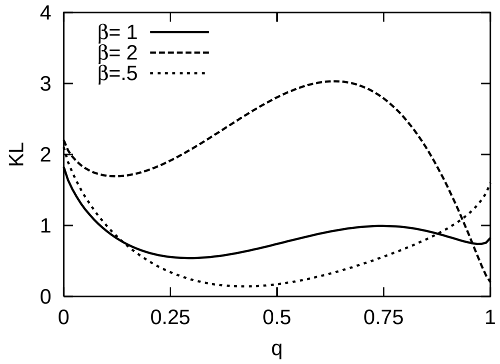

Abbildung 10.1: Kullback-Leibler-Divergenz für *Deceptive*-4

- $1.11 < \beta$: Die beiden Minima wandern zu den Rändern, wobei das Minimum $\tilde{q} > 0.5$ das globale Minimum ist. Für große β wird der Verlauf der Divergenz in (10.9) dominiert von dem Term $-\beta \cdot W_f^u$ (Z_β spielt keine Rolle, weil es unabhängig von q ist). Damit ergibt sich wie in Abschnitt 3.7.3, dass asymptotisch für großes β bei $q \approx 0.58$ das Maximum liegt, welches zu überwinden ist.

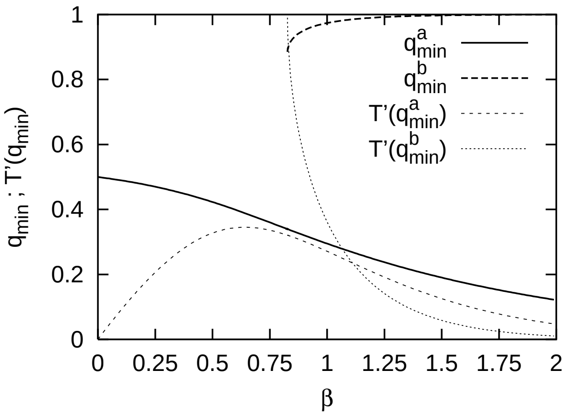

Abbildung 10.2: Lage der lokalen Minmima für verschiedene Werte von β sowie Wert der Ableitung von $T(q)$ an diesen Stellen. Bei $\beta \approx 0.827$ entsteht ein zweites Minimum. Ableitungen von $T(q) < 1$ bedeuten stabile Minima.

In Abbildung 10.2 ist die Lage der Minima für β-Werte im Intervall $[0, 2]$ dargestellt. Sie ergaben sich als Fixpunkte der Abbildung T aus Gleichung (10.17) mit Startwerten 0 beziehungsweise 1. Man sieht deutlich, dass das erste Minimum q^a_{min} von 0.5 zur Null hin wandert, während das zweite Minimum für $\beta \approx 0.827$ spontan entsteht. Für kleinere Werte von β existieren vermutlich noch komplexe Lösungen. Diese sind aber sehr wahrscheinlich nicht stabil. Eine stabile Lösung, die durch die Iterationsgleichung (10.17) gefunden werden kann, ist nämlich durch eine Ableitung $T'(q_{\min}) < 1$ charakterisiert. Wie man an dem Graphen sehen kann, sind die Ableitungen hier immer kleiner 1, nur nicht dort, wo das zweite Minimum entsteht.

Es ergibt sich also ein stetiger Übergang von der Entropieminimierung für $\beta = 0$ (lokales Minimum für $\tilde{q} = {}^1\!/_2$) und der verschobenen negativen durchschnittlichen Fitness. Da das globale Minimum für große β (also das eigentliche Ziel) erst sehr spät als Minimum sichtbar ist und von einem lokalen Maximum verdeckt ist, wird auch dieser Algorithmus das globale Optimum der *Funktion* nicht finden können.

10.1.2 Algorithmus

Algorithmus 10.1 setzt diese Überlegungen um.

Algorithmus 10.1: *KL-UMDA*

$t \Leftarrow 1$. $\beta_0 \Leftarrow 0$. Setze die Schätzer $q_{i,0} \Leftarrow 0.5$.
do {
  Generiere N Punkte mit der Verteilung $\prod_i q_{i,t-1}(x_i)$ und schätze daraus die Standardabweichung der Fitness-Werte σ. Setze $\beta_t = \beta_{t-1} + c/\sigma$.
  Beginnend mit der Schätzung $q_{i,t-1}$ aus dem vorherigen Schritt, iteriere die *Mean-Field*-Gleichungen, um $q_{i,t}$ zu erhalten.
  $t \Leftarrow t + 1$.
} **until** (Abbruchkriterium erreicht).
Gib den besten gefundenen Punkt aus.

Dabei ist eine kritische Frage noch ausgespart worden: Wie setzt man die *Mean-Field*-Gleichungen auf? Wenn die Zerlegung bekannt ist, liefert Abschnitt 10.1.1 eine Methode, die $\bar{f}_i$ zu bestimmen. Ansonsten ist man gezwungen, diese Werte zu schätzen, indem man Punkte generiert. In jedem Iterationsschritt werden aber wieder neue Parameter q^i erzeugt, mit denen man dann neue Punkte generiert, aus denen man erneut $\bar{f}_i$ schätzt.

Dieser Algorithmus ist also deutlich aufwändiger als der einfache *UMDA*.

10.1.3 Diskussion

Der *FDA* mit exakter Faktorisierung approximiert stochastisch die Boltzmann-Verteilung, die gegen das globale Optimum konvergiert. Wenn aber die Berechnung der *echten* Verteilung zu aufwändig ist, muss eine andere Verteilung gewählt werden. Der *UMDA* geht dabei einen möglichen Weg. Zusammen mit Boltzmann-Selektion kann man argumentieren, dass die Verteilungen in jedem Schritt die Boltzmann-Verteilung approximieren. Dabei tritt aber das Problem auf, dass sich die Verteilungen mit jeder Generation immer weiter von der Boltzmann-Verteilung weg entfernen, da es keinen Mechanismus gibt, der dem entgegenwirkt.

Der *KL-UMDA* geht ganz anders vor. Um den Bezug zur Boltzmann-Verteilung nicht zu verlieren, wird in jeder Generation wieder die beste Approximation (bezüglich der Kullback-Leibler-Divergenz) zur Boltzmann-Verteilung mit einem vorgegebenen β gesucht.

Unter der Annahme, dass es ein eindeutiges Maximum gibt, besteht die Boltzmann-Verteilung für große β fast nur noch aus diesem Punkt und ließe sich problemlos durch eine Produkt-Verteilung approximieren. Der *KL-UMDA* wäre demnach also ein hervorragendes Verfahren, wenn sich das globale Minimum der Divergenz ermitteln ließe. Bisher hat sich aber noch kein Beispiel gefunden, bei dem dies einfacher gewesen wäre als die Suche nach dem Optimum der Funktion. Man kann hier also nur hoffen, dass man durch geeignete Wahl der β_t zu einer guten Lösung geleitet wird.

10.2 Dynamisches Programmieren

Eine andere Methode um diskrete, additiv zerlegte Funktionen zu optimieren, ist eine Variante des dynamischen Programmierens. Dabei werden die Variablen nacheinander eliminiert, ähnlich wie bei dem Verschmelzen von Teilfunktion aus Abschnitt 4.2.5.

Betrachte als Beispiel die Funktion *IsoC* aus Abschnitt 6.2.4. Für $n = 4$ Variablen ist $m = 3$. Die Funktion ist wie folgt definiert:

$$f(x) = IsoC_1(x_1, x_2) + IsoC_1(x_2, x_3) + IsoC_2(x_3, x_4); \qquad \begin{array}{c|c|c|c|c} & 00 & 01 & 10 & 11 \\ \hline IsoC_1 & 3 & 0 & 0 & 2 \\ IsoC_2 & 0 & 0 & 0 & 3 \end{array} \tag{10.18}$$

Zerlege nun zunächst die Funktion $f(x)$ in die drei Teilfunktionen $f_{12}(x_1, x_2)$, $f_{23}(x_2, x_3)$ und $f_{34}(x_3, x_4)$. Dann werden nacheinander die Variablen eliminiert.

x_1: Die Variable x_1 kommt nur in f_{12} vor. Bestimme daher eine neue Funktion $f_2(x_2)$, die die Maximierung von f_{12} bezüglich x_1 darstellt. Weiterhin wird die entsprechende Belegung gespeichert. Eine korrekte Notation würde hier nur den Blick auf das Verfahren verstellen.

$$f_2(x_2): \qquad \begin{array}{c|c|l} x_2 & \max_{x_1} f_{12} & \text{Belegung} \\ \hline 0 & 3 & X_1 \Leftarrow 0 \\ 1 & 2 & X_1 \Leftarrow 1 \end{array} \tag{10.19}$$

Die neue Fitness-Funktion ist dann $f_2 + f_{23} + f_{34}$, wobei die Belegungsspalte natürlich ignoriert wird.

x_2: Die Variable x_2 kommt in f_2 und in f_{23} vor. Eliminieren schafft eine neue Funktion f_3, wobei die Belegungen aus f_2 übernommen werden:

$$f_3(x_3): \quad \begin{array}{c|c|l} x_3 & \max_{x_2} f_2 + f_{23} & \text{Belegung} \\ \hline 0 & 6 & X_1 \Leftarrow 0, X_2 \Leftarrow 0 \\ 1 & 4 & X_1 \Leftarrow 1, X_2 \Leftarrow 1 \end{array} \tag{10.20}$$

Die aktuelle Funktion ist also $f_3 + f_{34}$.

x_3: x_3 kommt in f_3 und f_{34} vor und es ergibt sich

$$f_4(x_4): \quad \begin{array}{c|c|l} x_4 & \max_{x_3} f_3 + f_{34} & \text{Belegung} \\ \hline 0 & 6 & X_1 \Leftarrow 0, X_2 \Leftarrow 0, X_3 \Leftarrow 0 \\ 1 & 7 & X_1 \Leftarrow 1, X_2 \Leftarrow 1, X_3 \Leftarrow 1 \end{array} \tag{10.21}$$

Als Fitness-Funktion bleibt also $f_4(x_4)$ übrig. Hier kann man einfach ablesen, dass das Maximum für $X_4 = 1$ erreicht wird mit Wert 7 und dass die übrigen Variablen auch alle den Wert 1 haben müssen.

Offenbar ist der Aufwand für diese Funktionsklasse linear in n. Dies ist aber nicht immer so. Wenn eine Variable etwa in k verschiedenen Funktionen vorkommt, die alle von 2 (jeweils unterschiedlichen) Variablen abhängen, so wird die neu gebildete Funktion eine Funktion von k Variablen sein, die Tabelle hat dann also 2^k Einträge.

Bei dem Gitter aus Abbildung 4.2 taucht dieses Problem auf. Nach der Eliminierung der ersten Reihe entsteht eine Funktion, die von allen Variablen der zweiten Reihe abhängt. Von nun an werden die Tabellen zwar nicht mehr größer, da das Eliminieren der Variablen der zweiten Reihe jeweils nur die Variable der Reihe drunter hinzufügt, die Tabellengröße ist aber mit $2^{\sqrt{n}}$ bereits nicht mehr polynomiell in n. Es entsteht also der gleiche Aufwand, wie wenn eine Faktorisierung mit *running intersection property* verwendet wird wie in Abschnitt 4.2.6.

10.2.1 Die *running intersection property* und dynamisches Programmieren

Zur Erinnerung noch einmal die Definition der *running intersection property*, Definitionen 4.10 und 4.13 von Seite 64. $f(x)$ sei zerlegt in $f(x) = \sum_{i=1}^{m} f_{s_i}(x_{s_i})$. Dann setze

$$d_i := \bigcup_{j=1}^{i} s_j \qquad b_i := s_i \setminus d_{i-1} \qquad c_i := s_i \cap d_{i-1} \tag{10.22}$$

mit $d_0 := \emptyset$. Die s_i erfüllen die *running intersection property*, wenn

- $\forall 1 \leq i \leq m : \quad b_i \neq \emptyset$,
- $d_m = I$
- $\forall i \geq 2 \quad \exists j < i : \quad c_i \subseteq s_j$

Es gilt folgender

Satz 10.1. *Es sei $|s_i| < k_{\max}$ für alle i. Wenn die Zerlegung s_i die* running intersection property *erfüllt, so findet dynamisches Programmieren das globale Optimum mit Aufwand $O(n \cdot 2^{k_{\max}})$.*

Da die Spezifikation einer Teilfunktion mit $k_{\max}$ Variablen bereits $2^{k_{\max}}$ Einträge benötigt, kann man also von einem linearen Aufwand reden.

Beweis: Starte mit der Teilfunktion $s_m = b_m \cup c_m$. Zunächst können die Variablen aus b_m eliminiert werden, da sie nur in der Teilfunktion f_{s_m} vorkommen. Es ergibt sich also die neue Teilfunktion f_{c_m}. Eliminiere nun diese Variablen. Nach Voraussetzung gibt es ein j, so dass $c_m \subseteq s_j$. Verschmelze nun f_{c_m} mit f_{s_j}. Dabei wird die Tabelle von f_{s_j} nicht größer, da keine neuen Variablen vorkommen.

Das System $s_1, \ldots, s_{m-1}$ erfüllt nun ebenfalls die *running intersection property* und es gilt $f(x) = \sum_{i=1}^{m-1} f_{s_i}(x_{s_i})$, wobei f_{s_j} ausgetauscht wurde. Per Induktion folgt damit der Satz. (Satz 10.1) ■

10.3 Zusammenfassung

Die beiden in diesem Kapitel vorgestellten Ansätze sind sehr unterschiedlich. Die *Mean Field* Approximation versucht unter Ausnutzung der einfachen Produktverteilung die Boltzmann-Verteilung zu approximieren. Es ist allerdings nicht klar, wie die prinzipiellen Einschränkungen der Produktverteilung dadurch überwunden werden können, also die Unmöglichkeit, Abhängigkeiten darzustellen. Für einen benutzbaren Algorithmus ist hier noch Arbeit nötig.

Der hier benutzte Ansatz der dynamischen Programmierung ist ein etabliertes Verfahren zur Lösung von diskreten Optimierungsproblemen. Er wurde hier aufgenommen, um die Resultate der früheren Kapitel in einen Kontext zu setzen: Wenn nämlich die *running intersection property* erfüllt ist und die Variablenmengen klein bleiben, so gibt es hiermit einen *linearen* Algorithmus, der das globale Optimum garantiert findet. Damit bleibt als (große) Nische für den *LFDA* der Fall, dass die *running intersection property* nur mit großen Variablenmengen zu erfüllen ist.

11 Zusammenfassung

In dieser Arbeit wurden die Stärken und Schwächen von drei populationsbasierten Optimierungsalgorithmen vorgestellt, *UMDA*, *FDA* und *LFDA*. Alle drei Algorithmen verwenden ein probabilistisches Modell, um die Population der selektierten Individuen zu beschreiben, und generieren neue Punkte nach diesem Modell.

Der *UMDA* wählt dabei als Modell das Produkt der univariaten Randverteilungen. Theoretische sowie empirische Resultate zeigen, dass er sich sehr ähnlich verhält wie ein einfacher genetischer Algorithmus. Sein Verhalten lässt sich beschreiben durch Analyse der durchschnittlichen Fitness als Funktion der univariaten Randverteilungen. Dazu wurden in dieser Arbeit ältere Resultate verallgemeinert und einige neue Sätze gezeigt. Die durchschnittliche Fitness verwandelt das diskrete Optimierungsproblem in ein stetiges. In diesem kontinuierlichen Raum führt der *UMDA* einen gewichteten Gradientenaufstieg durch. Dabei korrespondieren die Maxima in dieser Darstellung zu Maxima im ursprünglichen (diskreten) Raum.

Obwohl durch die Wahl der einfachen Produkt-Verteilung *Abhängigkeiten* zwischen Variablen nicht direkt erfasst werden können, ist der *UMDA* in der Lage, auch multi-modale Probleme zu lösen. Dies wird aus der Analyse der Landschaft der durchschnittlichen Fitness deutlich. Wenn aber die Natur der Interaktion den *UMDA* in die falsche Richtung lenkt, wie bei der *Deceptive*-Funktion, ist ein anderer Algorithmus vonnöten.

Der *FDA* legt als Modell die Boltzmann-Verteilung zu der zu optimierenden Fitnessfunktion $f(x)$ zugrunde. Um mit dieser Verteilung arbeiten zu können, wird sie faktorisiert. Es zeigt sich, dass man diese Faktorisierung aus einer additiven Zerlegung der Funktion $f(x)$ gewinnen kann. Die meisten Selektionsverfahren verwandeln die Boltzmann-Verteilung allerdings in eine andere, nicht näher spezifizierbare Verteilung. Um während des Verfahrens die Boltzmann-Verteilung beizubehalten ist es notwendig, Boltzmann-Selektion zu verwenden. Dies wurde bisher nicht getan, weil kein entsprechendes Abkühlungsverfahren zur Parametrisierung der Boltzmann-Selektion zur Verfügung stand. In dieser Arbeit wurde ein solches Verfahren vorgestellt, welches über eine Reihenentwicklung der durchschnittlichen Fitness der Boltzmann-Verteilung gewonnen wurde. Das Verfahren bietet weiterhin den Vorteil, dass der *FDA* dann invariant unter monotonen linearen Transformationen der Fitness-Funktion ist.

Wenn die Struktur des Problems nicht bekannt ist, muss sie gelernt werden. Die Theorie der graphischen Modelle liefert dazu den Rahmen. Die Verknüpfung des *FDA* mit dieser Theorie, um in jeder Generation aus den selektierten Individuen ein Bayes'sches Netzwerk und somit eine Faktorisierung zu lernen, liefert den *LFDA*. Dieser Algorithmus hat in der

Praxis eine deutlich größere Komplexität sowohl in der Implementierung als auch in der Laufzeit. Dafür lassen sich mit diesem Verfahren eine Reihe von Optimierungsproblemen lösen, die der *UMDA* nicht bewältigen kann. Über die Informationstheorie lässt sich auch ein Satz zeigen, der es ermöglicht, die Kenntnis einer additiven Zerlegung der Fitness-Funktion auch im Rahmen des Lernprozesses des Bayes'schen Netzwerkes zu verwenden. Dadurch kann der Zeitaufwand des Verfahrens deutlich sinken.

Die hier vorgestellten Algorithmen laufen in der Praxis in polynomieller Zeit. Für bestimmte Klassen von zu optimierenden Funktionen erreichen sie fast sicher das Optimum, sofern die verwendeten Populationen groß genug sind. Für den *UMDA* ist dies die Klasse der linearen Funktionen, für den *FDA* die Klasse der Funktionen, deren additive Zerlegung auf eine Faktorisierung der Boltzmann-Verteilung führt, die identisch ist zu der Ursprungsverteilung. Die Verfahren finden aber in weit mehr Fällen ebenfalls das globale Optimum. So erweist sich insbesondere der *UMDA* in der Praxis als ein sehr gutes Suchverfahren. Durch Verwendung des Bayes'schen Schätzers und der dadurch implizierten Mutation wird er noch robuster. Auf Grund des stochastischen Prozesses kann man eine 100%-ige Garantie für das Finden des globalen Optimums für keines dieser Verfahren geben, nicht einmal für einfachste Funktionen. Es ist auch noch nicht gelungen, die Wahrscheinlichkeit zu quantifizieren.

Für den Fall, dass die Boltzmann-Verteilung mit polynomiellen Aufwand faktorisiert, ist der Algorithmus aus Kapitel 10, der dynamisches Programmieren nutzt, besser geeignet, da er dann in polynomieller Zeit das Optimum garantiert findet. Die Domäne der Algorithmen liegt also im Bereich der *approximativen* Faktorisierungen, wie es beim *UMDA* auch schon der Fall ist.

Alle in dieser Arbeit vorgestellten Verfahren sind nicht direkt dafür geeignet, Probleme mit Nebenbedingungen direkt zu lösen. In Verbindung mit einem Hillclimber, der die Nebenbedingungen sicherstellt, wurde für die Graph Bipartitionierung gezeigt, dass sich die Verfahren auch für in der Praxis einsetzen lassen. Der Rechenaufwand des *LFDA* ist dabei zwar deutlich höher als bei in der Literatur verwandten Algorithmen, gemessen an Funktionsevaluierungen schneidet er aber sehr gut ab.

UMDA mit Mutation und *LFDA* stellen zwei Basisalgorithmen dar, die grundsätzlich für diskrete Optimierung ohne Nebenbedingungen eingesetzt werden können. Für Probleme mit Nebenbedingungen ist eine Sonderbehandlung erforderlich. Grundsätzlich kann man immer zunächst den *UMDA* verwenden, der leicht programmierbar ist und wenig Parameter hat. Wenn der *UMDA* aber aufgrund der Interaktionen zwischen den Variablen versagt, kann man auf den *LFDA* zurückgreifen, der allerdings eine teils deutlich erhöhte Komplexität hat.

A Allgemeine Definitionen

A.1 Aufgabenstellung

Aufgabe: *Gegeben sei die Funktion*

$$f(x) : \mathcal{R} \to \mathbb{R} \tag{A.1}$$

für diskretes $\mathcal{R}$. Finde ein $x_{\max} \in \mathcal{R}$ mit

$$f(x_{\max}) = \max_{x \in \mathcal{R}} f(x) =: f_{\max} \tag{A.2}$$

*Das Maximum muss nicht eindeutig sein, die Menge der Optima heiße $\mathcal{M}$. Die zu optimierende Funktion heißt auch **Fitness-Funktion**.*

Der Raum $\mathcal{R}$ sei dabei ein Produktraum,

$$\mathcal{R} = \prod_{i=1}^{n} \mathcal{R}_i, \qquad \mathcal{R}_i = \{0, \ldots, l_i\} \tag{A.3}$$

so dass das Argument x in n Komponenten zerfällt:

$$x = (x_1, \ldots, x_n) \qquad \text{mit } x_i \in \mathcal{R}_i \tag{A.4}$$

Zur Vereinfachung der Darstellung gilt in vielen Fällen $\mathcal{R}_i = \{0, 1\}$.

A.1.1 Variablennamen und Belegungen

Im folgenden wird es häufig nötig sein, zwischen einer Variablen und ihrer Belegung zu unterscheiden. Dazu gelten folgende Konvention:

Variablen werden durch Großbuchstaben X_1 bis X_n bezeichnet, *Belegungen* durch Kleinbuchstaben x_1 bis x_n. Somit bezeichnet $g(X_1, X_2)$ die Funktion g, die von zwei Variablen abhängt, während $g(x_1, x_2)$ den Funktionswert an der Stelle $(X_1 = x_1, X_2 = x_2)$ beschreibt. $g(X_1, x_2)$ ist dann die Funktion (von X_1), die durch Fixieren von $X_2 = x_2$ entsteht. In den meisten Fällen macht dies keinen Unterschied, so bezeichnen sowohl $f(x_1, \ldots, x_n)$ wie auch $f(X_1, \ldots, X_n)$ die Funktion f.

I sei die Indexmenge $\{1, \ldots, n\}$, $\bar{X}$ die Menge der Variablen $\{X_1, \ldots, X_n\}$.

Nun sei $s \subseteq I$ eine Menge von Variablenindizes. Für $i \in I$ sei $\chi_s(i)$ die Indikatorfunktion:

$$\chi_s(i) = \begin{cases} 1 & \text{für } i \in s \\ 0 & \text{für } i \notin s \end{cases} \tag{A.5}$$

Definiere die Projektion auf s für $x \in \mathcal{R}$ durch

$$\bar{P}_s x := (\hat{x}_1, \ldots, \hat{x}_n) \qquad \text{mit } \hat{x}_i := x_i \cdot \chi_s(i) \tag{A.6}$$

Also gilt zum Beispiel $\bar{P}_{1,2}(x_1, x_2, x_3) = (x_1, x_2, 0)$. Zur Vereinfachung der Notation identifiziere auf kanonische Weise den Bildbereich von $\bar{P}_s$ mit $\prod_{i \in s} \mathcal{R}_i =: \mathcal{R}_s$. Definiere weiterhin $\mathcal{R}_i^u := \{1, \ldots, l_i\}$ und analog $\mathcal{R}_s^u$.

A.1.2 Dualität von Vektoren / Teilmengen

Zu jedem binären Vektor $x = (x_1, \ldots, x_n)$ gibt es genau eine Teilmenge $\check{x} \subseteq I$ mit $i \in \check{x} \;\Leftrightarrow\; x_i = 1$. Ferner gibt es zu einer Teilmenge $s \subseteq I$ genau einen Vektor $\hat{s}$ mit $\hat{s}_i = 1 \;\Leftrightarrow\; i \in s$. Es gilt natürlich $\hat{\check{x}} = x$ und $\check{\hat{s}} = s$.

Definiere weiterhin

$$x^s := \prod_{i=1}^{n} x_i^{\hat{s}_i} \qquad = \begin{cases} 1 & \text{wenn } \check{x} \subseteq s \\ 0 & \text{sonst} \end{cases} \tag{A.7}$$

A.1.3 Additive Zerlegung in Subfunktionen

Eine Funktion, die nur von den Parametern aus $s \in I$ abhängt, wird durch den Index s gekennzeichnet. Formal gesehen sind alle Funktionen auf ganz $\mathcal{R}$ definiert.

So kann man zum Beispiel definieren

$$g_s(x) := \sum_{i \in s} x_i \tag{A.8}$$

Um die Notation zu vereinfachen, lässt sich dies zum Beispiel für $s = \{1, 2\}$ auch so schreiben:

$$g_{1,2}(x_1, x_2) := x_1 + x_2 \tag{A.9}$$

Die Fitness-Funktion f wird in vielen Fällen als Summe von Teilfunktionen darstellbar sein, die jeweils nur von einer Teilmenge von Variablen abhängt.

Definition A.1. *Es seien $s_1, \ldots, s_m$ Indexmengen mit $s_i \subseteq I$. Ferner seien f_{s_1} bis f_{s_m} Funktionen der Variablen X_j mit $j \in s_i$. Dann ist*

$$f(x) = \sum_{i=1}^{m} f_{s_i}(x) \tag{A.10}$$

*die **additive Zerlegung** der Fitness-Funktion f.*

Eine additive Zerlegung kann die Optimierung des Problems stark vereinfachen.

A.2 Wahrscheinlichkeiten und Zufallsvariablen

Definition A.2. *Eine (diskrete)* ***Wahrscheinlichkeitsdichte*** *p ist eine Funktion $p : \mathcal{R} \to \mathbb{R}$ mit $\sum_{x \in \mathcal{R}} p(x) = 1$ sowie $p(x) \geq 0$. Assoziiert ist ein* ***Wahrscheinlichkeitsmaß****, welches ebenfalls mit p bezeichnet wird. Das Wahrscheinlichkeitsmaß p ist eine Funktion, die jeder Teilmenge von $\mathcal{R}$ eine Zahl zuordnet gemäß*

$$\forall M \subseteq \mathcal{R} : \quad p(M) = \sum\nolimits_{x \in M} p(x) \tag{A.11}$$

Der Raum $\mathcal{R}$ wird zusammen mit einem (diskreten) Wahrscheinlichkeitsmaß p zu einem ***Wahrscheinlichkeitsraum****.*

Die Benutzung des gleichen Symbols für die Dichte und das Maß ist sinnvoll, weil nach Definition $p(\{x\}) = p(x)$ gilt.

Definition A.3. *Eine* ***Zufallsvariable*** *ist eine Funktion $V : \mathcal{R} \to Y$. Für $Y = \mathbb{R}$ nennt man V eine* ***reelle Zufallsvariable****. Die* ***Dichte*** *der Zufallsvariablen ist die Funktion*

$$\forall y \in \mathit{Bild}(V) : \qquad \rho_{p,V}(y) := p(\{x | V(x) = y\}) \quad =: p(V = y) \tag{A.12}$$

Damit ist jedem Element $y \in \mathit{Bild}(V)$ des Bildraums eine Wahrscheinlichkeit $p(V = y)$ zugeordnet. Die Menge $\{(y \in \mathit{Bild}(V), p(V = y)\}$ nennt man ***Verteilung*** *von V.*

Beispiel A.1. Die Fitness-Funktion f kann als (reelle) Zufallsvariable betrachtet werden.

Beispiel A.2. $X_1, \ldots, X_n$ lassen sich als Zufallsvariablen auffassen, in dem man sie einfach als Koordinatenprojektionen $X_i(x_1, \ldots, x_n) := x_i$ definiert mit $\mathrm{Bild}(X_i) = \mathcal{R}_i$. Die Dichten der Koordinatenprojektionen heißen **univariate Randverteilungen** und haben folgende Form:

$$p_i(x_i) := \rho_{p,X_i}(x_i) = p(X_i = x_i) = \sum_{\substack{y \in \mathcal{R} \\ y_i = x_i}} p(y) \tag{A.13}$$

Für binären Wertebereich definiere die Abkürzung

$$p_i := p_i(1) = p(X_i = 1) \tag{A.14}$$

$p_i(x_i)$ lässt sich ferner auch als Wahrscheinlichkeitsdichte $p_i(X_i)$ auf dem Raum $\mathcal{R}_i$ auffassen. Dort ist es außerdem die Dichte der Zufallsvariable id_{W_i}, also der Identität auf W_i.

Beispiel A.3. n-Tupel von Zufallsvariablen kann man auch als vektorwertige Zufallsvariablen ansehen. Die Verteilung dieser neuen Zufallsvariablen heißt **gemeinsame Verteilung**. Wenn die beteiligten Zufallsvariablen die Koordinatenprojektionen sind, so ergeben sich für die Dichte die **Randverteilungen** als Verallgemeinerung der univariaten Randverteilungen:

$$p_s(x_s) := p(X_s = x_s) = \sum_{\substack{y \in \mathcal{R} \\ y_i = x_i}} p(y) \qquad \text{für } s \subseteq I \tag{A.15}$$

Randverteilungen haben die Eigenschaft $\sum_{x_s} p_s(x_s)=1$, das heißt, p_s kann man wieder als Wahrscheinlichkeitsdichte $p_s(X_s)$ auf $\mathcal{R}_s$ betrachten. Im Spezialfall $s=I$ ist die Randverteilung p_I identisch mit der Ausgangsverteilung p (weil die Koordinatenprojektion auf *alle* Koordinaten die Identität ist).

Wenn sich die betrachtete Menge von Variablen aus dem Argument und dem Kontext ergibt, definiere $p(x_s) := p_s(x_s)$. Mit $b, c \subseteq I$ und disjunkt schreibe auch $p(x_a, x_b)$ anstelle von $p(x_{a\cup b})$.

Weiterhin gilt, wenn s und t disjunkte Teilmengen von I sind: $p_s(x_s) \cdot p_t(x_t)$ ist Wahrscheinlichkeitsdichte auf $\mathcal{R}_{s\cup t}$. Insbesondere ist $\tilde{p}(x) = \prod_{i=1}^{n} p_i(x_i)$ eine Wahrscheinlichkeitsdichte auf $\mathcal{R}$, die aber im Allgemeinen *nicht* mit $p(x)$ übereinstimmt. Allgemein induziert jede disjunkte Zerlegung $\mathcal{S} = \{s_1, \ldots, s_m\}$ von I eine Wahrscheinlichkeitsdichte auf $\mathcal{R}$ über

$$p^{\mathcal{S}}(X) = \prod_{i=1}^{m} p_{s_i}(X_{s_i}) \tag{A.16}$$

In dieser Arbeit kommen als Zufallsvariablen nur Mengen von Koordinatenprojektionen beziehungsweise die (eventuell transformierte) Fitness-Funktion vor.

Definition A.4. *Die **konditionalen Wahrscheinlichkeiten** definieren sich aus den Randverteilungen durch*

$$p(x_b|x_c) := p_{b|c}(x_b|x_c) := \frac{p(x_b, x_c)}{p(x_c)} \qquad \textit{für } p(x_c) > 0 \tag{A.17}$$

Mit Hilfe der konditionalen Wahrscheinlichkeiten lassen sich allgemeinere Faktorisierungen der Verteilung p beschreiben.

Definition A.5. *Eine **Faktorisierung** einer Verteilung p ist eine neue Verteilung*

$$\tilde{p}(X) = \prod_{i=1}^{l} p(X_{b_i}|X_{c_i}) \qquad \textit{mit } b_i, c_i \subseteq I \tag{A.18}$$

Es müssen dabei bestimmte Bedingungen an die b_i und c_i gestellt werden, damit $\tilde{p}$ überhaupt eine Verteilung ist. Dies wird in Kapitel 4 im Detail diskutiert. Ferner gilt im Allgemeinen nicht $\tilde{p}(X) = p(X)$.

A.2.1 Momente

Da man f als Zufallsvariable auffassen kann, kann man für ein Wahrscheinlichkeitsmaß p die **durchschnittliche Fitness** $W_{f,p}$ definieren als

$$W_{f,p} := \sum_{x\in\mathcal{R}} f(x)p(x) \tag{A.19}$$

Die Wahl des Buchstaben W kommt aus der Populationsgenetik und wurde von Sewell Wright (1932) eingeführt.

$W_{f,p}$ ist natürlich nichts anderes als der Erwartungswert von f. Man kann nun auch die höheren Momente betrachten:

Definition A.6. *Die **Momente** M_k der reellen Zufallsvariablen f sind definiert durch*

$$M_{k,f}(p) := \sum_{x \in \mathcal{R}} f(x)^k p(x) \tag{A.20}$$

*Die **zentrierten Momente** M_k^z sind*

$$M_{k,f}^z(p) := \sum_{x \in \mathcal{R}} \left[f(x) - W_{f,p}\right]^k p(x) \tag{A.21}$$

Es gilt trivialerweise: $M_{1,f}(p) = W_{f,p}$, $M_{1,f}^z = 0$, $M_{2,f}^z(p) = \sigma_p^2(f)$, die Varianz der Verteilung.

A.3 Definitionen der Informationstheorie

Für die Definitionen und Sätze aus diesem Abschnitt siehe (Cover and Thomas, 1991; Taneja, 1998). $\log(x)$ bezeichne den Logarithmus zur Basis 2.

Definition A.7 (Shannonsche Entropie). *Die **Shannonsche Entropie** einer Zufallsvariablen X mit der Dichte p ist definiert durch*

$$H_p(X) := - \sum_{x \in Bild(X)} p(X = x) \log p(X = x) \tag{A.22}$$

Da sich $t \log(t)$ bei $t = 0$ durch 0 stetig ergänzen lässt, ist die Entropie für jede Verteilung definiert. Sie ist ein Maß für die Zufälligkeit einer Variablen. Es gilt $H(X) \geq 0$, ihr Maximum erreicht sie bei einer gleichverteilten Zufallsvariable.

Wenn die Zufallsvariable die Koordinatenprojektion X_i ist, dann lässt sich die Summe in (A.22) auf zwei Arten deuten. Einmal ist der zugrunde liegende Raum ganz $\mathcal{R}$ und $p(X_i = x_i)$ stellt die Randverteilung dar. Oder aber man betrachtet $p_i(X_i)$ als Verteilung auf $\mathcal{R}_i$ (vergleiche Abschnitt A.2), die Zufallsvariable X_i ist dann die Identität auf $\mathcal{R}_i$ und es ergibt sich $H_{p_i}(X_i) = -\sum_{x_i \in \mathcal{R}_i} p_i(x_i) \log p_i(x_i)$. Dies sind aber nur zwei Sichten derselben Formel. Dies ermöglicht es aber, für solche Zufallsvariablen abkürzend zu schreiben:

$$H_p(id) = -\sum_x p(x) \log p(x) \tag{A.23}$$

Die Entropie kann im Kontext des Samplings auch als Maß für die *Unsicherheit* über die Verteilung angesehen werden. Sie hat eine ganze Reihe von Eigenschaften. Wichtig ist die

Lemma A.8 (Shannon-Gibbs-Ungleichung). *Wenn p und q zwei Wahrscheinlichkeitsdichten sind, so gilt*

$$-\sum_x p(x)\log p(x) \leq -\sum_x p(x)\log q(x) \tag{A.24}$$

Definiere weiterhin die **gemeinsame** und die **konditionale Entropie** als

$$H_p(X,Y) := -\sum_{x,y} p(X=x,Y=y)\log p(X=x,Y=y) \tag{A.25}$$

$$H_p(X|Y) := H_p(X,Y) - H_p(Y) = -\sum_{x,y} p(X=x,Y=y)\log p(X=x|Y=y) \tag{A.26}$$

Für Koordinaten X_i, X_j ist die gemeinsame Entropie demnach die Entropie der Identität auf dem Produktraum $\mathcal{R}_i \times \mathcal{R}_j$. Daher ist sie identisch zu der Entropie der gemeinsamen Zufallsvariablen (X_i, X_j).

Als Unsicherheit interpretiert, lesen sich diese Gleichungen so: Die Unsicherheit von (X,Y) ist die Unsicherheit von Y plus die durchschnittliche Unsicherheit von X gegeben Y. Außerdem kann man die konditionale Entropie auffassen als Maß dafür, wieviele Bits nötig sind, um X zu kodieren, wenn der Wert von Y bekannt ist.

Definition A.9 (Kullback-Leibler-Divergenz). *Die **Kullback-Leibler-Divergenz** zwischen zwei Verteilungen p und q ist definiert als*

$$D^{KL}(p\,\|\,q) := \sum_x p(x)\log\frac{p(x)}{q(x)} \tag{A.27}$$

Definition A.10 (Wechselseitige Information). *Die **wechselseitige Information** zwischen zwei Zufallsvariablen X und Y ist*

$$\mathrm{Inf}_p(X;Y) := H_p(X) + H_p(Y) - H_p(X,Y) = H_p(X) - H_p(X|Y) \tag{A.28}$$

*Diese Definition erweitert sich auf natürliche Weise auf mehr als zwei Variablen. Definiere weiterhin die **konditionale wechselseitige Information***

$$\mathrm{Inf}_p(X;Y|Z) := H_p(X|Z) - H_p(X|Y,Z) \tag{A.29}$$

Im folgenden wird die Entropie immer auf eine feste Verteilung p bezogen, die nicht mehr dazugeschrieben wird. Die Zufallsvariablen sind Koordinatenprojektionen auf eine oder mehrere Koordinaten.

Lemma A.11. *Es gilt: (siehe (Cover and Thomas, 1991; Taneja, 1998))*

1. $H(X) \geq 0$
2. $D^{KL}(p\,\|\,q) \geq 0$, $D^{KL}(p\,\|\,q) = 0 \;\Leftrightarrow\; p = q$

3. *$D^{KL}(p \,\|\, q)$ ist nicht symmetrisch.*
4. $X \perp\!\!\!\perp Y \;\Leftrightarrow\; H(X,Y) = H(X) + H(Y)$
5. $\mathrm{Inf}(X;Y) \geq 0,\; X \perp\!\!\!\perp Y \;\Leftrightarrow\; \mathrm{Inf}(X;Y) = 0$
6. $\mathrm{Inf}(X;Y|Z) \geq 0,\; X \perp\!\!\!\perp Y|Z \;\Leftrightarrow\; \mathrm{Inf}(X;Y|Z) = 0$
7. $\mathrm{Inf}(X;Y) = D^{KL}(p(X,Y) \,\|\, p(X) \cdot p(Y))$
8. $\mathrm{Inf}(X;Y,Z) = \mathrm{Inf}(X;Y) + \mathrm{Inf}(X;Z|Y)$

A.4 Bayes'sche Schätzer und gezählte Häufigkeiten

Hier soll nicht tiefer auf die Hintergründe der Bayes'schen Theorie eingegangen werden, eine für diesen Rahmen ausreichende Einführung findet sich zum Beispiel in (Heckerman, 1999). Es seien nur die Unterschiede zur klassischen *Maximum-Likelihood*-Schätzung kurz am Beispiel der gezinkten Münze vorgestellt.

Es geht dabei um folgende Frage: *Wenn in einem Münzwurf mit gezinkter Münze bei N Würfen m mal Zahl kommt, wie ist dann die Wahrscheinlichkeit θ für „Zahl" einzuschätzen?*

Es gibt dabei folgende mögliche Antworten:

Gezählte Häufigkeiten

Die zunächst naheliegende Lösung ist es, die *Maximum-Likelihood*-Lösung zu wählen. Dabei wird die *Likelihood* für jedes mögliche θ berechnet, also die Wahrscheinlichkeit, dass für dieses θ bei N Würfen m Mal Zahl kam. Es ergibt sich

$$L(\theta) = \theta^m (1-\theta)^{N-m} \tag{A.30}$$

Wenn man nun das Maximum dieser Gleichung sucht, ergibt sich $\theta^* = {}^m\!/_N$. Diese Schätzung kann man als Bewertung der vergangenen N Experimente ansehen.

Eine wichtige Eigenschaft dieser Schätzung ist es, dass für $m=0$ (also nie Zahl geworfen) die Wahrscheinlichkeit $\theta^*=0$ herauskommt, ebenso führt $m=N$ zu $\theta^*=1$.

Bayes'sche Schätzer

Der Ansatz der Bayesianer ist ein anderer. Hierbei verändern Messungen, also Würfe x_1 bis x_N, die subjektive Einschätzung eines Beobachters über die Wahrscheinlichkeiten der Münze. Ausgehend von einer A-Priori-Verteilung für p, also einer Annahme über die „Gezinktheit" der Münze, *bevor* noch ein Wurf gefallen ist, werden die Ergebnisse der N Würfe mittels des Bayes'schen Updateverfahren aktualisiert, um so die Wahrscheinlichkeit von „Zahl" für den *nächsten* Wurf abzuschätzen.

Konditionale Wahrscheinlichkeiten spielen hier eine zentrale Rolle. $p(x|\xi)$ bezeichnet dabei eine Verteilung von x für jemand, der Hintergrundwissen ξ hat. Dieses Hintergrundwissen kann eine Grundannahme sein („ich vermute, dass die Münze nicht gezinkt ist"), aber auch beliebiges anderes Wissen, wie Ergebnisse von vorhergehenden Würfen. θ bezeichnet jetzt also einen Parameter der Münze, den man aus den Experimenten bestimmen will.

Wenn $p(\theta|\xi)$ die A-Priori-Verteilung für diesen Parameter ist, so ist das Ziel, $p(x_{N+1}|D,\xi)$ daraus zu berechnen.

Die Bayes'sche Regel liefert die Wahrscheinlichkeits-Verteilung von θ bei gegebenen Würfen D und Hintergrundwissen ξ:

$$p(\theta|D,\xi) = \frac{p(\theta|\xi)\,p(D|\theta,\xi)}{\int p(D|\tilde{\theta},\xi)\,p(\tilde{\theta}|\xi)d\tilde{\theta}} \tag{A.31}$$

Der Nenner ist dabei nur für die Normalisierung notwendig ist. Oft schreibt man daher nur $p(\theta|D,\xi) \propto p(\theta|\xi)\,p(D|\theta,\xi)$. Diese Verteilung nennt man die A-Posteriori-Verteilung von θ.

Für $p(D|\theta,\xi)$ ergibt sich wie im klassischen Fall die Wahrscheinlichkeit der Daten, gegeben den Parameter θ, also

$$p(D|\theta,\xi) = \theta^m(1-\theta)^{N-m} \tag{A.32}$$

Ein entscheidender Unterschied zu gezählten Häufigkeiten ist es nun, dass man eine *Verteilung* über die möglichen Werte von θ hat, nämlich $p(\theta|D,\xi)$. Für die Wahrscheinlichkeit, im nächsten Wurf eine Zahl zu bekommen, wird deshalb der *Erwartungswert* genommen:

$$p(x_{N+1}\!=\!\text{Kopf}|D,\xi) = \int p(x_{N+1}\!=\!\text{Kopf},\theta|D,\xi)\,d\theta \tag{A.33}$$

$$= \int p(x_{N+1}\!=\!\text{Kopf}|\theta,D,\xi)\,p(\theta|D,\xi)\,d\theta \tag{A.34}$$

$$= \int \theta\cdot p(\theta|D,\xi)\,d\theta \tag{A.35}$$

An dieser Stelle wird manchmal aus Effizienzgründen auch einfach nur das θ genommen, welches $p(\theta|D,\xi)$ maximiert. Zulässig ist dies aber eigentlich nur, wenn diese Verteilung im Wesentlichen aus einem Peak rund um das Maximum besteht.

Die drei Formeln lassen sich nun zusammensetzen, wenn man sich auf eine A-Priori-Verteilung geeinigt hat. Für das Problem des Münzwerfens eignet sich besonders die *Beta*-Verteilung:

$$p(\theta|\xi) = Beta(\theta|N_1',N_2') = \frac{\Gamma(N_1'+N_2')}{\Gamma(N_1')\Gamma(N_2')}\cdot\theta^{N_1'-1}(1-\theta)^{N_2'-1} \tag{A.36}$$

Die Parameter N_1' und N_2' müssen dabei größer Null sein, um eine gültige Verteilung zu haben. Die *Beta*-Verteilung eignet sich deshalb besonders gut, weil die A-Posteriori-Verteilung dann ebenfalls eine *Beta*-Verteilung ist:

$$p(\theta|D,\xi) = Beta(\theta|N_1'+m,N_2'+(N-m)) \tag{A.37}$$

Die Menge der *Beta*-Verteilungen nennt sich daher *konjugierte Familie von Verteilungen* für binomielles Samplen. Man sieht daran auch, dass die A-Posteriori-Verteilung einfach als A-Priori-Verteilung für eine weitere Anzahl an Experimenten genutzt werden kann. Insbesondere ist es für die *Beta*-Verteilung nicht relevant, ob man die Verteilung nach jedem Wurf aktualisiert oder erst ganz am Ende.

Wenn man alle Formeln zusammenführt, ergibt sich mit einem *Beta*-Prior als Voraussage

$$p(x_{N+1} = \text{Kopf}|D, \xi) = \frac{N'_1 + m}{(N'_1 + N'_2) + N} \tag{A.38}$$

Diese Formel ist identisch zur *Maximum-Likelihood*-Schätzung, wenn man $N'_1 = N'_2 = 0$ setzt. Das ist allerdings keine gültige Wahl, weil die A-Priori-Verteilung dafür nicht definiert ist.

Im Gegensatz zu gezählten Häufigkeiten wird die Bayes'sche Methode, selbst wenn Kopf überhaupt nicht gefallen ist, dennoch eine Wahrscheinlichkeit größer Null für Kopf im nächsten Wurf errechnen. Für $N'_1 = N'_2 = 1$ ergibt sich diese Wahrscheinlichkeit zu $^1/_{(N+2)}$. Dies entspricht auch der Intuition in diesem Modell: Wenn man wenig über die Münze weiß, so wird man selbst nach 10 Mal Zahl immer noch nicht mit Sicherheit sagen wollen, dass nie Kopf erscheinen wird.

A.4.1 Wahl der Hyperparameter

Neben der Wahl der Familie müssen die Hyperparameter N'_1 und N'_2 nun noch gewählt werden. Dazu gilt folgendes (siehe Abbildung A.1).

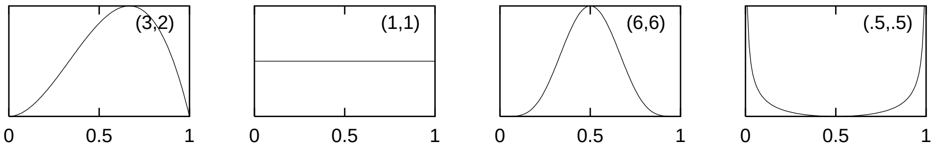

Abbildung A.1: Verschiedene *Beta*-Verteilungen

- Wenn N'_1 größer als N'_2 ist (erstes Bild), so sind Werte von θ größer 0.5 wahrscheinlicher. So lässt sich also etwa die Vermutung, dass „Kopf" häufiger sein wird, ausdrücken. In den Anwendungen gibt es dafür allerdings keinen Grund, so dass immer $N'_1 = N'_2 =: N'$ gilt.
- $N' = 1$ (zweites Bild) ist die Gleichverteilung für den Parameter θ.
- $N' > 1$ (drittes Bild) liefert eine Verteilung, die für wachsendes N' immer stärker um den Peak bei 0.5 konzentriert ist. Wachsendes N' drückt also ein größeres Gewicht für die Hypothese aus, dass die Münze fair ist.

- Wenn $N' < 1$ (viertes Bild), so werden die Ränder der Verteilung stärker bevorzugt. Die Vermutung ist also, dass die Münze nicht fair ist, es wird aber keine Annahme darüber gemacht, in welcher Richtung sie gezinkt ist.

Man muss sich darüber im Klaren sein, dass *keine* Verteilung mit $N'_1 = N'_2$ eine der beiden Seiten bevorzugt. Es geht dann nur noch darum, wie stark man a priori die Wahrscheinlichkeit einschätzt, dass die Münze nicht fair ist.

Die Wahl der A-Priori-Verteilung ist eine schwierige Frage und einer der Kritikpunkte an der Theorie. In Abschnitt 3.8 wird ein Kriterium hergeleitet, wie die Parameter für diese Anwendung gewählt werden können.

B Symbolverzeichnis

$$
\begin{aligned}
N &:= \text{Populationsgröße der Population } \mathfrak{X} \\
\hat{N} &:= \text{Anzahl an selektierten Individuen} \\
n &:= \text{Anzahl an Variablen} \\
\mathfrak{X}^i &:= \text{Individuum } i, 1 \leq i \leq N \\
\mathfrak{X}^i_j &:= \text{Allel } j \text{ von Individuum } i,\ 1 \leq j \leq n \\
\hat{\mathfrak{X}}^i &:= \text{Selektiertes Individuum}, 1 \leq i \leq \hat{N} \\
X_1, \ldots, X_n &:= \text{Variablennamen}, \quad \bar{X} := \bigcup_{i=1}^n X_i \\
\mathcal{R}_1, \ldots, \mathcal{R}_n &:= \text{Definitionsmengen},\ \mathcal{R}_i = \{0, \ldots, l_i\},\ \mathcal{R}_i^u := \{1, \ldots, l_i\} \\
\mathcal{R} &:= \text{Definitionsbereich von } f \text{ und } p, \quad \prod_{i=1}^n \mathcal{R}_i, \quad \hat{\mathcal{R}} := \max_i |\mathcal{R}_i| \\
x_1, \ldots, x_n &:= \text{Zuweisungen},\ x_i \in \mathcal{R}_i \\
\check{x}, \hat{s} &:= \text{Duale Darstellung von binären Vektoren / Teilmengen} \\
x^s &:= \prod_{i=1}^n x_i^{\hat{s}_i} \\
\mathcal{G} = (V, E) &:= \text{(gerichteter) Graph mit Knoten } V = \bar{X} \text{ und Kanten } E = (X_i \to X_j) \\
\Pi_i &:= \text{Elternvariablen von } X_i \text{ im Graphen } \mathcal{G} \\
k_{\max} &:= \text{Maximale Anzahl an Eltern im Graphen} \\
N_{ijk} &:= X_i \text{ in Zustand } k \text{ gegeben Eltern in Zustand } j;\ 0 \leq k \leq l_i,\ 0 \leq j \leq q_i - 1 \\
N_{ij} &:= \sum_{k=0}^{l_i} N_{ijk} \\
r_i &:= l_i + 1, \quad q_i := \prod_{j: X_j \in \Pi_i} r_j \\
N_i', N_{ij}' &:= \text{Dirichlet-Hyperparameter der A-Priori-Verteilung} \\
p(x_s) &:= p_s(x_s) := \sum_{y, y_s = x_s} p(y) \qquad \text{Beispiel: } p(X_1 = 1) = \sum_{y, y_1 = 1} p(y) \\
p(x_{b_i} | x_{c_i}) &:= \frac{p(x_{b_i}, x_{c_i})}{p(x_{c_i})}
\end{aligned}
$$

C Literaturverzeichnis

S. Arora, D. Karger, und M. Karpinski, 1999. Polynomial time approximation schemes for dense instances of NP-hard problems. In *Journal of Computer and System Sciences*, Band 58, Seiten 193–210.

S. Baluja und R. Caruana, 1995. Removing the genetics from the standard genetic algorithm. In *Proc. of the 12th Intern. Conf. on Machine Learning*, Seiten 38–46. Morgan Kaufmann.

L.E. Baum und J.A. Eagon, 1967. An inequality with applications to statistical estimation for probabilistic functions of markov processes and to a model for ecology. *Bull. Am. Math. Soc.*, 73:360–363.

A. Berny, 2000. Statistical machine learning and combinatorial optimization. In L. Kallel, B. Naudts, und A. Rogers, Herausgeber, *Theoretical Aspects of Evolutionary Computing*, Natural Computing. Springer Verlag.

A.D. Bethke, 1981. *Genetic Algorithms as Function Optimizers*. Doktorarbeit, University of Michigan.

J.S. De Bonet, Ch. L. Isbell, und P. Viola, 1997. Mimic: Finding optima by estimating probability densities. In M.C. Mozer & M.I. Jordan & Th. Petsche, Herausgeber, *Advances in Neural Information Processing Systems 9*, Band 9, Seiten 424–430, Cambridge. MIT Press.

R.R. Bouckaert, 1993. Belief network construction using the minimum description length principle. In *Proceedings ECSQARU*, Seiten 41–48.

R.R. Bouckaert, 1994. Properties of Bayesian network learning algorithms. In R. Lopez de Mantaras und D. Poole, Herausgeber, *Proc. Tenth Conference on Uncertainty in Artificial Intelligence*, Seiten 102–109, San Francisco. Morgan Kaufmann.

T.N. Bui und C. Jones, 1992. Finding good approximate vertex and edge partitions is np-hard. *Information Processing Letters*, 42:153–159.

D.M. Chickering, D. Geiger, und D. Heckerman, November 1994. Learning bayesian networks is NP-hard. Technischer Report MSR-TR-94-17, Microsoft Research, Microsoft Corporation, Redmond, WA 98052.

G.F. Cooper und E.A. Herskovits, 1992. A Bayesian method for the induction of probabilistic networks from data. *Machine Learning*, 9:309–347.

Th.M. Cover und J.A. Thomas, 1991. *Elements of Information Theory.* John Wiley & Sons.

I. Csiszár und J. Körner, 1981. *Information Theory: Coding Theorems for Discrete Memoryless Systems.* Academic Press, New York.

Y. Davidor, 1991. Epistasis variance: A viewpoint on GA-hardness. In G. Rawlins, Herausgeber, *Foundations of Genetic Algorithms*, Seiten 23–35, San Mateo. Morgan Kaufmann.

M. de la Maza und B. Tidor, 1993. An analysis of selection procedures with particular attention paid to proportional and Boltzmann selection. In S. Forrest, Herausgeber, *Proc. of the Fifth Int. Conf. on Genetic Algorithms*, Seiten 124–131, San Mateo, CA. Morgan Kaufman.

D. S. Falconer, 1981. *Introduction to Quantitative Genetics.* Longman, London.

U. Feige und R. Krauthgamer, 2000. A polylogarithmic approximation of the minimum bisection. In *Proceedings of the 41st Annual Symposium on Foundations of Computer Science.* to appear.

C.M. Fiduccia und R.M. Mattheyses, 1982. A linear-time heuristic for improving network partitions. In *Proceedings of the Nineteenth ACM/IEEE Design Automation Conference*, Seiten 175–181, Piscataway, New Jersey. IEEE Computer Society Press.

R.A. Fisher, 1958. *The Genetical Theory of Natural Selection.* Dover, New York.

N. Friedman und M. Goldzmidt, 1999. Learning Bayesian networks with local structure. In Jordan (1999), Seiten 421–459.

N. Friedman und Z. Yakhini, 1996. On the sample complexity of learning Bayesian networks. In *Proceedings of the 12 Conference on Uncertainty in AI*, Seiten 274–282.

M.R. Garey, D.S. Johnson, und L. Stockmeyer, 1976. Some simplified np-complete graph problems. In *Theoretical Computer Science*, Band 1, Seiten 237–267.

H. Geiringer, 1944. On the probability theory of linkage in mendelian heredity. *Annals of Math. Stat.*, 15:25–57.

D.E. Goldberg, 1989a. Genetic algorithms and walsh functions: part i, a gentle introduction. *Complex systems*, 3:129–152.

D.E. Goldberg, 1989b. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley, Reading.

B. Hajek, 1988. Cooling schedules for optimal annealing. *Mathematics of Operations Research*, 13(2):311–329.

B. Hajek und G. Sasaki, 1989. Simulated annealing – to cool or not. *Systems and Control Letters*, 12:443–447.

R.B. Heckendorn, 2000. Polynomial time summary statistics for two general classes of functions. In *Proceedings of the Genetic and Evolutionary Computation Conference GECCO-2000*, Seiten 919–926, San Francisco. Morgan Kaufmann.

D. Heckerman, 1999. A tutorial on learning with Bayesian networks. In Jordan (1999), Seiten 301–354.

D. Heckerman, D. Geiger, und D. Chickering, 1995. Learning Bayesian networks: The combinations of knowledge and statistical data. *Machine Learning*, 20:197–243.

J.H. Holland, 1975. *Adaptation in Natural and Artificial Systems*. Univ. of Michigan Press, Ann Arbor.

R. Howard und J. Matheson, 1981. Influence diagrams. In R. Howard und J. Matheson, Herausgeber, *Readings on the Principles and Applications of Decision Analysis*, Band II, Seiten 721–762. Strategic Decisions Group, Menlo Park, CA.

X. Hu, G. Greenwood, und S. Ravichandran, 2000. Modeling epistasis interactions in fitness landscapes. In *Proceedings of the 2000 Congress on Evolutionary Computation*, Seiten 932–937, New Jersey. IEEE Press.

F.V. Jensen und F. Jensen, 1994. Optimal junction trees. In R.L. Mantaras und D. Poole, Herausgeber, *Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence*, Seiten 360–366, San Francisco. Morgan Kaufmann.

M.I. Jordan, Herausgeber, 1999. *Learning in Graphical Models*. MIT Press, Cambrigde.

M.I. Jordan, Z. Ghahramani, T.S. Jaakkola, und L.K. Saul, 1999. An introduction to variational methods for graphical models. In Jordan (1999), Seiten 105–162.

St.A. Kauffman und S. Levin, 1987. Towards a general theory of adaptive walks on rugged landscapes. *Journal of Theoretical Biology*, 128:11–45.

St.A. Kauffman und E. Weinberger, 1989. The $n-k$ model of rugged fitness landscapes and its application to maturation of the immune response. *Journal of Theoretical Biology*, 141(2).

B.W. Kernighan und S. Lin, 1970. An efficient heuristic procedure for partitioning graphs. *Bell System Technical Journal*, 2:291–307.

Y.-H. Kim und B.-R. Moon, 2000. A hybrid genetic search for graph partitioning based on lock gain. In *Proceedings of the Genetic and Evolutionary Computation Conference GECCO-2000*, Seiten 167–174, San Francisco. Morgan Kaufmann.

S. Kirkpatrick, C.D. Gelatt, und M.P. Vecchi, 1983. Optimization by simulated annealing. *Science*, 220:671–680.

St. L. Lauritzen, 1996. *Graphical Models.* Clarendon Press, Oxford.

J.A. Lozano, P. Larrañaga, M. Graña, und F. Albizuri, 1999. Genetic algorithms: bridging the convergence gap. *Theoretical Computer Science*, 229:11–22.

P. Merz und B. Freisleben, 2000. Fitness landscapes, memetic algorithms, and greedy operators for graph bipartitioning. *Evolutionary Computation*, 8(1):61–91.

H. Mühlenbein, 1991. Evolution in time and space - the parallel genetic algorithm. In G. Rawlins, Herausgeber, *Foundations of Genetic Algorithms*, Seiten 316–337, San Mateo. Morgan Kaufmann.

H. Mühlenbein, 1992. How Genetic Algorithms Really Work: Mutation and Hill-climbing. In R. Männer und B. Manderick, Herausgeber, *Parallel Problem Solving from Nature*, Seiten 15–26, Amsterdam. North-Holland.

H. Mühlenbein, 1998. The equation for response to selection and its use for prediction. *Evolutionary Computation*, 5:303–346.

H. Mühlenbein und Th. Mahnig, 1999a. Convergence theory and applications of the factorized distribution algorithm. *Journal of Computing and Information Technology*, 7:19–32.

H. Mühlenbein und Th. Mahnig, 1999b. FDA – a scalable evolutionary algorithm for the optimization of additively decomposed functions. *Evolutionary Computation*, 7(4): 353–376.

H. Mühlenbein und Th. Mahnig, 2000a. Evolutionary algorithms: From recombination to search distributions. In L. Kallel, B. Naudts, und A. Rogers, Herausgeber, *Theoretical Aspects of Evolutionary Computing*, Natural Computing, Seiten 137–176. Springer Verlag.

H. Mühlenbein und Th. Mahnig, 2000b. Evolutionary optimization using graphical models. *New Generation Computing*, 18(2):157–166.

H. Mühlenbein, Th. Mahnig, und A. Rodriguez Ochoa, 1999. Schemata, distributions and graphical models in evolutionary optimization. *Journal of Heuristics*, 5(2):215–247.

H. Mühlenbein und G. Paaß, 1996. From recombination of genes to the estimation of distributions i. binary parameters. In H.-M. Voigt, W. Ebeling, I. Rechenberg, und H.-P. Schwefel, Herausgeber, *Lecture Notes in Computer Science 1141: Parallel Problem Solving from Nature - PPSN IV*, Seiten 178–187, Berlin. Springer-Verlag.

H. Mühlenbein und D. Schlierkamp-Voosen, 1993a. Analysis of Selection, Mutation and Recombination in Genetic Algorithms. *Neural Network World*, 3:907–933.

H. Mühlenbein und D. Schlierkamp-Voosen, 1993b. Predictive Models for the Breeder Genetic Algorithm I. Continuous Parameter Optimization. *Evolutionary Computation*, 1:25–49.

H. Mühlenbein und H.-M. Voigt, 1996. Gene pool recombination in genetic algorithms. In J.P. Kelly und I.H. Osman, Herausgeber, *Metaheuristics: Theory and Applications*, Seiten 53–62, Norwell. Kluwer Academic Publisher.

B. Naudts, 1998. The effect of spin-flip symmetry on the performance of the simple GA. In A.E. Eiben, T. Bäck, M. Schoenauer, und H.-P. Schwefel, Herausgeber, *Proceedings of the Fifth International Conference on Parallel Problem Solving from Nature*, LNCS 1498, Seiten 67–76. Springer.

M.S. Ortiz, A.O. Rodriguez, S. Acid, und L.M. de Campos, 1999. Introducing the polytree approximation of distribution algorithm. In A. Rodriguez, M. Ortiz, und R. Santana, Herausgeber, *Second Symposium on Artifical Intelligence*, Seiten 360–367, Cuba. Institue of Cybernetics, Mathematics and Physics (ICIMAF).

J. Pearl, 1988. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufman, San Mateo.

M. Pelikan, D.E. Goldberg, und E. Cantú-Paz, 1999. BOA: The Bayesian Optimization Algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference GECCO-99*, Band 1, Seiten 525–532, San Francisco. Morgan Kaufmann.

J. Rissanen, 1987. Stochastic complexity. *Journal of the Royal Statistical Society B*, 49 (3):223–239.

R.B. Robbins, 1918. Some applications of mathematics to breeding problems III. *Genetics*, 3:375–389.

R.W. Robinson, 1977. Counting unlabeled acyclic digraphs. In C.H.C. Little, Herausgeber, *Combinatorial mathematics V*, Lecture Notes in Mathematics 622, Seiten 28–43, New York. Springer-Verlag.

G. Schwarz, 1978. Estimating the dimension of a model. *Annals of Statistics*, 6(2): 461–464.

D.J. Spiegelhalter und St.L. Lauritzen, 1990. Sequential updating of conditional probabilities on directed graphical structures. *Networks*, 20:579–605.

G. Syswerda, 1989. Uniform crossover in genetic algorithms. In H. Schaffer, Herausgeber, *3rd Int. Conf. on Genetic Algorithms*, Seiten 2–9, San Mateo. Morgan Kaufmann.

G. Syswerda, 1993. Simulated crossover in genetic algorithms. In L. D. Whitley, Herausgeber, *Foundations of Genetic Algorithms 2*, Seiten 239–255, San Mateo. Morgan Kaufmann.

I.J. Taneja, 1998. Generalized information measures and their applications. `http://mtm.ufsc.br/~taneja/bhtml/bhtml.html`.

R.K. Thompson und A.H. Wright, 1996. Additively decomposable fitness functions. Technischer Report, University of Montana.

P.J.M. van Laarhoven, 1987. *Theoretical and Computational Aspects of Simulated Annealing.* Doktorarbeit, Erasmus Universiteit Rotterdam.

V. Vapnik, 1998. *Statistical Learning Theory.* Wiley, New York.

I. Wegener, 1987. *The Complexity of Boolean Functions.* Teubner, Stuttgart.

E. D. Weinberger, 1996. $\mathcal{NP}$ completeness of Kauffman's N-k model, a tuneably rugged fitness landscape. Technischer Report 96–02–003, Santa Fe.

J. Whittaker, 1990. *Graphical Models in Applied Multivariate Statistics.* John Wiley & Sons, New York.

G. Winkler, 1995. *Image Analysis, Random Fields and Dynamic Monte Carlo Methods.* Springer, Berlin.

D.H. Wolpert und W. Macready, 1997. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 4:67–82.

S. Wright, 1932. The roles of mutation, inbreeding, crossbreeding and selection in evolution. In *Proc. 6th Int. Congr. on Genetics*, Seiten 356–366.

S. Wright, 1935. The analysis of variance and the correlations between relatives with respect to deviation from an optimum. *J. Genetics*, 30:243–256.