Implementierung eines informationstheoretischen Ansatzes zur Bilderkennung

P. Bergmann +, J. Keller +, T. Malter +, S. M. Müller +, W. J. Paul +, T. Pöschel + +,
O. Schlüter + und L. Thiele + + +

+ : Fachbereich Informatik, Universität des Saarlandes
+ + : Sektion Physik, Humboldt-Universität zu Berlin
+ + + : Fachbereich Elektrotechnik, Universität des Saarlandes

Abstract:

In [BPT] wurde ein informationstheoretischer Ansatz für Systeme zur Bilderkennung vorgestellt. Es wird eine erste Implementierung dieses Ansatzes mit vernetzten SINIX-MX2-Rechnern beschrieben.

1. Informationstheorie und Verständnis von Bildern

Wir fassen den Ansatz aus [BPT] zusammen. Occam's Razor besagt, daß von zwei Theorien, welche die gleichen Phänomene beschrieben, diejenige vorzuziehen ist, welche die einfacheren Beschreibungen liefert.

Nach Kolmogorov und Solomonov [K], [S] ist Theoriebildung sogar nichts weiter als die Suche von kurzen Beschreibungen für große Mengen von beobachteten Daten. Beispielsweise beschreibt das einfache Newton'sche Gesetz "Kraft = Masse * Beschleunigung" aus der Theorie "Physik" die Meßergebnisse aus einer unglaublich großen Anzahl von Beobachtungen (Experimenten). Formal sind nach Kolmogorov und Solomonov Beschreibungen binär kodierte Programme in irgendeiner universellen Programmiersprache L, und die Kompliziertheit eines Programms ist seine Länge. Kolmogorov und Solomonov interessieren sich nur für das asymptotische Wachstum der Länge von unendlichen Folgen von Programmen. Deswegen ist für sie die spezielle Wahl der Sprache L nicht von Bedeutung (denn ist B' eine Beschreibung in Sprache L', so erhält man eine äquivalente Beschreibung in Sprache L, indem man einen in Sprache L geschriebenen Interpreter für L' auf B' anwendet; kürzeste Beschreibungen in Sprache L sind also höchstens um die Länge dieses

Interpreters länger als äquivalente kürzeste Beschreibungen in L'). Laufzeit und Speicherplatzbedarf von Programmen werden ignoriert.

In [BPT] werden Rechnersysteme vorgeschlagen, welche im obigen Sinne Theorien über die für sie beobachtbare Welt bilden. Geht es um das Verständnis unbewegter Schwarz/Weiß-Bilder, so ist die Welt einfach ein Feld von N*N Pixeln mit Werten 1 oder 0. Die Systeme erzeugen einen Vorrat V von Programmen, die irgendwann einmal kurze Beschreibungen für große Teile der Welt geliefert haben. Anfangs ist dieser Vorrat leer. Konfrontiert mit einer Beobachtung der Welt (in unserem Fall einem Bild) versucht das System ein möglichst kurzes Programm zu finden, das die Beobachtung reproduziert. Dabei stehen dem System folgende Mittel zur Verfügung:

- das Durchprobieren ganz einfacher Programme,
- das Mutieren von Programmen aus dem Vorrat V durch Probieren,
- das hierarchische Zusammenbauen von Programmen aus V durch sehr einfache Programme. Auch hier wird systematisch probiert.

Beispiel 1:
Ein völlig zufälliges Schwarz/Weiß-Bild mit N*N Pixeln kann man beschreiben durch "Die Pixel von rechts oben nach links unten haben die Werte p(1,1),..., p(N,N)" mit $p(i,j) \in 0,1$ für alle i,j. Die Beschreibung hat $N^2 + O(1)$ bits. Es liegt keine Erkenntnis vor.

Beispiel 2:
Ein ganz weißes Bild kann man etwa in der Programmiersprache PASCAL beschreiben als

for i = 1 to N do for j = 1 to N do p(i,j) = 0 od od

Man braucht nur noch log N + O(1) bits. Dies zeugt von erheblich höherem Verständnis als bei Beispiel 1, und das Programm für ein weißes Quadrat wird in den Vorrat V aufgenommen.

Beispiel 3:
Durch Mutation der Grenzen der Laufindizes des Programms aus Beispiel 2 erhält man Rechtecke. Mutation von p(i,j) = 0 in p(i,j) = 1 liefert schwarze Quadrate und Rechtecke.

Beispiel 4:
Durch Hintereinanderhängen des Programms aus Beispiel 2 und von zwei Programmen aus Beispiel 3 erhält man zwei Rechtecke auf weißem Hintergrund.

Beispiel 5:
Die Beobachtung bestehe aus zwei schwarzen Punkten $P_1 = (x_1, y_1)$ und $P_2 = (x_2, y_2)$ auf weißem Hintergrund. Der Hintergrund wird durch das Programm aus Beispiel 2 kurz beschrieben. Die Punkte werden in trivialer Weise beschrieben durch

$$p(x_1, y_1) = 1 ; p(x_2, y_2) = 1$$

Letzteres erfordert $4 * \log N + O(1)$ bits. Liegen die Punkte nahe beieinander (und N ist groß), so ist die folgende Beschreibung kürzer:

$a := x_1; b := y_1; p(a,b) := 1; a := a + x_2 - x_1; b := b + y_2 - y_1 ; p(a,b) := 1$

Liegen nämlich P_1 und P_2 nahe beieinander, so sind die Binärdarstellungen der Konstanten $x_2 - x_1$ und $y_2 - y_1$ kurz.

Ein System der oben beschriebenen Art würde also Konzepte wie "Rechteck" und "Nähe" selbständig finden und speichern.

Für das Durchprobieren aller ganz kurzen Programme einer Programmiersprache L oder das Durchprobieren von Mutationen wird der Einsatz von Parallelität vorgeschlagen. Die Wahl der Sprache L wird nun wesentlich. Ebenso müssen Rechenzeit und Speicherplatz in die Bewertung der Kompliziertheit eines Programms eingehen.

Da der Vorrat V im Laufe der Zeit groß wird, ist die Auswahl von Programmen aus V, die man auf einer Hierachiestufe s zusammenzubauen versucht, ein Problem. Es wird folgendes vorgeschlagen:

Man gehe aus von "aktiven Programmen" aus Stufen $< s$; das sind Programme, die bereits große Teile der aktuellen Beobachtung kurz beschreiben. Nun probiere man

i) solche Programme auf Stufe s in einer Weise zu kombinieren, die neu ist oder die früher erfolgreich war (bottom up).

ii) solche Programme in Stufen $< s$ zu aktivieren, deren Kombination auf Stufe s mit bereits aktiven Programmen aus Stufen $< s$ früher erfolgreich war (top down).

Als Datenstruktur zur Unterstützung hiervon wird ein gewichteter Graph vorgeschlagen. Die Knoten sind Programme aus V. Erfolgreiches Kombinieren von zwei Programmen erhöht das Gewicht der Kante zwischen diesen Programmen. (Wird die Kante lange nicht benutzt, diffundiert das Gewicht langsam).

2. Ziel der aktuellen Implementierung

Mit der Implementierung, die hier beschreiben wird, sollte möglichst schnell ein System der oben beschriebenen Art implementiert werden. Das System sollte zumindest in der Lage sein, die obigen Beispiele nachzuvollziehen. Ziel der Implementierung war es, sich ein erstes Vehikel für Experimente über die Leistungsfähigkeit und den Bedarf an Rechenleistung (real, nicht asymptotisch) solcher Systeme zu schaffen. Die verwendete Programmiersprache L wurde auf 2-dimensionale Graphik-Anwendungen hingetrimmt und extrem einfach gehalten. Sie ist insbesondere nicht universell.

3. Die Sprache L

Programme der Sprache L erzeugen quadratische Bilder B mit N*N Pixeln. Die Pixel können schwarz, weiß oder grau sein. N kann auf Werte zwischen 1 und 64 eingestellt werden.

Es gibt 2 Gruppen von je 8 Variablen: die gewöhnlichen Variablen V0, ..., V7 und die Laufvariablen L0, ..., L7. Die Variablen V1, ..., V7 und L1, ..., L7 können Werte aus $W = \{0, ..., N-1\}^2 \cup \{\Omega\}$ annehmen. Sie haben anfangs den Wert Ω. Die speziellen Variablen V0 und L0 haben stets den Wert (0, 0).

Es gibt ein Pixel-Array B[0:N-1;0:N-1]. Pixelelemente B[i,j] können Werte aus {SCHWARZ, WEISS, GRAU} annehmen. Anfangs haben alle Pixelelemente den Wert GRAU.

Programme aus L bestehen aus einem Initialisierungsteil gefolgt von einem Anweisungsteil. Ein Initialisierungsteil hat die Form

SETA $X_1\ Y_1$
...
$X_i\ Y_i$

mit $1 \leqq i \leqq 7$ und $X_j, Y_j \in \{0, ..., N-1\}$ für $1 \leqq j \leqq i$. Dadurch wird für alle j mit $1 \leqq j \leqq i$ der gewöhnlichen Variablen Vj der Wert (X_j, Y_j) zugewiesen. Ein Initialisierungsteil für i Variablen trägt zur Länge und Laufzeit eine Programms jeweils den Wert i bei.

Der Anweisungsteil besteht aus einer Folge von Anweisungen. Es gibt 4 Sorten von Anweisungen: Zuweisung, Addition, Pixel setzen und For-Loop.

Zuweisungen haben die Form

SETV Vj X Y

mit $1 \leqq j \leqq 7$ und $X, Y \in \{0, ..., N-1\}$. Dadurch wird der gewöhnlichen Variablen Vj der Wert (X,Y) zugewiesen. Zuweisungen haben Länge 2 und Laufzeit 1.

Additions-Anweisungen haben die Form

ADDV Vj D E

mit $1 \leqq j \leqq 7$ und $D, E \in \{-3, ..., 3\}$. Der aktuelle Wert von Variable Vj wird um (D, E) erhöht. Generell bricht ein Programm ohne Output ab, falls eine Variable mit aktuellem Wert Ω ausgewertet wird oder falls ihr ein Wert außerhalb ihres Wertebereichs zugewiesen wird. Additions-Anweisungen haben Länge 1 und Laufzeit 1.

Pixel werden durch den Befehl

SETP U F

gesetzt, wobei U eine Variable und $F \in \{SCHWARZ, WEISS\}$ eine Farbe ist. Ist (X, Y) der aktuelle Wert von Variable U, so wird Pixel B[X, Y] Farbe F zugewiesen. Diese Anweisungen haben Länge 1 und Laufzeit 1.

Laufanweisungen sind speziell auf 2-dimensionale geometrische Anwendungen zugeschnitten. Untergrenze und Obergrenze der Laufvariablen sind Punkte P1 und P2. Die Laufvariable variiert auf der Geraden von P1 nach P2. Laufanweisungen haben die Form

FOR Li = Vj + Lk TO Vl Rumpf END

oder

FOR Li = Vj + Lk TO Vl + Lk Rumpf END.

Hierbei sind Li und Lk Laufvariable mit $i \neq 0$ und Vj und Vl gewöhnliche Variable. Li heißt die Laufvariable dieser Anweisung. Der Rumpf ist eine Folge von Anweisungen. Laufanweisungen des Rumpfes haben Laufvariablen $\neq$ Li. Die Länge von Laufanweisungen ist 2 + Länge des Rumpfs. Die Laufzeit ist 1 + #Durchläufe * (Laufzeit des Rumpfs + 2)

Programme für die Beispiele des ersten Abschnitts sind:

Beispiel 1:
SETA 0 0
SETP V1 F(0. 0)
ADDV V1 0 1
SETP V1 F(0,1) ...

Triviales Spezifizieren von n zusammenhängenden Pixeln ist also stets durch ein Programm der Länge ungefähr 2n möglich.

Beispiel 2:

```
SETA   0 0
       0 N-1
       N-1 0
FOR L1 = V1 + L0 TO V2
FOR L2 = V0 + L1 TO V3 + L1
SETP L2 WEISS
END
END
```

Beispiel 5:

```
- Programm aus Beispiel 2 -
SETV V1 X1 Y1
SETP V1 SCHWARZ
ADDV V1 X2-X1 Y2-Y1
SETP V1 SCHWARZ
```

4. Systemarchitektur

Eingaben für das System sind sog. Muster; das sind Felder M[0: N-1 ; 0: N-1] mit Werten aus SCHWARZ und WEISS. Beim Versuch, einfache Beschreibungen für ein gegebenes Muster M zu finden, arbeitet das System in Runden. Jede Runde R hat 4 Teile:

1) Erstellen eines Programms P.
2) Ausführen und Anpassen des Programms P an Muster M.
3) Bewerten des Programms.
4) Vergleich von P mit anderen Programmen aus Vorrat V und ggf. Aufnahme von P in V.

Zu 1) Es wird eine von 5 möglichen Strategien gewürfelt. Die Wahrscheinlichkeiten beim Würfeln werden generell dynamisch bestimmt, wobei statistisch erfolgreiche Strategien bevorzugt werden. Der Mechanismus hierfür wird später beschrieben.

1.1) Aufzählen. Sei P' das letzte Programm, das in einer früheren Runde bei Strategie 1 erstellt wurde. P ist der Nachfolger von P' in der Ordnung, in der Programme ihrer Länge nach und bei gleicher Länge lexikographisch geordnet sind.

1.2) Würfeln. Die Länge $\lambda \in \{2, ..., 50\}$ von P wird gewürfelt. Die Befehle von P werden der Reihe nach gewürfelt. Die Wahrscheinlichkeit beim Würfeln jedes Befehls hängt vom vorigen Befehl ab.

1.3) Konkatenation. Zwei Programme P' und P" aus V werden gewürfelt. P = P';P".

1.4) Einsetzen. Zwei Programme P' und P" aus V sowie eine Anweisung A aus P' werden gewürfelt. P erhält man durch Ersetzen von Anweisung A in Programm P' durch Programm P".

1.5) Modifikation. Ein Programm P aus V und eine von 6 Unterstrategien werden gewürfelt. Zwischen Alternativen innerhalb der Unterstrategien wird ebenfalls durch Würfeln entschieden.

1.5.1) In einem SETP-Befehl in P wird die Farbe geändert.

1.5.2) In einer Anweisung in P wird eine Variable durch eine andere Variable gleichen Typs ersetzt.

1.5.3) Eine Anweisung in P wird durch eine andere Anweisung ersetzt

1.5.4) Eine Anweisung aus P wird gestrichen.

1.5.5) Eine Anweisung aus P wird an eine andere Stelle verschoben.

1.5.6) Eine neue Anweisung wird in P eingefügt.

Zu 2) Das Programm P wird einem einfachen Syntaxtest unterzogen und dann interpretiert. Ist das Programm syntaktisch korrekt und erzeugt es überhaupt einen output $B_{P'}$, so setzt der Anpassungsmechanismus ein. Dieser Mechanismus versucht durch Variation der in Programm P vorkommenden Konstanten eine Zielfunktion Z(M, B_P) zu maximieren, welche das Ausmaß der Übereinstimmung von M und B_P mißt. Im folgenden werden die Zielfunktion Z und die verwendeten Optimierungsstrategien beschrieben.

2.1) Sei F eine Farbe, M ein Muster und $Q \in \{0, ..., N-1\}^2$ ein Punkt. Sei D(F, M, Q) der Abstand von Q zum am nächsten liegenden Punkt in Muster M mit Farbe F. Das Potential POT(F, M, Q) von Muster M bezüglich Farbe F an Punkt Q wird definiert durch

$$POT(F, M, Q) = \begin{cases} 1 & \text{falls } D(F, M, Q) = 0 \\ 1/(1 + D(F, M, Q)) - 1 & \text{sonst.} \end{cases}$$

Sei B ein output. Dann ist die Zielfunktion Z(M, B, F) bezüglich Farbe F definiert durch

$$Z(M, B, F) = \sum_{B[Q]=F} POT(F, M, Q)$$

Die Zielfunktion Z(M, B) ist definiert durch

$Z(M, B) = Z(M, B, SCHWARZ) + Z(M, B, WEISS)$.

Zur Berechnung der Abstände D(F, M, Q) werden mit Hilfe des Bresenham-Algorithmus [B] immer weitere konzentrische Kreise um Q abgesucht, bis man zum erstenmal einen Punkt von Muster M mit Farbe F findet.

2.2) Die Optimierung verwendet eine Kombination aus Gradientenverfahren und Simulated Annealing [KGV], wobei sich 4 Runden Gradientenverfahren durch 3 Runden Simulated Annealing abwechseln.

Beim Gradientenverfahren wird versucht, die Zielfunktion dadurch zu vergrößern, daß eine oder beide Komponenten einer Konstanten in Programm P um eine Schrittweite s erhöht oder erniedrigt werden. Die erste solche Änderung, die gefunden wird, wird auch ausgeführt. Man steigt also nicht notwendig in Richtung des steilsten Aufstiegs auf. Man beginnt mit Schrittweite N/4. Erhöht keine Änderung mit Schrittweite s die Zielfunktion, wird die Schrittweite halbiert.

Gradientenverfahren können in lokale Maxima laufen. Aus diesen kann man mit Hilfe des Simulated Annealing-Verfahrens entkommen. Es wird eine Veränderung einer Konstanten gewürfelt. Schrittweite und Richtungen sind dabei beliebig. Erhöht die Veränderung die Zielfunktion, so wird sie durchgeführt. Erniedrigt sie die Zielfunktion um ΔZ, so wird sie abhängig von einem Parameter T (Temperatur) nur mit einer Wahrscheinlichkeit $p = \exp(-\Delta Z/T)$ ausgeführt. Die Temperatur T wird gemäß einem "Annealing Schedule" zwischen sukzessiven Annealing Schritten von einer hohen Anfangstemperatur (hohe Wahrscheinlichkeit, aus lokalen Maxima zu entkommen) zu einer niedrigen Endtemperatur (niedrige Wahrscheinlichkeit) erniedrigt.

Ist das Maximum, welches in der nächsten Runde des Gradientenverfahrens gefunden wird, schlechter als das Maximum der vorausgehenden Runde des Gradientenverfahrens, so wird die nächste Runde des Annealing-Verfahrens im besseren der beiden Maxima gestartet.

Zu 3) Sei P ein Programm aus L, das gefunden wurde beim Versuch, Muster M zu beschreiben. Sei B der Output von P. Sei b die Anzahl von Werten W, so daß $B[W] \in \{SCHWARZ, WEISS\}$. Programm P beschreibt also b Pixel. Sei e die Anzahl von Werten W mit B[W] =

M[W]. Dies ist die Anzahl der korrekt beschriebenen Pixel. Sei λ die Länge und τ die Laufzeit von Programm P. Dann wird die Güte G(P) definiert durch

G(P)	=	$\alpha*(2b - \lambda)$	(Informationskompression)
	+	$\beta*(2b - \tau)$	(i. A. negativ; Strafe für Laufzeit)
	+	$\gamma*b$	(Belohnung für Beschreibung von etwas Großem)
	+	$\delta*e$	(Belohnung für korrekte Beschreibung).

Die Gewichte α, β, γ und δ können eingestellt werden.

Zu 4) Ist G(P) positiv, so wird P in Vorrat V aufgenommen, sofern es nicht ein äquivalentes Programm höherer Güte bereits in V gibt. Zunächst wird P durch Umbenennen von Variablen in eine Form gebracht, so daß die ersten Vorkommen der Variablen in P in der Reihenfolge V1, V2,... bzw. L1, L2, ... erscheinen. Ist P bereits in V, geschieht weiter nichts.

Ist P noch nicht in V, so wird für alle Programme P' in V versucht, mit Hilfe des Anpassungsmechanismus den Output B_P von Programm P zu erzeugen. Dabei wird B_P als Muster für P' benutzt. Die obige Definition des Potentials funktioniert auch dann, wenn das Muster graue Pixel hat. Produziert ein Programm P' den gleichen Output wie P, so wird P' durch P ersetzt, falls G(P) > G(P'). Produziert kein Programm P' den gleichen Output wie P, wird P in den Vorrat V aufgenommen.

Die Wahrscheinlichkeiten beim Würfeln werden in Abhängigkeit von einer einstellbaren Konstanten ρ wie folgt modifiziert. Der einfachste Fall liegt vor, wenn die Anzahl a der Alternativen für alle Zeiten fest bleibt. Jede Alternative A erhält eine Zählvariable Z_A. Wurde Alternative A gewählt beim Erzeugen eines Programms P, das in Vorrat V aufgenommen wurde, so wird Z_A um 1 erhöht. Nach jedem ρ-ten Programm P, das in Vorrat V aufgenommen wurde, werden die Wahrscheinlichkeiten für Alternativen A im wesentlichen auf $Z_A/(\Sigma\ Z_B)$ gesetzt (zusätzlich werden gewisse Untergrenzen nicht unterschritten).

Für die Auswahl von Programmen P aus V sei

$$G(V) = \sum_{P\ in\ V} G(P)$$

Programm P wird mit Wahrscheinlichkeit G(P)/G(V) gewählt.

Die Auswahl einer Anweisung aus einem Programm P, das beliebig lang werden kann, geschieht in zwei Schritten. Im ersten wird ein Typ von Anweisung gewählt. Die Anzahl der Alternativen ist 4. Unter Anweisungen gleichen Typs wird mit Gleichverteilung gewürfelt.

5. Parallelarbeit

Das System läuft auf mehreren SINIX-MX2-Rechnern, die über das CANTUS-Netz kommunizieren. Von diesen ist einer als Master ausgezeichnet. Der Master verfügt über graphische Ein-/Ausgabe (z. Zt. realisiert über einen Atari 1040ST). Der Master kann prinzipiell allein arbeiten. Er verwaltet den Vorrat V und führt die Statistik. Periodisch prüft er, ob andere Rechner (Slaves) bereit sind, mitzuarbeiten. Bei der Anmeldung erhält jeder Slave einen individuellen Startwert für seinen Pseudozufallszahlengenerator. Dadurch wird verhindert, daß Slaves, die gemeinsam die gleiche Aufgabe behandeln, ständig identische Alternativen wählen. An die Slaves, welche mitarbeiten, schickt der Master periodisch Updates des Vorrats V und der Statistik.

Die Standardaufgabe, welche die Slaves zu bearbeiten haben, besteht darin, zu gegebenem Muster M und Güte g ein Programm zu suchen, das bezüglich M eine Güte $> g$ hat. Muster M und Güte g werden vom Master verteilt. Hat ein Slave ein solches Programm P' mit Güte g' gefunden, wird dies dem Master gemeldet.

6. Ausblick

Zunächst ist beabsichtigt, das System mit vielen Mustern laufen zu lassen und sich von den produzierten Beschreibungen überraschen zu lassen. Darüber hinaus sind im wesentlichen alle Komponenten des Systems zu verbessern. Die Größe der Muster soll auf die Größe guter Monitore ($N = 1000$) erhöht werden. Die Eingabe soll über eine Kamera erfolgen. Die Sprache L muß durch eine universelle Sprache ersetzt werden. Die Definition der Länge λ von Programmen sollte in etwa gleich der Anzahl der verwendeten Bits sein. Die Schrittzahl τ wird durch Timer gemessen werden. Die im ersten Abschnitt genannte Hierarchie von Programmen muß implementiert werden.

Die folgenden Ausdehnungen auf weitere Aufgaben bieten sich an:

1) Input über 2 Kameras (Stereovision)
2) Input über Mikrophone
3) Folgen von Mustern als Input.

Eine wesentliche Erweiterung wäre es, dem System die Möglichkeit zu geben, seine Inputs selbst zu beeinflussen (auf angetriebene Räder montieren, Robotarm, Lautsprecher). In diesem Fall muß sich das System nicht nur Programme zum Beschreiben des Inputs auswürfeln, sondern auch Programme zum Betreiben der Antriebe der Räder, des Robotarms und des Lautsprechers. Es müssen die Wahrscheinlichkeiten für solche Alternativen verstärkt

werden, in Folge derer ein Input M generiert wurde, für den ein Programm P hoher Güte gefunden wurde.

7. Literatur

[BPT] Bergmann P, Paul W. J. and Thiele L:
An Information Theoretic Approach to Computer Vision. To appear in Proc. Workshop on Dynamical Networks, Eisenach, 1988.

[B] Bresenhamm J. E.:
A linear algorithm for incremental digital display of circular area.
Communications of the ACM 20(2), Feb. 77, pp. 100-106.

[KGV] Kirkpatrik S., Gelott C. D., Vecchi M. P.:
Optimization by simulated annealing.
Science 220, 1983, Number 1598.

[K] Kolmogoróv, A. N.:
Three approaches to the quantitative definition of information problems.
information transmission vol. 1, No. 1, pp. 1-7, 1965.

[S] Solomonoff R. J.:
A formal theory of inducture inference Part 1 & 2.
Information and Control, Vol. 7, pp. 1-22, 224-254, 1964.