Gedächtnisprotokoll Diplomprüfung Vertiefungsfach Wissenbasierte Systeme

Prüfungsinhalt: Artificial Intelligence (WS)
Knowledge Representation (SS 2002)
Logikprogrammierung (WS)
Prüfer: Lakemeyer

Datum: 4.3.2003
Prüfling: Lutz Ißler

Ich schlage vor, wir machen Künstliche Intelligenz, Wissensrepräsentation und dann Logikprogrammierung.

Künstliche Intelligenz

Wir haben informierte und uninformierte Suchverfahren kennegelernt. Welche uninformierten würden Sie denn benutzen?
Es gibt Breitensuche. Günstiger als Breitensuche ist Tiefensuche, diese kann allerdings in unendlichen Pfaden im Suchbaum steckenbleiben. Eine Möglichkeit, das zu verhindern, ist Iterative Deepening.

Was wollen Sie denn nun verwenden? Breitensuche?
Nein! Die ist viel zu teuer. Iterative Deepening ist die erste Wahl.

Iterative Deepening, ist das denn optimal?
Ja, unter der Voraussetzung, dass die Kosten auf allen Pfaden monoton steigen, und dass die Kosten auf allen Pfaden eine Funktion der Tiefe der Knoten sind.

Genaugenommen, wenn die Kosten überall gleich sind. Sonst finden Sie nur eine tiefenoptimale Lösung.
Hm.

Welche Art von Vorwissen über den Suchraum kann man denn verwenden?
Eine Heuristik. Diese muss admissible, dh. zulässig sein. Das ist eine Heuristik, die die Kosten zum Ziel niemals überschätzt. Eine solche Heuristik erfüllt die Dreiecksungleichung.

Und wie finden Sie eine solche Heuristik?
Durch Vereinfachung des Problems. Wenn man z.B. die kürzeste Wegstrecke von *A* nach *B* sucht, kann man als Heuristik die Luftlinienentfernung verwenden.

Betrachten wir mal die Planung. Was ist da der Suchraum?
Die möglichen Pläne.

Und was ist das?
Äääh...

Na STRIPS-Operatoren. Was können Sie denn da vereinfachen, um eine Heuristik zu finden?
Äääh... nun ja, die Operatoren.

Fangen wir anders an. Woraus besteht so ein STRIPS-Operator?
Aus Vorbedingung, Delete-Liste und Add-Liste.

So eine Vorbedingung, was ist das?
Eine Konjunktion von Literalen.

Literale, hm. Können die auch negativ sein?
Äääh... nein.

Wie ist das mit der Delete-Liste und der Add-Liste, können da negative Literale vorkommen?
Ja.

Und was können Sie jetzt vereinfachen?
Äääh... (kam das eigentlich jemals in der Vorlesung?)

Sie können die Delete-Liste weglassen. Wenn Sie damit einen Plan finden, finden Sie auch mit den vollwertigen Operatoren einen Plan.
(Aha.)

Bleiben wir bei der Planung. Wir hatten den POP-Algorithmus. Wie sieht denn der Anfang beim POP-Algorithmus aus?
Man hat einen Startzustand mit bestimmten Bedingungen und einen Endzustand, dessen Bedingungen man erfüllen möchte.

Aha. Und was ist das, so ein Zustand?
Äääh...

Malen Sie doch mal so eine Anfangssituation beim POP auf.
(Aufgemalt.)

So, und was ist das jetzt, das, wo Sie „Start“ reingeschrieben haben?
Der Startzustand?

Das ist ein STRIPS-Operator. Wissen Sie, Sie planen ja die ganze Zeit mit STRIPS-Operatoren, da sind „Start“ und „Ziel“ natürlich auch STRIPS-Operatoren. Und die haben Vor- und Nachbedingungen, die Sie erfüllen müssen.
Hm.

So, was fehlt jetzt noch?
Die Ordnung.

Malen Sie mal einen Pfeil.
(Pfeil gemalt.)

Und was ist das jetzt?
Ein... abstrakter Plan?

Ein partieller Plan. Induktives Lernen. Was haben wir da an Möglichkeiten kennengelernt?
Decision Trees, Decision Lists und Neuronale Netze.

Warum haben wir denn die Decision Lists betrachtet?
Weil die eine größere Klasse von Funktionen repräsentieren können als die Decision Trees.

Falsch, die Klasse ist kleiner.
Ah, ja, natürlich – und wenn man die Anzahl der Literale je Test unbeschränkt lässt, ist sie gleich groß.

Genau. Also, was ist der Vorteil, wenn man die Klasse der repräsentierbaren Funktionen einschränkt?
Dass der Suchraum kleiner wird.

Fällt Ihnen dazu noch was ein?
Ja, PAC-Learning. Man kann die Anzahl der Beispiele bestimmen, die man lernen muss, um mit einer hohen Wahrscheinlichkeit eine nahezu korrekte Hypothese zu erhalten.

Und wie viele Beispiele waren das bei Decision Lists?
Äääh... also die PAC-Formel benötigt die Größe des Hypothesenraums, und dann kann man das ausrechnen.

Ja, aber was kam denn da raus, zum Beispiel bei so einer 3-DL?
Äääh... 3 Literale pro Test, jeweils zwei Werte macht schonmal 2^3... und dann...

...kommt in jedem Fall eine Anzahl mit hoch n raus. Kennen Sie Funktionen, die sich zum Beispiel mit Neuronalen Netzen besser darstellen lassen als mit Decision Trees?
Ja, die Mehrheitsfunktion.

Wie sieht denn so ein Netz aus, das die Mehrheitsfunktion darstellt?
Dazu reicht ein Perceptron, also ein Netz ohne Zwischenschicht. (Ein Perceptron aufgezeichnet.)

Und wie sind die Gewichte und die Aktivierungsfunktion?
Gewichte sind alle eins, Aktivierungsfunktion ist Step-Funktion mit Schwellwert bei $\lceil \frac{n}{2} \rceil$.

Zum Beispiel, ja. Wie sieht es denn mit Booleschen Funktionen aus, geht das auch mit so einem Netz?
Nein, dazu braucht man Zwischenschichten.

Wie viele Schichten oder Knoten brauchen Sie denn, um so eine allgemeine Boolesche Funktion darzustellen?
Äääh... einige.

Man kann ja die einfachen Booleschen Verknüpfungen mit jeweils einer Einheit darstellen...
...und da jede Boolesche Funktion in KNF darstellbar ist, braucht man in der Ausgabeschicht einen Knoten für die Konjunktion, davor eine Zwischenschicht für die Disjunktionen und dann entweder noch eine Zwischenschicht oder die doppelte Zahl an Eingabeeinheiten, um negative Eingaben darzustellen.

Mal abgesehen davon, dass man Boolesche Funktionen ja sowieso nicht mit Neuronalen Netzen repräsentiert, sondern dafür besser eine Wissensbasis verwendet. Und damit...

Knowledge Representation

...sind wir schon bei der Wissensrepräsentation. Für eine effiziente Repräsentation haben Beschreibungslogiken betrachtet. Was für Inferenzen interessieren uns denn da?
Zum Beispiel Subsummierung und die Frage, ob eine Konstante zu einem Konzept gehört.

Wie berechnet man denn die Subsummierung?
Durch Struktur-Matching. (Algorithmus grob erklärt.)

Der Algorithmus arbeitet teilweise rekursiv. Können Sie erklären, wo?
Äääh... beim **AND**.

Nein, beim **ALL**. *Wenn oben zum Beispiel steht* (**ALL** *r C*) *und unten steht* (**ALL** *r D*), *dann müssen Sie den Algorithmus für C und D rekursiv aufrufen. Wir hatten einen Operator* **SOME**. *Wie ist die informelle Semantik von* (**SOME** *r C*)*?*
Das sind alle Objekte, die über die Rolle *r* mit mindestens einem Objekt des Konzepts *C* verbunden sind.

Und wie ist die formale Semantik?
Das macht man mit dem Φ. (Aufgeschrieben: Φ[(**SOME** *r C*)]...)

...und das Φ *ist Teil einer Interpretation mit einer Domäne D und einer Belegung...*
Ja, richtig... (aufgeschrieben: $\Im = \langle D, \Phi \rangle$. Dann den Rest der Semantik von **SOME** hergeleitet und notiert.)

Was würden Sie sagen, wird (**SOME** *r C*) *von* (**ALL** *r C*) *subsummiert?*
Ja. Denn wenn alle Erfüller von *r C*s sind, dann gibt es auch mindestens einen Erfüller, der *C* ist. Oha, es gibt einen Sonderfall: Das **ALL** trifft auch dann zu, wenn es keine Erfüller von *r* gibt. Also korrigiere ich mich zu: nein.

Logikprogrammierung

Was ist denn Resolution? Erklären Sie mal.
Man geht im aussagenlogischen Fall von einer Formel in KNF aus und betrachtet die einzelnen Disjunktionen, das heisst Klauseln. Zwei Klauseln sind resolvierbar, wenn ein Literal in einer Klausel negativ und in der anderen Klausel positiv vorkommt. Das Ergebnis ist eine Klausel, die das Literal nicht mehr enthält.

Und warum macht man das?
Um Unerfüllbarkeit einer Klauselmenge, dh. der KNF-Formel, zu zeigen.

Und warum kann man die Literale so einfach resolvieren?
Äääh... (ich stehe auf dem Schlauch.) Weil die Klauseln damit unerfüllbar sind.

Das gilt nur für Einheitsklauseln.
Hm, ja. Weil die Literale in den Klauseln zur Unerfüllbarkeit beitragen.

Nun ja. Wie schwer ist denn Resolution?
Exponentiell.

Unentscheidbar. Sie haben vom aussagenlogischen Fall gesprochen, was macht man denn im Fall der Logik erster Stufe, also mit Variablen?
Dann muss man zwei Literale unifizieren, bevor man sie resolvieren kann. Das heisst, man muss die Literale strukturgleich machen.

Und wie geht das?
Durch einen Strukturvergleich: Treffen Sie auf eine Stelle, an der eine Variable und... (Unifikationsalgorithmus erklärt.)

Wie komplex ist der Algorithmus?
Exponentiell.

Und warum? Mal eben gucken, ob eine Variable in einem Term vorkommt, ist doch nicht so schwer?
Äääh...

Denken Sie mal dran, was Sie alles angucken müssen, bis Sie die Variable gefunden haben.
Die Länge der Terme.

Und welche Länge?
Die maximale Länge.

Das ist eine Tautologie... lassen wir's mal dabei. Bei Logikprogrammen betrachten Sie Hornklauseln und eine Einschränkung der Resolution, die SLD-Resolution. Wo ist denn da die Einschränkung?
Lineare Resolution, dabei dürfen nur Klauseln verwendet werden, die aus dem Programm kommen oder die als Resolventen aufgetaucht sind. Bei Logikprogrammen starten man mit der Zielklausel und erhält dann nur negative Resolventen.

Und das ist eine Einschränkung?
Äääh... nein.

Sondern, wo ist diese?
Man erhält nur negative Resolventen, weil man immer mit Hornklauseln resolviert.

Ja, und das sind die Programmklauseln. Das ist die Einschränkung – dadurch, dass Sie nur mit Programmklauseln resolvieren dürfen, ist die Resolution nicht mehr vollständig. Wie schwer ist SLD-Resolution denn?
Unentscheidbar. Noch immer sehr schwer.

Exponentiell, um genau zu sein. Prolog hat eine Möglichkeit, unendliche Pfade im SLD-Baum zu verhindern, den Cut. Was macht der?
Der schneidet Teilbäume im SLD-Baum ab. Diese können unendliche Pfade und nicht erfolgreiche Rechnungen enthalten, aber auch erfolgreiche Rechnungen. Im letzteren Fall wird durch den Cut die Semantik des Programms verändert. Das kann gewollt sein, aber auch gefährlich.

Ja, der Programmierer weiss manchmal nicht, was er da tut. Wir hatten Negation as Failure betrachtet. Wie können Sie das mit dem Cut simulieren?
Zum Beispiel mit einem Prädikat `not(A)`. (Klauseln aufgeschrieben.)

Das `true` *in der zweiten Klausel können Sie sich sparen. Wie sieht denn die Auswertung aus?*
(Auswertung erklärt.)

Und wenn Sie den Cut weglassen, was passiert dann?
(Auswertung erklärt.)

Knowledge Representation, 2. Teil

Wir haben über Abductive Reasoning gesprochen. Was versteht man darunter?
Das Schließen von Beobachtungen auf Ursachen.

Und das bedeutet was?
Dass man von einer gegebenen Beobachtung auf eine Ursache schließt...?

Passen Sie auf. Man hat das hier gegeben: $A, A \supset B$. Was sagt Ihnen das?
Da kann man dann von *B* auf *A* schließen.

Und das ist dann abductive? Oder doch deductive?
(Diese Schreibweise kommt mir bekannt vor, aber wie war das doch jetzt gleich genau? Außerdem sieht das nach einer Fangfrage aus, also ist es wohl doch eher deductive. Wobei... ich entscheide mich für...) äääh...

Das ist deductive. Schauen Sie mal, daraus möchten wir B ableiten. (Schreibt dahinter ein $\models B$.)
Ah, ja.

Und was ist dann abductive?
Äääh....

Das hier: $B, A \supset B$. Und daraus schließen Sie A.
Ah ja.

Eine Anwendung ist die Fehlererkennung in Schaltkreisen. Wie funktioniert das?
Man benötigt das Modell eines Schaltkreises und ein Fehlermodell. Wenn ein Fehler, dh. eine falsche Ausgabe des Schaltkreises, vorliegt, können Sie anhand dieser Angaben auf mögliche Fehler schließen. Das Problem dabei ist, dass zum einen die Diagnose minimal ist, obwohl nicht nur minimal viele Bauteile defekt sein müssen, und dass im Fehlermodell nicht alle möglichen Fehler erfasst sein könnten.

Wie berechnet man denn so ein minimales Fehlerszenario?
Durch Berechnung aller Resolventen der Eingabeklauseln.

Und dann?
Minimiert man die Klauseln.

Minimierung heisst anhand der Anzahl der Literale?
Nein, bezüglich Untermengen.

Und was ist das, was Sie da berechnen?
Äääh...

Na Primimplikanten. Wie schwer ist das denn?
Schwer, weil man im schlimmsten Fall alle Resolventen berechnen müsste.

Und wie viele Resolventen gibt es so?
Exponentiell viele.

Ja, und damit bleibt es exponentiell, selbst wenn Sie nicht alle berechnen müssten. Es gibt sogar Varianten, da brauchen Sie gar kein Fehlermodell, und es funktioniert trotzdem nicht. Da hat Intel schon viele Millionen für ausgegeben, das können Sie mir glauben. Das ist nämlich eine der Standard-Methoden zum Berechnen von Fehlern.
Und das trotz der Probleme mit der minimalen Fehlerdarstellung?

Ja.

Fazit:

Die Athmosphäre war sehr entspannt. Ich fühlte mich vorher gut vorbereitet, während der Prüfung sind mir dann aber doch sehr viele Lücken aufgefallen. Ich kann nur empfehlen, jedes kleinste bisschen, was man liest, gründlich zu hinterfragen („Wie schwer ist Unifikation – warum?“).