Apprendimento di regole del primo ordine: Aleph

Aleph

- Disponibile a

http://web.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/aleph_toc.html

- Scritto in Prolog, open source
- Aleph, come Progol, usa la bottom clause per limitare dal basso lo spazio di ricerca
- Utilizza un algoritmo branch and bound per cercare una clausola
- Molte possibili opzioni, che lo rendono in grado di emulare molti altri sistemi di ILP

Ricerca di una clausola

```
begin
   active:= {0}; (comment: ``0" is a conventional starting point)
   C:= inf;
   currentbest:= anything;
   while active is not empty do begin
      remove first node k from active; (comment: k is a branching node)
      generate the children i=1,...,Nk of node k, and
         compute corresponding costs Ci and
            lower bounds on costs Li;
      for i = 1,...,Nk do
         if Li >= C then prune child i
         else begin
            if child i is a complete solution and Ci < C then begin
                  C:= Ci, currentbest:= child i;
                  prune nodes in active with lower bounds more than Ci
            end
            add child i to active
         end
   end
end
```

Strategia di ricerca

- Nodo=clausola
- L'ordinamento dei nodi in active: dipende dalla strategia di ricerca scelta
  - Breadth-first: le clausole sono ordinate dalle piu' corte alle piu' lunghe. A parita' di lunghezza sono ordinate in base all'euristica (strategia di default)
  - Depth first; le clausole sono ordinate dalle piu' lunghe alle piu' corte. A parita' di lunghezza sono ordinate in base all'euristica
  - Best first: le clausole sono ordinate in base al valore dell'euristica
  - Piu' altre

Euristiche

P, N are the number of positive and negative examples covered by the clause

- accuracy
  - Clause utility is P/(P+N)
- auto_m
  - Clause utility is the m estimate with the value of m automatically set to be the maximum likelihood estimate of the best value of m.
- compression
  - Clause utility is P - N - L + 1, where L is the number of literals in the clause.
- coverage
  - Clause utility is P - N

Euristiche

- entropy
  - Clause utility is p log p + (1-p) log (1-p) where p = P/(P + N)
- gini
  - Clause utility is -2p(1-p) where p = P/(P + N)
- laplace
  - Clause utility is (P+1)/(P+N+2)

Euristiche

- mestimate
  - Clause utility is its m estimate. The value of m is set by set(m,M).
- wracc
  - Clause utility is calculated using the weighted relative accuracy function

Osservazioni

- Generazione dei figli: operatore di raffinamento
  - Aggiunta di un letterale alla volta preso dalla bottom clause.
    - Controllo per evitare l'aggiunta di letterali duplicati
  - Raffinamento definito dall'utente: possibilita' di aggiungere piu' letterali alla volta

Osservazioni

- Lower bounds: minimo costo che puo' essere ottenuto dal sottoalbero sotto il nodo.
- Il calcolo dipende dall'euristica.
- Ad esempio, per la coverage, il costo e' N-P e il lower bound e' –P, equivalente a suppore che da un nodo k si possa ottenere una clausola che copre i P positivi di k e nessun negativo
- Se non si riesce a calcolare il bound, viene assunto = 0

Osservazioni

- Si puo' interrompere la ricerca anche prima che active sia vuoto mettendo un limite al numero di nodi esplorati

Aleph

- Lanciare l'ambiente Prolog Yap
- Caricare aleph:

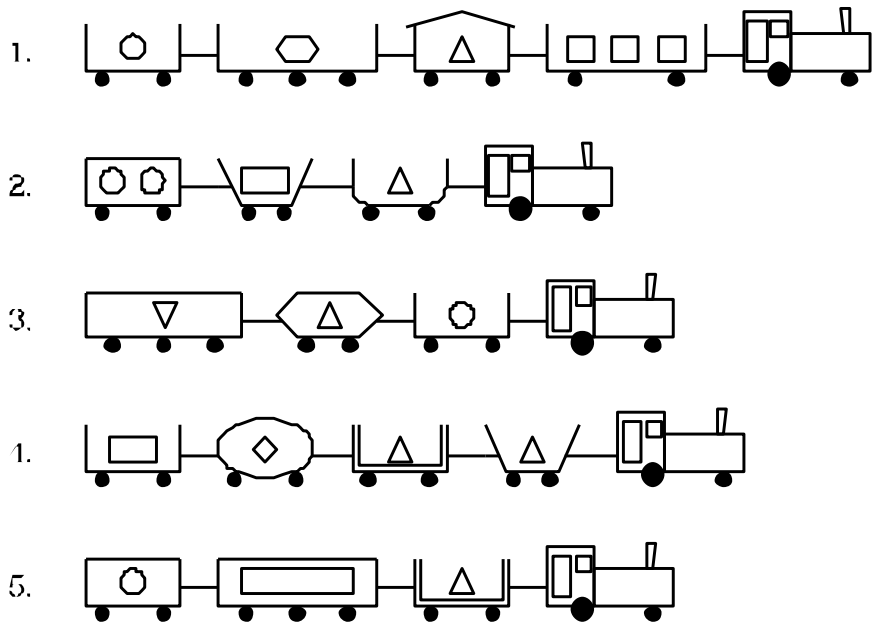
  ?- [aleph].
- Aleph richide 3 file per costruire teorie. L'uso più semplice di Aleph consiste nel
  1. Costruire 3 file di dati chiamati `filestem.b, filestem.f, filestem.n'.
  2. Leggere tutti i dati con il comando read_all(filestem).
  3. Costruire una teoria con il comando induce.

Un semplice esempio: train-spotting

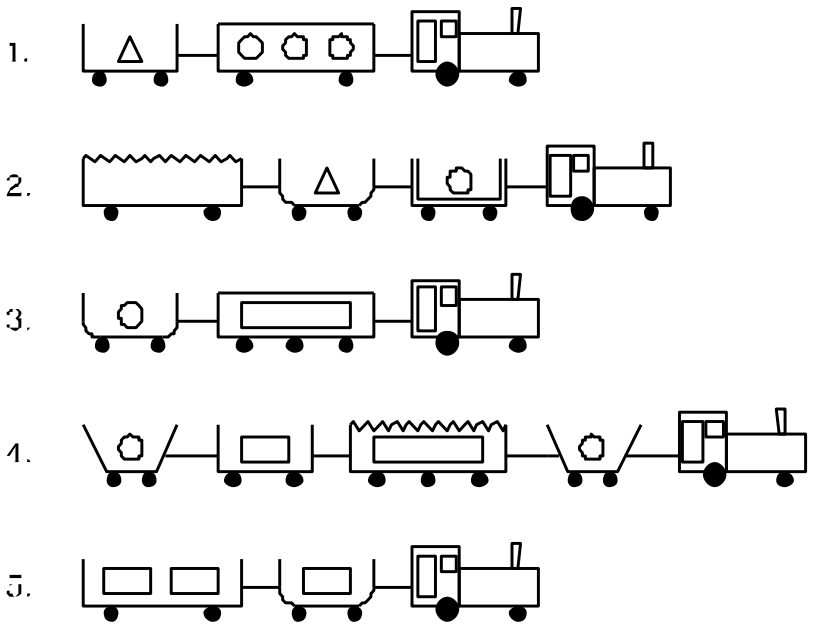
- Un semplice esempio di input per Aleph è la codifica di un problema proposto da Ryszard Michalski
- L'obiettivo è quello di trovare una spiegazione plausibile per distinguere tra due tipi di treni: treni che vanno ad est (eastbound) e treni che vanno ad ovest (westbound).
- Filestem= train

Trainspotting

1. TRAINS GOING EAST



2. TRAINS GOING WEST



File di background knowledge

- La background knowledge deve essere contenuta nel file filestem.b
- Ha la forma di clausole Prolog
- Il file può contenere inoltre direttive per il compilatore Prolog come :-consult(someotherfile).
- Il file contiene inoltre le restrizioni di linguaggio e di ricerca per Aleph:
  - *modi*
  - *tipi*
  - *determinazioni*

Esempio di background knowledge

- Trainspotting: background (nel file train.b):

```
% eastbound train 1
has_car(east1,car_11).
has_car(east1,car_12).
short(car_12).
closed(car_12).
long(car_11).
open_car(car_11).
shape(car_11,rectangle).
load(car_11,rectangle,3).
wheels(car_11,2).
wheels(car_12,2).

....
```

Dichiarazioni di modi

- Sintassi

:- modeb(RecallNumber,PredicateMode).

:- mode(RecallNumber,PredicateMode).

- RecallNumber può essere un numero oppure '*'. E' generalmente più semplice specificare RecallNumber come *.
- Indica il numero di risposte alle query da includere nella bottom clause

Dichiarazioni di modi

- PredicateMode è un template della forma:

p(ModeType, ModeType,...)

- Ecco alcuni esempi di come appaiono in un file:

```
:- modeb(1,mem(+number,+list)).
:- modeb(1,dec(+integer,-integer)).
:- modeb(1,mult(+integer,+integer,-integer)).
:- modeb(1,plus(+integer,+integer,-integer)).
:- modeb(1,(+integer)=(#integer)).
:- modeb(*,has_car(+train,-car)).
```

Dichiarazioni di modi

- Ogni ModeType può essere:
  - **Semplice:**
    - (a) **+T** variabile di “input”di tipo **T**;
    - (b) **-T** variabile di “output” di tipo **T**; oppure
    - (c) **#T** costante di tipo **T**.
  - **Strutturato:** è della forma **f(..)** dove **f** è un simbolo di funzione e ogni argomento può essere sia semplice che strutturato. Ecco un esempio:

```
:- modeb(1,mem(+number,[+number|+list])).
```

Dichiarazioni di tipi

- I tipi devono essere specificati per ciascun argomento di tutti i predicati che devono essere usati nella costruzione di una ipotesi. Questa specifica è fatta all'interno dei comandi mode(...,...).
- Per Aleph i tipi sono semplicemente nomi e nessun type-checking è fatto. Variabili di tipo diverso sono trattate distintamente anche se una è un sub-tipo dell'altra.
- Quando si variabilizza la bottom clause, una variabile di input (+T) deve apparire come variabile di output dello stesso tipo in un letterale precedente e una costante (#T) non viene sostituita con una variabile

Dichiarazioni di determinazione

- Le dichiarazioni di determinazione dichiarano i predicati che possono essere utilizzati nelle teste e nei body. Prendono la forma

:-determination(TargetName/Arity,BackgroundName/Arity).

  - Significa che devo imparare il predicato TargetName/Arity e che nelle regole per TargetName/Arity puo' comparire il predicato BackgroundName/Arity nel body
- Tipicamente ci saranno tante dichiarazioni di determinazione per un predicato target, corrispondenti ai predicati che si pensa siano rilevanti nel costruire ipotesi.

Dichiarazioni di determinazione

- Se nessuna determinazione è presente, Aleph non costruisce alcuna ipotesi.
- Le determinazioni sono ammesse per un solo predicato target per ciascuna esecuzione di Aleph: se compaiono determinazioni multiple viene scelta la prima. Esempi:

```
:- determination(eastbound/1,has_car/2).
:- determination(mult/3,mult/3).
:- determination(p/1,'='/2).
```

File di esempi

- **File di esempi positivi:** ha estensione **.f** (foreground, es. train.f). Gli esempi positivi sono semplicemente dei fatti ground. Ad esempio
- eastbound(east1). eastbound(east2). eastbound(east3).
- **File di esempi negativi**: ha estensione .n (es. train.n). Gli esempi negativi sono semplicemente dei fatti ground. Ad esempio
- eastbound(west1). eastbound(west2). eastbound(west3).

Lettura dei file di input

- Una volta che i file `filestem.b, filestem.f' e `filestem.n' sono pronti, possono essere letti con:

read_all(filestem).

Es.: read_all(train).

- Questo tipicamente produce una lunga lista delle impostazioni correnti

Costruire una teoria

- Il comando di base per costruire una teoria è:

induce.

- Il risultato è un elenco che mostra le bottom clause e la clausole esplorate insieme alla loro copertura di esempi positivi e negativi

Costruire una teoria

[bottom clause]

```
eastbound(A) :-
   has_car(A,B), has_car(A,C), has_car(A,D), has_car(A,E),
   short(E), short(C), closed(C), long(D),
   long(B), open_car(E), open_car(D), open_car(B),
   shape(E,rectangle), shape(D,rectangle), shape(C,rectangle),
   shape(B,rectangle),
   wheels(E,2), wheels(D,3), wheels(C,2), wheels(B,2),
   load(E,circle,1), load(D,hexagon,1), load(C,triangle,1),
   load(B,rectangle,3).
```

…..

Costruire una teoria

...

eastbound(A) :- has_car(A,B). [5/5]

eastbound(A) :- has_car(A,B), short(B). [5/5]

...

- Il risultato finale ha l'aspetto

[theory] [Rule 1][Pos cover = 5 Neg cover = 0]

eastbound(A) :- has_car(A,B), short(B), closed(B).

[pos-neg] [5]

Costruire una teoria

- induce stampa anche la prestazione della teoria sui dati di training come una matrice di confusione

```
[Training set performance]
         Actual
        +   -
      + 5   0   5
Pred  - 0   5   5
        5   5   10
Accuracy = 100%
```

- Anche la prestazione su dati di test è riportata se i valori per i parametri test_pos e test_neg sono settati

Altri comandi

- **Salvare una teoria:** con

write_rules(FileName).

- memorizza in FileName la teoria prodotta.
- **Valutare una teoria:** la teoria prodotta può essere valutata su esempi in un qualunque file di dati con il comando

test(File,Flag,Covered,Total)

  - File è il nome del file contenente gli esempi.
  - Flag può essere show o noshow per mostrare gli esempi coperti o non coperti. Sia File che Flag devono essere forniti.
- test/4 restituisce:
  - Covered è il numero degli esempi coperti dalla teoria corrente
  - Total è il numero totale di esempi nel file di dati

Parametri

- Aleph ha una serie di parametri che ne regolano il funzionamento.
- Possono essere settati con

:-set(Parameter,Value).

- Parametri importanti
  - nodes: è il numero massimo di nodi da esplorare quando si cerca una clausola accettabile. Di default vale 5000.
  - clauselength: numero massimo di letterali nel body di una clausola (default 4).
  - depth: valore massimo della profondità delle dimostrazioni (default 5).

Parametri

- i: profondita' massima delle variabili (default 2)
- test_pos: nome di un file contenente un elenco di esempi positivi da testare
- test_neg: nome di un file contenente un elenco di esempi negativi da testare
- verbosity: livello di dettaglio dei messaggi stampati dall'algoritmo. E' un intero >=0 (default 1)
- evalfn: il valore e' uno di: coverage, compression, posonly, pbayes, accuracy, laplace, auto_m, mestimate, entropy, gini, sd, wracc, or user (default coverage). Imposta la funzione euristica.

Esempio: father

- Apprendimento del predicato father:
- father.b:

:- set(i,2). :- set(verbose,1).

:- mode(*,father(+person,-person)).

:- mode(*,parent(+person,-person)).

:- mode(*,male(+person)). :- mode(*,female(+person)).

:- determination(father/2,parent/2).:- determination(father/2,male/1).

:- determination(father/2,female/1).

parent(john,mary). male(john).parent(david,steve).
male(david).parent(kathy,ellen). female(kathy).

Esempio: father

- father.f:

father(john,mary).

father(david,steve).

- father.n:

father(kathy,ellen).

father(john,steve).

Esempio: father

- Risultato

[bottom clause]

father(A,B) :-

parent(A,B), male(A).

Esempio: father

[theory]
[Rule 1] [Pos cover = 2 Neg cover = 0]
father(A, B) :-
   parent(A, B), male(A).
[Training set performance]

		Actual		
		+	-	
Pred	+	2	0	2
	-	0	2	2
		2	2	4

Accuracy = 1
[Training set summary] [[2, 0, 0, 2]]
[time taken] [0]
[total clauses constructed] [4]