seit 1558

Friedrich-Schiller-Universität Jena

Fakultät für Mathematik und Informatik
INSTITUT FÜR INFORMATIK
Praktische Informatik / Musteranalyse

Merkmalauswahlverfahren zur Lokalisierung der Bindungsstellen von Transkriptionsfaktoren

Diplomarbeit im Fach Informatik

vorgelegt
von

Sven Schütze
geb. am 21. Dezember 1979 in Schmölln

Betreuer:	*Prof. Dr. E.G. Schukat-Talamazzini*
	R. Pudimat (Lehrstuhl für Bioinformatik)
Beginn der Arbeit:	*24. Januar 2005*
Abgabe der Arbeit:	*19. Juli 2005*

Kurzfassung

Die vorliegende Arbeit beschäftigt sich mit dem Einsatz von Merkmalauswahlverfahren zur Verbesserung der Leistung eines Systems zur Lokalisierung der Bindungsstellen von Transkriptionsfaktoren (TFBS).

Dazu wird zuerst ein Überblick über bereits aus der Literatur bekannte Merkmalauswahlverfahren gegeben, um zu prüfen, welche sich davon für diese Problemstellung eignen. Danach werden noch einige eigene Verfahren vorgeschlagen und zusammen mit einigen bekannten getestet. Für die Tests werden neben den TFBS-Daten auch noch aus der Literatur bekannte Sonardaten heranngezogen, um eine Vergleichbarkeit mit anderen Arbeiten zu erreichen.

Im Unterschied zu den meisten bisherigen Arbeiten werden die Tests aber nicht nur mit dem Ziel einer möglichst hohen Güte bei gegebener Merkmalzahl erfolgen. Im Mittelpunkt dieser Arbeit steht die Effizienz, also eine möglichst hohe Güte bei gegebener Berechnungszahl.

Im Anschluss an die Vorstellung der Testergebnisse wird noch eine kurze Zusammenfassung und ein Ausblick gegeben.

Inhaltsverzeichnis

Kapitel 1

Einführung

In diesem Kapitel soll eine Annäherung an das Anwendungsgebiet der vorliegenden Arbeit sowie eine Übersicht über den inhaltlichen Aufbau der vorliegenden schriftlichen Ausarbeitung gegeben werden.

1.1 Hintergrund der Arbeit

Eine grundlegende Herausforderung der aktuellen biologischen Forschung ist das Verständnis der Regulation der Aktivität von Genen. Dass heißt, es wird versucht den Steuerungsmechanismus zu verstehen, welcher dafür sorgt, dass manche Gene nur zu bestimmten Zeitpunkten aktiv sind. Aktiv bedeutet dabei, dass von dem entsprechenden DNA-Abschnitt (Abbildung 1.1) eine RNA transkribiert und bis hin zum fertigen Protein weiterverarbeitet wird. Diese Steuerung erfolgt über eine Art Genschalter. Diese Genschalter sind spezielle Bereiche in der DNA, die meist vor dem eigentlichen zu transkribierenden Bereich liegen. Dabei gibt es Bereiche welche immer vorhanden seinen müssen um eine Transkription überhaupt zu ermöglichen, und andere, welche nur unter bestimmten Umständen eine Transkription ermöglichen.

Diese Bereiche nennt man Promotoren. An diese Promotorregion binden sich u. a. zahlreiche Transkriptionsfaktoren. [Bro03]

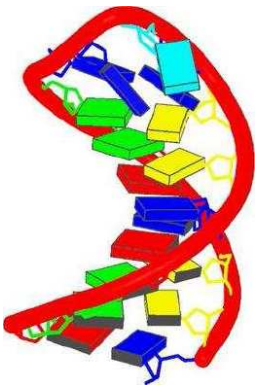

Abbildung 1.1: Schematische Darstellung eines DNA-Abschnitts

Die zu lösende Aufgabe besteht nun darin, diese Bindungsstellen der Transkriptionsfaktoren (kurz TFBS) zu lokalisieren.

Trotz der relativ starken Sequenz-Ähnlichkeit unter den Bindungsstellen bestimmter Transkriptionsfaktoren ist die Entwicklung von hoch spezifischen und genauen computergestützten Detektionsansätzen noch ein ungelöstes Problem. Da die verhältnismäßig kurzen Sequenzen, die an Bindungsstellen auftreten, einen bestimmten Grad an Veränderlichkeit zeigen können und überall in einem Genom, ohne regelnde Funktionalität zu haben, zufällig anwesend sein können, leiden die meisten gegenwärtigen Lösungen unter einer hohen falsch-positiven Rate (Erkennungsfehler 1. Ordnung).

1.2 Anliegen der Arbeit

In der Arbeit von Rainer Pudimat et al. ([Pud04]) wird ein Lösungsansatz vorgestellt, welcher auf einem Bayesnetz basiert. Es wird dort festgestellt, dass eine höhere Erkennungsleistung des Bayesnetzes dadurch erreicht werden kann, dass anstatt nur die Basen der einzelnen Positionen als Merkmale zu verwenden, Merkmalen verschiedenen Typs genutzt werden. Es ist aber auch angemerkt, dass ein Algorithmus zur Auswahl der besten Untermenge aus der Menge der zur Verfügung stehenden Merkmale noch nicht entwickelt ist. An dieser Stelle setzt diese Arbeit an.

Es soll also zuerst überprüft werden, mit welchem der aus der Literatur bekannten Algorithmen die besten Ergebnisse erzielt werden können. Danach wird versucht mit eigenen Algorithmen diese Ergebnisse möglichst noch zu verbessern.

1.3 Aufbau der Arbeit

Im Anschluss an dieses einleitende Kapitel folgt in den Kapiteln 2, 3 und 4 eine Zusammenfassung der aus der Literatur bekannten Algorithmen und eine Vorstellung eigener Algorithmen zur Merkmalauswahl. Dabei widmet sich jedes Kapitel einer Gruppe von Algorithmen. Einige der Algorithmen aus den Kapiteln 2 bis 4 werden dann im Kapitel 5 vergleichend evaluiert. Abgeschlossen wird die Arbeit mit einem nochmals zusammenfassenden Kapitel 6, welches auch einen Ausblick auf mögliche Verbesserungsansätze geben soll.

Kapitel 2

Sequentielle Algorithmen

In diesem und den folgenden Kapiteln sollen einige Algorithmen zur Merkmalauswahl – engl.: Feature Subset Selection (FSS) – vorgestellt werden. Nach Pudil et al. ([Pud94a, Kap. 1, Abs. 1]) versteht man unter Merkmalauswahl die Auswahl des Inputs eines Mustererkennungssystems. Folglich ist das Hauptziel der Merkmalauswahl, aus einer gegebenen Menge X mit D Merkmalen eine Untermenge Y ($Y \subseteq X$) mit d Merkmalen ($\text{card}(Y) = d < D$) auszuwählen. Dabei sollte sich die Erkennungsrate des Mustererkennungssystems nicht nennenswert verschlechtern, sondern sie sollte sich so weit wie möglich verbessern.

Ein wesentlicher Punkt bei der Merkmalauswahl ist die Bestimmung der Güte einer Merkmalmenge Y. Im Folgenden wird die Funktion zur Bestimmung der Güte mit $J(Y)$ bezeichnet. Sie kann z. B. klassifikatorabhängig durch das Anlernen und Testen des resultierenden Mustererkennungssystems bestimmt werden. Es gäbe aber auch die Möglichkeit einer klassifikatorunabhängigen Berechnung der Güte, welche aber in dieser Arbeit nicht genutzt werden soll.

Eine wichtige Fragestellung zur Auswahl eines Algorithmus für ein bestimmtes Problem ist die Monotonie des Güte-Kriteriums. Dabei versteht man nach [Yu93, Kap. 1] unter Monotonie, dass gilt:

$$X_s \supseteq X_t \Rightarrow J(X_s) \geq J(X_t).$$

In Worten heißt dies, dass die Güte einer Merkmaluntermenge nicht besser sein darf, als die einer beliebigen Obermengen. Im Folgenden werden sowohl Algorithmen vorgestellt, welche ein monotones Güte-Kriterium voraussetzen als auch solche die nicht darauf angewiesen sind.

Mit dieser Güte-Funktion wird die Merkmalauswahl zu einem Suchproblem. Gesucht ist dann die Merkmaluntermenge Y, für die $J(Y)$ maximal wird. Formal bedeutet dies also:

$$Y^* = \underset{Y \subseteq X}{\text{argmax}}\, J(Y)$$

Bei der folgenden Vorstellung der Algorithmen wird eine Einteilung in sequentielle und randomisierte Algorithmen sowie in Graph-Suchalgorithmen vorgenommen, welche sich in der Kapiteleinteilung widerspiegelt. Als erstes sollen in diesem Kapitel die sequentiellen Algorithmen vorgestellt werden.

Die sequentiellen Algorithmen versuchen sich durch wiederholtes Hinzufügen und/oder Entfernen einzelner oder mehrerer Merkmale Schritt für Schritt zu einer Merkmaluntermenge vorzuarbeiten, welche der optimalen in der Güte so nahe wie möglich kommt. Bei vielen sequentiellen Algorithmen gibt es zwei Varianten:

1. Start mit leerer Menge und Hinzufügen (Forward) und
2. Start mit voller Menge und Entfernen (Backward).

Alle anderen Algorithmen versuchen diese Varianten zu kombinieren.

2.1 Sequential Forward (Backward) Selection

Die einfachsten sequentiellen Algorithmen sind Sequential Forward Selection (SFS) und die entsprechende Rückwärts-Variante Sequential Backward Selection (SBS). Das Verfahren wurde erstmals von A. W. Whitney 1971 [Whi71] genutzt (vgl. auch [Ste76], [Kit78])

SFS beginnt mit einer leeren Menge. Danach wird Schritt für Schritt jeweils das Merkmal aus der ungenutzten Merkmalmenge hinzugenommen, welches die Güte der Merkmaluntermenge am meisten steigert. Der Abbruch des Algorithmus erfolgt, sobald die vorgegebene Anzahl d von Merkmalen ausgewählt wurde. Danach wird in diesem Fall die beste Merkmaluntermenge mit $k = d$ Merkmalen zurückgegeben. Man kann den Abbruch aber auch nach einer vorgegebenen Anzahl von Berechnungen durchführen. In diesem Fall wird die Merkmaluntermenge zurückgegeben, welche im gesamten Verlauf der Suche die höchste Güte erzielt hat. Im letztgenannten Fall wird bei Gleichheit der Güten mehrerer Merkmalmengen die mit der kleineren Kardinalität ausgewählt.

SBS beginnt entsprechend mit einer Merkmalmenge, welche alle zur Verfügung stehenden Merkmale enthält. Danach wird jeweils das Merkmal entfernt, welches die Güte am wenigsten verringert. Zum Schluß wird wieder die beste Merkmalmenge noch obigen Kriterien ausgewählt.

Der Pseudocode des SFS-Algorithmus ist Abbildung 2.1 zu entnehmen. Die zeitliche Komplexität beider Verfahren beträgt $\Theta(d \cdot D)$.

Einschätzungen

Beide Verfahren sind in etwa gleichwertig, wobei die Vorwärtsmethode erwartungsgemäß schneller als die Rückwärtsmethode ist (vgl. [Jai96, Kap. 3]). Außerdem leiden beide Algorithmen unter Vernestungsproblemen (vgl. [Jai96, Kap. 3], [Som99, Kap. 1]). Dabei versteht man unter dem Begriff des Vernestungsproblems, dass ein Merkmal, welches einmal „gierig“ hinzugefügt (entnommen) wurde, nicht wieder entfernt (hinzugefügt) werden kann, selbst wenn dies die Güte verbessern würde. Es handelt sich dabei also um ein Problem welches aus der „gierigen“ Natur des SFS-Algorithmus erwächst, indem frühzeitig Entscheidungen getroffen werden, welche später nicht wieder korrigiert werden können. Vorteil beider Verfahren, und im speziellen von SFS, ist ihre Geschwindigkeit, welche auf der oben genannten zeitliche Komplexität von $\Theta(d \cdot D)$ basiert.

Die Ergebnisse beider Algorithmen sind aber unter den sequentiellen Algorithmen die schlechtesten, was dem einfachen Aufbau zu schulden ist, welcher mit dem Vernestungsproblem nicht umgehen kann. Daher finden diese Verfahren in der Praxis eher weniger Anwendung, sondern eher die folgenden verbesserten Ansätze.

SFS

Input: X Merkmalmenge

Output: Y ausgewählte Merkmaluntermenge

Pseudocode:

1. $Y_0 = \emptyset$
2. **FOR** $k = 1$ **TO** d
   (a) $x^+ := \text{argmax}_{x \in X \setminus Y_{k-1}} J(Y_{k-1} \cup \{x\})$
   (b) $Y_k := Y_{k-1} \cup \{x^+\}$
3. **NEXT** k
4. Gib $\text{argmax}_{Y_i, i \in \{1 \ldots d\}} J(Y_i)$ zurück

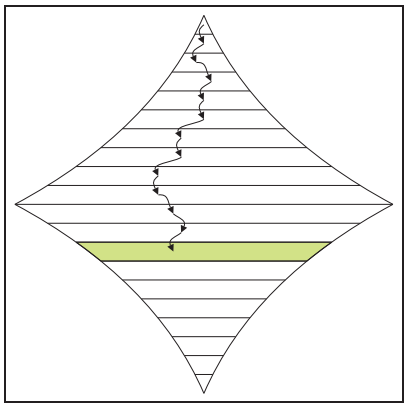

Abbildung 2.1: Pseudocode nach [Kun99, Fig. 4] und Schematisierung des SFS-Algorithmus (d ist die gewünschte Merkmalzahl)

2.2 Generalized Sequential Forward (Backward) Selection

Generalized Sequential Forward Selection (GSFS(g)) bzw. Generalized Sequential Backward Selection (GSBS(g)) sind Verallgemeinerungen der entsprechenden einfachen Varianten SFS bzw. SBS (vgl. [Kit78]). Generalisierung bedeutet hierbei, dass jetzt bei jeder Iteration nicht mehr nur genau ein Merkmal hinzugefügt bzw. entfernt wird, sondern genau g Merkmale (vgl. [Kud00a, S. 4]). Der Grad der Generalisierung hängt also vom Parameter g ab. Hierbei sind die folgenden Beziehungen offensichtlich:

1. GSFS(1) = SFS
2. GSBS(1) = SBS

Der zum GSFS-Algorithmus gehörende Pseudocode ist Abbildung 2.2 zu entnehmen. Die zeitliche Komplexität beider Verfahren beträgt $\Theta(\frac{d}{g} \cdot D^g)$.

Einschätzungen

Auch hier sind die Algorithmen wieder in etwa gleichwertig, mit einer etwas schnelleren Vorwärtsvariante ([Jai96, Kap. 3]). Da bei diesen Algorithmen einmal aufgenommene (entfernte) Merkmale ebenfalls nicht wieder entfernt (aufgenommen) werden können, leiden sie ebenso wie SFS und SBS unter einem Vernestungsproblem ([Jai96, Kap. 3]). Die besseren Ergebnisse haben den Preis, dass man im Voraus einen Parameter g festlegen muss, welcher den Grad der Generalisierung bestimmt. [Som99, Kap. 3] Ein zu großes g erfordert einen eventuell zu großen Rechenaufwand, ein zu kleines g verschenkt eventuell ein Mehr an Güte.

GSFS

Input:	X	Merkmalmenge
	g	Grad der Generalisierung
Output:	Y	ausgewählte Merkmaluntermenge

Pseudocode:

1. $Y_0 = \emptyset$
2. **FOR** $k = 1$ **TO** d/g
   (a) $X^+ := \text{argmax}_{X \subseteq X \setminus Y_{k-1}, \text{card}(X)=g} J(Y_{k-1} \cup X)$
   (b) $Y_k := Y_{k-1} \cup X^+$
3. **NEXT** k
4. Gib $\text{argmax}_{Y_i, i \in \{1 \ldots d/g\}} J(Y_i)$ zurück

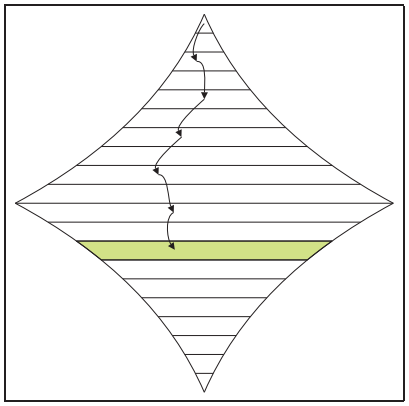

Abbildung 2.2: Pseudocode und Schematisierung des GSFS-Algorithmus (d ist die gewünschte Merkmalzahl, $d = g \cdot n$, $n \in \mathbb{N}$)

2.3 Plus l – Take Away r

Plus l – Take Away r (PTA(l, r)) wurde 1976 von S. D. Stearns in [Ste76] eingeführt (vgl. auch [Pud94a, Kap. 2.1], [Kit78]). Bei diesem Verfahren handelt es sich um eine Kombination aus SFS und SBS ([Pud94a, Kap. 2.1 Abs. 2]). Es werden nicht mehr nur ausschließlich Merkmale zur Merkmaluntermenge hinzugefügt bzw. entfernt, sondern hier ist jetzt beides möglich. Dies wird realisiert, indem abwechselnd l mal genau ein Merkmal hinzugenommen und anschließend r mal genau ein Merkmal wieder entfernt wird. Der Abbruch des Algorithmus erfolgt auch hier wieder, wenn die gewünschte Größe der Merkmaluntermenge erreicht ist. Aber auch hier könnte man den Abbruch nach einer vorgegebenen Anzahl von Berechnungen durchführen.

Für $l > r$ resultiert eine Vorwärts-Methode und für $l < r$ eine Rückwärts-Methode ([Pud94a, Kap. 2.1]). Dabei sind l und r im Vorfeld festzulegen.

Hierbei sind wieder die folgenden Beziehungen offensichtlich:

1. PTA(1, 0) = SFS
2. PTA(0, 1) = SBS

Da jetzt die Möglichkeit besteht, einmal hinzugefügte Elemente wieder zu entfernen, ist dieser Algorithmus nicht vom Vernestungsproblem betroffen.

In Abbildung 2.3 ist der Pseudocode des PTA(l, r)-Algorithmus dargestellt. Die zeitliche Komplexität dieses Verfahrens beträgt $\Theta(\frac{d}{l-r} \cdot (l + r) \cdot D)$.

Einschätzungen

Wie oben schon beschrieben, ist dieser Algorithmus, anders als SFS, SBS und ihre generalisierten Versionen, nicht vom Vernestungsproblem betroffen. Dies wird aber damit „erkauft“, dass zwei zusätzliche Parameter l und r angegeben werden müssen. Es gibt

PTA(l, r)

Input: Y Merkmalmenge
l Inklusions-Schritte
r Exklusions-Schritte

Output: X_k ausgewählte Merkmaluntermenge

Pseudocode:

IF $l > r$ **THEN** $k := 0$; $X_0 := \emptyset$; **GOTO** Step1
ELSE $k := D$; $X_D := Y$; **GOTO** Step2

1. **Step1** (Inklusion):
   (a) Wiederhole l mal
      i. $x^+ := \text{argmax}_{x \in Y \setminus X_k} J(X_k \cup \{x\})$
      ii. $X_{k+1} := X_k \cup \{x^+\}$; $k := k + 1$
      iii. **IF** $k = d$ **THEN GOTO** Stop
   (b) **GOTO** Step2
2. **Step2** (Exklusion):
   (a) Wiederhole r mal
      i. $x^- := \text{argmax}_{x \in X_k} J(X_k \setminus \{x\})$
      ii. $X_{k-1} := X_k \setminus \{x^-\}$; $k := k - 1$
      iii. **IF** $k = d$ **THEN GOTO** Stop
   (b) **GOTO** Step1
3. **Stop**: Gib X_k zurück

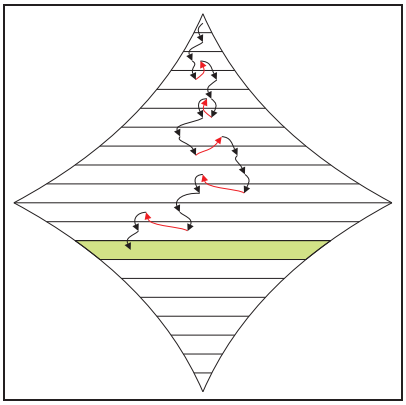

Abbildung 2.3: Pseudocode nach [Pud94a, Kap. 2.1] und Schematisierung des PTA(l, r)-Algortihmus (d ist die gewünschte Merkmalzahl; D die Anzahl der verfügbaren Merkmale)

aber keine Möglichkeit, die optimalen Werte für l und r vorherzusagen ([Pud94a, Kap. 2.1]).

Jain et al. kommen in [Jai96, Kap. 3] bei ihren Untersuchungen zu dem Ergebnis, dass PTA(l, r) nahezu optimale Ergebnisse liefert aber viel langsamer als Branch & Bound ist.

Dieser Algorithmus kann außerdem mit nichtmonotonen Güte-Kriterien zur Bewertung der Merkmaluntermenge umgehen ([Pud94a, Kap. 2.3 Abs. 2]).

2.4 Generalized Plus l – Take Away r

Der Algorithmus Generalized Plus l – Take Away r (GPTA(l, r)) funktioniert ähnlich wie PTA(l, r). Jedoch werden hier nicht l SFS-Schritte und r SBS-Schritte angewandt, sondern ein GSFS(l)-Schritt und ein GSBS(r)-Schritt (vgl. [Kud00a, Kap. 2]). GPTA(l, r) ist also nicht mehr eine Kombination aus SFS und SBS, sondern aus GSFS(l) und GSBS(r). Dieser Algorithmus ist ebenso nicht vom Vernestungsproblem betroffen (aus dem gleichen Grund wie PTA(l, r)).

Der Pseudocode des GPTA(l, r)-Algorithmus ist Abbildung 2.4 zu entnehmen. Die

zeitliche Komplexität dieses Verfahrens beträgt $\Theta(D^{max\{l+1,r+1\}})$ (vgl. [Kud00a, Table 1]).

GPTA(l, r)

Input:
- Y Merkmalmenge
- l Inklusions-Schritte
- r Exklusions-Schritte

Output:
- X_k ausgewählte Merkmaluntermenge

Pseudocode:

IF $l > r$ **THEN** $k := 0$; $X_0 := \emptyset$; **GOTO** Step1
ELSE $k := D$; $X_D := Y$; **GOTO** Step2

1. **Step1** (Inklusion):
   (a) $X^+ := \text{argmax}_{X \subseteq Y \setminus X_k, \text{card}(X)=l} J(X_k \cup X)$
   (b) $X_{k+1} := X_k + X^+$; $k := k + 1$
   (c) **IF** $k = d$ **THEN GOTO** Stop
   (d) **GOTO** Step2
2. **Step2** (Exklusion):
   (a) $X^- := \text{argmax}_{X \subseteq X_k, \text{card}(X)=r} J(X_k \setminus X)$
   (b) $X_{k-1} := X_k \setminus X^-$; $k := k - 1$
   (c) **IF** $k = d$ **THEN GOTO** Stop
   (d) **GOTO** Step1
3. **Stop**: Gib X_k zurück

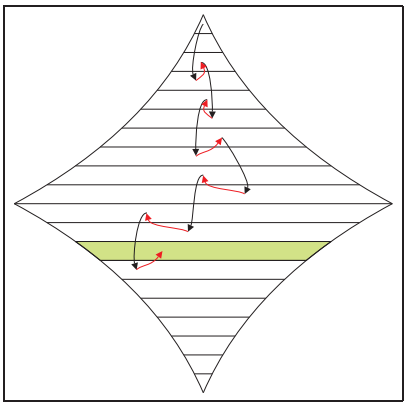

Abbildung 2.4: Pseudocode und Schematisierung des GPTA(l, r)-Algortihmus (d ist die gewünschte Merkmalzahl; D die Anzahl der verfügbaren Merkmale). Das Terminierungsmanagement wurde vereinfacht! Um eine Terminierung sicherzustellen müssen ggf. l und/oder r am Ende angepasst werden, um die Zielgröße d zu treffen.

Einschätzungen

Kudo et al. kommen bei ihren Untersuchungen in [Kud00a, Tabel 5] zu dem Schluss, dass GPTA(l, r) in etwa dieselben guten Leistungen wie die im Folgenden beschriebenen Sequential Forward (Backward) Floating Selection-Algorithmen (SFFS/SBFS) zeigt, aber deutlich zeitaufwendiger ist. Nach diesen Untersuchungen von Kudo et al. [Kud00a, Kap. 4.4.2] ist GPTA(1, 2) der beste einzelne sequentielle Algorithmus. In Kombination (d.h. beide ausführen und besseres Ergebnis verwenden) sind dort aber SFFS und SBFS besser.

2.5 Sequential Forward (Backward) Floating Selection

Sequential Forward Floating Selection (SFFS) und Sequential Backward Floating Selection (SBFS) wurden 1994 von Pudil et al. in [Pud94b] eingeführt. Teilweise werden sie auch „... Search“ genannt.

SFFS

Input: Y Merkmalmenge

Output: X_k ausgewählte Merkmaluntermenge

Pseudocode:

$X_0 := \emptyset$; $k := 0$

1. **Step1** (Inklusion):
   (a) **IF** $k = d$ **THEN GOTO** Stop
   (b) $x^+ := \text{argmax}_{x \in Y \setminus X_k} J(X_k \cup \{x\})$
   (c) $X_{k+1} := X_k \cup \{x^+\}$; $k := k + 1$
2. **Step2** (Bedingte Exklusion):
   (a) $x^- := \text{argmax}_{x \in X_k} J(X_k \setminus \{x\})$
   (b) **IF** $J(X_k \setminus \{x^-\}) > J(X_{k-1})$ **THEN**
      i. $X_{k-1} := X_k \setminus \{x^-\}$
      ii. $k := k - 1$
      iii. **GOTO** Step2

   ELSE GOTO Step1
3. **Stop**: Gib X_k zurück

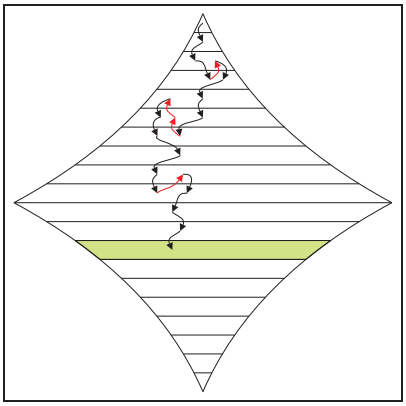

Abbildung 2.5: Pseudocode nach [Pud94a, Kap. 2.2] und Schematisierung des SFFS-Algorithmus (d ist die benötigte Merkmalzahl)

Sie sind verwandt mit PTA(l, r), aber die Anzahl der Vorwärts- und Rückwärts-Schritte wird dynamisch kontrolliert, anstatt fest vorgegeben zu sein ([Pud94a, Kap. 1, Abs. 6]). Der SFFS-Algorithmus setzt dies um, indem er nach jedem Vorwärtsschritt solange Rückwärtsschritte durchführt, wie sich dadurch Merkmalmengen mit Güten finden lassen, welche besser als die bisher für diese Größe gefundenen sind („Floating Search“). Dies bedeutet auch, dass überhaupt kein Rückwärtsschritt durchgeführt wird, wenn dies die Güte nicht verbessert. Somit ist es nicht nötig, irgendeinen Parameter in Vorfeld bestimmen und festlegen zu müssen.

Der Algorithmus terminiert, sobald die Merkmalmenge eine vorgegebene Größe erreicht. Eine mögliche Modifikation dieses Verfahrens besteht darin, den Abbruch nicht gleich bei erstmaliger Erfüllung des Kriteriums $k = d$ zu veranlassen. Der Algorithmus könnte vielmehr mehrmals über diesen Punkt hinweglaufen, was man z. B. durch folgendes Abbruchkriterium bewirken kann: $k = d + \Delta$ ($\Delta \leq D - d$).

Es ist aber auch wieder möglich den Abbruch nach einer vorgegebenen Anzahl von Berechnungen durchzuführen.

Den Pseudocode des SFFS-Algorithmus kann man Abbildung 2.5 entnehmen. Die zeitliche Komplexität dieses Verfahrens beträgt $O(2^D)$ (vgl. [Kud00a, Table 1]). Es ist also im worst-case keine Verbesserung gegenüber einer exhaustiven Suche zu erreichen. Im Normalfall ist der SFFS-Algorithmus aber deutlich schneller.

Einschätzungen

In den Untersuchungen von Jain et al. ([Jai96, Kap. 3]) lieferten SFFS und SFBS vergleichbare Ergebnisse wie Branch & Bound, waren aber meist schneller. Auch diese beiden Algorithmen können mit nichtmonotonen Güte-Kriterien umgehen ([Pud94a, Kap. 2.3 Abs. 2]).

Abbildung 2.6 sind einige vergleichende Ergebnisse verschiedener sequentieller Algorithmen und Exhaustiver Suche zu entnehmen ([Pud94a]). Es ist zu erkennen, dass SFFS/SBFS meist, aber nicht immer, der Beste unter den bisherigen sequentiellen Algorithmen ist.

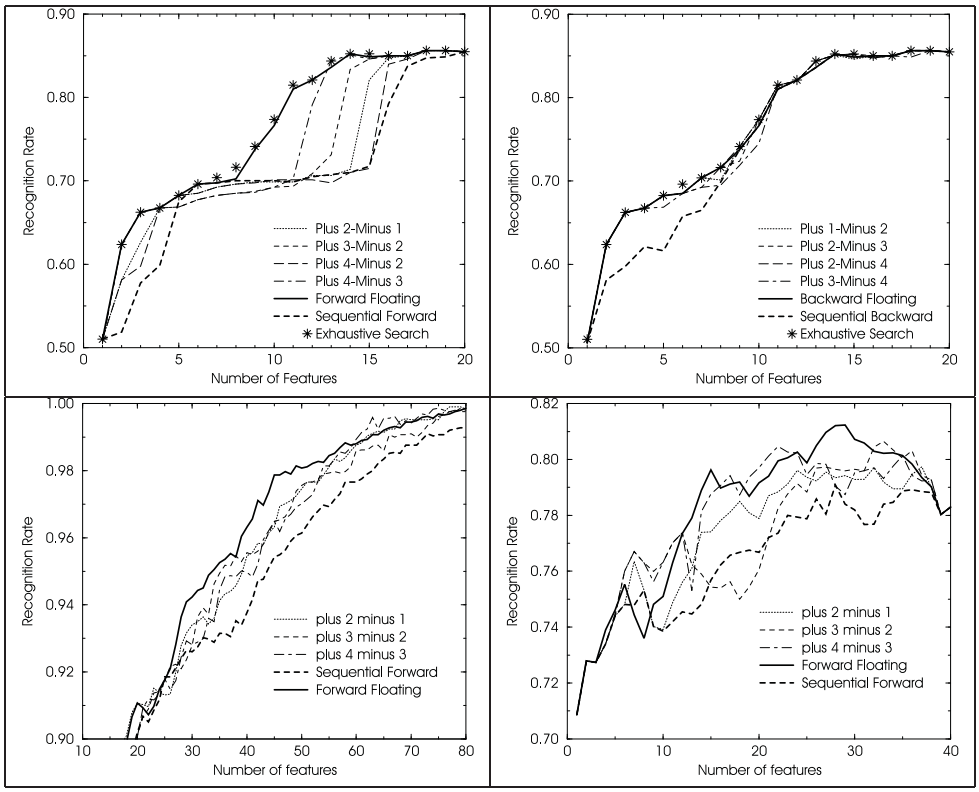

Abbildung 2.6: Vergleich verschiedener Algorithmen anhand ihrer Ergebnisse (aus [Pud94a]) an einem Diagnoseproblem (oben) und einem Dokumentenerkennungsproblem (unten)

2.6 Adaptive Sequential Forward (Backward) Floating Selection

Adaptive Sequential Forward Floating Selection (ASFFS(r_{max}, b)) und Adaptive Sequential Backward Floating Selection (ASBFS(r_{max}, b)) sind Weiterentwicklungen von SFFS und SBFS. Sie werden z.B. in [Som99] beschrieben.

Zusätzlich zur Floating-Suche im Rückwärtsschritt wird eine komplexere Suche im Vorwärtsschritt angewandt. Dass heißt, es wird statt eines SFS-Schrittes ein GSFS(o)-Schritt durchgeführt, wobei o durch r nach oben begrenzt ist. Die Variable r wird wiederum durch den Parameter r_{max} nach oben begrenzt und wird so festgelegt, dass, wenn k in einer b-Umgebung der gewünschten Merkmalzahl d ist, r nahe bei r_{max} liegt und sonst klein ist (siehe Abbildung 2.7).

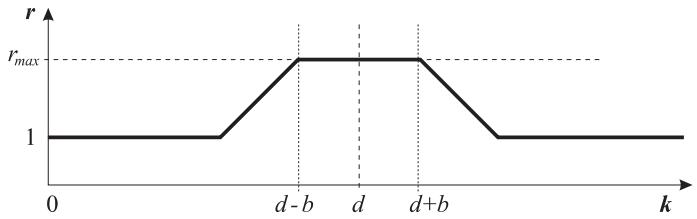

Abbildung 2.7: Einfluss der Nutzerparameter r_{max} und b auf die Variable r im ASFFS-Algorithmus (aus [Kud00b, Fig. 2])

Der Abbruch sollte bei diesem Algorithmus nicht gleich beim erstmaligen Erreichen der Zielgröße d der Merkmalmenge durchgeführt werden. Der Algorithmus sollte mehrmals über diesen Punkt hinweglaufen (im folgenden Flussdiagramm ist dies durch das Abbruchkriterium $k \geq d + \Delta$ dargestellt), da dies unter Umständen zu besseren Ergebnissen führt [Kud00b, Kap. 3.2].

Auch hier wäre es wieder möglich den Abbruch nicht über die Merkmalmengengöße sondern über die Anzahl der vorgenommenen Berechnungen zu steuern.

Das Flussdiagramm des ASFFS-Algorithmus ([Som99, Fig. 3], [Kud00b, Fig. 3]) ist Abbildung 2.8 zu entnehmen.

Einschätzungen

ASFFS liefert im Allgemeinen bessere Ergebnisse als SFFS, aber manchmal ist auch SFFS besser im Ergebnis ([Kud00b, Kap. 5.1]). Abbildung 2.9 zeigt einige Vergleiche bezüglich des Ergebnisses zwischen ASFFS/ASBFS und SFFS/SBFS aus [Kud00b]. Es ist zu erkennen, dass ASFFS bzw. ASBFS meist, aber nicht immer, bessere Ergebnisse als SFFS bzw. SBFS liefert.

Dafür ist aber ASFFS bzw. ASBFS deutlich langsamer als SFFS bzw. SFBS, da diese Algorithmen zur Klasse der generalisierten Algorithmen gehören, also mehr als ein Merkmal pro Schritt hinzufügen oder entfernen. Kudo et al. kommen in [Kud00b, Kap. 5.1] zu dem Ergebnis, dass ASFFS bzw. ASBFS in ihrem Beispiel 1000-mal langsamer als SFFS bzw. SBFS ist. Abbildung 2.10 zeigt einige Vergleiche bezüglich des Rechenaufwands zwischen ASFFS und SFFS von Somol et al. aus [Som99].

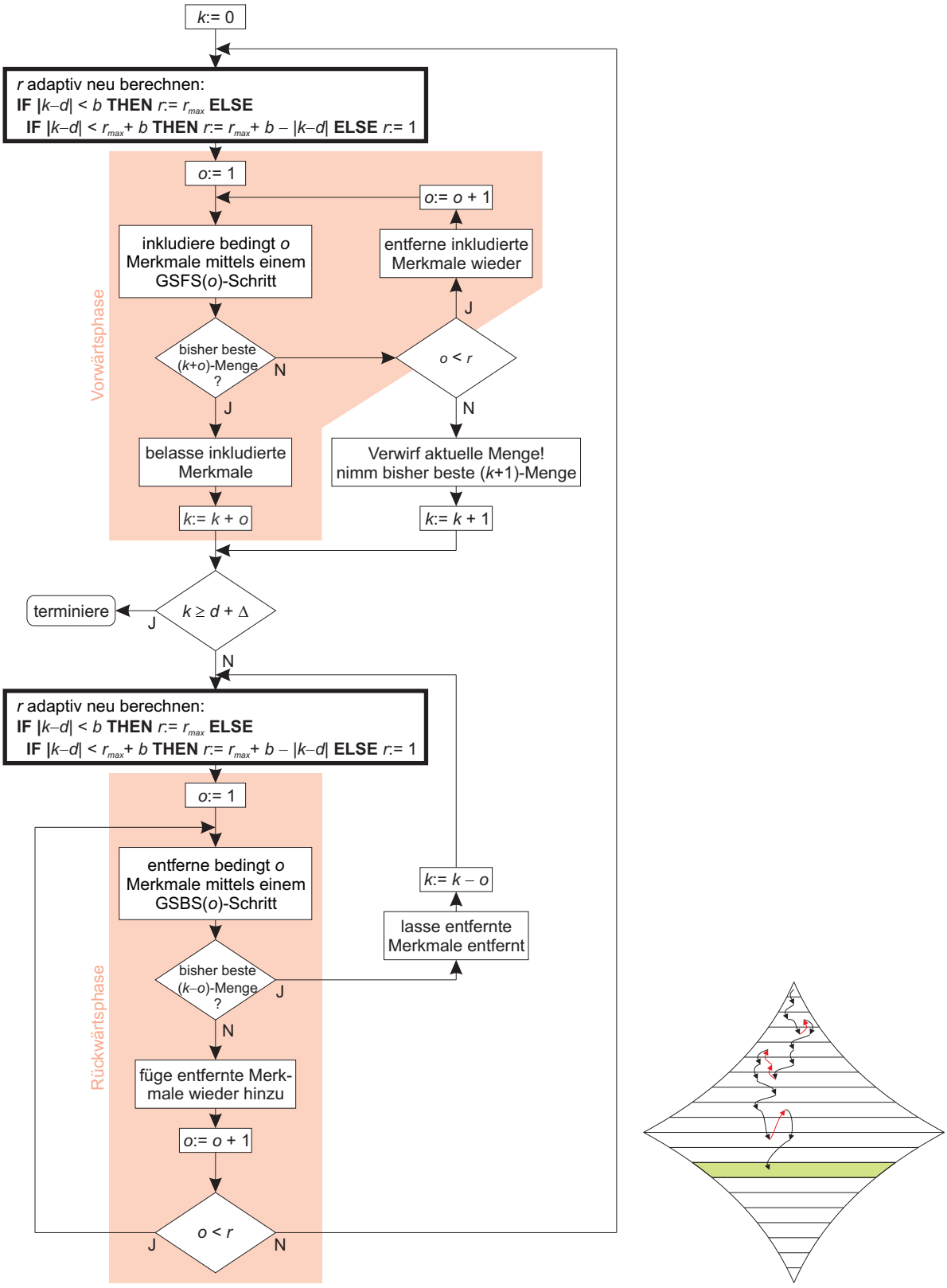

Abbildung 2.8: Flussdiagramm nach [Som99, Fig. 3] und [Kud00b, Fig. 3] und Schematisierung des ASFFS(r_{max}, b)-Algorithmus

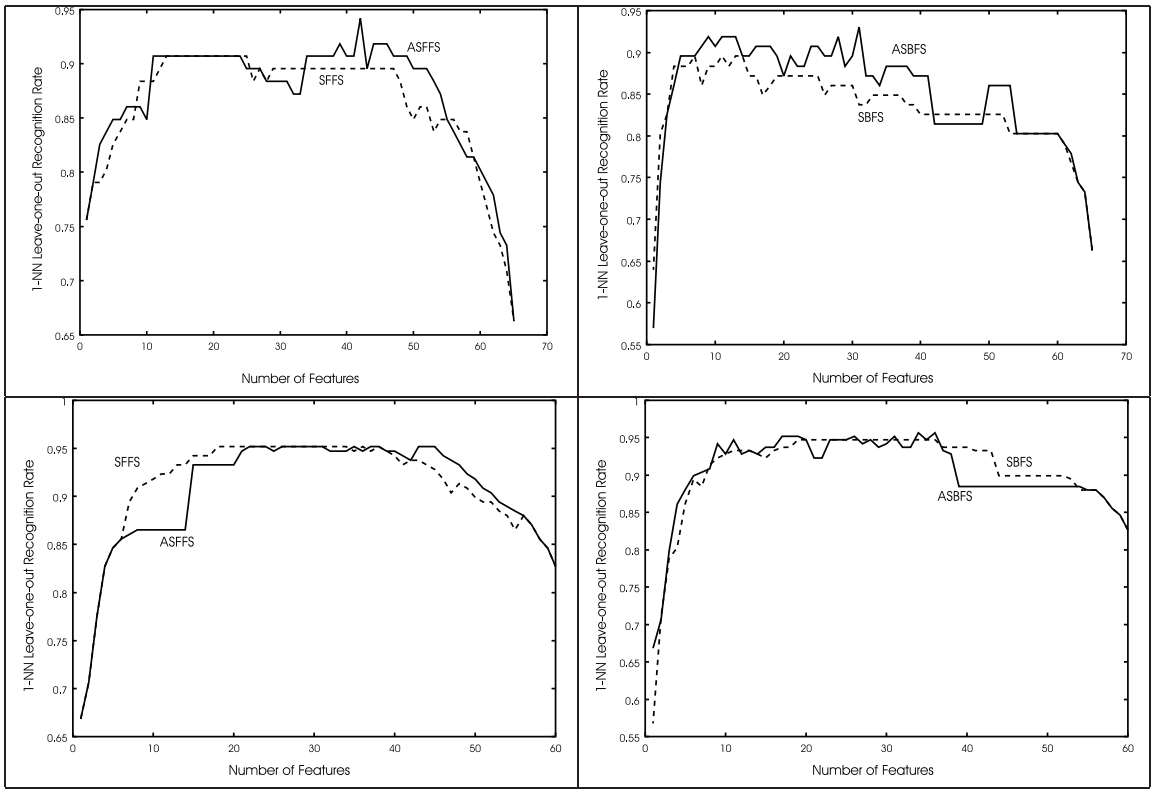

Abbildung 2.9: Vergleich zw. ASFFS/ASBFS(r_{max}, b) und SFFS/SBFS (Abbruchkriterium $k = D$) anhand ihrer Ergebnisse an Mammographiedaten (oben) und Sonardaten (unten) (aus [Kud00b, Fig. 5,6,8,9])

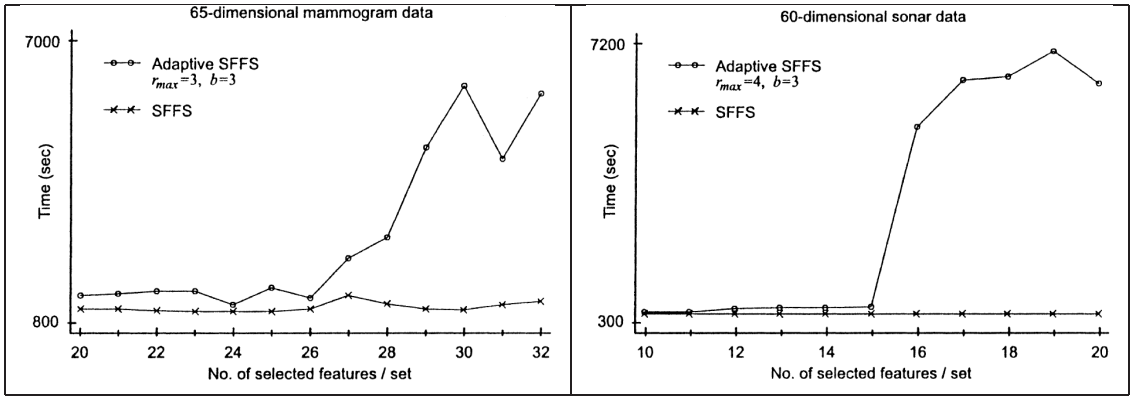

Abbildung 2.10: Vergleich zw. ASFFS(r_{max}, b) und SFFS (Abbruchkriterium $k = D$) anhand ihrer Rechenzeit an Mammographiedaten (links) und Sonardaten (rechts) (aus [Som99, Fig. 4, 5])

2.7 Max-Min Search

Die Methode Max-Min Search (MM) wurde 1977 von Backer et al. in [Bac77] eingeführt (siehe auch [Kit78]). Pudil et al. zeigten aber in [Pud93], dass die theoretische Grundlage, auf der MM basiert, fehlerhaft ist. Daher ist dieser Algorithmus nicht weit verbreitet und soll hier auch nicht detailliert betrachtet werden. Es sei nur gesagt, dass der Algorithmus

mit SFS, basierend auf individuellen und paarweisen Vorzügen der Merkmale, arbeitet ([Ser01, Kap. II]).

Einschätzungen

Der Max-Min-Algorithmus arbeitet sehr schnell, liefert aber im Vergleich zu anderen Algorithmen schlechte Ergebnisse ([Jai96, Kap. 3]).

2.8 ParallelSFFS

Bei den in dieser Arbeit bereits vorgestellten Verfahren und denen die später noch vorgestellt werden, gibt es zwei verschiedene Tendenzen. Zum einen sind dies die in diesem Kapitel beschriebenen sequentiellen Verfahren, welche dazu neigen eher schnell in die Tiefe zu gehen, also relativ schnell relativ große Merkmalmengen generieren. Zum anderen sind da die Verfahren A^* bzw. BestFirst im Allgemeinen (siehe Kapitel 4), welche dazu neigen eher erst in die Breite zu gehen, also erst relativ viele kleine Merkmalmengen zu untersuchen, ehe große untersucht werden.

Beide Gruppen erzielen mit ihrem Vorgehen recht akzeptable Ergebnisse (vgl. auch Kapitel 5). Es wäre also sicher sinnvoll einen Kompromiss aus beidem zu suchen. Dies wird in den Folgenden Abschnitten versucht.

Der beim ParallelSFFS(n)-Algorithmus gewählte Ansatz besteht darin, n SFFS-Algorithmen parallel laufen zu lassen. Würden diese jedoch völlig frei, also ohne Verknüpfung laufen, so würden sie alle genau dieselben Merkmalmengen untersuchen. Es ist also nötig die einzelnen Teilalgorithmen miteinander zu verknüpfen.

Dies soll hier so geschehen, dass alle Teilalgorithmen auf eine gemeinsame CLOSED-Liste zugreifen. In diese trägt dann ein Teilalgorithmus jeweils die Merkmalmenge ein, welche nach einem Vorwärtsschritt und den darauf folgenden möglichen Rückwärtsschritten als aktuelle Merkmalmenge hervorgeht. Danach sind alle Merkmalmengen in der CLOSED-Liste für alle Teilalgorithmen als Zielmenge eines Vorwärts- oder Rückwärtsschritts tabu.

Der Gesamtalgorithmus terminiert, wenn alle Teilalgorithmen terminiert haben. Die Terminierung der Teilalgorithmen kann dabei entweder nach einer vorgegebenen Anzahl durchgeführter Berechnungen oder beim Erreichen einer vorgegebenen Merkmalzahl erfolgen.

2.9 LazyGSFFS

Ein anderer Ansatz zur Verbreiterung der Suche besteht darin, immer zwei oder mehr statt nur einem Element pro Vorwärtsschritt zur aktuellen Merkmalmenge hinzuzufügen. Man nennt dies Generalisierung.

Ein Algorithmus der dies praktiziert wurde auch bereits in diesem Kapitel vorgestellt, nämlich der ASFFS-Algorithmus. Dabei wird aber die Anzahl der beim Vorwärtsschritt hinzuzufügenden Elemente in Abhängigkeit von der gewünschten Merkmalzahl dynamisch geregelt. Da wir aber von einer festen Merkmalzahl unabhängig bleiben wollen ist dieses Verfahren so nicht praktikabel.

Eine einfachere Möglichkeit zur Generalisierung besteht darin, die hinzuzufügende Merkmalzahl im Vorfeld zu fixieren. Man kann z. B. pro Vorwärtsschritt l Elemente hinzufügen und pro Rückwärtsschritt r wieder entfernen (GSFFS(l, r)). Wobei die folgende Beziehung gilt: GSFFS(1, 1) = SFFS.

Dieses Vorgehen hat allerdings den Nachteil, dass der Aufwand pro Vorwärtsschritt mit der Anzahl der zur Verfügung stehenden Merkmale zur Potenz $\max\{l, r\}$ ansteigt. Dies ist aber bei den in dieser Arbeit untersuchten Merkmalmengen zur Lokalisierung der Bindungsstellen von Transkriptionsfaktoren problematisch. Daher wäre es sinnvoll die Anzahl der durchzuführenden Berechnungen wieder etwas reduzieren zu können.

Eine Möglichkeit für diese Reduktion besteht darin, erst alle Güten nur zu schätzen und erst dann eine gewisse Anzahl (min. 1) an Güten der am besten eingeschätzten Merkmalmengen echt zu berechnen. Dieser Algorithmus soll dann LazyGSFFS(l, r, p) heißen, wobei p den Anteil der echt zu berechnenden Güten angibt. Für die genaue Erläuterung der Schätzung sei hier auf Anhang A verwiesen. Nachdem neue Güten echt berechnet wurden, wird auf Basis dieser Ergebnisse der Schätzer aktualisiert.

Die Terminierung des Algorithmus kann dabei entweder wieder nach einer vorgegebenen Anzahl durchgeführter Berechnungen oder beim Erreichen/Überschreiten einer vorgegebenen Merkmalzahl erfolgen.

2.10 LazySFFS

Die beim LazyGSFFS-Algorithmus genutzte Reduktion der nötigen Berechnungen kann man natürlich auch auf den SFFS-Algorithmus übertragen und gelangt so zum LazySFFS-Algorithmus. Es gilt dann die offensichtliche Beziehung: LazySFFS(p) = LazyGSFFS(1, 1, p).

Im Unterschied zum normalen SFFS-Algorithmus wird die Auswahl des hinzuzufügenden Merkmals jetzt wieder in zwei Stufen durchgeführt:

1. alle Güten nur schätzen und daraus einen gewissen Anteil (aber min. 1) der am besten geschätzten Güten auswählen

2. die Güten in diesem Anteil echt berechnen und daraus auswählen

Nachdem neue Güten echt berechnet wurden, wird auf Basis dieser Ergebnisse der Schätzer wieder aktualisiert.

2.11 Zusammenfassung

In Tabelle 2.1 sind nochmals die wichtigsten Eigenschaften der in diesem Kapitel vorgestellten Algorithmen zusammengefasst.

Name	Komplexität	Parameter	Bemerkung
SFS	$\Theta(d \cdot D)$	keine	
GSFS(g)	$\Theta(\frac{d}{g} \cdot D^g)$	1	
PTA(l, r)	$\Theta(\frac{d}{l-r} \cdot (l+r) \cdot D)$	2	
GPTA(l, r)	$\Theta(D^{max\{l+1,r+1\}})$	2	
SFFS	$O(2^D)$	keine	Aufw. in der Praxis meist geringer
ASFFS(r_{max}, b)		2	
Max-Min		keine	fehlerhafte theor. Basis
ParallelSFFS(n)		1	
LazyGSFFS(l, r, p)		3	
LazySFFS(p)		1	

Tabelle 2.1: Übersicht der Eigenschaften der vorgestellten sequentiellen Algorithmen

Kapitel 3

Randomisierte Algorithmen

Bei den randomisierten Verfahren wird versucht, die Wahrscheinlichkeit zu verringern, dass man nur ein lokales Maximum statt des globalen Maximum findet. Dies versucht man, indem man zufällige Entscheidungen oder Operationen einführt.

Bei allen randomisierten Verfahren ist es sicher sinnvoll, sich parallel zum Algorithmus die beste im Verlauf gefundene Merkmalmenge zu merken. Außerdem ist es meist vorteilhaft diese Algorithmen mehrmals laufen zu lassen, und aus den Ergebnissen das beste auszuwählen.

3.1 Genetischer Algorithmus

Der Genetischer Algorithmus (GA(N, T, p_c, p_m)) wurde 1989 von W. Siedlecki et al. in [Sie89] für die Merkmalauswahl eingeführt. Er ist an der natürlichen Evolution orientiert und arbeitet im Wesentlichen mit den randomisierten Methoden der „Kreuzung“ und der „Mutation“ und einem „Fitness“-Maß für die getroffenen Auswahlen.

Beim GA werden die Merkmaluntermengen durch so genannte Chromosomen binär repräsentiert. Diese sind Strings der Länge D, wobei D der Anzahl der zur Verfügung stehenden Merkmale entspricht. Dabei bedeutet eine 1, dass ein Merkmal ausgewählt ist und eine 0, dass es nicht ausgewählt ist. Die ermittelte Merkmaluntermenge ist also die Menge aller Merkmale, die im zurückgegebenen Chromosom mit einer 1 markiert sind.

Die Verfahrensweise (siehe z.B.: [Fer94, Kap. 2.2] oder [Kud00b, Kap. 3.3]) des Algorithmus ist so, dass zu Beginn eine Menge (mit vorgegebener Größe N) von Chromosomen, eine so genannte „Population“ P, initialisiert wird. Diese Population stellt die erste „Generation“ dar. Danach wird jeweils aus der Population einer Vorgänger-Generation durch Kreuzung (engl.: Crossover), Mutation, Fitnessberechnung und Selektion eine Population (der Größe N) der Nachfolger-Generation erzeugt. Dabei wird die Kreuzung durch die vorgegebene Kreuzungsrate p_c und die Mutation durch die ebenfalls vorgegebene Mutationsrate p_m gesteuert. Nach einer vorgegebenen Zahl von Generationen, wird aus der letzten Generation das beste Chromosom ausgewählt und zurückgeliefert.

Für die Initialisierung der ersten Population, nennen Kudo et al. ([Kud00b, Kap. 3.3]) zwei Möglichkeiten:

1. Auswahl aller $2D$ Chromosomen, mit 1 bzw. $(D-1)$ Merkmalen und zusätzlich $(N-2D)$ Chromosomen mit 2 oder $(D-2)$ Merkmalen. (Schreibweise $\{1, D-1\}$)

2. Zufällige Auswahl von N Chromosomen, mit einer Merkmalzahl im Intervall $[m, M]$. (Schreibweise $[m, M]$)

Der Pseudocode des Genetic Algorithm ist Abbildung 3.1 zu entnehmen. Die zeitliche Komplexität dieses Verfahrens ist nur von den gewählten Parametern abhängig. Werden diese Parameter unabhängig von D konstant gewählt, so beträgt die zeitliche Komplexität $\Theta(1)$. Kudo et al. wählen den Parameter N in linearer Abhängigkeit von D und kommen somit zu einer zeitlichen Komplexität von $\Theta(D)$ [Kud00a, Kap. 2.1, Abs. 5].

GA(N, T, p_c, p_m)

Input:	N	Größe der Population
	T	maximale Anzahl an Generationen
	p_c	Kreuzungs-Rate
	p_m	Mutations-Rate
Output:	x	Chromosom der Merkmaluntermenge

Pseudocode:

1. $k := 0$
2. Initialisiere P mit N zufälligen Chromosomen
3. $EvaluiereFitness(P)$
4. **WHILE** $(k < T)$ AND $(KeineKonvergenz(P))$ **DO**
   (a) $M := Rekombination(P)$
   (b) $O := Kreuzung(M, p_c)$
   (c) $O := Mutation(O, p_m)$
   (d) $EvaluiereFitness(P)$
   (e) $P := Selektiere(P, O)$
   (f) $k := k + 1$

   ENDWHILE
5. Gib bestes Chromosom x aus P zurück

Abbildung 3.1: Pseudocode des GA(N, T, p_c, p_m) nach [Kud00b, Fig. 4]/[Fer94, Fig. 3]

Einschätzungen

Eine größere Generationenzahl oder eine größere Populationszahl führt im Allgemeinen zu besseren Ergebnissen ([Kud00b, Kap. 5.2.1]). Mit wenig Training (wenigen Generationen) kann man vergleichbare Ergebnisse wie SFFS erzielen ([Kud00b, Kap. 5.2.2]). Mit viel Training (vielen Generationen) sind vergleichbare Ergebnisse wie mit ASFFS zu erreichen, mit mehreren Anläufen manchmal auch bessere ([Kud00b, Kap. 5.3.1]). Für große Merkmalmengen ist GA schneller als ASFFS ([Kud00b, Kap. 5.3.3]).

In Abbildung 3.2 sind vergleichende Ergebnisse von ASFFS/ASBFS und GA zu sehen ([Kud00b]). Es ist zu erkennen, dass ASFFS/ASBFS und GA in etwa die gleichen Ergebnisse erzielen, aber in Einzelfällen GA bessere Ergebnisse erzielt.

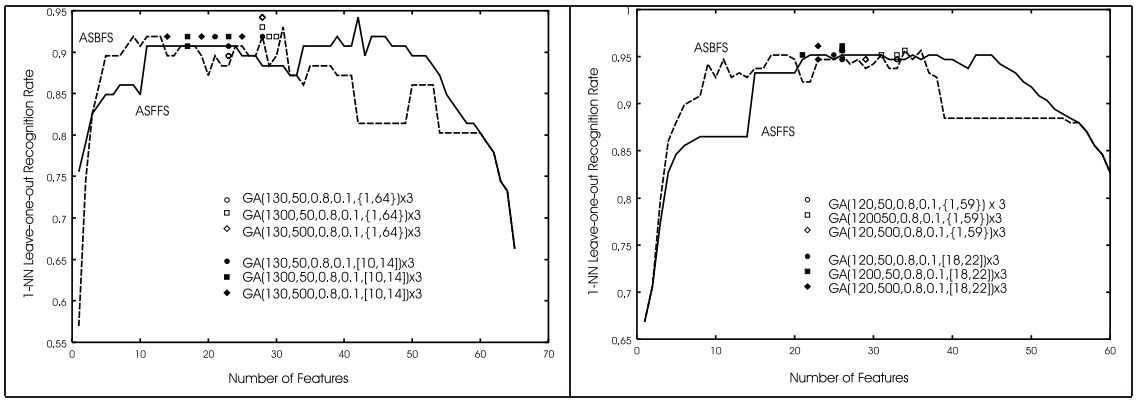

Abbildung 3.2: Vergleich von GA(N, T, p_c, p_m, IA) mit ASFFS/ASBFS anhand ihrer Ergebnisse an Mammographiedaten (links) und Sonardaten (rechts) (aus [Kud00b, Fig. 7, 8]). Der 5. Parameter IA gibt die Art der Initialisierung der ersten Population an. Es wurden je 3 Anläufe unternommen.

3.2 Simulated Annealing

Simulated Annealing (SA(t_0, α, d)) wurde 1983 von Kirkpatrick et al. in [Kir83] vorgestellt.

Wie das Wort Annealing (engl.: Ausglühen) schon sagt, ist dieser Algorithmus an einem Abkühlungsprozess orientiert. Wie in [BD01, Kap. 5] beschrieben, wird jetzt eine Temperatur t eingeführt, welche permanent sinkt. Es gilt dabei im k-ten Schritt: $t = t_0\alpha^{k/d}$, wobei t_0 die Anfangstemperatur ist, und $\alpha < 1$ sowie d Parameter zur Beschreibung der genauen Form der Abkühlkurve sind. Diese Temperatur wird nun dazu genutzt, die Wahrscheinlichkeit zu bestimmen, mit der eine Verringerung der Güte von einer Merkmaluntermenge zu einer neuen in Kauf genommen wird.

Diese Methode kann jetzt mit den obigen sequentiellen Algorithmen kombiniert werden. Man kann dies erreichen, indem man von der Übergangswahrscheinlichkeit = 1 für die Merkmaluntermenge mit der höchsten Güte zu Übergangswahrscheinlichkeiten < 1 übergeht. Dass heißt also, man wählt nicht mehr immer genau die Merkmaluntermenge mit der höchsten Güte, sondern mit einer bestimmten Wahrscheinlichkeit auch eine andere.

Sei g die Güte der aktuellen Merkmaluntermenge. Aus den durch Weglassen oder Hinzufügen eines oder mehrerer Merkmale entstehenden neuen Merkmaluntermengen, wird eine zufällig ausgewählt. Diese habe die Güte g'. Dann wird mit Wahrscheinlichkeit $\min(1, \exp(\frac{s'-s}{t}))$ zu dieser gewechselt.

Dies bedeutet, dass bei hohen Temperaturen (also zu Beginn) die Wahrscheinlichkeit eines Schrittes, welcher die Güte verringert, nahe 1 ist, aber später, bei niedrigeren Temperaturen, nahe bei 0.

3.3 Sintflut-Algorithmus

Der Sintflutalgorithmus (Great Deluge Algorithm (GDA(r, t))) wurde 1989 von Gunter Dueck in [Due89, Due93] vorgeschlagen.

Die Funktionsweise des Algorithmus soll wie folgt bildlich dargestellt werden: Ein (imaginärer) Wanderer wird an einem zufälligen Startpunkt in einer Funktion „ausgesetzt“ und soll einen möglichst hohen Gipfel dieser Funktion finden. Dabei steht der höchste Gipfel dieser Funktion für das zu findende Optimum, in unserem Falle also für die optimale Güte einer Merkmalmenge mit vorgegebener Größe.

In dieser Verbildlichung funktioniert GDA wie folgt: Der Wanderer bewegt sich bei jedem Schritt, ausgehend von der aktuellen Position, in eine zufällige Richtung und um eine feste Weite (meist *ein* Schritt). Dadurch entsteht anfangs ein zielloses Umherirren. Währenddessen steigt die „Sintflut“ durch permanenten Regen in vorgegebener Menge (r) pro Schritt. Die Sintflut symbolisiert in unserem Fall eine untere Schranke für die Güte.

Dem Wanderer ist es dabei verboten, eine Bewegung auszuführen, deren Zielpunkt unter Wasser liegt, dass heißt in unserem Fall, eine Merkmalmenge zu wählen, deren Güte unterhalb der aktuellen unteren Schranke liegt. Wählt er dennoch eine solche unzulässige Bewegung, wird diese verworfen und eine neue Bewegung gewählt. Ist der Wanderer ringsum von Wasser umgeben, kann er also keine zulässige Bewegung mehr ausführen, terminiert der Algorithmus. Der aktuelle Punkt, also die aktuelle Merkmalmenge, ist das gefundene (lokale) Optimum.

Falls es zu einer Merkmalmenge zu viele Nachbarmengen gibt, muss man die Terminierungsforderung, dass keine zulässige Nachbarmenge mehr existiert, eventuell vereinfachen. So könnte man z.B. nur noch fordern, dass man nach einer vorgegebenen Anzahl (t) von Versuchen keine zulässige Nachbarmenge mehr gefunden hat.

Das Flussdiagramm des Sintflut-Algorithmus ist Abbildung 3.3 zu entnehmen.

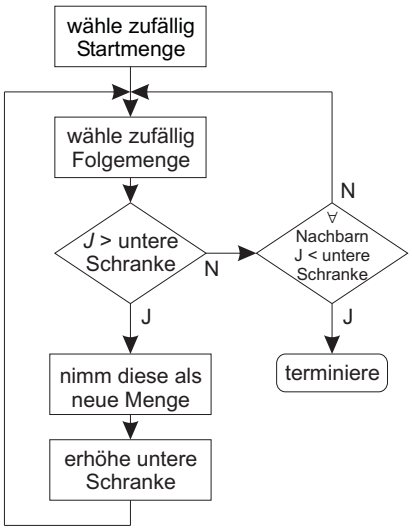

Abbildung 3.3: Flussdiagramm des Sintflut-Algorithmus

Einschätzungen

Im Vergleich zu den anderen randomisierten Algorithmen hat dieser Algorithmus den Vorteil, dass lediglich ein (oder ggf. zwei) Parameter, nämlich die Regenmenge pro Bewegung und ggf. die Anzahl der Versuche eine zulässige Nachbarmenge zu finden, festgelegt werden muss. Somit ist dieser Algorithmus deutlich leichter konfigurierbar.

3.4 Zusammenfassung

In Tabelle 3.1 sind nochmals die wichtigsten Eigenschaften der in diesem Kapitel vorgestellten Algorithmen zusammengefasst.

Name	Komplexität	Parameter	Bemerkung
GA(N, T, p_c, p_m)	$\Theta(1)$ bzw. $\Theta(D)$	4	$\Theta(D)$ z.B. bei $N = 2D$
SA(t_0, α, d)		3	
GDA(r, t)		2	

Tabelle 3.1: Übersicht der Eigenschaften der vorgestellten randomisierten Algorithmen

Kapitel 4

Graph-Suchalgorithmen

Bei den im Folgenden aufgeführten Algorithmen wird die Merkmaluntermengen-Suche als Suche in einem Lösungsgraph aufgefasst, dessen Knoten Merkmaluntermengen repräsentieren. Daher kommen Graph-Suchalgorithmen zum Einsatz. Da die Merkmalauswahl ein NP-hartes Problem ([Tah04, Kap. 1 Abs. 5]) ist, führt dies zu exponentiellem Aufwand.

4.1 Exhaustive Search

Beim Verfahren Exhaustive Search (siehe z.B. [Cov77]) wird, wie das Wort „exhaustive“ (engl.: erschöpfend) schon sagt, der Lösungsraum vollständig abgesucht. Dies liefert natürlich in jedem Fall das optimale Ergebnis ([Som99, Kap. 1]), ist aber mit exponentiellem Aufwand verbunden und somit nur für sehr kleine Aufgabenstellungen praktikabel. Für größere Aufgabenstellungen muss man Einschränkungen treffen und somit das Verfahren deutlich beschleunigen. Dies wird bei den folgenden Algorithmen getan.

4.2 Branch and Bound

Der Branch and Bound-Algorithmus (BB) wurde 1977 von Narendra und Fukunaga in [Nar77] vorgeschlagen.

Mit dem BB-Algorithmus wird versucht, mit einer geringeren Rechenzeit als bei exhaustiver Suche auszukommen, aber trotzdem wieder die optimale Merkmaluntermenge zu finden. Dies wird dadurch erreicht, dass BB jetzt von einer monotonen Güte ausgeht.

Beim BB-Algorithmus kommt ein Lösungsbaum wie in Abbildung 4.1 zum Einsatz, welcher eine Durchnummerierung der Merkmale voraussetzt. Dieser ist rekursiv so aufgebaut, dass alle möglichen Merkmaluntermengen in *genau einem* Blatt vertreten sind.

Der genaue Aufbau geschieht dabei nach den folgenden zwei Regeln:

- der Wurzelknoten repräsentiert die Menge, welche alle zur Verfügung stehenden Merkmale enthält
- unter einem beliebigen, nichtterminalen Knoten n_i, hängen Knoten n_j mit allen Merkmalen aus n_i, abzüglich je einem Merkmal zwischen $\text{maxfeature}(n_i) + 1$ und

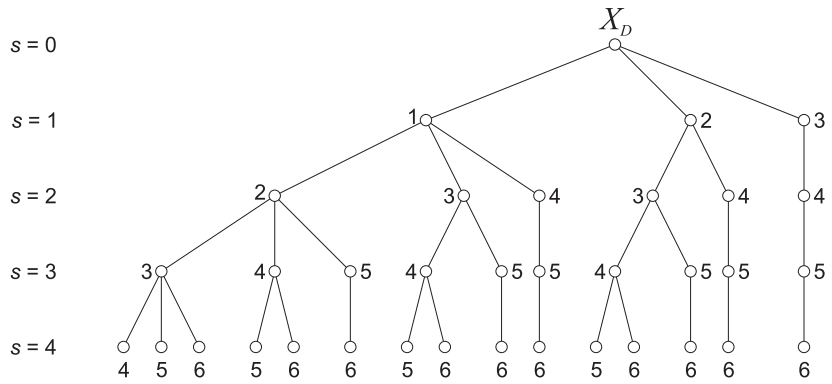

Abbildung 4.1: Branch and Bound-Lösungsbaum für Auswahl von 2 aus 6 Merkmalen. (Zahlen neben den Knoten bezeichnen das Merkmal, welches im Vergleich zum Vorgänger entfernt wurde.)

$$D - d + \mathrm{card}(n_i) + 1.^1$$

Aus diesem Aufbau folgt, dass die Unterbäume unter einem Knoten so angeordnet sind, dass sie von rechts nach links komplexer (größer) werden. Bei der Abarbeitung der Unterbäume unter einem Knoten wird daher auch von rechts nach links vorgegangen, was bedeutet, dass immer zuerst der einfachste Unterbaum verarbeitet wird.

Jedoch wird jetzt nicht der komplette Baum blind durchsucht, sondern man lässt alle überflüssigen Unterbäume nach folgendem Prinzip weg: Sei B die beste bisher gefundene Güte (Bound) eines terminalen Knotens und sei X_s der aktuelle Knoten. Falls $B \geq J(X_s)$, gilt auf Grund der Monotoniebedingung offensichtlich für alle Knoten X_t unterhalb des aktuellen Knotens $B \geq J(X_t)$. Dass heißt, unterhalb des aktuellen Knotens liegt sicher kein Knoten mehr, welcher besser wäre als der beste bisherige Knoten. Dies bedeutet, dass der komplette Unterbaum unterhalb des aktuellen Knotens nicht mehr exploriert werden muss.

Das Flussdiagramm des BB-Algorithmus ist in Abbildung 4.2 dargestellt. Die zeitliche Komplexität dieses Algorithmus liegt bei $O(2^D)$ (vgl. [Kud00a, Table 1]).

Einschätzungen

Der BB-Algorithmus liefert die optimale Merkmaluntermenge, ist aber an die Monotonie der Güte der Merkmaluntermengen gebunden ([Som99, Kap. 1]).

Des Weiteren handelt es sich beim BB-Algorithmus um einen Rückwärts-Algorithmus. Dass bedeutet, dass mit einer Merkmalmenge begonnen wird, welche alle Merkmale enthält. Dies ist bei großen Merkmalzahlen im Hinblick auf den Rechenaufwand natürlich problematisch.

[1] $\mathrm{maxfeature}(n) = \begin{cases} 0 & n = \emptyset \\ \text{die höchste Nummer eines Merkmals in } n & \textit{sonst} \end{cases}$

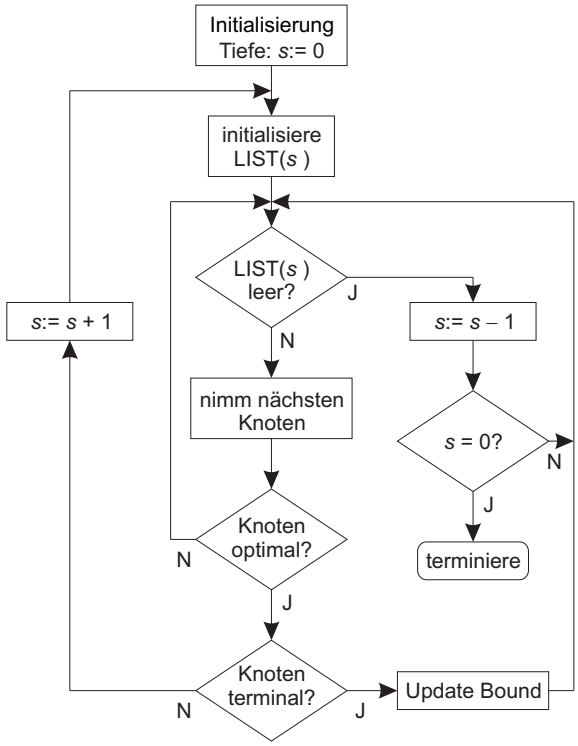

Abbildung 4.2: Flussdiagramm des BB-Algorithmus nach [Yu93, Fig. 4]. Dabei werden in LIST(s) für jede Ebene s jeweils die noch nicht untersuchten Geschwisterknoten der Knoten auf dem Weg vom Wurzelknoten zum aktuellen Knoten gespeichert.

4.3 Branch and Bound$^+$

Der Algorithmus Branch and Bound$^+$ (BB$^+$) wurde 1993 von Yu und Yuan in [Yu93] vorgeschlagen.

BB$^+$ arbeitet vom Grundprinzip her wie BB. Es werden jetzt aber alle Unterbäume, welche die Form einer linearen Liste haben, schnell bis zum terminalen Knoten (Blatt) durchlaufen, ohne die einzelnen Knoten der Liste zu bewerten. Dadurch kann man sich den Lösungsbaum jetzt wie in Abbildung 4.3 vorstellen.

Es ist offensichtlich, dass jetzt weniger Knoten bewertet werden müssen, und somit die Geschwindigkeit von BB$^+$ höher liegt als die von BB.

Die Bestimmung, ob ein zu untersuchender Unterbaum die Form einer linearen Liste hat ist dabei trivial. Denn aufgrund des rekursiven Aufbaus des Lösungsbaums ist immer genau der Unterbaum eine lineare Liste, welcher am ersten Kindknoten von rechts hängt.

Das Flussdiagramm des BB$^+$ Algorithmus ([Yu93, Fig. 3]) ist Abbildung 4.4 zu entnehmen.

Einschätzungen

BB$^+$ liefert genau wie BB die optimale Merkmaluntermenge, ist aber auch an die Monotonie der Güte der Merkmaluntermengen gebunden und ist ebenso ein Rückwärts-Algorithmus.

Einen Vergleich des Rechenaufwandes von BB$^+$ mit dem von BB, kann man Tabelle

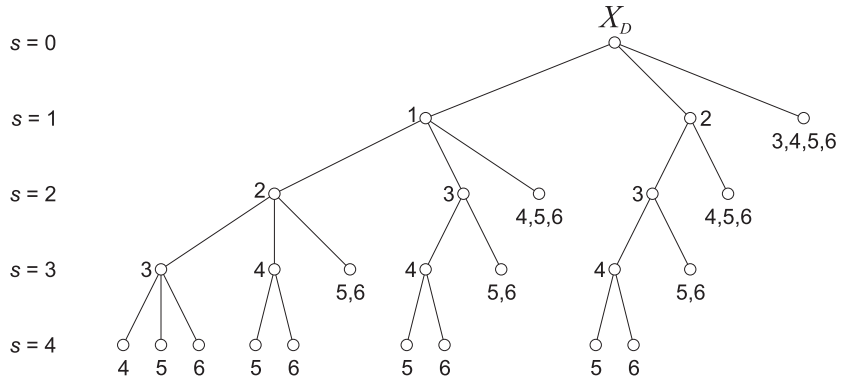

Abbildung 4.3: Branch and Bound$^+$-Lösungsbaum für Auswahl von 2 aus 6 Merkmalen. (Zahlen neben den Knoten bezeichnen die Merkmale, welche im Vergleich zum Vorgänger entfernt wurden.)

4.1 entnehmen. Es ist erwartungsgemäß zu erkennen, dass bei BB$^+$ weniger Knoten expandiert wurden. Einen ähnlichen Vergleich mit äquivalenten Ergebnissen kann man auch Abbildung 4.8 entnehmen.

Problem	Algorithmus	Knoten	expandierte Knoten
2 aus 12	BB$^+$	121	17
2 aus 12	BB	286	41
2 aus 9	BB$^+$	64	15
2 aus 9	BB	120	29

Tabelle 4.1: Vergleich des Rechenaufwandes von BB und BB$^+$ (nach [Yu93, Table 1])

Es ist aber auch anzumerken, dass dieser Algorithmus eine unmotivierte Abhängigkeit von der Reihenfolge der Merkmale besitzt.

4.4 Fast Branch and Bound

Der Algorithmus Fast Branch and Bound (FBB(γ, δ)) wurde 2000 von Somol et al. in [Som00] eingeführt. Er sucht ebenso wie BB und BB$^+$ nach der optimalen Merkmaluntermenge einer vorgegebenen Größe, ist aber auch auf die Monotonie der Güte angewiesen.

Das neue an diesem Algorithmus ist, dass die Güte nicht immer *berechnet* wird, sondern unter bestimmten Bedingungen (z.B. nicht bei Blättern) *vorhergesagt* wird. Die Vorhersage basiert dabei auf der Annahme, dass die Entfernung des gleichen einzelnen Merkmals bei verschiedenen Merkmaluntermengen ähnliche Veränderungen der Güte verursacht (vgl. [Som00, Kap. 3]). Dies führt natürlich zu einer Reduktion des Rechenaufwandes, speziell im Wurzelbereich, und ist die Basis der erhöhten Geschwindigkeit gegenüber BB bzw. BB$^+$.

Danach werden berechnete und vorhergesagte Güten im Wesentlichen gleich behandelt. Lediglich wenn die vorhergesagte Güte nahe dem bisherigen Optimum (Bound) liegt, wird zusätzlich zur Vorhersage noch die echte Berechnung durchgeführt. Das Abschneiden von Unterbäumen wird ebenfalls nur auf Basis der berechneten echten Güte durchgeführt.

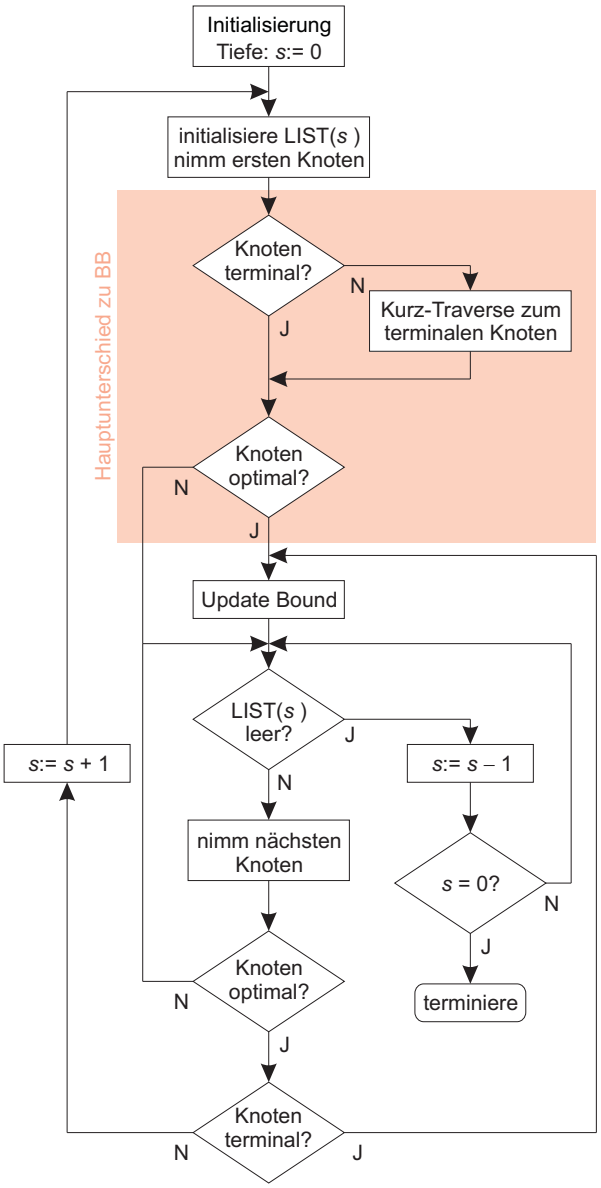

Abbildung 4.4: Flussdiagramm des BB$^+$-Algorithmus ([Yu93, Fig. 3])

Der Parameter δ gibt die Zahl der Güte-Berechnungen an, die für die Entfernung eines Merkmals vorliegen müssen, ehe mit der Vorhersage gestartet wird. Der Parameter γ gibt an, wie optimistisch mit den Vorhersagen umgegangen wird. Werte $\gamma > 1$ verringern das Vorkommen optimistischer Vorhersagefehler, Werte $0 < \gamma < 1$ verringern das Vorkommen pessimistischer Vorhersagefehler. Für $\gamma = 0$ verhält sich der Algorithmus wie Exhaustive Search. Somol et al. geben als praktikable werte $\delta = 5$ und $\gamma = 1.1$ an (vgl. [Som00, Kap. 5]).

Dieser Vorhersagemechanismus beeinflusst aber nicht den Optimalitätsanspruch (vgl. [Som00, Kap. 3]). Eine falsche Vorhersage führt im schlimmsten Fall dazu, dass ein Unterbaum exploriert wird, welcher beim klassischen BB nicht exploriert worden wäre. Dies wird aber üblicherweise durch Geschwindigkeitsgewinne bei anderen Knoten wieder kompensiert.

Das Flussdiagramm des FBB Algorithmus ([Som00, Fig. 3]) ist Abbildung 4.5 zu entnehmen.

Einschätzungen

Der FBB-Algorithmus liefert genau wie BB und BB$^+$ die optimale Merkmaluntermenge, ist aber auch an die Monotonie der Güte der Merkmaluntermengen gebunden und ist ebenso ein Rückwärts-Algorithmus.

Einen Vergleich des Rechenaufwandes von FBB mit BB$^+$ bzw. BB kann man Abbildung 4.8 entnehmen. Es ist zu erkennen, dass FBB weniger Güte-Berechnungen als BB$^+$ oder BB vornimmt, und somit die Rechenzeit geringer ist als bei BB$^+$ oder BB.

4.5 Improved Branch and Bound

Der Algorithmus Improved Branch and Bound (IBB) wurde 2003 von Chen in [Che03] eingeführt. Er basiert auf BB$^+$, versucht aber noch mehr Unterbäume zurückzuweisen, indem Informationen von früheren Zurückweisungen genutzt werden.

Die Nutzung der vorhergehenden Informationen erfolgt wie im Folgenden: Sei z.B. ein Unterbaum unter einem Knoten X abgeschnitten worden. Dabei sei der Knoten X dadurch entstanden, dass nacheinander die Merkmale 2 und 3 entfernt wurden. Dann kann man aufgrund der Monotonie z.B. auch den Unterbaum unter einem Knoten Z entfernen, wenn Z entstand, indem nacheinander die Merkmale 1, 2 und 3 entfernt wurden (siehe Abbildung 4.7). Man nennt dann den Pfad (1, 2, 3) auch Elternpfad des partiellen Pfades (2, 3).

Bei diesem Algorithmus ist es wichtig, auf einer Ebene stets von rechts nach links vorzugehen. Denn Elternpfade treten immer nur auf der linken Seite von partiellen Pfaden auf.

Diese Idee wird praktisch umgesetzt, indem die Pfade zu allen bisher zurückgewiesenen Knoten in einer Liste (PARTIAL) gespeichert werden. Bei jedem neuen Knoten wird dann geprüft, ob einer der Pfade aus PARTIAL ein Teilpfad des Pfades zum aktuellen Knoten ist. Falls dem so ist, kann aufgrund der Monotonie der Unterbaum unter dem aktuellen Knoten abgeschnitten werden.

Das Flussdiagramm des IBB Algorithmus ([Che03, Fig. 2]) ist Abbildung 4.6 zu entnehmen.

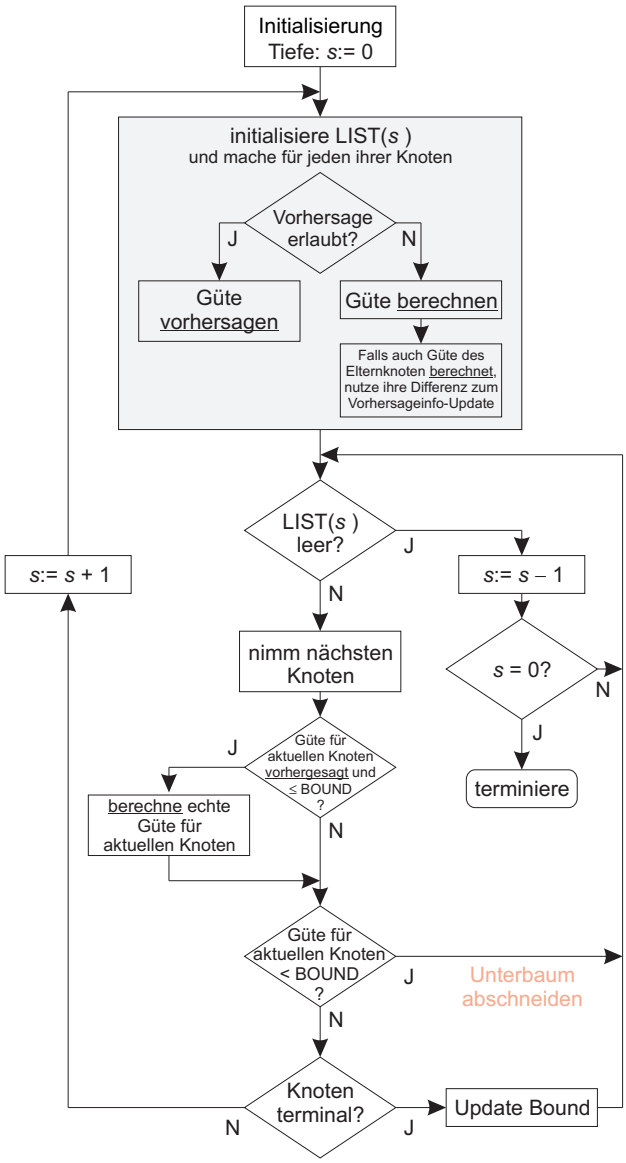

Abbildung 4.5: Flussdiagramm des FBB-Algorithmus ([Som00, Fig. 3])

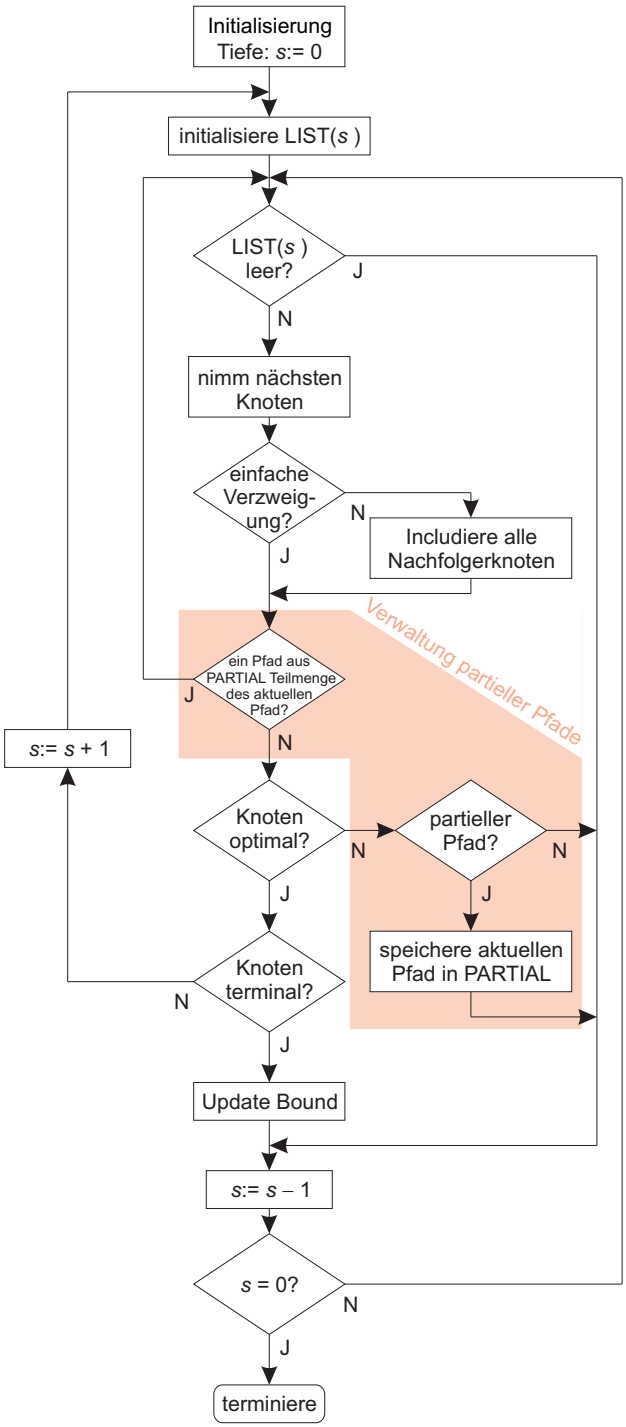

Abbildung 4.6: Flussdiagramm des IBB-Algorithmus ([Che03, Fig. 2])

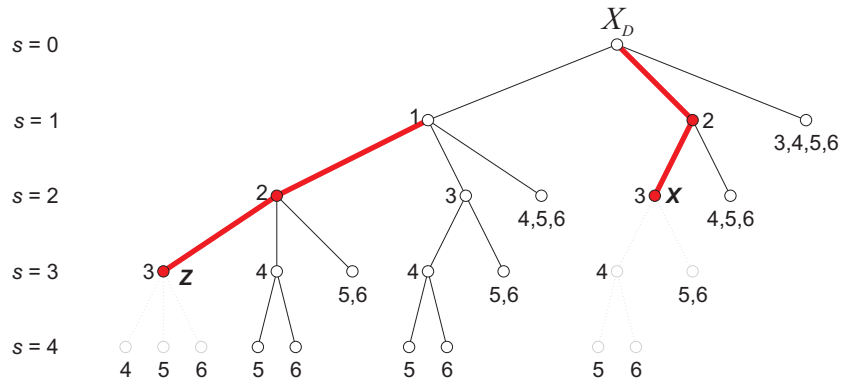

Abbildung 4.7: Anschauliche Darstellung der zusätzlichen Nutzung von Information zur Verkleinerung des Suchbaums bei IBB (nach [Che03, Fig. 1])

Einschätzungen

Auch der IBB-Algorithmus liefert wie die bisherigen BB-Varianten die optimale Merkmaluntermenge, ist aber auch an die Monotonie der Güte der Merkmaluntermengen gebunden. Des Weiteren ist auch IBB ein Rückwärts-Algorithmus.

Abbildung 4.8 zeigt einen Vergleich zwischen IBB und den bisher vorgestellten BB-Varianten. Dabei ist zu erkennen, dass Fast Branch and Bound zwar etwas weniger Knoten exploriert, aber IBB im Endeffekt eine deutlich geringere Rechenzeit als BB, BB$^+$ oder auch FBB benötigt.

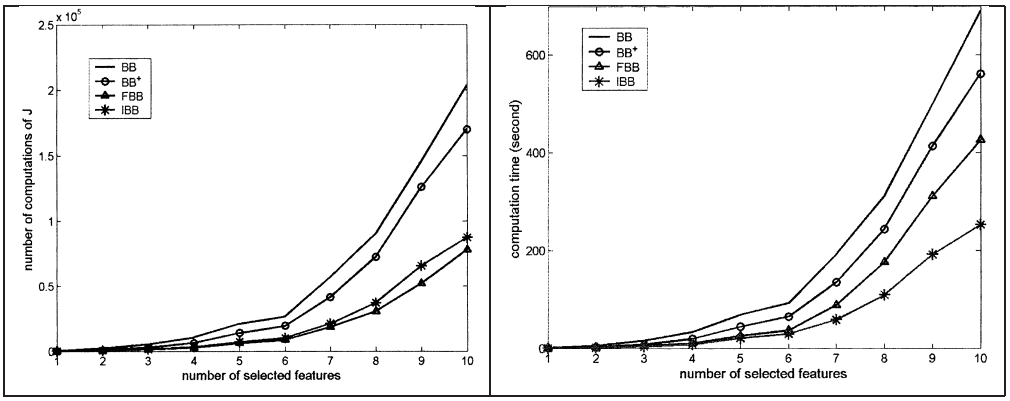

Abbildung 4.8: Vergleich zw. BB, BB$^+$, FBB und IBB anhand ihres Aufwandes (explorierte Knoten (links) und Rechenzeit (rechts)) an Mammographiedaten (aus [Che03, Fig. 7,8])

4.6 A^*-Algorithmus

4.6.1 Aufgabenstellung

Die Merkmalauswahl soll auch hier wie bei den Branch and Bound-Varianten als Graphensuche betrachtet werden. Dabei repräsentiert jeder Knoten n des gerichteten Suchgraphen eine Merkmalmenge A_n. Zwischen zwei Knoten n und m existiert genau dann eine Kante

von n nach m, wenn $A_n \subset A_m$ und $\text{card}(A_n) = \text{card}(A_m) - 1$. Die Kanten zwischen n und m repräsentieren also eine Nachfolgerrelation zwischen dem *Vorgänger* n und dem *Nachfolger* m.

Im Folgenden soll nichtmehr zwischen dem Knoten und der Merkmalmenge, welche er repräsentiert, unterschieden werden. Es wird also nur n statt A_n geschrieben.

4.6.2 Der Algorithmus A^*

Der A^*-Algorithmus ist ein weit verbreitetes heuristisches Graphensuchverfahren (siehe z.B. [Rus03]). Es wurde 1980 von Nilsson in [Nil80] eingeführt. Ursprüngliches Ziel ist das Aufspüren eines möglichst kostengünstigen Weges zwischen einer Menge von möglichen Startknoten und einer Menge möglicher Zielknoten. Dabei werden die Kosten eines Knotens durch Aufsummieren der Kantengewichte auf dem Weg vom Startknoten bis zu diesem Knoten bestimmt.

Hier soll jetzt aber der Maximierungsansatz verfolgt werden. Es soll also ein möglichst lukrativer Weg gefunden werden, dass heißt ein Weg mit möglichst hoher Belohnung. Die Belohnung für einen Knoten soll dabei aus dem Knoten (bzw. aus den von dem Knoten repräsentierten Daten) selbst bezogen werden und nicht aus den Kanten.

Voraussetzungen

Der Graph, auf den der folgende A^*-Algorithmus angewendet werden soll, muss ein

- endlicher,
- gerichteter,
- zyklenfreier und
- zusammenhängender

Graph sein, in dem es aus später klar werdenden Gründen keine Kante zwischen zwei Zielknoten geben darf. Die Menge der Zielknoten soll also einschichtig sein.

Das Vorgehen von A^*

Ein Flussdiagramm des A^*-Algorithmus kann man Abbildung 4.9 entnehmen.

In der auch für Graphen geeigneten allgemeineren Version (siehe z.B.: [Cun94, Kap. 2.1]) nutzt der A^*-Algorithmus zwei Listen: Die so genannte OPEN- und die CLOSED-Liste. Die CLOSED-Liste enthält die Knoten, die schon expandiert wurden, die OPEN-Liste jene, welche noch expandiert werden können. Unter Expansion versteht man dabei, dass ein Knoten in der OPEN-Liste durch all seine Nachfolgerknoten ersetzt wird.

Die OPEN-Liste wird beim Start mit den Startknoten initialisiert. Im Spezialfall der Suche im Baum also mit dem Wurzelknoten. Danach wird bei jedem Schritt nach dem Best-First-Prinzip der am besten bewertete Knoten ausgewählt. Dieser Knoten wird dann von der OPEN- in die CLOSED-Liste verschoben und seine Nachfolger in die OPEN-Liste eingefügt und bewertet, sofern sie sich noch nicht in der OPEN- oder CLOSED-Liste befinden. Falls sie sich schon in einer der beiden Listen befinden, wird ein Update der

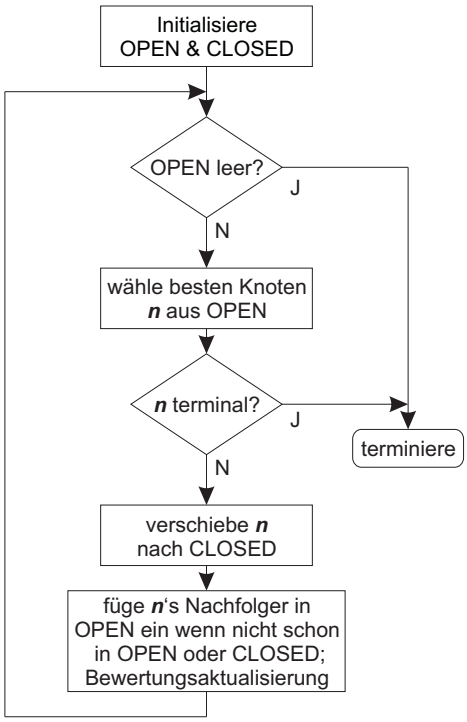

Abbildung 4.9: Flussdiagramm des A^*-Algorithmus

Bewertung vorgenommen und der Knoten ggf. von der CLOSED- in die OPEN-Liste verschoben, falls die neue Bewertung g besser als die alte ist.

Die Bewertung eines Knoten n erfolgt dabei durch eine Funktion $f(n)$. Diese setzt sich wiederum aus den beiden Funktionen $g(\cdot)$ und $h(\cdot)$ wie folgt zusammen:

$$f(n) = g(n) + h(n)$$

Dabei gibt die Funktion $g(n)$ die Belohnung an, die auf dem bisher lukrativsten Weg von einem Startknoten zum Knoten n bereits angefallen ist. Die Funktion $h(n)$ ist eine Schätzung der maximalen Belohnung, welche auf dem Weg vom Knoten n bis zu einem Zielknoten noch anfallen wird. Die Funktionen $g^*(n)$ und $h^*(n)$ bezeichnen jeweils die optimalen Werte für einen Knoten n. Die Abweichung von $g(n)$ gegenüber $g^*(n)$ kann dabei durch Wahl eines suboptimalen Weges von einem Startknoten zum Knoten n entstehen, wohingegen die Abweichung von $h(n)$ gegenüber $h^*(n)$ durch eine falsche Schätzung entsteht. $f^*(n)$ ergibt sich dann wie folgt:

$$f^*(n) = g^*(n) + h^*(n)$$

Der Algorithmus terminiert, sobald der erste Zielknoten zur Expansion ausgewählt wird. Sollte die OPEN-Liste leer werden, so terminiert der Algorithmus mit einem Fehlschlag.

Sowohl die zeitliche als auch die räumliche Komplexität von A^* beträgt $O(b^t)$ (vgl. [Cun94, Kap. 2.1]), wobei b die durchschnittliche Verzweigung und t die Tiefe des Baums ist.

Anforderungen an A^*

Definition 4.1 *Die Funktion $g(\cdot)$ heißt zulässig, wenn sie nicht überschätzt, dass heißt wenn gilt:*

$$\forall n : g(n) \leq g^*(n)$$

Definition 4.2 *Die Funktion $h(\cdot)$ heißt optimistisch, wenn sie nicht unterschätzt, dass heißt wenn gilt:*

$$\forall n : h(n) \geq h^*(n)$$

Definition 4.3 *Die Funktion $h(\cdot)$ heißt zulässig, falls sie optimistisch ist und zusätzlich für Knoten aus der Menge $\mathcal{T}$ der terminalen Knoten gilt:*

$$\forall n \in \mathcal{T} : h(n) = 0$$

Beim A^*-Algorithmus mit Maximierungsziel müssen $g(\cdot)$ und insbesondere $h(\cdot)$ im Sinne dieser Definitionen zulässig sein. Sind sie es nicht, spricht man lediglich vom A-Algorithmus. Dieser hat jedoch keinen Optimalitätsanspruch.

Eigenschaften von A^*

Falls die oben genannten Forderungen an $g(\cdot)$ und $h(\cdot)$ erfüllt sind, besitzt der A^*-Algorithmus für endliche Graphen die im Folgenden genannten Eigenschaften (1–3 nach [Cun94, Kap.2.1]; 4 nach [Bec02, Kap. „Heuristische Suche“ S. 11]).

Eigenschaft 4.1 *Der A^*-Algorithmus terminiert und findet einen Zielknoten, falls einer existiert und erreichbar ist.*

Beweis 4.1: Zuerst soll per Induktion gezeigt werden, dass der A^*-Algorithmus allen möglichen Wegen, ausgehend von den Startknoten, folgt.

- **Induktionsanfang:** Durch die Initialisierung der OPEN-Liste mit allen Startknoten, ist jeweils ein Knoten von jedem möglichen Weg in der OPEN-Liste.
- **Induktionshypothese:** Zu jeder Zeit befindet sich mindestens ein Knoten eines jeden Weges, ausgehend von einem Startknoten, in der OPEN-Liste.
- **Zu zeigen:** Befinden sich zum Zeitpunkt i von jedem Weg mindestens ein Knoten in der OPEN-Liste, so auch zum Zeitpunkt $i + 1$.

Da bei der Expansion eines Knotens *alle* Nachfolgerknoten in die OPEN-Liste aufgenommen werden, welche nicht schon in der OPEN- oder CLOSED-Liste sind, werden alle Wege weiter beschritten, welche nicht schon zuvor beschritten wurden. Also ist wieder zu jedem möglichen Weg mindestens ein Knoten in der OPEN-Liste. Damit ist gezeigt, dass allen Wegen, ausgehend von einem Startknoten gefolgt wird.

Da bei jedem Schritt auf einem Weg vorangegangen wird und der Graph endlich ist, sind nach endlicher Zeit alle erreichbaren Knoten untersucht. Dass heißt, der Algorithmus terminiert genau dann, wenn ein Zielknoten erreicht ist oder alle erreichbaren Knoten untersucht wurden.

Also findet der A^*-Algorithmus immer dann einen Zielknoten, wenn einer existiert und erreichbar ist. □

Eigenschaft 4.2 *Der A^*-Algorithmus liefert einen bezüglich der Belohnung optimalen Zielknoten zurück, falls einer existiert und erreichbar ist.*

Beweis 4.2:

- Falls ein Zielknoten existiert und erreichbar ist, so terminiert A^* nach Eigenschaft 4.1 mit einem Erfolg.
- Angenommen, A^* hat ausgehend vom Startknoten s mit einem suboptimalen Zielknoten t terminiert.
  Dann gilt:
  $$f(t) = g(t) < f^*(s)$$
- Basierend auf den Eigenschaften 4.1 und 4.2 gibt es aber in der OPEN-Liste immer, also auch unmittelbar vor der Terminierung,
  - einen Knoten n
  - auf einem Pfad vom Startknoten s zum optimalen Zielknoten
  - mit $f(n) \geq f^*(s)$
- Das ergibt zusammen:
  $$f(n) \geq f^*(s) > f(t)$$
- Damit hätte A^* aber statt dem Knoten t den Knoten n zur Expansion ausgewählt.
- A^* kann also, falls ein Zielknoten existiert, nicht mit einem suboptimalen Zielknoten terminieren. Also terminiert er mit dem optimalen Zielknoten. □

Eigenschaft 4.3 *Sei $h(\cdot)$ zulässig und monoton, dass heißt, für alle Nachfolger n_j von n_i gilt:*
$$h(n_i) - h(n_j) \geq c(n_i, n_j).$$
Dann betrachtet A^* denselben Knoten nicht zweimal, unabhängig vom Vorhandensein einer CLOSED-Liste. Dabei sei $c(n, m)$ die Belohnung die beim Übergang vom Knoten n zum Knoten m hinzugewonnen wird.

Beweis 4.3:

- $h(n_i) - h(n_j) \geq c(n_i, n_j)$
- $h(n_i) + g(n_i) - h(n_j) \geq c(n_i, n_j) + g(n_i)$
- $f(n_i) - h(n_j) \geq g(n_j)$
- $f(n_i) \geq g(n_j) + h(n_j)$
- $f(n_i) \geq f(n_j)$

Bei einem zweiten Expandieren eines Knotens n müsste $f(n)$ größer geworden sein. Dies ist aber wie eben gezeigt nicht möglich. Also kann ein Knoten kein zweites Mal expandiert werden. □

Eigenschaft 4.4 *Falls $h_1(\cdot)$ und $h_2(\cdot)$ zulässig sind und $\forall n : h_1(n) \geq h_2(n)$ gilt: A^* expandiert mit h_2 nicht mehr Knoten als mit h_1.*

Beweis 4.4:
Vereinbarung:

- A_1^* verwende die Schätzfunktion $f_1(n) = g_1(n) + h_1(n)$
- A_2^* verwende die Schätzfunktion $f_2(n) = g_2(n) + h_2(n)$

Jetzt Induktion über die Tiefe k der Knoten im Baum (nach [Bec02, Musterlösung ÜS2]):

- **Induktionsanfang:** Wenn A_2^* einen Knoten n auf Tiefe $k = 0$ (also $n = s$) expandiert (Startknoten ist nicht terminal), dann wird dies auch A_1^* tun.
- **Induktionshypothese:** A_1^* expandiert alle Knoten, die von A_2^* expandiert werden und im Suchbaum von A_2^* eine Tiefe $\leq k$ besitzen.
- **Zu zeigen:** Jeder Knoten n, der von A_2^* im Suchbaum von A_2^* auf der Tiefe $k + 1$ expandiert wird, wird auch von A_1^* expandiert.

Unter Ausnutzung der Induktionshypothese wissen wir, dass jeder Vorgänger von n im Suchbaum von A_2^* auch von A_1^* expandiert wird. Daraus folgt: Der Knoten n liegt auch im Suchbaum von A_1^* und es gibt einen Pfad von s nach n im Suchbaum von A_1^*, dessen Belohnung nicht die Belohnung eines Pfades von s nach n im Suchbaum von A_2^* unterschreitet:

$$g_1(n) \geq g_2(n)$$

Nehmen wir nun an, dass A_1^* nicht auch den von A_2^* expandierten Knoten n expandiert. Dann muss, bei Terminierung von A_1^*, der Knoten n in der OPEN-Liste von A_1^* stehen (A_1^* expandierte einen Eltern-Knoten von n). Andererseits wissen wir, dass A_1^* terminiert und einen Pfad mit maximaler Belohnung als Ergebnis liefert ohne dass der Knoten n expandiert wurde (aufgrund der Annahme). Also:

$$f_1(n) = g_1(n) + h_1(n) \leq f^*(s)$$

Mit $g_1(n) \geq g_2(n)$ gilt:

$$h_1(n) \leq f^*(s) - g_2(n)$$

Da A_2^* den Knoten n expandiert, gilt weiter

$$f_2(n) \geq f^*(s)$$

also

$$g_2(n) + h_2(n) \geq f^*(s)$$

und damit

$$h_2(n) \geq f^*(s) - g_2(n).$$

Zusammen mit der bereits gezeigten Ungleichung $h_1(n) \leq f^*(s) - g_2(n)$ ergibt dies aber

$$h_2(n) \geq h_1(n).$$

Dies steht aber im Widerspruch zur Bedingung. □

4.6.3 Besonderheiten von A^* bei der Merkmalauswahl

Eine Besonderheit bei der Anwendung des A^*-Algorithmus zur Merkmalauswahl ist, dass man

$$g^* = g$$

setzen kann. Dies ist klar, da sich die Güte einer Merkmalmenge nicht von einer zur anderen Berechnung ändern kann.

Dies hat zur Folge, dass ein Knoten, welcher einmal in der CLOSED-Liste ist, dort verbleibt und nicht wieder zurück in die OPEN-Liste kommen kann. Man kann sich also das Update der Knoten in der CLOSED-List sparen.

4.6.4 Der Lösungsgraph

Als Lösungsgraph kann man trivialer Weise einen Graphen nutzen, welcher $d+1$ Ebenen hat (Ebene 0 bis Ebene d). In der Ebene 0 gibt es nur einen Knoten. Dieser repräsentiert die leere Merkmalmenge und ist der Startknoten. In der Ebene 1 gibt es so viele Knoten wie Merkmale und jeder Knoten repräsentiert eine Merkmalmenge mit genau einem Merkmal. In Ebene 2 sind dann alle Kombinationen von 2 Merkmalen repräsentiert usw., bis schließlich in Ebene d alle Kombinationen von d Merkmalen repräsentiert sind.

Sei A_i die vom Knoten n_i repräsentierte Merkmalmenge. Dann ist zwischen den Knoten n_i in Ebene k und n_j in Ebene $k+1$ genau dann eine Kante vorhanden, wenn $A_i \subset A_j$. In Abbildung 4.10 ist solch ein Graph für die Auswahl von 4 aus 5 Merkmalen dargestellt.

Natürlich könnte man diesen Graphen auch als Baum darstellen.

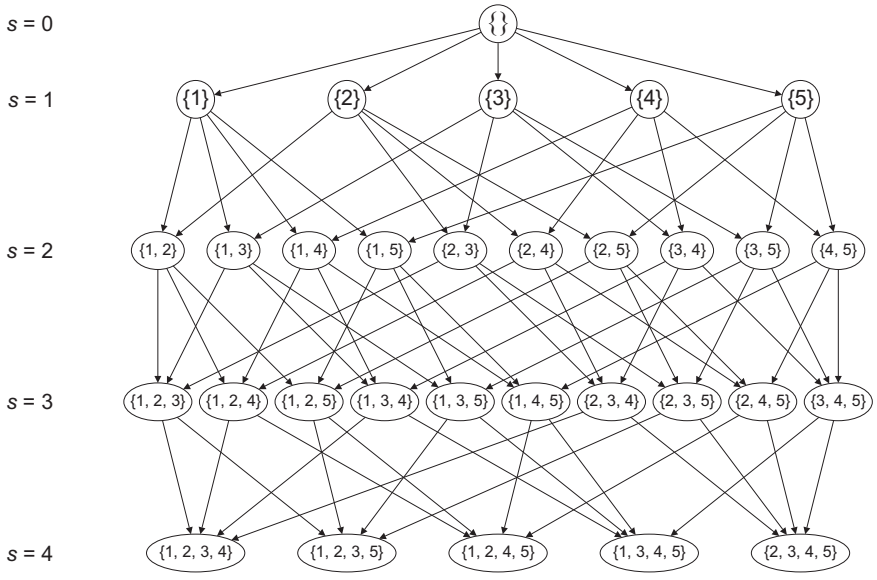

Abbildung 4.10: Einfacher Lösungsgraph für A^*-Algorithmus

4.6.5 Die Funktion f

Die triviale Funktion f

Am einfachsten ist es, für die Funktion $g(n)$ die Güte $J(n)$, mit $\forall n : 0 \leq J(n) \leq 1$, zu nutzen. Dann ist die optimale Restschätzung $h^*(n)$ durch $1 - g(n)$ nach oben beschränkt. Da $h(\cdot)$ nicht unterschätzen darf, ist die Restschätzung $h(n) = 1 \geq 1 - g(n)$ auf jeden Fall zulässig. Es muss aber noch eine Fallunterscheidung getroffen werden, um die Restschätzung für terminale Knoten auf 0 zu setzten. Dies ergibt:

$$h(n) = \begin{cases} 1 & \text{card}(n) < d \\ 0 & \text{card}(n) = d \end{cases}$$

Die daraus resultierende Funktion $f(\cdot)$ hat dann die folgende Form:

$$f(n) = \begin{cases} J(n) + 1 & \text{card}(n) < d \\ J(n) & \text{card}(n) = d \end{cases}$$

Diese Funktion gewährleistet ein optimales Ergebnis, es werden aber zwangläufig zuerst alle nichtterminalen Knoten expandiert, bevor der erste terminale Knoten expandiert wird. Dies ist aber äußerst uneffizient. Daher ist diese Funktion — wenn überhaupt — nur für sehr kleine Probleme praktikabel.

Die lineare Funktion f

Eine effizientere, aber nicht immer optimale Methode besteht darin, die Funktion $h(\cdot)$ linear von 1, für die leere Menge, auf 0, für die terminalen Mengen, fallen zu lassen. Dabei wird $g(\cdot)$ wieder wie oben gewählt, also $g(n) = J(n)$. Die Funktion $h(\cdot)$ wird jetzt aber durch eine lineare Funktion ersetzt. Dies ergibt:

$$h(n) = 1 - \frac{\text{card}(n)}{d}$$

Die daraus resultierende Funktion $f(\cdot)$ hat dann die folgende Form:

$$f(n) = J(n) + 1 - \frac{\text{card}(n)}{d}$$

Diese Funktion kann zwar nicht immer ein optimales Ergebnis gewährleisten, sie sollte aber ein nahoptimales Ergebnis in deutlich schnellerer Zeit liefern.

Die hyperbolische Funktion f

Eine noch effizientere, aber auch nicht immer optimale Methode besteht darin, die Funktion $h(\cdot)$ hyperbolisch von etwa 1, für die leere Menge, auf 0, für die terminalen Mengen, fallen zu lassen. Dabei wird $g(\cdot)$ erneut wie oben gewählt, also $g(n) = J(n)$. Die Funktion $h(\cdot)$ wird jetzt aber durch eine hyperbolische Funktion ersetzt. Das ergibt

$$h(n) = \frac{1}{\text{card}(n) + 1} - \frac{1}{d + 1},$$

wobei der rechte Bruch dafür sorgt, dass $h(n)$ für terminale Knoten 0 wird.

Die daraus resultierende Funktion $f(\cdot)$ hat dann die folgende Form:

$$f(n) = J(n) + \frac{1}{\text{card}(n)+1} - \frac{1}{d+1}$$

Diese Funktion kann zwar noch seltener ein optimales Ergebnis gewährleisten, sie sollte aber noch einmal schneller arbeiten.

Die Funktion f mit Vorhersage

Auch hier wird $g(\cdot)$ wieder wie oben gewählt, also $g(n) = J(n)$.

Komplizierter gestaltet sich jetzt hingegen die Funktion $f(\cdot)$. Hier wird jetzt die Güte einer Merkmalmenge S mit Zielgröße d prognostiziert, welche als Teilmenge die Merkmalmenge M_n des aktuellen Knotens n enthält ($M_n \subseteq S$, $\text{card}(S) = d$). Dabei entsteht S, indem die aktuelle Merkmalmenge M_n mit den $d - \text{card}(M_n)$ Merkmalen m_i aufgefüllt wird, welche die besten Güten $J(m_i)$ haben und noch nicht in A sind. Diese Schätzung sei mit $\hat{J}(n, d)$ bezeichnet.

Die genaue Funktionsweise der hier angesprochenen Schätzung kann man Anhang A entnehmen.

Die Funktion $h(\cdot)$ entsteht jetzt aus der Differenz der prognostizierten Güte und der aktuell erreichten Güte:

$$h(n) = \hat{J}(n, d) - J(n)$$

Die resultierende Funktion $f(\cdot)$ hat dann allerdings die folgende Form:

$$f(n) = \hat{J}(n, d)$$

Es wird also nur noch die Schätzung verwendet, die echte aktuelle Güte hat keinen direkten Einfluss mehr. Sie geht zwar noch über die Schätzung ein, aber trotzdem ist diese Variante eher unschön.

Die Funktion f mit Vorhersage mit schwindendem Einfluss

Die Funktion $g(\cdot)$ wird auch hier wieder wie oben gewählt, also $g(n) = J(n)$.

Die Funktion $h(\cdot)$ entsteht jetzt aber nicht mehr aus der Differenz der prognostizierten Güte und der aktuell erreichten Güte, sondern es wird die prognostizierte Gesamtgüte mit einer linear fallenden Konstante multipliziert. Es ergibt sich so die folgende Gleichung:

$$h(n) = \hat{J}(n, d) \cdot \left(1 - \frac{\text{card}(n)}{d}\right)$$

Die resultierende Funktion $f(\cdot)$ erhält so die folgende Form:

$$f(n) = J(n) + \hat{J}(n, d) \cdot \left(1 - \frac{\text{card}(n)}{d}\right)$$

Zusammenfassung

In Tabelle 4.2 sind die einzelnen Varianten nochmal zusammenfassend dargestellt:

	$g(\cdot)$	$h(\cdot)$
triviale Funktion	$g(n) = J(n)$	$h(n) = \begin{cases} 1 & \text{card}(n) < d \\ 0 & \text{card}(n) = d \end{cases}$
lineare Funktion	$g(n) = J(n)$	$h(n) = 1 - \frac{\text{card}(n)}{d}$
hyberbolische Funktion	$g(n) = J(n)$	$h(n) = \frac{1}{\text{card}(n)+1} - \frac{1}{d+1}$
Vorhersage	$g(n) = J(n)$	$h(n) = \hat{J}(n,d) - J(n)$
Vorhersage mit schw. Einfl.	$g(n) = J(n)$	$h(n) = \hat{J}(n,d) \cdot \left(1 - \frac{\text{card}(n)}{d}\right)$

Tabelle 4.2: Übersicht der Varianten der Funktionen $g(\cdot)$ und $h(\cdot)$ für A^*

4.7 Floating-A^*

Ein möglicher Verbesserungsansatz ist das Floating-Prinzip des SFFS-Algorithmus. Man kann es z. B. auf den A^*-Algorithmus übertragen, indem man bei der Expansion eines Knotens nicht nur seine Nachfolger, sondern auch seine Vorgänger mit in die OPEN-Liste aufnimmt, falls nicht schon vorhanden. Inwieweit dies wirklich einen Vorteil verschafft, muss noch experimentell geprüft werden.

4.8 $LazyA^*$

Ein anderer möglicher Verbesserungsansatz besteht im $LazyA^*$-Algorithmus. Hierbei handelt es sich eigentlich nicht um einen eigenen Algorithmus, sondern nur um eine modifizierte Variante des A^*-Algorithmus.

Im Unterschied zu A^* wird die Expansion eines Knotens bei $LazyA^*(p)$ in zwei Stufen durchgeführt:

1. die Güte aller Knoten nur schätzen
2. einen gewissen Anteil p der besten geschätzten Knoten echt berechnen

Zur Schätzung der Güten kommt dabei der selbe Schätzer zum Einsatz, welcher bei der prädiktiven Funktion h genutzt wurde.

Wird dann im weiteren Verlauf des Algorithmus ein Knoten mit nur geschätzter Güte zur Expansion ausgewählt, wird dessen Güte erst echt berechnet und die Auswahl dann nochmals überprüft (d. h. es wird neu ausgewählt). Nachdem eine neue Güte echt berechnet wurde, wird auf Basis des Ergebnisses der Schätzer aktualisiert. Danach werden alle Schätzungen neu berechnet.

Durch dieses Verfahren genügt $g(\cdot)$ zwar nicht mehr den Anforderungen von A^*, aber bei großen Problemen entsteht möglicher Weise eine deutliche Geschwindigkeitssteigerung.

4.9 Best-First mit mehreren Sortierkriterien

Der in diesem Kapitel vorgestellte A^*-Algorithmus ist genau genommen eine Spezialform des Best-First-Algorithmus. Das besondere, namensgebende Merkmal des Best-First-Algorithmus ist, dass jeweils das bestbewertetste Element aus der OPEN-Liste entnommen wird. Die Bewertung erfolgt beim A^*-Algorithmus durch eine Funktion, welche sich

additiv aus der aktuellen Güte und einer Restschätzung für den noch erreichbaren Gütezuwachs zusammensetzt.

Es liegt nun nahe zu versuchen andere Kriterien für die Bewertung zu nutzen. Prinzipiell gibt es für den Umgang mit mehreren Sortierkriterien mehrere Möglichkeiten. Zwei Möglichkeiten sind die folgenden:

- Kriterien normieren und dann per Addition zu einem zusammenfassen
- nacheinander sortieren, bei Sortierung $n+1$ nur die bestbewertetsten aus Sortierung n weiterführen — erfordert aber eine Reihenfolge der Kriterien

Der in dieser Arbeit vorgestellte A^*-Algorithmus ist also eine Variante der ersten Möglichkeit.

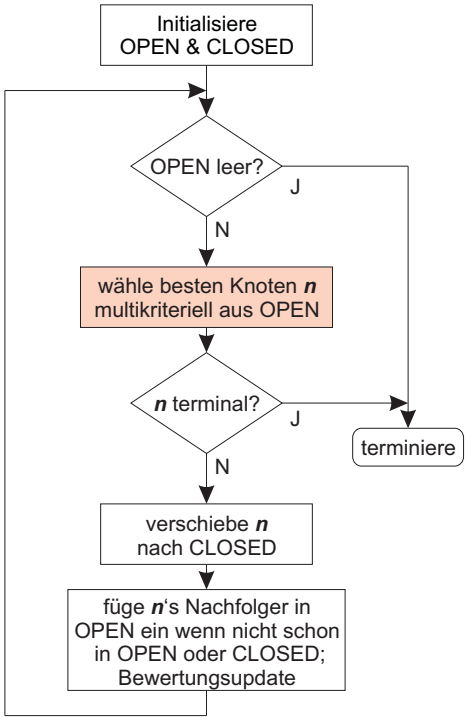

Abbildung 4.11: Flussdiagramm des Best-First-Algorithmus

Im Folgenden sollen einige mögliche Kriterien genannt werden. Dabei kommt für die Verknüpfung die zweite genannte Möglichkeit zum Einsatz, also mehrere hintereinandergeschaltete Sortierungen mit unterschiedlichen Sortierkriterien.

Variante 1

- zuerst Sortierung nach maximaler Güte
- innerhalb der Besten nach Güte, Sortierung nach minimaler Größe

Variante 2

- zuerst Sortierung nach maximaler Güte pro Merkmal
- innerhalb der dort Besten, Sortierung nach maximaler Größe

Wird wahrscheinlich nur Anfangs konkurrenzfähige Ergebnisse liefern, da für die letzten Verbesserungen der Güte nur geringe Verbesserungen pro Merkmal zu erwarten sind.

Variante 3

- zuerst Sortierung nach maximaler Güte
- innerhalb der Besten nach Güte, Sortierung größter Verbesserung der Güte bei letzter Änderung der Merkmalmenge

4.10 Floating-Best-First

Floating-Best-First ist wieder eine Variation des Best-First-Algorithmus. Bei der Expansion eines Knotens werden dabei nicht nur seine Nachfolger, sondern auch seine Vorgänger mit in die OPEN-Liste aufgenommen, falls nicht schon vorhanden.

4.11 Zusammenfassung

In Tabelle 4.3 sind nochmals die wichtigsten Eigenschaften der in diesem Kapitel vorgestellten Algorithmen zusammengefasst.

Name	Komplexität	Parameter	Bemerkung
ES	$O(2^D)$	keine	
BB	$O(2^D)$	keine	nur Rückwärtsvariante
BB+	$O(2^D)$	keine	nur Rückwärtsvariante
FBB(γ, δ)		2	nur Rückwärtsvariante
IBB		keine	nur Rückwärtsvariante
A^*		keine	
Floating-A^*		keine	
$LazyA^*(p)$		1	
Best-First		keine	
Floating-Best-First		keine	

Tabelle 4.3: Übersicht der Eigenschaften der vorgestellten Graph-Suchalgorithmen

Kapitel 5

Realisierung und Evaluierung

In diesem Kapitel sollen einige der in dieser Arbeit vorgestellten Algorithmen umgesetzt und ihre Ergebnisse vorgestellt werden. Dabei sollen die nachstehenden Verfahren implementiert werden:

- SFS und SFFS
- ParallelSFFS
- LazySFFS und LazyGSFFS
- A^* und $LazyA^*$
- Best-First
- Floating-$A*$ und Floating-Best-First

Die Algorithmen sollen dabei unter dem Gesichtspunkt einer möglichst guten Effizienz betrachtet werden. Dass heißt, es soll eine möglichst hohe Güte mit einer fest vorgegebenen Anzahl von Berechnungen erreicht werden, unabhängig von der Merkmalzahl. Darin unterscheidet sich diese Untersuchung von den meisten bisherigen, welche versuchten, eine möglichst gute Güte bei einer fest vorgegebenen Anzahl von Merkmalen zu erreichen, unabhängig von der Anzahl der dafür benötigen Güteberechnungen.

Dabei stehen für die Tests die folgenden zwei Problemstellungen mit insgesamt vier verschiedenen Datensätzen zur Verfügung:

- Lokalisierung der Bindungsstellen der Transkriptionsfaktoren (TFBS) AP1, MEF2 und SP1 [Pud04]
- Klassifikation von Sonardaten [Bla98]

Die Problemstellung der Klassifikation von Sonardaten wurde in die Arbeit mit aufgenommen, um einen besseren Vergleich der neu vorgestellten Algorithmen mit älteren Arbeiten zu ermöglichen.

5.1 Die Testdaten

5.1.1 Transkriptionsfaktoren-Bindungsstellen-Daten

Die Berechnung der Güte $J(\cdot)$ soll bei der Lokalisierung der Bindungsstellen von Transkriptionsfaktoren durch das von Pudimat et al. in [Pud04] vorgestellte System erfolgen. Dabei kommt ein Bayes-Netz zum Einsatz, dessen Leistung mittels eines F-Measures bewertet wird, welches immer einen Wert aus dem Intervall [0,1] annimmt. Das F-Measure wird dabei mittels der folgenden Formel aus den bekannten Werten *Recall* und *Precision* berechnet:

$$F = 2 \cdot \frac{Recall \cdot Precision}{Recall + Precision}$$

Für die drei Transkriptionsfaktoren kommen dabei unterschiedliche Datensätze zum Einsatz. Die Merkmalzahlen und die Anzahl der Datensätze, der positiven und der negativen Klasse, kann man Tabelle 5.1 entnehmen.

Name	Merkmalzahl	Anzahl pos. Datensätze	Anzahl neg. Datensätze
AP1	495	77	4577
MEF2	653	26	4577
SP1	802	108	4578

Tabelle 5.1: Beschaffenheit der Daten zur Lokalisierung der Bindungsstellen von Transkriptionsfaktoren

5.1.2 Sonardaten

Ziel ist es, zwischen Signalen, welche von Minen reflektiert wurden und solchen die von Felsen reflektiert wurden, zu unterscheiden. Es stehen 208 Datensätze mit 60 Merkmalen zur Verfügung, welche in 2 Klassen mit 111 und 97 Datensätze aufgeteilt sind. Bei der Klassifikation der Sonardaten (von [Bla98]) soll die Güte, wie in [Kud00a], durch einen Leave-One-Out-1-NN-Klassifikator berechnet werden.

5.2 Implementierungsdetails

5.2.1 SFS, SFFS, LazySFFS und LazyGSFFS

Der SFS-, der SFFS-, der LazySFFS und der LazyGSFFS-Algorithmus wurden bei der Implementierung so abgeändert, dass immer die Merkmalmenge mit der besten gefundenen Güte zurückgegeben wird. Die Größe der Merkmalmenge spielt also beim Ergebnis keine Rolle mehr. Deshalb wurde auch das „normale“ Abbruchkriterium (erreichen der gewünschten Merkmalzahl) entfernt und der Abbruch nach einer vorgegebenen Anzahl von Berechnungen durchgeführt, sofern nicht vorher schon alle Merkmale ausgewählt wurden.

5.2.2 ParallelSFFS

Auch der ParallelSFFS-Algorithmus wurde so implementiert, dass immer die Merkmalmenge mit der besten gefundenen Güte zurückgegeben wird, unabhängig von der Merkmalzahl.

Das zweite wichtige Implementierungsdetail ist die Anzahl der parallel laufenden Teilalgorithmen. Diese wurde pauschal auf 5 festgelegt.

Eine Erhöhung der Anzahl sollte nie dazu führen, dass ein gutes Ergebnis übersehen wird. Es wird aber immer zu einer Erhöhung des Rechenaufwandes führen. Umgekehrt erreicht man mit einer Absenkung einen geringeren Rechenaufwand, aber es sollte dadurch kein besseres Ergebnis gefunden werden. Deshalb wurde als Kompromiss zwischen möglichst geringem Rechenaufwand und möglichst breitangelegter Suche der obige Wert ohne weitere Untersuchungen festgelegt.

5.2.3 A^*, Floating-A^* und $LazyA^*$

Die einzige wesentliche Freiheit bei der Umsetzung des A^*-Algorithmus und seiner Variationen besteht in der Wahl der Restschätzung h. Dabei wurden die triviale und die lineare Restschätzung gar nicht erst implementiert, da sie viel zu aufwendig sind. Implementiert wurden aber die hyperbolische und für A^* selbstverständlich die prädiktive Restschätzung.

Auch hier soll der Abbruch des Algorithmus wieder nach einer fest vorgegebenen Anzahl von Berechnungen erfolgen. Dies hat aber zur Folge, dass für die prädiktive Restschätzung eigentlich die Zielgröße fehlt, welche nötig ist um die aktuelle Merkmalmenge auffüllen zu können. Daher wurde die prädiktive Restschätzung so implementiert, dass die aktuelle Merkmalmenge immer mit einer fest vorgegebenen Anzahl von Merkmalen aufgefüllt wird.

Ein anderes Implementierungsdetail ist die Reihenfolge der Abarbeitung der OPEN-Liste. Falls dabei im Verlauf des Algorithmus mehrere Knoten mit gleicher Bewertung $f(\cdot)$ zur Auswahl stehen, wird der Knoten zuerst expandiert, welcher zuletzt zur OPEN-Liste hinzugefügt wurde.

5.2.4 Best-First und Floating-Best-First

Beim Best-First-Algorithmus mit mehreren Sortierkriterien wurden alle drei vorgestellten Varianten und die entsprechenden Floating-Best-First-Varianten auch implementiert, also:

- V1: 1. maximale Güte; 2. minimale Größe
- V2: 1. maximale Güte pro Merkmal; 2. maximale Größe
- V3: 1. maximale Güte; 2. maximale Verbesserung bei Erzeugung der Merkmalmenge

Ein kleines Implementierungsdetail ist wie beim verwanden A^*-Algorithmus die Reihenfolge der Abarbeitung der OPEN-Liste. Falls dabei im Verlauf des Algorithmus mehrere Knoten mit gleicher Bewertung zur Auswahl stehen, wird der Knoten zuerst expandiert, welcher zuletzt zur OPEN-Liste hinzugefügt wurde.

5.3 Testergebnisse

In diesem Abschnitt sollen die Algorithmen auf ihre Effizienz hin getestet werden. Dazu laufen die Algorithmen ohne den beschriebenen Abbruch bei einer bestimmten Merkmalmengengröße. Sie halten erst nach einer vorgegebenen Anzahl von Güteberechnungen und liefern die Merkmalmenge mit der besten überhaupt gefundenen Güte zurück. Somit erhält man einen guten Vergleich, welcher Algorithmus in einer bestimmten Zeit welche Ergebnisse liefern kann.

Dies ist dann praxisrelevant, wenn die benötige Rechenzeit im Vordergrund steht, weil der Algorithmus z. B. regelmäßig wiederholt werden muss.

5.3.1 SFS und SFFS

Die Ergebnisse des SFS-Algorithmus und des SFFS-Algorithmus sind in Abbildung 5.1 dargestellt.

Dabei sind die Abbruchzeitpunkte bei TFBS(AP1) und TFBS(SP1) an Stellen, an denen ein weiterrechnen kaum noch sinnvoll ist. Dies hat seine Ursache darin, dass zu diesen Zeitpunkten bereits relativ große Merkmalmengen untersucht werden, was beim verwendeten Klassifikator einen sehr hohen Rechenaufwand hervorruft.

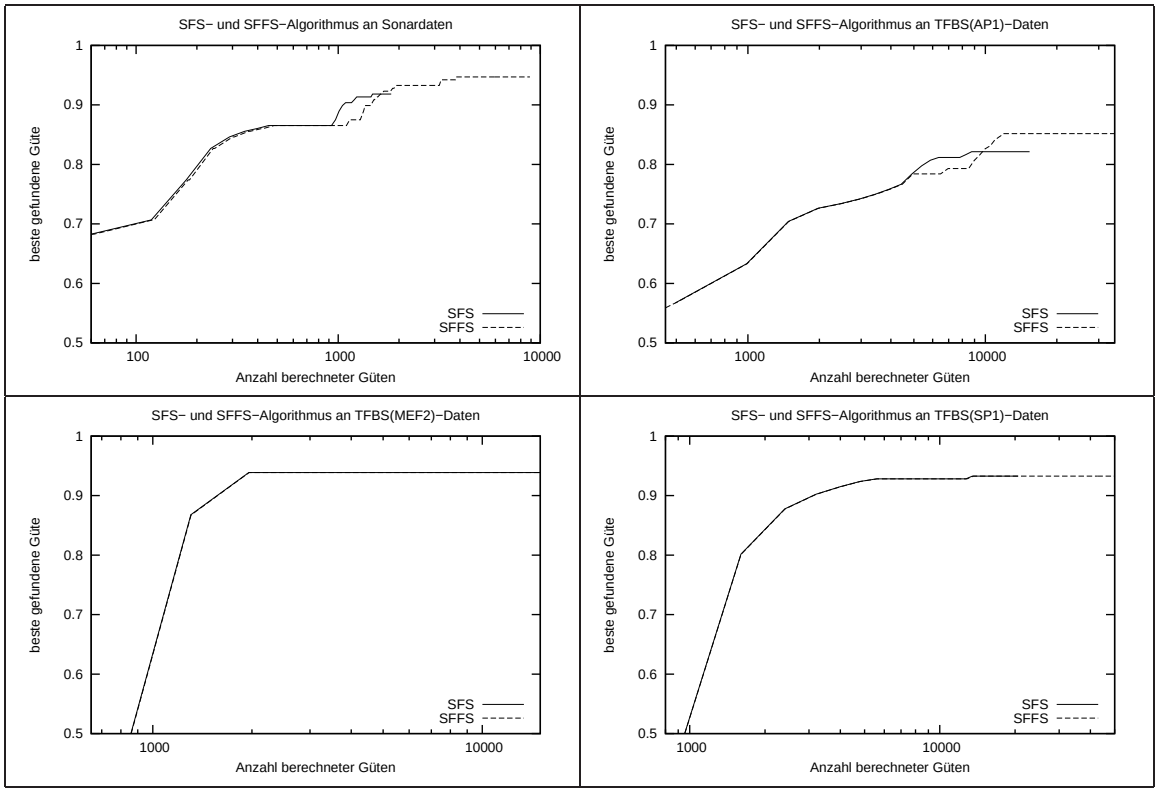

Abbildung 5.1: Effizienztestergebnisse des SFS- und des SFFS-Algorithmus an Sonardaten (oben links) und TFBS-Daten (o. r.: AP1; u. l.: MEF2; u. r.: SP1)

Es ist zu erkennen, dass der SFS-Algorithmus nur dann einen vorübergehenden Effizientvorteil bietet, wenn der SFFS-Algorithmus mittelfristig bessere Ergebnisse liefert.

Dies hat seine Ursache in dem in der Einführung des Algorithmus genannten Vernestungsproblem des SFS-Algorithmus. Der SFFS-Algorithmus kann ein einst als optimal hinzugefügtes Merkmal wieder entfernen, sobald es störend wirkt. Dies kostet jedoch Zeit, was zu einem vorübergehenden Effizienznachteil im Vergleich zum SFS-Algorithmus führt. Der SFS-Algorithmus kann ein solches Merkmal jedoch nicht wieder entfernen, was aber mittelfristig zu schlechteren Güten führt.

Die Effizienzvorteile des SFS-Algorithmus sollten danach nur temporär sein, der SFFS-Algoritmus also mittel- bis langfristig immer besser.

5.3.2 LazySFFS

Die Testergebnisse des LazySFFS-Algorithmus sind in den Abbildungen 5.2 und 5.3 dargestellt. Dabei gibt der erste der beiden eingeklammerten Werte den Anteil an besten Merkmalmengen an, für welche die Güte echt berechnet wurde. Der zweite Wert gibt an, nach wie vielen Berechnungen das Anlernen des Schätzers abgebrochen wird. Dies ist nötig, da der Aufwand für das Lernen quadratisch ansteigt und somit nicht beliebig fortgeführt werden kann. Bei der Anzahl der berechneten Güten wurden auch die Güten mit gezählt, welche zur Initialisierung des Schätzers verwendet wurden.

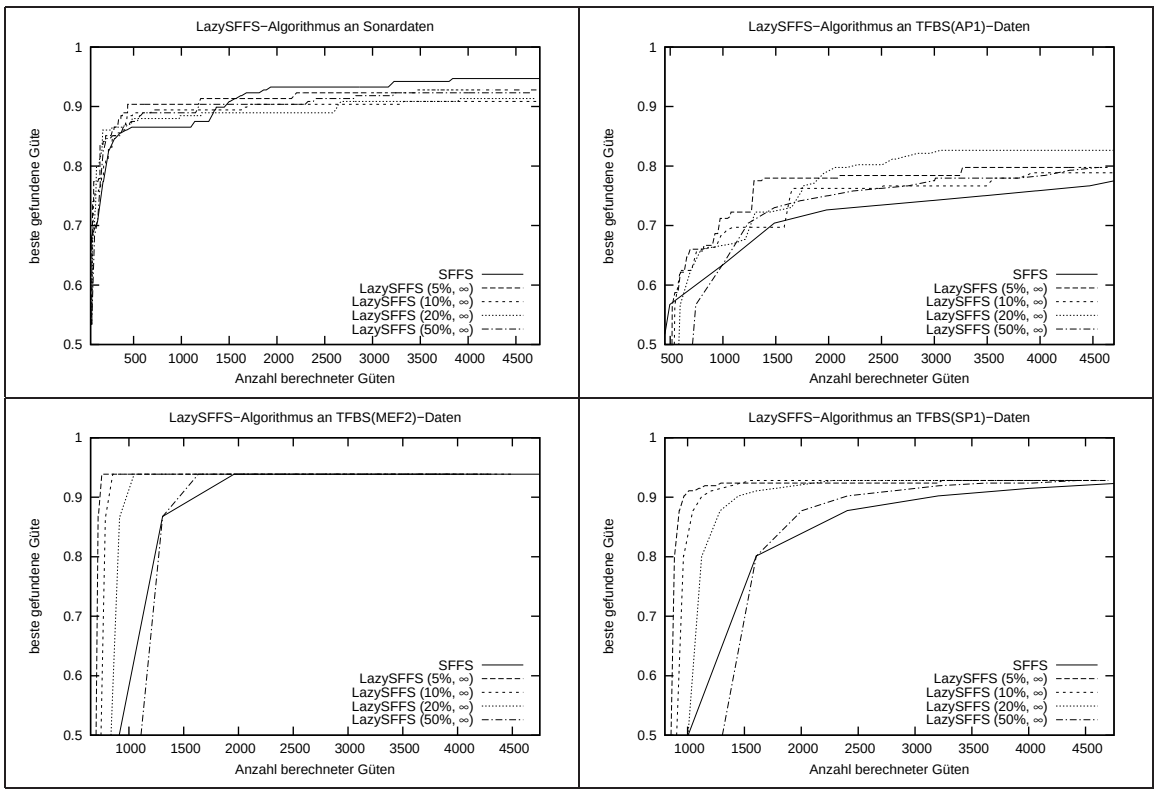

Abbildung 5.2: Effizienztestergebnisse des LazySFFS-Algorithmus bzgl. des Berechnungsanteil an Sonardaten (oben links) und TFBS-Daten (o. r.: AP1; u. l.: MEF2; u. r.: SP1)

In Abbildung 5.2 sind die Ergebnisse in Abhängigkeit vom Anteil echt berechneter Güten dargestellt. Es ist zu erkennen, dass die Variante mit einem Anteil von 5% zwar anfangs am effizientesten unter den getesteten Algorithmen arbeitet, im späteren Verlauf

jedoch von den anderen übertroffen wird. Den besten Kompromiss zwischen anfänglicher Effizienz und hoher Güte im späteren Verlauf stellt offensichtlich ein Anteil von 20% dar.

Dieser Anteil wurde dann für die weiteren Tests des LazySFFS und die späteren Tests anderer Lazy-Algorithmen beibehalten.

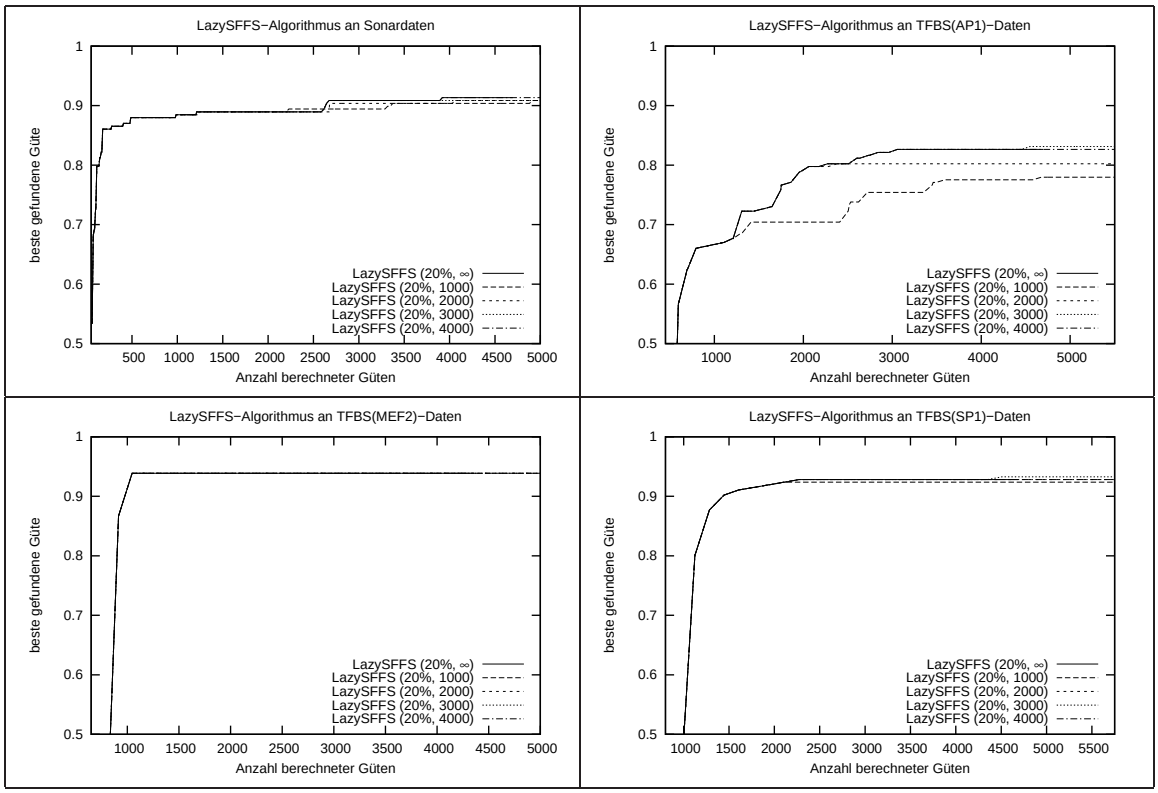

Abbildung 5.3: Effizienztestergebnisse des LazySFFS-Algorithmus bzgl. des Lernabbruchs an Sonardaten (oben links) und TFBS-Daten (o. r.: AP1; u. l.: MEF2; u. r.: SP1)

In Abbildung 5.3 sind die Ergebnisse in Abhängigkeit vom Zeitpunkt das Lernabbruchs dargestellt. Es ist erwartungsgemäß zu erkennen, dass besonders frühe Abbrüche einen deutlichen Verlust an Güte verursachen. Bei etwa 3000 Lernschritten ist dann bei den meisten Testdaten das Maximum an Güte erreicht und fällt dann teilweise mit 4000 Lernschritten schon wieder ab. Also ist es am besten, den Lernabbruch nach 3000 Schritten vorzunehmen.

Diese Anzahl an Lernschritten wurde dann für die weiteren Tests des LazySFFS und allen späteren Tests, welche auf den Schätzer zurückgreifen, beibehalten.

Im Vergleich zum SFFS-Algorithmus bietet der LazySFFS-Algorithmus anfangs einen starken Effizienzvorteil, welcher aber im weiteren Verlauf schwindet. Im Fall der TFBS(AP1)-Daten konnte der LazySFFS-Algorithmus im untersuchten Bereich auch nicht die Spitzen-Güte erreichen, welche der SFFS-Algorithmus erreichte. Dafür konnte er im Fall der TFBS(SP1)-Daten eine höhere Güte erreichen. Dies ist wahrscheinlich auf ein häufigeres Entfernen von Merkmalen zurückzuführen.

5.3.3 ParallelSFFS und LazyGSFFS

Die Testergebnisse des ParallelSFFS- und des LazyGSFFS-Algorithmus sind gemeinsam in Abbildung 5.4 dargestellt.

Die Angabe in Klammern beim ParallelSFFS-Algorithmus gibt die Anzahl der parallel laufenden Teilalgorithmen an (hier immer 5).

Die beiden Werte in der ersten Klammer beim LazyGSFFS-Algorithmus gibt die Anzahl der hinzuzufügenden Merkmale pro Vorwärtsschritt (erster Wert) und die Anzahl der zu entfernenden Merkmale pro Rückwärtsschritt (zweiter Wert) an. Die Werte in der zweiten Klammer geben wieder die vom LazySFFS-Algorithmus bekannten Parameter für die Schätzung an (Erster: Anteil echt zu berechnender Güten; Zweiter: Lernabbruch) und wurden auch nach den dortigen Erkenntnissen gewählt.

Für die Sonardaten wurde auch ein Testlauf für den GSFFS-Algorithmus durchgeführt. Dabei entfällt natürlich die zweite Angabe.

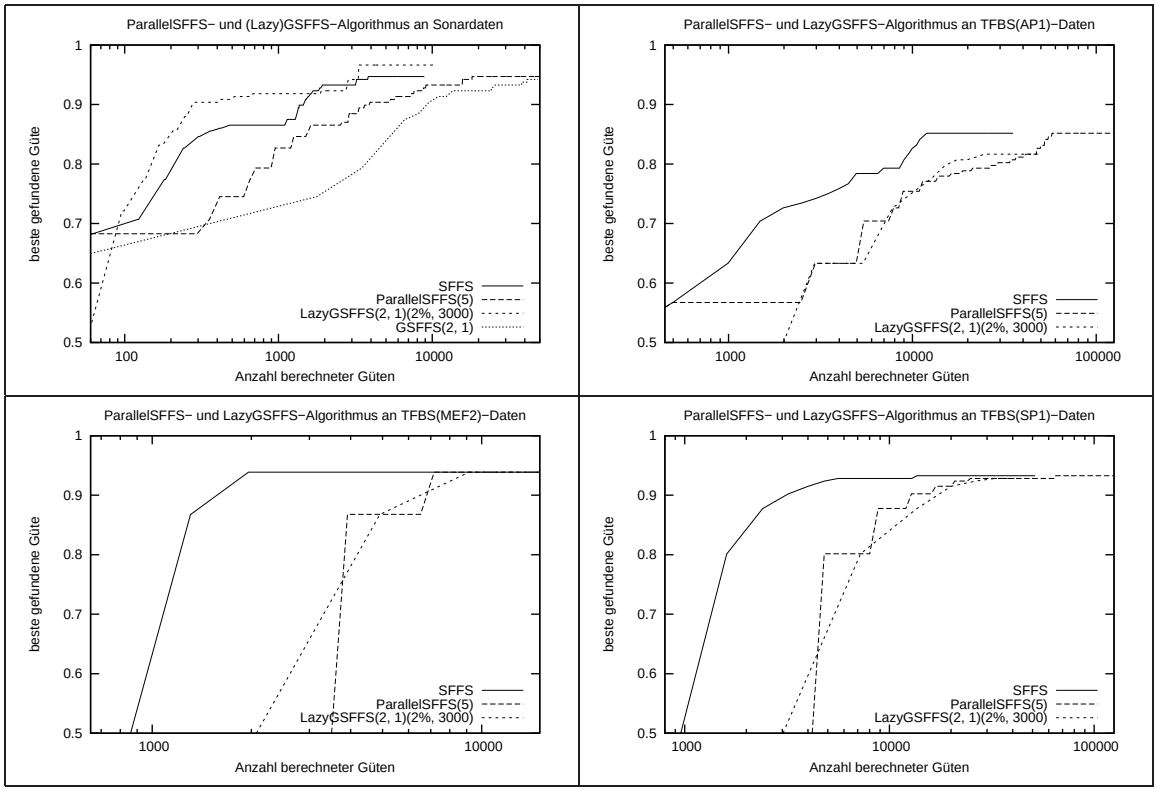

Abbildung 5.4: Effizienztestergebnisse des ParallelSFFS- und das LazyGSFFS-Algorithmus im Vergleich mit SFFS an Sonardaten (oben links) und TFBS-Daten (o. r.: AP1; u. l.: MEF2; u. r.: SP1)

Es ist aber festzustellen, dass diese Erweiterungen an den untersuchten TFBS-Daten keine Verbesserung erzielen konnten. Einzig an den Sonardaten war mit dem LazyGSFFS-Algorithmus eine Verbesserung der bisherigen Ergebnisse möglich. Dafür blieben aber die Ergebnisse dieses Algorithmus an den TFBS-Daten hinter denen des zugrunde liegenden SFFS-Algorithmus teilweise deutlich zurück.

Der ParallelSFFS-Algorithmus lieferte als Endergebnis jeweils die gleichen Resultate wie der SFFS-Algorithmus, benötigte dafür aber deutlich mehr Berechnungen.

5.3.4 A^*, Floating-A^* und $LazyA^*$

Die Ergebnisse des A^*- und des Floating-A^*-Algorithmus sind in Abbildung 5.5 dargestellt. In Klammern ist dabei jeweils die Art der verwendeten Funktion h angegeben.

Die Ergebnisse des $LazyA^*$-Algorithmus sind ebenfalls mit in Abbildung 5.5 dargestellt. Dabei gibt der erste der beiden eingeklammerten Werte wieder den Anteil an besten Merkmalmengen an, für welche die Güte echt berechnet wurde. Der zweite Wert gibt an, nach wie vielen Berechnungen das Anlernen des Schätzers abgebrochen wird. Beide Werte wurden entsprechend den Testergebnissen des LazySFFS-Algorithmus gewählt. In den zweiten Klammern ist wieder die Art der verwendeten Funktion h angegeben. Bei der Anzahl der berechneten Güten wurden auch hier wieder die Güten mit gezählt, welche zur Initialisierung des Schätzers verwendet wurden.

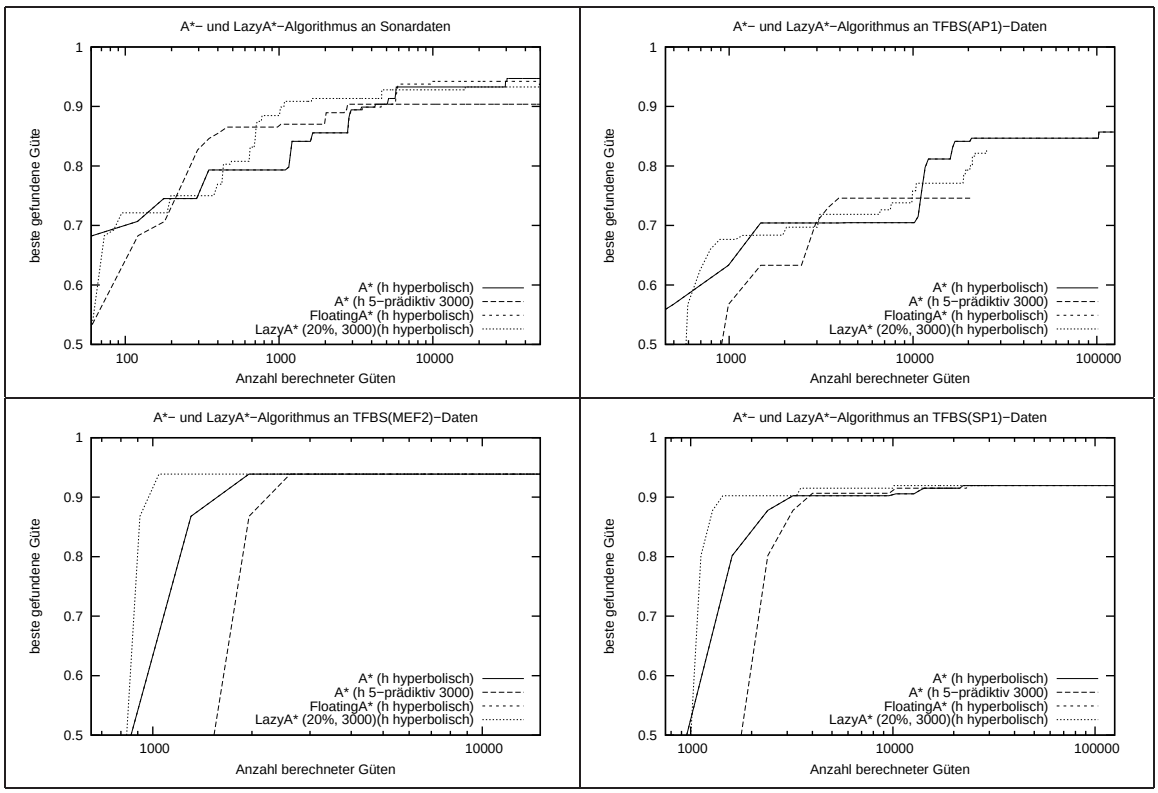

Abbildung 5.5: Effizienztestergebnisse des A^*-, Floating-A^*- und $LazyA^*$-Algorithmus an Sonardaten (oben links) und TFBS-Daten (o. r.: AP1; u. l.: MEF2; u. r.: SP1)

Man kann erkennen, dass die Floating-Erweiterung keinen nennenswerten Vorteil verschafft.

Beim Vergleich der Restschätzfunktionen kann man erkennen, dass die prädiktive Restschätzung nur im mittleren Teil einen Vorteil bietet. Am Anfang ist dies damit zu erklären, das sie durch die Initialisierung des Schätzers ins Hintertreffen gerät. Im späteren Verlauf fehlt ihr es vermutlich an Präzision.

Es ist zu erkennen, dass der $LazyA^*$-Algorithmus am Anfang teilweise einen Effizienzvorteil gegenüber A^* bietet, welcher sich aber im Verlauf der Berechnung umkehrt. Außerdem musste festgestellt werden, dass der $LazyA^*$-Algorithmus sehr schnell zu größeren Merkmalmengen übergeht. Dies führt speziell bei den TFBS-Daten zu einem schnellen

Ansteigen des Rechenaufwandes. Daher wurden die Berechnungen für diesen Algorithmus vorzeitig abgebrochen. Der $LazyA^*$-Algorithmus ist also in Fällen, in denen der Rechenaufwand mit steigender Merkmalzahl schnell steigt im Nachteil. Der A^*-Algorithmus ist im Vorteil, wenn zu erwarten ist, dass die optimale Merkmalmenge eine geringe Merkmalzahl aufweist.

5.3.5 Best-First und Floating-Best-First

Die Ergebnisse des Best-First- und des Floating-Best-First-Algorithmus sind in Abbildung 5.6 dargestellt.

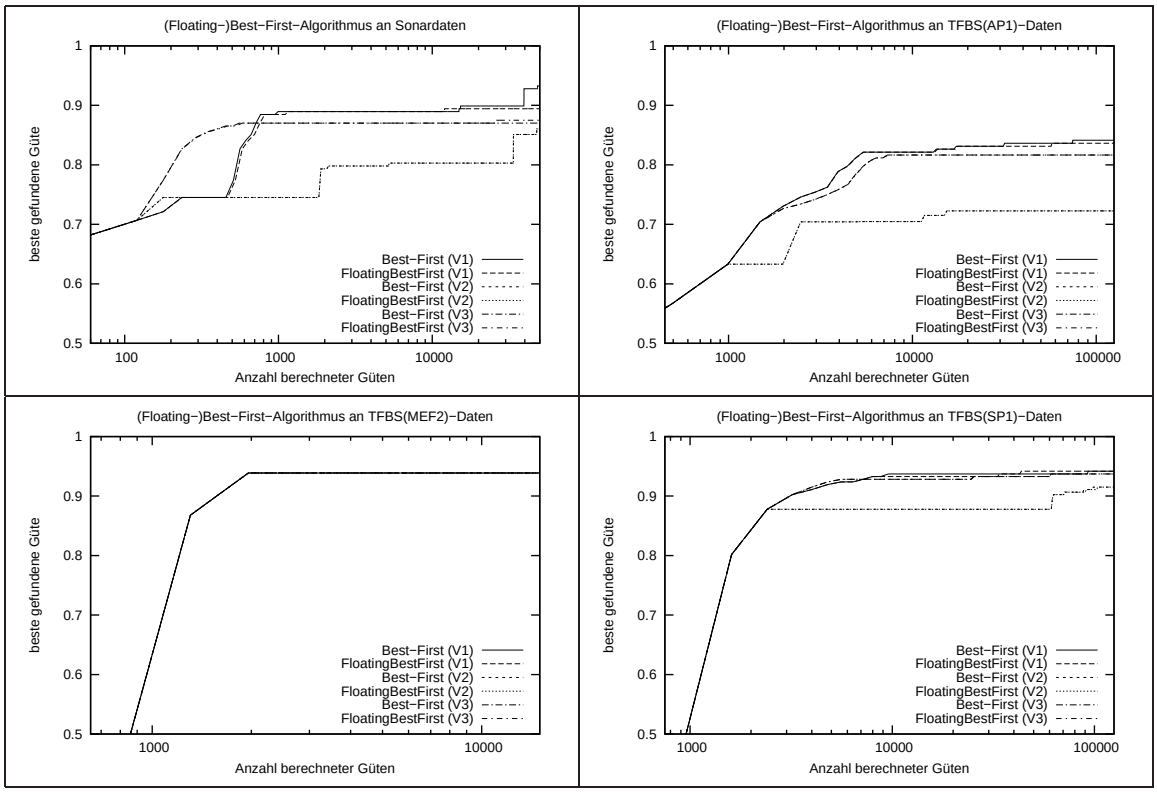

Abbildung 5.6: Effizienztestergebnisse des Best-First- und des Floating-Best-First-Algorithmus an Sonardaten (oben links) und TFBS-Daten (o. r.: AP1; u. l.: MEF2; u. r.: SP1)

Es ist zu erkennen, dass die Varianten 1 meist effizienter ist als die anderen beiden. Besonders deutlich wird dies im Vergleich zu Variante 2. Daher ist Variante 1 sicher zu bevorzugen.

Ebenfalls in Abbildung 5.6 sind die Ergebnisse der Floating-Variante des Best-First-Algorithmus dargestellt. Es ist aber zu erkennen, dass diese Erweiterung keine nennenswerten Verbesserungen erbringt.

5.3.6 Gesamtvergleich

In Abbildung 5.7 sind nochmal die Ergebnisse, bei vorgegebener Anzahl von Güteberechnungen, der in ihren Gruppen besten Algorithmen zusammengefasst. Außerdem ist zum

Vergleich die beste Güte eingetragen, welche durch manuelle Auswahl der Merkmale bei den TFBS-Daten erzielt werden konnte. Anhand dieser kann man erkennen, dass alle in diesem Diagramm aufgeführten Algorithmen bessere Ergebnisse liefern konnten als es mit einer manuellen Auswahl möglich war.

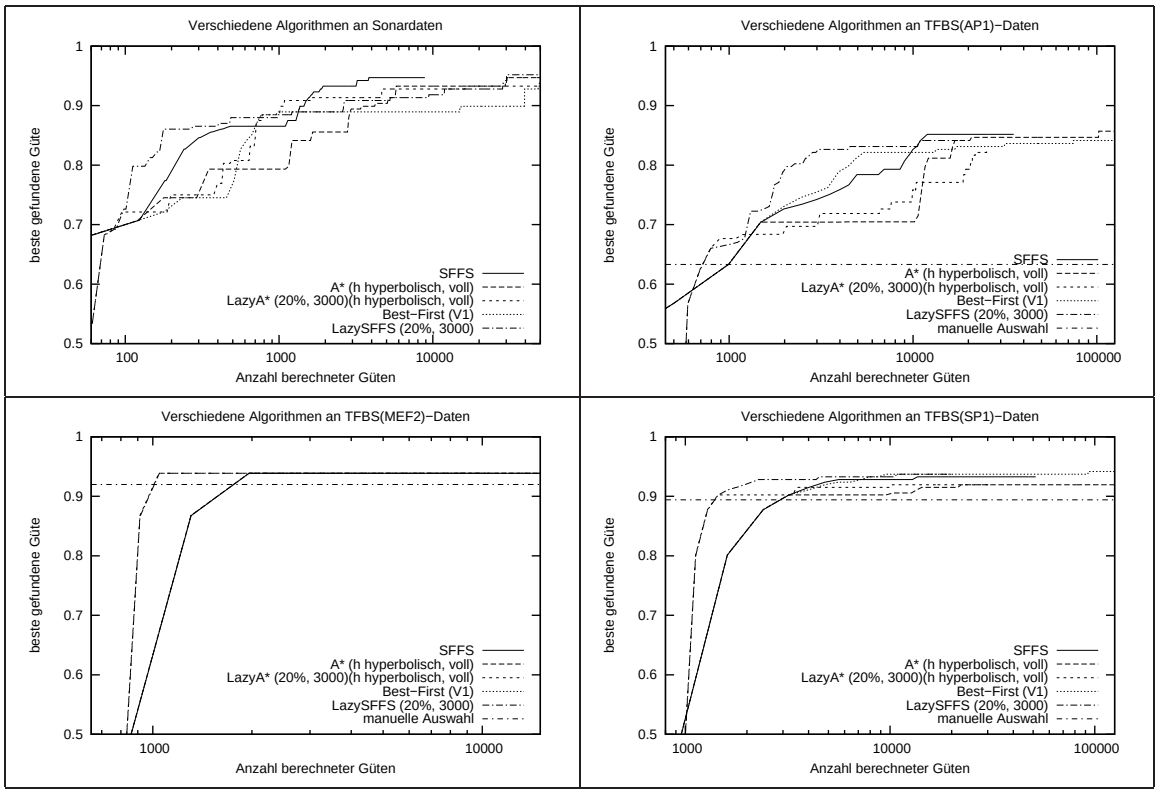

Abbildung 5.7: Vergleich der Effizienztestergebnisse der verschiedenen untersuchten Algorithmen bei gegebener Anzahl an Güteberechnungen an Sonardaten (oben links) und TFBS-Daten (o. r.: AP1; u. l.: MEF2; u. r.: SP1)

Es ist außerdem zu erkennen, dass der LazySFFS-Algorithmus für kleine Güteberechnungsanzahlen immer am effizientesten ist. Langfristig wird er aber meistens vom normalen SFFS-Algorithmus geschlagen.

Man kann auch erkennen, dass der A^*-Algorithmus für die TFBS-Daten für den Faktor AP1 und der Best-First-Algorithmus in Variante 1 für den Faktor SP1 das jeweils beste Ergebnis lieferen. Dies lässt sich aber, wie an den anderen Ergebnissen zu erkennen ist, nicht verallgemeinern.

Kapitel 6

Zusammenfassung und Ausblick

In diesem Kapitel soll die vorliegende Arbeit noch einmal kurz zusammengefasst werden und die Ergebnisse mit den zuvor erreichten verglichen werden. Anschließend soll ein Ausblick auf weitere mögliche Verbesserungsansätze gegeben werden.

6.1 Zusammenfassung

Das der Arbeit zugrunde liegende Problem bestand darin zu prüfen ob, und falls ja, mit welchen Ergebnissen es möglich ist, Merkmalauswahlverfahren (FSS) bei der Lokalisierung der Bindungsstellen von Transkriptionsfaktoren (TFBS) einzusetzen. Bisher erfolgte die Auswahl der Merkmale für dieses in der Genetik sehr interessante Problem rein manuell.

Dazu wurde zuerst eine Literaturrecherche durchgeführt um zu prüfen welche Merkmalauswahlverfahren bereits existierten und welche sich für dieses Problem eigneten. Danach wurden noch einige eigene Verfahren vorgeschlagen und zusammen mit einigen bekannten an den TFBS-Daten sowie, zur Vergleichbarkeit mit anderen Arbeiten, an Sonardaten getestet.

Im Unterschied zu den meisten bisherigen Arbeiten erfolgten die Tests aber nicht mit dem Ziel einer möglichst hohen Güte bei gegebener Merkmalzahl. Im Mittelpunkt stand hier die Effizienz, also eine möglichst hohe Güte bei gegebener *Berechnungs*zahl.

In Tabelle 6.1 sind die besten in dieser Arbeit gefundenen Ergebnisse und die vor der Arbeit vorliegenden Ergebnisse zusammengestellt. Dabei ist auch mit vermerkt, wie viele Merkmale die gefundene Merkmalmenge umfasst, durch welchen Algorithmus das jeweilige Ergebnis gefunden werden konnte und wie viele Güteberechnungen dafür nötig waren. Die besten gefundenen Merkmalmengen sind im Anhang B noch mal genauer aufgeführt.

Man kann erkennen, dass bei den TFBS-Daten für alle drei Transkriptionsfaktoren teils *deutliche Verbesserungen* gegenüber der manuellen Merkmalauswahl erzielt werden konnten. Beim Transkriptionsfaktor AP1 konnte so z.B. die Fehlerrate von 0.366972 auf 0.142857 gesenkt werden. Dies ist eine Verringerung von mehr als 60%. Für den Transkriptionsfaktor SP1 konnte immerhin noch eine Verringerung der Fehlerrate von 45% erreicht werden (von 0.105991 auf 0.058252). Diese Verringerung der Fehlerrate ging dabei in beiden Fällen zusätzlich mit einer Verringerung der benötigten Merkmalzahl einher.

Besonders auffällig ist jedoch der Transkriptionsfaktor MEF2. Bei diesem konnte die Fehlerrate zwar nur von 0.08 auf 0.061224, also um 23% gesenkt werden, dafür konnte

Daten	AP1	MEF2	SP1	Sonar
beste Güte	0.857143	0.938776	0.941748	0.966346
Merkmalzahl	10	3	13	15
Algorithmus	A^*(h hyperb.)	LazySFFS[1]	Best-First(V1)	LazyGSFFS(2,1)[2]
Berechnungen	102512	1047	93124	3361
Güte b. man. Aw.[3]	0.633028	0.92	0.894009	—
Merkmalzahl	15	14	15	—

Tabelle 6.1: Übersicht der besten Ergebnisse

dieses Ergebnis mit nur drei Merkmalen erzielt werden. Dies ist einer Verringerung der benötigten Merkmalzahl um knapp 80%.

Es war jedoch nicht möglich *einen* Algorithmus zu finden, welcher für *alle* Beispiele die jeweils besten Resultate liefern konnte.

In Tabelle 6.2 ist für einige Algorithmen zusammengefasst, wie lange sie brauchten um ein Ergebnis zu erreichen, welches 98% des besten gefundenen Ergebnisses entspricht. Dabei sind die Berechnungszahlen der drei jeweils schnellsten Algorithmen kursiv dargestellt, die Striche bedeuten, dass der jeweilige Algorithmus im untersuchten Bereich kein solches Ergebnis erzielen konnte. Man kann erkennen, dass der LazySFFS-Algorithmus als einziger in allen untersuchten Fällen jeweils zu den drei schnellsten gehört.

Daten (Referenzgüte)	AP1 (0.840)	MEF2 (0.920)	SP1 (0.923)	Sonar (0.947)
SFS	—	1.956	*4.797*	—
SFFS	*10.973*	1.960	*4.813*	*3.839*
LazySFFS[1]	*10.906*	*1.047*	*2.106*	*30.129*
A^* (h hyperbolisch)	*16.914*	1.957	—	30.353
$LazyA^*$ (h hyperbolisch)[1]	—	*1.044*	—	—
BestFirst (V1)	74.576	*1.957*	5.594	—
LazyGSFFS (2,1)[2]	—	9.158	26.386	*3.310*

Tabelle 6.2: Zeitpunk des Erreichens von 98% der besten gefundenen Güte

Die Ergebnisse lassen sich wie folgt zusammenfassen:

- Falls möglichst schnell eine relativ gute Merkmalmenge gefunden werden soll, ist sicher der LazySFFS-Algorithmus die erste Wahl.
- Falls sehr viel Rechenzeit zur Verfügung steht ist zu erwägen, mehrere Algorithmen wie A^* (h hyperbolisch) oder Best-First (Version 1) zu testen und das beste Ergebnis auszuwählen.

Einen guten Kompromiss zwischen Geschwindigkeit und bester erreichbarer Güte stellt der bekannte SFFS-Algorithmus dar.

[1] Berechnungsanteil von 20% und Lernabbruch nach 3000 Berechnungen

[2] Berechnungsanteil von 2% und Lernabbruch nach 3000 Berechnungen

[3] Die Güten nach manueller Auswahl sind bei den TFBS-Daten die Güten von Merkmalmengen welche von Herrn Pudimat ohne Merkmalauswahlverfahren zusammengestellt wurden.

6.2 Ausblick

Ein möglicher Ansatz für zukünftige Forschungsaktivitäten stellt sicherlich der verwendete Schätzer dar. Eine Verbesserung hin zum theoretischen optimalen Schätzer würde auch eine Verbesserung der prädiktiven A^*-Variante hin zum optimalen Algorithmus bewirken.

Dazu könnte man z.B. versuchen die speziellen Eigenheiten der Merkmale der TFBS-Daten, welche auf die Struktur der DNA-Sequenzen zurückgehen, zu nutzen um den Schätzer zu verbessern. Dies würde unter Umständen die Möglichkeit bieten die Schätzgenauigkeit zu verbessern und/oder die Anzahl der möglichen Lernschritte und damit die Anpassung an den Datensatz zu erhöhen.

Literaturverzeichnis

[Bac77] Backer, E.; de Schipper, J. A.: *On the max-min approach for feature ordering and selection*, in *The Seminar on Pattern Recognition, Liege University, Sart-Tilman, Belgium*, 1977, S. 2.4.1–2.4.7.

[BD01] Ben-Dor, A.; Friedman, N.; Yakhini, Z.: *Class discovery in gene expression data*, *RECOMB*, 2001, S. 31–38.

[Bec02] Beckstein, C.: *Script und Übungslösungen zur Vorlesung: Einführung in die KI*, FSU Jena, 2002.

[Bla98] Blake, C.; Merz, C.: *UCI Repository of machine learning databases*, University of California, Irvine, Dept. of Information and Computer Sciences, 1998.

[Bro03] Brockhausredaktion, (Hrsg.): *Der Brockhaus multimedial 2003 Premium*, Bibiographisches Institut & F. A. Brockhaus AG, Mannheim, 2003.

[Che03] Chen, X.: *An improved branch and bound algorithm for feature selection*, *Pattern Recogn. Lett.*, Bd. 24, Nr. 12, 2003, S. 1925–1933.

[Cov77] Cover, T. M.; van Campenhout, J. M.: *On the possible orderings in the measurement selection problem*, *IEEE Transactions on Systems, Man and Cybernetics*, Bd. 7, 1977, S. 657–661.

[Cun94] Cung, V.-D.; Le Cun, B.: *An Efficient Implementation of Parallel A**, in *Canada-France Conference on Parallel and Distributed Computing*, 1994, S. 153–168.

[Due89] Dueck, G.: *The Great Deluge Algorithm and the Record-to-Record Travel*, TR 89.06.011, IBM Heidelberg Scientific Center, 1989.

[Due93] Dueck, G.: *New optimization heuristics: The great deluge algorithm and the record-to-record-travel*, *Journal of Computational Physics*, Bd. 104, Nr. 1, 1993, S. 86–92.

[Fer94] Ferri, F.; Pudil, P.; Hatef, M.; Kittler, J.: *Comparative Study of Techniques for Large-Scale Feature Selection*, *Pattern Recognition in Practice IV*, 1994, S. 403–413.

[Jai96] Jain, A.; Zongker, D.: *Algorithms for Feature Selection: An Evaluation*, in *13th International Conference on Pattern Recognition*, 1996, S. 18–22.

[Kir83] Kirkpatrick, S.; Gelatt, C. D.; Vecchi, M. P.: *Optimization by Simulated Annealing*, *Science, Number 4598, 13 May 1983*, Bd. 220, 4598, 1983, S. 671–680.

[Kit78] Kittler, J.: *Feature set search algorithms*, *Pattern Recognition and Signal Processing*, 1978, S. 41–60.

[Kud00a] Kudo, M.; Sklansky, J.: *Comparison of Algorithms that Select Features for Pattern Classifiers*, *Pattern Recognition*, Bd. 33, Nr. 1, 2000, S. 25–41.

[Kud00b] Kudo, M.; Somol, P.; Pudil, P.; Shimbo, M.; Sklansky, J.: *Comparison of Classifier-Specific Feature Selection Algorithms*, *Lecture Notes in Computer Science*, Bd. 1876, 2000, S. 677–686.

[Kun99] Kuncheva, L. I.; Jain, L. C.: *Nearest neighbor classifier: simultaneous editing and feature selection*, *Pattern Recognition Letters*, Bd. 20, Nr. 11-13, 1999, S. 1149–1156.

[Nar77] Narendra, P. M.; Fukunaga, K.: *A branch and bound algorithm for feature subset selection*, *IEEE Transactions on Computers*, Bd. 26, Nr. 9, 1977, S. 917–922.

[Nil80] Nilsson, N. J.: *Principles of Artificial Intelligence*, Tioga, Palo Alto, Calif, 1980.

[Pud93] Pudil, P.; Novovičová, J.; Choakjarernwanit, N.; Kittler, J.: *An analysis of the max-min approach to feature selection and ordering*, *Pattern Recognition Letters*, Bd. 14, Nr. 11, 1993, S. 841–847.

[Pud94a] Pudil, P.; Ferri, F. J.; Novovičová, J.; Kittler, J.: *Floating Search Methods for Feature Selection with Nonmonotonic Criterion Functions*, in *Proceedings of the 12th ICPR*, Bd. 2, 1994, S. 279–283.

[Pud94b] Pudil, P.; Novovičová, J.; Kittler, J.: *Floating Search Methods in Feature Selection*, *Pattern Recognition Letters*, Bd. 15, Nr. 11, 1994, S. 1119–1125.

[Pud04] Pudimat, R.; Schukat-Talamazzini, E. G.; Backofen, R.: *Feature Based Representation and Detection of Transcription Factor Binding Sites*, in *German Conference on Bioinformatics*, 2004, S. 43–52.

[Rus03] Russell, S.; Norvig, P.: *Artificial Intelligence: A Modern Approach*, Prentice-Hall, Englewood Cliffs, NJ, 2. Ausg., 2003.

[Ser01] Serpico, S.; Bruzzone, L.: *A new search algorithm for feature selection in hyperspectral remote sensing images*, *GeoRS*, Bd. 39, Nr. 7, July 2001, S. 1360–1367.

[Sie89] Siedlecki, W.; Sklansky, J.: *A note on genetic algorithms for large-scale feature selection*, *Pattern Recognition Letters*, Bd. 10, Nr. 5, 1989, S. 335–347.

[Som99] Somol, P.; Pudil, P.; Novovičová, J.; Paclík, P.: *Adaptive floating search methods in feature selection*, *Pattern Recognition Letters*, Bd. 20, 1999, S. 1157–1163.

[Som00] Somol, P.; Pudil, P.; Ferri, F. J.; Kittler, J.: *Fast branch & bound algorithm in feature selection*, in *Proc. SCI 2000 Conf., Orlando, FL*, Bd. IIV, 2000, S. 646–651.

[Ste76] Stearns, S. D.: *On selecting features for pattern classifiers*, in *Third International Conference on Pattern Recognition*, 1976, S. 71–75.

[Tah04] Tahir, M. A.; Bouridane, A.; Kurugollu, F.; Amira, A.: *Feature Selection using Tabu Search for Improving the Classification Rate of Prostate Needle Biopsies.*, in *ICPR (2)*, 2004, S. 335–338.

[Whi71] Whitney, A. W.: *A direct method of nonparametric measurement selection*, *IEEE Transactions on Computers*, Bd. 20, Nr. 9, 1971, S. 1100–1103.

[Yu93] Yu, B.; Yuan, B.: *A more efficient branch and bound algorithm for feature selection*, *Pattern Recognition*, Bd. 26, Nr. 6, 1993, S. 883–889.

Abbildungsverzeichnis

Tabellenverzeichnis

Anhang A

Vorhersage der Güte

In diesem Kapitel soll die in der vorliegenden Arbeit verwendete Vorhersage der Güten von Merkmalmengen näher beschrieben werden. Das zugrunde liegende System wurde mir freundlicherweise inklusive dieser Beschreibung von Herrn Prof. Dr. E.G. Schukat-Talamazzini zur Verfügung gestellt.

A.1 Funktionale über Potenzmengen

A.1.1 Aufgabenstellung

Gegeben ist eine endliche Indexmenge $\mathcal{I} = \{1, 2, \ldots, n\}$ der Kardinalität $n \in \mathbb{N}$ und ihre Potenzmenge

$$2^{\mathcal{I}} \stackrel{\text{def}}{=} \mathfrak{P}(\mathcal{I}_n) \;=\; \{A \mid A \subseteq \mathcal{I}\}$$

und es sei

$$f:\; 2^{\mathcal{I}} \;\rightarrow\; \mathbb{R}$$

ein Funktional darauf. Der Funktionsverlauf ist nicht bekannt, aber es steht eine Probe

$$\omega \;=\; \{(A_t, z_t) \mid t = 1, \ldots, T\}\;, \quad A_t \in 2^{\mathcal{I}}\;, \quad z_t = f(A_t)$$

von Beispielen zur Verfügung. Gesucht ist eine geeignete Näherungsfunktion

$$\hat{f}:\; 2^{\mathcal{I}} \;\rightarrow\; \mathbb{R}\;,$$

die aus einem Raum effizient berechenbarer Potenzmengenfunktionen stammt und mit dem Original $f(\cdot)$ auf der Probe ω möglichst gut übereinstimmt.

A.1.2 Lineare kombinatorische Familien

Wir gehen davon aus, daß jeder Funktionswert $f(A)$ davon abhängt, welche Indizes $i \in \mathcal{I}$ das Argument A enthält, welche Indexpaare $(i, j) \in \mathcal{I}^2$ enthalten sind, welche Tripel $(i, j, k) \in \mathcal{I}^3$ und so weiter. Wir bezeichnen das Teilmengensystem

$$\mathfrak{B} \;\subseteq\; 2^{\mathcal{I}}$$

in diesem Sinne relevanter Indexkombinationen als die Mengenbasis unseres Modells. Wir entscheiden uns für einen linearen Ansatz, bei dem $f(A)$ eine Summendarstellung besitzt mit je

einem Summanden w_S für jede Basismenge $S \in \mathfrak{B}$ mit der Inklusionseigenschaft $S \subseteq A$:

$$f(A|\boldsymbol{w}) \;=\; \sum_{S\in\mathfrak{B}} \delta_{S\subseteq A} \cdot w_S$$

Dabei bezeichne δ die charakteristische Funktion und $\boldsymbol{w}$ sei eine Kurzschreibweise für das reellwertige Parameterfeld

$$\boldsymbol{w} \;=\; (w_S \mid S \in \mathfrak{B}) \;\in\; \mathbb{R}^{\mathfrak{B}}$$

der $|\mathfrak{B}|$ Koeffizienten für die gewählte Mengenbasis. Wir bezeichnen die resultierende Funktionenfamilie mit

$$\mathfrak{F} \;=\; \mathfrak{F}_{\mathfrak{B}} \stackrel{\text{def}}{=} \left\{ f(\cdot|\boldsymbol{w}) \mid \boldsymbol{w} \in \mathbb{R}^{\mathfrak{B}} \right\} .$$

Wir können die Berechnungsvorschrift dieser Funktionen etwas eleganter als Skalarprodukt

$$f(A|\boldsymbol{w}) \;=\; \sum_{S\in\mathfrak{B}} \phi_S(A) \cdot w_S \;=\; \langle \boldsymbol{\phi}(A), \boldsymbol{w} \rangle \;=\; \boldsymbol{w}^\top \boldsymbol{\phi}(A)$$

schreiben, wenn wir für jede Menge $A \subseteq \mathcal{I}$ den Indikatorvektor

$$\boldsymbol{\phi}(A) \in \{0,1\}^{\mathfrak{B}} , \quad \phi_S(A) \stackrel{\text{def}}{=} \begin{cases} 1 & S \subseteq A \\ 0 & S \not\subseteq A \end{cases}$$

definieren; diese Definition hängt natürlich von der Wahl der Mengenbasis $\mathfrak{B}$ ab.

A.1.3 Quadratmittelapproximation

Für gegebene Parametrisierung $\boldsymbol{w} \in \mathbb{R}^{\mathfrak{B}}$ ergibt sich für die Stützstelle $(A_t, z_t) \in 2^{\mathcal{I}} \times \mathbb{R}$ eine Abweichung

$$d_t(\boldsymbol{w}) \stackrel{\text{def}}{=} z_t \;-\; f(A_t|\boldsymbol{w}) \;=\; z_t \;-\; \langle \boldsymbol{w}, \boldsymbol{\phi}(A_t) \rangle ,$$

die wir im quadratischen Mittel minimieren wollen:

$$\varepsilon(\omega, \boldsymbol{w}) \;=\; \sum_{t=1}^{T} d_t^2(\boldsymbol{w}) \;=\; \sum_{t=1}^{T} (z_t \;-\; \langle \boldsymbol{w}, \boldsymbol{\phi}(A_t) \rangle)^2$$

Mit der kompakteren Vektor- und Matrixschreibweise

$$\boldsymbol{z} \;=\; \begin{pmatrix} z_1 \\ z_2 \\ \vdots \\ z_T \end{pmatrix} , \qquad \boldsymbol{X} \;=\; \begin{pmatrix} \boldsymbol{x}_1^\top \\ \boldsymbol{x}_2^\top \\ \vdots \\ \boldsymbol{x}_T^\top \end{pmatrix} , \qquad \boldsymbol{x}_t \;=\; \boldsymbol{\phi}(A_t)$$

für Sollwerte und Indikatorvektoren gewinnen wir die vertraute Form

$$\varepsilon(\omega, \boldsymbol{w}) \;=\; \|\boldsymbol{z} - \boldsymbol{X}\boldsymbol{w}\|^2$$

für die Quadratmittelzielgröße.

A.1.4 Gaußsche Normallösung

Wir berechnen für die Zielgröße den Vektor

$$\nabla_{\boldsymbol{w}}\varepsilon(\omega,\boldsymbol{w}) \;=\; \mathbf{0} - 2\boldsymbol{X}^\top\boldsymbol{z} + 2\boldsymbol{X}^\top\boldsymbol{X}\boldsymbol{w}$$

aller partiellen Ableitungen nach den Funktionsparametern; nach Nullsetzen ergibt sich das System der Gaußschen Normalengleichungen:

$$\boldsymbol{X}^\top\boldsymbol{X}\boldsymbol{w} \;=\; \boldsymbol{X}^\top\boldsymbol{z}$$

Die Koeffizientenmatrix $\boldsymbol{R} = \boldsymbol{X}^\top\boldsymbol{X}$ des Gleichungssystems enthält bis auf den Faktor T alle gemischten zweiten Momente der Indikatorvektoren $\boldsymbol{x}_1,\ldots,\boldsymbol{x}_T$. Ist $\boldsymbol{R}$ invertierbar, so lautet die eindeutige Lösung der Minimierungsaufgabe

$$\hat{\boldsymbol{w}} \;=\; (\boldsymbol{X}^\top\boldsymbol{X})^{-1}\boldsymbol{X}\boldsymbol{z} \;=\; \boldsymbol{R}^{-1}\boldsymbol{m}$$

mit der Abkürzung $\boldsymbol{m} = \boldsymbol{X}\boldsymbol{z}$. Ist die symmetrische Matrix $\boldsymbol{R}$ lediglich positiv-semidefinit, so kann das Gleichungssystem nach Gratregulierung

$$\boldsymbol{R}(\sigma^2) \stackrel{\text{def}}{=} \boldsymbol{R} + \sigma^2\boldsymbol{E}\;, \qquad \sigma^2 > 0$$

eindeutig gelöst werden.

A.1.5 Pseudonormallösung

Wenn die Koeffizientenmatrix $\boldsymbol{R}$ singulär ist, besitzt das Gleichungssystem einen Unterraum gleichwertiger Lösungen $\boldsymbol{w}$ der Dimension $b - r$ (Rangdefizit), wobei b die Dimension der Datenvektoren $\boldsymbol{x}_t$ bezeichnet — hier $b = |\mathfrak{B}|$ — und r den Rang der Matrix $\boldsymbol{X}$ bzw. $\boldsymbol{R}$.

Diejenige Lösung $\boldsymbol{w}$ mit der kleinsten Norm $\|\boldsymbol{w}\|^2 = \boldsymbol{w}^\top\boldsymbol{w}$ heißt Pseudonormallösung und besitzt die Darstellung

$$\hat{\boldsymbol{w}} \;=\; \boldsymbol{X}^+\boldsymbol{z}\;.$$

Es bezeichne dabei $\boldsymbol{X}^+$ die Pseudoinverse von $\boldsymbol{X}$, die sich aus der Singulärzerlegung $\boldsymbol{X} = \boldsymbol{V}\boldsymbol{D}\boldsymbol{U}^\top$ gemäß

$$\boldsymbol{X}^+ \stackrel{\text{def}}{=} \boldsymbol{U}\boldsymbol{D}^+\boldsymbol{V}^\top\;, \qquad d_i^+ \stackrel{\text{def}}{=} \begin{cases} 1/d_i & d_i \neq 0 \\ 0 & d_i = 0 \end{cases}\;, \qquad i = 1,\ldots,\min(b,T)$$

ergibt. Die Singulärwerte d_i und die Rechtssingulärvektoren $\boldsymbol{u}_i$ erhalten wir aus der Eigenzerlegung der Gleichungskoeffizienten, denn es gilt:

$$\boldsymbol{R} \;=\; \boldsymbol{X}^\top\boldsymbol{X} \;=\; \boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^\top\cdot\boldsymbol{V}\boldsymbol{D}\boldsymbol{U}^\top \;=\; \boldsymbol{U}\boldsymbol{D}^2\boldsymbol{U}^\top$$

Wegen $\boldsymbol{X}^\top = \boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^\top$ und daher $\boldsymbol{U}^\top\boldsymbol{X}^\top = \boldsymbol{D}\boldsymbol{V}^\top$ und auch $\boldsymbol{D}^{+2}\boldsymbol{U}^\top\boldsymbol{X}^\top = \boldsymbol{D}^+\boldsymbol{V}^\top$ besitzt die Pseudoinverse die Darstellung[1]

$$\boldsymbol{X}^+ \;=\; \boldsymbol{U}\boldsymbol{D}^{+2}\boldsymbol{U}^\top\boldsymbol{X}^\top \;=\; \boldsymbol{R}^+\boldsymbol{X}^\top$$

und die Pseudonormallösung die Form

$$\hat{\boldsymbol{w}} \;=\; \boldsymbol{R}^+\boldsymbol{X}^\top\boldsymbol{z} \;=\; \boldsymbol{R}^+\boldsymbol{m}\;.$$

[1] Wir schreiben $\boldsymbol{A}^{+2}$ für $(\boldsymbol{A}^+)^2$.

Der quadratische Vorhersagefehler für die Daten bei Verwendung der (Pseudo-)Normallösung beträgt

$$\varepsilon(\omega,\hat{\boldsymbol{w}}) \;=\; \|\boldsymbol{z}\|^2 - \boldsymbol{m}^\top \boldsymbol{R}^{+}\boldsymbol{m} \qquad \text{bzw.} \qquad \varepsilon(\omega,\hat{\boldsymbol{w}}) \;=\; \|\boldsymbol{z}\|^2 - \boldsymbol{m}^\top \boldsymbol{R}^{-1}\boldsymbol{m}$$

und besitzt auch die Alternativdarstellung

$$\varepsilon(\omega,\hat{\boldsymbol{w}}) \;=\; \|\boldsymbol{z}\|^2 - \left\|\boldsymbol{V}^\top \boldsymbol{z}\right\|^2$$

unter Verwendung der Linkssingulärvektoren $\boldsymbol{v}_i$ von $\boldsymbol{R}$.

A.2 Dualisierte Aufgabenstellung

Obiges Quadratmittelverfahren ist leider nicht praktikabel, denn die zu invertierende Momentenmatrix $\boldsymbol{R} = \boldsymbol{X}^\top \boldsymbol{X}$ besitzt zwar höchstens den Rang T, aber gleichzeitig die Dimension $(b \times b)$, wobei $b = |\mathfrak{B}|$ im ungünstigsten Fall gleich dem Umfang 2^n der Potenzmenge $2^{\mathcal{I}}$ ist.

Auch die Funktionsauswertung $f(A|\boldsymbol{w})$ selbst ist impraktikabel, denn weder möchten wir gerne explizit den megalangen Indikatorenvektor $\boldsymbol{\phi}(A) \in \{0,1\}^b$ berechnen noch das entsetzliche Skalarprodukt $\boldsymbol{w}^\top \boldsymbol{\phi}(A)$ im Raum $\mathbb{R}^b$.

A.2.1 Rangdefizit und Lösungsraum

In dieser aussichtslosen Lage kommt uns zu Hilfe, daß jede Normallösung, aber auch jede Pseudonormallösung der Aufgabenstellung[2] notwendigerweise in einem T-Dimensionalen Unterraum des $\mathbb{R}^b$ angesiedelt ist. Die Lösung hat nämlich (s.o.) die jeweilige Form

$$\hat{\boldsymbol{w}} \;=\; \left\{ \begin{array}{ll} \boldsymbol{R}^{-1}\boldsymbol{m} & = \boldsymbol{U}\boldsymbol{D}^{-2}\boldsymbol{U}^\top \boldsymbol{X}^\top \boldsymbol{z} \\ \boldsymbol{R}^{+}\boldsymbol{m} & = \boldsymbol{U}\boldsymbol{D}^{+2}\boldsymbol{U}^\top \boldsymbol{X}^\top \boldsymbol{z} \end{array} \right. ,$$

beginnt also mit dem Produkt $\boldsymbol{U}\boldsymbol{D}^{-2}$ bzw. $\boldsymbol{U}\boldsymbol{D}^{+2}$. Diesen Lösungspräfix können wir gemäß

$$\boldsymbol{U}\boldsymbol{D}^{-2} \;=\; \boldsymbol{U}\boldsymbol{D}\boldsymbol{D}^{-3} \;=\; \boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^\top \boldsymbol{V}\boldsymbol{D}^{-3} \;=\; \boldsymbol{X}^\top \boldsymbol{V}\boldsymbol{D}^{-3}$$

umformen; für den unteren Fall verwenden wir $\boldsymbol{D}\boldsymbol{D}^{+}\boldsymbol{D}^{+} = \boldsymbol{D}^{+}$ und verfahren dann analog. In beiden Fällen besitzt die Lösung eine Darstellung $\hat{\boldsymbol{w}} = \boldsymbol{X}^\top \boldsymbol{\beta}$ mit $\boldsymbol{\beta} \in \mathbb{R}^T$, also einer Linearkombination

$$\hat{\boldsymbol{w}} \;=\; \beta_1 \boldsymbol{x}_1 + \beta_2 \boldsymbol{x}_2 + \beta_3 \boldsymbol{x}_3 + \ldots + \beta_T \boldsymbol{x}_T$$

der Datenvektoren, mit anderen Worten: es gilt die Aussage

$$\hat{\boldsymbol{w}} \;\in\; \mathrm{span}\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T\} \; .$$

A.2.2 Gaußsche Normallösung

Nach der obigen Beobachtung rechtfertigt sich die modifizierte Form

$$\varepsilon(\omega,\boldsymbol{\beta}) \;=\; \left\|\boldsymbol{z} - \boldsymbol{X}\boldsymbol{X}^\top \boldsymbol{\beta}\right\|^2 \;=\; \|\boldsymbol{z} - \boldsymbol{G}\boldsymbol{\beta}\|^2$$

für die Quadratmittelzielgröße. Die symmetrische Matrix $\boldsymbol{G} = \boldsymbol{X}\boldsymbol{X}^\top$ heißt Gramsche Matrix, besitzt die Dimension $(T \times T)$ und besitzt die wechselseitigen Skalarprodukte

$$G_{st} \;=\; \boldsymbol{x}_s^\top \boldsymbol{x}_t$$

[2] Das gilt aber — bei singulärer Momentenmatrix $\boldsymbol{R}$ — nicht für beliebige Lösungen!

der Datenvektoren als Einträge. Der Gradientenvektor lautet nunmehr

$$\nabla_{\boldsymbol{\beta}}\varepsilon(\omega,\boldsymbol{\beta}) \;=\; \mathbf{0} - 2\boldsymbol{G}^\top\boldsymbol{z} + 2\boldsymbol{G}^\top\boldsymbol{G}\boldsymbol{\beta} \;=\; -2\boldsymbol{G}\boldsymbol{z} + 2\boldsymbol{G}^2\boldsymbol{\beta}$$

und nach Nullsetzen ergibt sich das Gleichungssystem

$$\boldsymbol{G}^2\boldsymbol{\beta} \;=\; \boldsymbol{G}\boldsymbol{z}\;.$$

Ist die Gramsche Matrix $\boldsymbol{G}$ invertierbar, so besitzt das Gleichungssystem die eindeutige Lösung

$$\hat{\boldsymbol{\beta}} \;=\; \boldsymbol{G}^{-1}\boldsymbol{z}$$

und die Minimierungsaufgabe wird von

$$\hat{\boldsymbol{w}} \;=\; \boldsymbol{X}^\top\hat{\boldsymbol{\beta}} \;=\; \boldsymbol{X}^\top\boldsymbol{G}^{-1}\boldsymbol{z}$$

gelöst.

A.2.3 Pseudonormallösung

Ist die Gramsche Matrix $\boldsymbol{G}$ hingegen singulär, so muß wieder eine normminimale Lösung ausgespäht werden. Für die dualen Koeffizienten erhalten wir das Ergebnis

$$\hat{\boldsymbol{\beta}} \;=\; \boldsymbol{G}^{+}\boldsymbol{z}$$

und für die primären Koeffizienten

$$\hat{\boldsymbol{w}} \;=\; \boldsymbol{X}^\top\boldsymbol{G}^{+}\boldsymbol{z}\;.$$

Unter Verwendung der Singulärwertzerlegung von $\boldsymbol{X}$ und der resultierenden Darstellungen $\boldsymbol{G} = \boldsymbol{V}\boldsymbol{D}^2\boldsymbol{V}^\top$ und $\boldsymbol{G}^{+} = \boldsymbol{V}\boldsymbol{D}^{+2}\boldsymbol{V}^\top$ überzeugt man sich leicht davon, daß tatsächlich

$$\boldsymbol{X}^\top\boldsymbol{G}^{+}\boldsymbol{z} \;=\; \boldsymbol{X}^{+}\boldsymbol{z}$$

gilt.

A.2.4 Skalarprodukte von Indikatorvektoren

Was ist denn nun hinsichtlich des Berechnungsaufwandes gewonnen? Schließlich sind ja die T^2 Einträge[3] der Gramschen Matrix zu bestimmen. Diese Skalarprodukte haben jedoch die relativ simple Gestalt

$$G_{st} \;=\; \langle\boldsymbol{x}_s,\boldsymbol{x}_t\rangle \;=\; \langle\boldsymbol{\phi}(A_s),\boldsymbol{\phi}(A_t)\rangle \;=\; \sum_{S\in\mathfrak{B}}\delta_{S\subseteq A_s}\cdot\delta_{S\subseteq A_t} \;=\; \sum_{S\in\mathfrak{B}}\delta_{S\subseteq A_s\cap A_t}\;,$$

da hier selbstverständlich nicht beliebige Vektoren $\boldsymbol{x}\in\mathbb{R}^b$ zu behandeln sind, sondern lediglich Indikatoren $\boldsymbol{x} = \boldsymbol{\phi}(A)$ von Indexmengen $A\subseteq\mathcal{I}$. Wir bezeichnen die zu Grunde liegende Berechnungsvorschrift

$$\mathsf{K}:\left\{\begin{array}{lcl} 2^{\mathcal{I}}\times 2^{\mathcal{I}} & \rightarrow & \mathbb{R} \\ (A,B) & \mapsto & \sum_{S\in\mathfrak{B}}\delta_{S\subseteq A\cap B}\end{array}\right.$$

als Kernfunktion zum Operator $\boldsymbol{\phi}(\cdot)$ und konstatieren den Zusammenhang

$$\langle\boldsymbol{\phi}(A),\boldsymbol{\phi}(B)\rangle \;=\; \mathsf{K}(A,B)\qquad(\forall A,B\subseteq\mathcal{I})\;.$$

[3] Wegen der Symmetrie sind es nur $T/2\cdot(T+1)$ Produkte.

Sowohl Kernoperator $\mathsf{K}(\cdot,\cdot)$ als auch Indikatoroperator $\boldsymbol{\phi}(\cdot)$ hängen natürlich implizit von der gewählten Mengenbasis $\mathfrak{B}$ ab.

Zur Berechnung von $\mathsf{K}(A,B)$ ist die Schnittmenge $A{\cap}B$ zu bilden und es sind die mit $\mathfrak{B}$ gemeinsamen Teilmengen auszuzählen:

$$\mathsf{K}(A,B) \;=\; |\mathfrak{B} \cap \mathfrak{P}(A{\cap}B)|$$

Für einige sehr regelmäßig strukturierte Mengensysteme ergeben sich noch einfachere Resultate:

$$\mathsf{K}(A,B) \;=\; \begin{cases} \nu & \mathfrak{B} = \big\{S \in 2^{\mathcal{I}} \mid |S| = 1\big\} \\ \binom{\nu}{k} & \mathfrak{B} = \big\{S \in 2^{\mathcal{I}} \mid |S| = k\big\} \\ \sum_{j=0}^{k} \binom{\nu}{j} & \mathfrak{B} = \big\{S \in 2^{\mathcal{I}} \mid |S| \leq k\big\} \\ 2^{\nu} & \mathfrak{B} = 2^{\mathcal{I}} \end{cases} \qquad \text{mit } \nu \stackrel{\text{def}}{=} |A{\cap}B|$$

A.2.5 Vorhersageaufwand

Mit Hilfe des Kernoperators läßt sich problemlos die Gramsche Matrix erstellen, und auch die Berechnung ihrer (Pseudo-)Inversen sollte keine Schwierigkeiten bereiten. Der Koeffizientenvektor $\boldsymbol{w} = \boldsymbol{X}^{\top}\boldsymbol{G}^{-1}\boldsymbol{z}$ selbst ist allerdings nicht mehr ökonomisch auszurechnen, denn dafür wären die $\mathbb{R}^b$-Vektoren aus den T Zeilen von $\boldsymbol{X}$ (gewichtet) aufzuaddieren.

Stattdessen wird in der Lernphase nur $\boldsymbol{\beta} = \boldsymbol{G}^{-1}\boldsymbol{z}$ ermittelt und „abgespeichert“. Zum Vorhersagezeitpunkt nutzen wir die Gleichungskette

$$f(A|\boldsymbol{w}) \;=\; \Big\langle \boldsymbol{\phi}(A), \boldsymbol{X}^{\top}\boldsymbol{\beta} \Big\rangle \;=\; \boldsymbol{\phi}(A)^{\top}\boldsymbol{X}^{\top}\boldsymbol{\beta} \;=\; \underbrace{(\boldsymbol{X}\boldsymbol{\phi}(A))}_{\boldsymbol{g}(A)}{}^{\top} \cdot \boldsymbol{\beta}$$

aus; der Vektor $\boldsymbol{g}(A) = \boldsymbol{X}\boldsymbol{\phi}(A) \in \mathbb{R}^T$ besteht offenbar aus den Komponenten

$$g_t(A) \stackrel{\text{def}}{=} \langle \boldsymbol{x}_t, \boldsymbol{\phi}(A)\rangle \;=\; \langle \boldsymbol{\phi}(A_t), \boldsymbol{\phi}(A)\rangle \;=\; \mathsf{K}(A_t, A)\;, \qquad t = 1,\ldots,T$$

und wird unter Verwendung des Kernoperators ausgewertet. Abschließend bleibt nur noch das $\mathbb{R}^T$-Skalarprodukt $\boldsymbol{\beta}^{\top}\boldsymbol{g}(A)$ zu berechnen.

Zur Vorhersage stehen uns also zwei Summenformeln zur Verfügung, einmal die soeben hergeleitete Dualform

$$f(A|\boldsymbol{w}) \;=\; \sum_{t=1}^{T} \beta_t \cdot \mathsf{K}(A_t, A)$$

sowie die originale Primärform

$$f(A|\boldsymbol{w}) \;=\; \sum_{A \supseteq S \in \mathfrak{B}} w_S\;,$$

die im Falle kleiner Indexmengen A unter Umständen effizienter auswertbar ist als die Dualform. Allerdings wollen auch die wenigen Primärkoeffizienten w_S, $S \subseteq A$ erst einmal berechnet sein:

$$w_S \;=\; (\boldsymbol{X}^{\top}\boldsymbol{\beta})_S \;=\; \sum_{t|A_t \supseteq S} \beta_t\;, \qquad S \in \mathfrak{B}$$

A.2.6 Dünne Vorhersage

Der Rechenaufwand für die Lernphase wie auch für die Vorhersagephase kann noch erheblich reduziert werden, wenn die gesuchte Lösung bereits aus einem Inventar von nur S Vektoren, $S < T$, linear kombiniert werden kann:

$$\hat{\boldsymbol{w}} \in \text{span}\{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_S\} \quad \text{bzw.} \quad \hat{\boldsymbol{w}} = \boldsymbol{Y}^\top \boldsymbol{\beta}$$

Der zu minimierende Fehler besitzt dann die Darstellung

$$\varepsilon(\omega, \boldsymbol{\beta}) = \left\| \boldsymbol{z} - \boldsymbol{X}\boldsymbol{Y}^\top \boldsymbol{\beta} \right\|^2 = \|\boldsymbol{z} - \boldsymbol{G}\boldsymbol{\beta}\|^2$$

mit der gemischten Gramschen Matrix $\boldsymbol{G} = \boldsymbol{X}\boldsymbol{Y}^\top \in \mathbb{R}^{(T \times S)}$, welche die Einträge $G_{ts} = \boldsymbol{x}_t^\top \boldsymbol{y}_s$ besitzt. Nach Ableiten und Nullsetzen ergeben sich die notwendigen Minimalitätsbedingungen in Form des linearen Gleichungssystems

$$\boldsymbol{G}^\top \boldsymbol{G}\boldsymbol{\beta} = \boldsymbol{G}^\top \boldsymbol{z}$$

und daraus die Pseudonormallösung

$$\hat{\boldsymbol{\beta}} = \boldsymbol{G}^+ \boldsymbol{z}\,, \qquad \hat{\boldsymbol{w}} = \boldsymbol{Y}^\top \hat{\boldsymbol{\beta}}$$

sowie die Vorhersageformel

$$f(A|\hat{\boldsymbol{w}}) = \hat{\boldsymbol{w}}^\top \boldsymbol{\phi}(A) = \hat{\boldsymbol{\beta}}^\top \underbrace{\boldsymbol{Y}\boldsymbol{\phi}(A)}_{\boldsymbol{g}(A)}$$

mit $g_s(A) = \langle \boldsymbol{\phi}(A), \boldsymbol{y}_s \rangle$. Ist $\boldsymbol{G} = \boldsymbol{V}\boldsymbol{D}\boldsymbol{U}^\top$ die Singulärwertzerlegung der gemischten Grammatrix, so läßt sich die Pseudoinverse von $\boldsymbol{G}$ wegen

$$\boldsymbol{G}^\top = \boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^\top\,, \qquad \boldsymbol{G}^\top \boldsymbol{G} = \boldsymbol{U}\boldsymbol{D}^2\boldsymbol{U}^\top$$

und folglich[4]

$$\boldsymbol{G}^+ = (\boldsymbol{G}^\top \boldsymbol{G})^+ \boldsymbol{G}^\top$$

mit Hilfe der Eigenzerlegung der positiv-semidefiniten $(S \times S)$-Matrix $\boldsymbol{G}^\top \boldsymbol{G}$ berechnen.

A.2.7 Inkrementierbarkeit

Neues Lerndatum

Steht eine neue Stützstelle (A_{T+1}, z_{T+1}) zur Verfügung, muß nur die vergrößerte Gramsche Matrix $\tilde{\boldsymbol{G}}$ durch Anlagern einer neuen Zeile/Spalte an $\boldsymbol{G}$ erzeugt werden. Für die Berechnung von $\tilde{\boldsymbol{G}}^{-1}$ mit Hilfe von $\boldsymbol{G}^{-1}$ gibt es eine hocheffiziente Rekursionsformel; für $\tilde{\boldsymbol{G}}^+$ gibt es diese Möglichkeit nicht.

Es sei $\tilde{\boldsymbol{H}}$ die inverse Matrix zu $\tilde{\boldsymbol{G}}$, d.h. es gilt $\tilde{\boldsymbol{G}}\tilde{\boldsymbol{H}} = \boldsymbol{E}$, und wir betrachten die Blockzerlegungen

$$\tilde{\boldsymbol{G}} = \begin{pmatrix} \boldsymbol{G}_{11} & \boldsymbol{G}_{12} \\ \boldsymbol{G}_{21} & \boldsymbol{G}_{22} \end{pmatrix}\,, \qquad \tilde{\boldsymbol{H}} = \begin{pmatrix} \boldsymbol{H}_{11} & \boldsymbol{H}_{12} \\ \boldsymbol{H}_{21} & \boldsymbol{H}_{22} \end{pmatrix}\,.$$

[4]Das gilt wegen $\boldsymbol{G}^+ = \boldsymbol{U}\boldsymbol{D}^+\boldsymbol{V}^\top = \boldsymbol{U}\boldsymbol{D}^+\boldsymbol{D}^+\boldsymbol{D}\boldsymbol{V}^\top = \boldsymbol{U}\boldsymbol{D}^{+2}\boldsymbol{U}^\top \cdot \boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^\top$.

Dann gilt für die Blöcke der Inversen:

$$\begin{aligned}
\boldsymbol{H}_{11}^{-1} &= \boldsymbol{G}_{11} - \boldsymbol{G}_{12}\boldsymbol{G}_{22}^{-1}\boldsymbol{G}_{21} \\
\boldsymbol{H}_{22}^{-1} &= \boldsymbol{G}_{22} - \boldsymbol{G}_{21}\boldsymbol{G}_{11}^{-1}\boldsymbol{G}_{12} \\
\boldsymbol{H}_{12} &= -\boldsymbol{G}_{11}^{-1}\boldsymbol{G}_{12}\boldsymbol{H}_{22} \\
\boldsymbol{H}_{21} &= -\boldsymbol{G}_{22}^{-1}\boldsymbol{G}_{21}\boldsymbol{H}_{11}
\end{aligned}$$

Für den Spezialfall $\boldsymbol{G}_{22} \in \mathbb{R}$, $\boldsymbol{H}_{22} \in \mathbb{R}$, skalarer Südostblöcke liegt die Gestalt

$$\tilde{\boldsymbol{G}} = \begin{pmatrix} \boldsymbol{G} & \boldsymbol{g} \\ \boldsymbol{g}^\top & \gamma \end{pmatrix}, \qquad \tilde{\boldsymbol{H}} = \begin{pmatrix} \boldsymbol{H} & \boldsymbol{h} \\ \boldsymbol{h}^\top & \eta \end{pmatrix}$$

vor und es gelten die Beziehungen

$$\begin{aligned}
\boldsymbol{H}^{-1} &= \boldsymbol{G} - \frac{1}{\gamma} \cdot \boldsymbol{g}\boldsymbol{g}^\top \\
\eta^{-1} &= \gamma - \boldsymbol{g}^\top \boldsymbol{G}^{-1}\boldsymbol{g} \\
\boldsymbol{h} &= -\eta \cdot \boldsymbol{G}^{-1}\boldsymbol{g}
\end{aligned}$$

Den Nordwestblock $\boldsymbol{H}$ wiederum berechnen wir nach dem Rang-1-Spezialfall

$$\boldsymbol{H} = \left(\boldsymbol{G} - {}^1\!/_\gamma \cdot \boldsymbol{g}\boldsymbol{g}^\top\right)^{-1} = \boldsymbol{G}^{-1} + \frac{1}{\gamma - \boldsymbol{g}^\top \boldsymbol{G}^{-1}\boldsymbol{g}} \cdot (\boldsymbol{G}^{-1}\boldsymbol{g})(\boldsymbol{G}^{-1}\boldsymbol{g})^\top$$

der Formel von *Sherman-Morrison-Woodbury*:

$$(\boldsymbol{A} + \boldsymbol{U}\boldsymbol{V}^\top)^{-1} = \boldsymbol{A}^{-1} - \boldsymbol{A}^{-1}\boldsymbol{U} \cdot (\boldsymbol{E} + \boldsymbol{V}^\top \boldsymbol{A}^{-1}\boldsymbol{U})^{-1} \cdot \boldsymbol{V}^\top \boldsymbol{A}^{-1}$$

Vorsicht ist im Falle $\gamma \approx \boldsymbol{g}^\top \boldsymbol{G}^{-1}\boldsymbol{g}$ geboten; ohne geeignete Gratregulierung $\gamma \mapsto \gamma + \sigma^2$ gerät die aufgebohrte Grammatrix $\tilde{\boldsymbol{G}}$ singulär und kann auch durch die Inkrementierungsformel nicht invertiert werden.

Insgesamt lassen sich alle Bausteine der gesuchten Graminversen $\tilde{\boldsymbol{H}} = \tilde{\boldsymbol{G}}^{-1}$ aus der alten Inversen $\boldsymbol{G}^{-1}$, dem Vektor $\boldsymbol{g}$ der Kernfunktionswerte $g_t = \mathsf{K}(A_t, A_{T+1})$ und dem Wert $\gamma = \mathsf{K}(A_{T+1}, A_{T+1})$ berechnen.

Neues Vorhersagedatum

Ist $f(A|\boldsymbol{w})$ bereits bekannt, so ist $f(A'|\boldsymbol{w})$ mit $A' = A \uplus \{i\}$ besonders effizient zu berechnen:

$$\Delta_i f(A|\boldsymbol{w}) \stackrel{\text{def}}{=} f(A'|\boldsymbol{w}) - f(A|\boldsymbol{w}) = \sum_{t=1}^{T} \beta_t \cdot \big\{\mathsf{K}(A_t, A') - \mathsf{K}(A_t, A)\big\}$$

Die inkrementelle Berechnung erfordert also nur das Auswerten der rechtsstehenden Summe. Alle Summanden mit $i \notin A_t$ verschwinden und können daher unberücksichtigt bleiben. Für die übrigen gilt

$$|A_t \cap A'| = |A_t \cap A| + 1$$

mit entsprechenden Konsequenzen für $\mathsf{K}(A_t, A')$.

Anhang B

Beste gefundene Merkmalmengen

F-Measure	0.8571428571428571
Algorithmus	A^* (h hyperbolisch)
Rechenschritte	102512
Merkmalzahl	10
Merkmale	*Nuc-1* *Nuc19* *C5* *bdnaMIN_GROOVE_DIST1_3* *bdnaTWIST34_6* *complexTILT4_5* *bdnaBEND7_9* *fdnaTWIST5_7* *bdnaMAJ_GROOVE_DEPTH8_10* *bdnaMIN_GROOVE_DEPTH4_6*

Tabelle B.1: Beste gefundene Merkmalmenge für die TFBS(AP1)-Daten

F-Measure	0.9387755102040816
Algorithmus	LazySFFS (20%)
Rechenschritte	1047
Merkmalzahl	3
Merkmale	*C3* *bdnaMOB_TO_BIND_TOWARD_MAJ_GROOVE6_7* *bdnaMIN_GROOVE_WIDTH6_8*

Tabelle B.2: Beste gefundene Merkmalmenge für die TFBS(MEF2)-Daten

F-Measure	0.941747572815534
Algorithmus	Best-First (V1)
Rechenschritte	93124
Merkmalzahl	13
Merkmale	*Nuc-15* *C3* *C4* *P2* *P8* *fdnaTILT2_4* *bdnaTWIST31_3* *fdnaROLL1_3* *bdnaPROPELLER2_3* *bdnaMELTING_TEMPERATURE8_9* *bdnaMIN_GROOVE_WIDTH1_2* *bdnaMIN_GROOVE_WIDTH5_6* *bdnaCLASH_STRENGTH10_12*

Tabelle B.3: Beste gefundene Merkmalmenge für die TFBS(SP1)-Daten

LOO-1NN-Erkennungsrate	0.966346
Algorithmus	LazyGSFFS (2, 1) (2%, 3000)
Rechenschritte	3361
Merkmalzahl	15
Merkmale[1]	6, 8, 9, 11, 12, 13, 21, 22, 24, 32, 35, 36, 40, 44, 47

Tabelle B.4: Beste gefundene Merkmalmenge für die Sonardaten

[1] nullbasierte Zählweise

Anhang C

Statistiken zu den untersuchten Daten

Bei den im Rahmen dieser Arbeit durchgeführten Testläufen wurde ein Cache-System eingesetzt, welches alle einmal berechneten Güten speicherte um später ggf. darauf zurückgreifen zu können. Nach Abschluss der Arbeit wurden diese Daten genutzt um die folgenden Statistiken zu erstellen. Dass heißt, für die Statistiken dieses Kapitel wurden nahezu alle im Verlauf der Arbeit berechneten Güten verwendet, also auch solche, die keinen direkten Eingang in die Arbeit fanden (z.B. wieder verworfene Algorithmen).

In der linken oberen Abbildung ist jeweils die beste überhaupt gefundene Güte in Abhängigkeit von der Merkmalzahl dargestellt. Die rechte obere Abbildung zeigt, wieviele Güten einer bestimmten Größe insgesammt berechnet wurden. Die linke untere Abbildung zeigt, wie große der Anteil der Merkmalmengen ist, die die gleiche Güte wie die als beste ausgewählte Merkmalmenge besitzen.

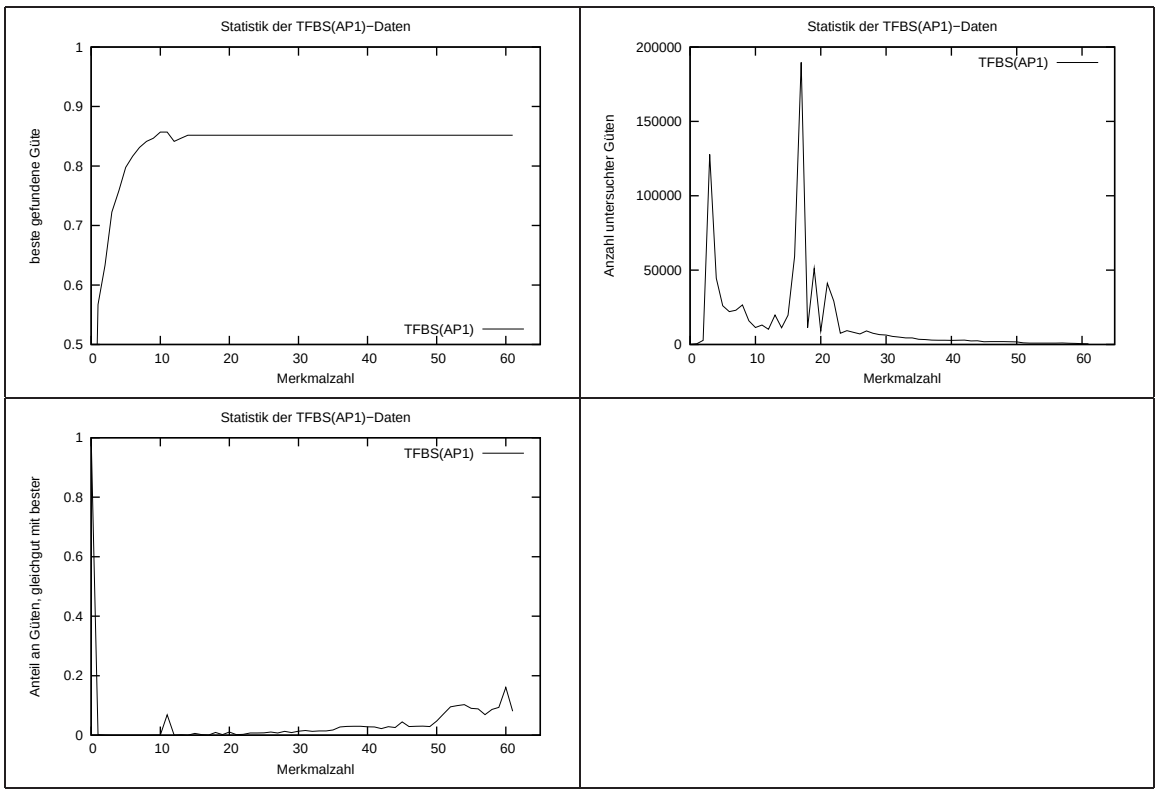

Abbildung C.1: Statistiken zu den TFBS(AP1)-Daten

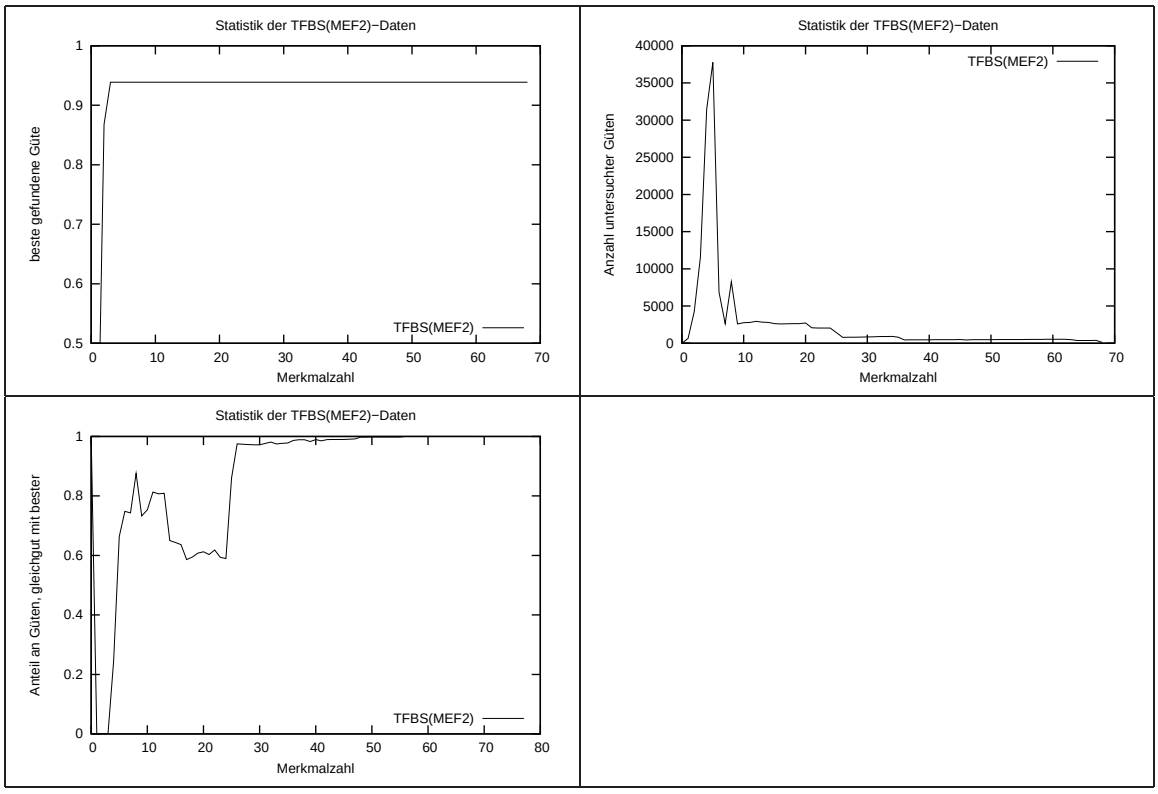

Abbildung C.2: Statistiken zu den TFBS(MEF2)-Daten

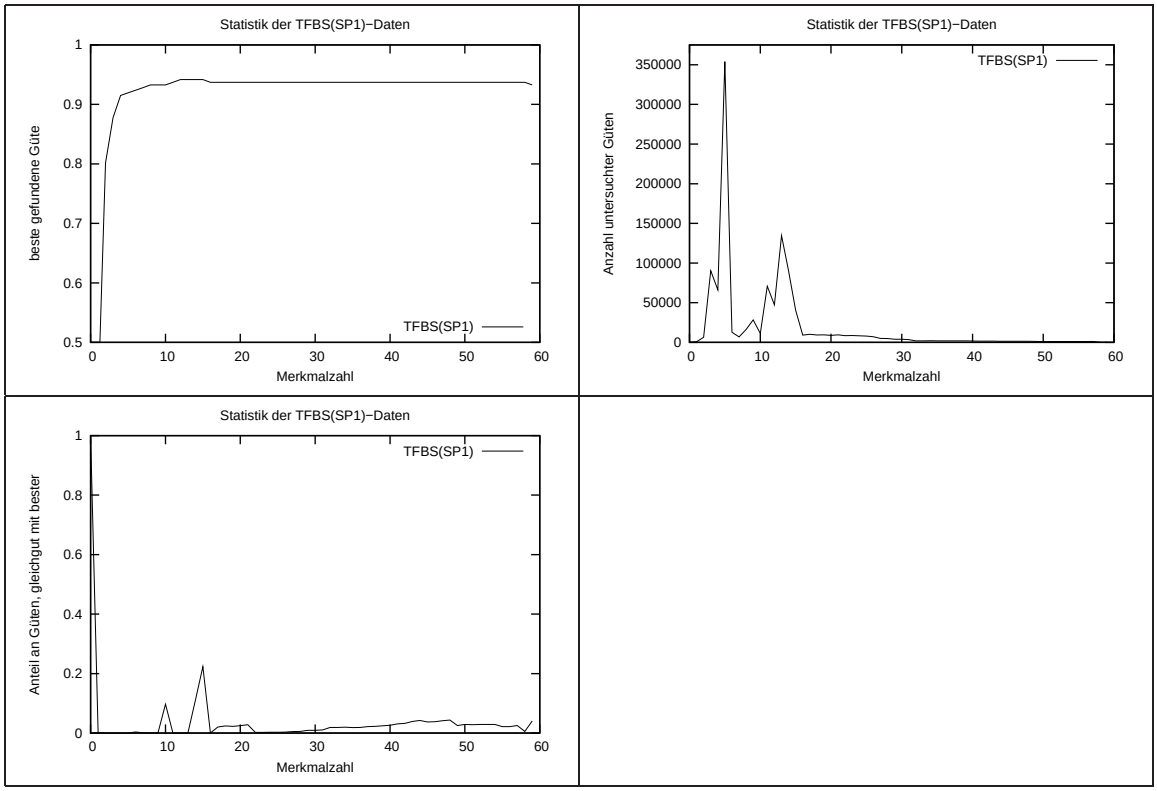

Abbildung C.3: Statistiken zu den TFBS(SP1)-Daten

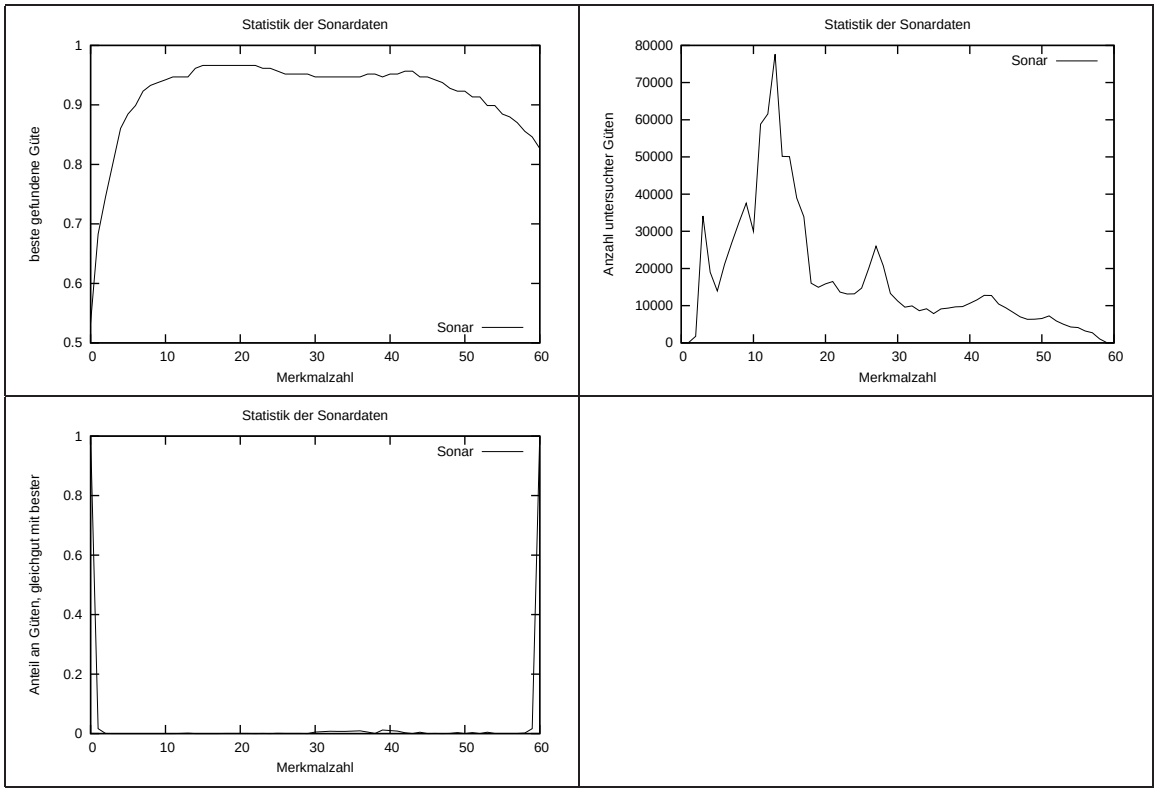

Abbildung C.4: Statistiken zu den Sonardaten

Selbständigkeitserklärung

Ich versichere, daß ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und daß die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Jena, den ....

(Sven Schütze)