Representation des connaissances et raisonnement 2

« Rapport du TP °3 : Logique possibiliste qualitative»

CHIKH Khadidja

Master 2 SII
Groupe 1

02/12/2019

Introduction :

Dans ce TP , il a été demandé de :

-Implémenter l'algorithme d'inférence utilisé pour le calcul d'une variable d'interet, en utilisant un prouver SAT.

-Générer des bases de connaissances pondérées.

-Calculer la variable d'interêt qui correspond à val(phi,sigma)

Logique possibiliste qualitative :

Elle repose sur l'idée d'une représentation ordinale sous la forme d'une relation de plausibilité entre les mondes possibles, qu'on peut projeter sur une échelle symbolique totalement ordonnée. La logique possibiliste est une extension de la logique classique à des bases de connaissances ordonnées selon leur degré de certitude**. [1]**

Implémentation :

Pour l'implémentation, nous avons utilisé le langage Python (3.7). Les figures en dessus illustrent le script répondant aux points précédents.

Generation d'une base de connaissances pondérée :

Cette classe « sigma » construit un ensemble de clauses pondérées en prenant en entrée un nombre de variables, un nombre de clauses et la taille d'une clause ; leur associe des poids aléaoitres entre 0 et 1, les tries selon ces poids et extrait la liste des strates en regroupant les poids uniques.

```
import random
import pycosat
class sigma:
      nbrvar=0
      nbrcls=0
      longcls=0
      listvar=[]
      clauses=[]
      poids=[]
      poidsstrates=[]
      l=0
      u=0

      def __init__(self,nbrvar,nbrcls,longcls):
         for i in range(nbrvar):
           self.listvar.append(i+1)
           self.listvar.append(-(i+1))
         for i in range(nbrcls):
           p=random.random()
           self.clauses.append(random.sample(self.listvar,longcls))
           self.poids.append(p)
      def sort(self):
        for i in range(len(self.poids)-1, 0, -1):
          for j in range(i):
            if self.poids[j]<self.poids[j+1]:
               self.poids[j],self.poids[j+1] = self.poids[j+1],self.poids[j]
               self.clauses[j], self.clauses[j+1] = self.clauses[j+1],self.clauses[j]

      def strates(self):
          s=set(self.poids)
          self.poidsstrates=sorted(s)
          self.u=len(s)
```

[1] :https://www.irit.fr/ADRIA/site/recherche/incertitude/qualitative/

Implémentation de l'algorithme :

Le programme principal consiste à intitialiser une clause (phi) aléatoire, avoir le nonphi, construire, trier sigma, ensuite appliquer l'lagorithme en utilisant le principe de réfutation et celui de la dichotomie.

Après avoir calculé le r, nous avons extrait les clauses répondant à la condition (dans cons) et les avons unifier avec nonphi.

Pour le test de consistance nous avons utilisé le solver ***pycosa,*** qui est fourni par **Python.** Dans le cas de consistance, le solver retourne une liste ; dans le cas contraire il retourne une chaine de caractère. Donc nous avons testé le type de retour pour avoir la réponse du test.

```
def algo(nbrvar,nbrcls,longcls):
    phi=[5,-2,3]
    nonphi=[]
    print("****Les clauses pondérées:****")
    for i in range(len(phi)):
        nonphi.append(-1*phi[i])
    basec=sigma(nbrvar,nbrcls,longcls)
    basec.sort()
    basec.strates()
    for i in range (len(basec.clauses)):
        print(basec.clauses[i],basec.poids[i])

    while basec.l<basec.u:
        cons=[]
        r=int((basec.l+basec.u+1)/2)
        for i in range(len(basec.clauses)):
            if(basec.poids[i]<basec.poidsstrates[r]):
                cons.append(basec.clauses[i])
        cons.append(nonphi)
        print(cons)
        if(type(pycosat.solve(cons))==type([])):
            basec.u=r-1
        else:
            basec.l=r
    print( "***Val(",str(phi),",Sigma)=",basec.poidsstrates[r],"***")
```

S'il y a consistance nous mettons à jour le u en le décalant sinon nous mettons à jour le l en l'avançant ; ceci jusqu'à violation de la condition principale.

Calcul de la variable d'interêt :

Pour le déroulement, nous avons pris une base à 10 clauses avec des poids associés. La figure suivante montre le résultat obtenu pour phi=[5,-2,3]

```
****Les clauses pondérées:****
[-5, 2, 3]
[-3, 1, 5] 0.9683456907772509
[-2, -3, -4] 0.962186984690537
[-4, 3, 5] 0.9393565454409709
[3, 2, -3] 0.7123146836754335
[2, -3, -5] 0.6842577967738797
[3, 5, -4] 0.6566615949566051
[5, -1, -2] 0.5895543263047116
[2, -3, -4] 0.5730956661260046
[2, -5, 1] 0.4883335955840131
[-4, -3, -5] 0.13055945741728847
[[-3, 1, 5], [-2, -3, -4], [-4, 3, 5], [3, 2, -3], [2, -3, -5], [-5, 2, 3]]
[[-3, 1, 5], [-2, -3, -4], [-5, 2, 3]]
[[-3, 1, 5], [-5, 2, 3]]
***Val( [5, -2, -3] ,Sigma)= 0.9683456907772509 ***
```

Conclusion :

A travers ce TP, nous avons pu voir une petite application de la logique possibiliste qualitative et ceci en : implémentant un algorithme d'inférence, générant une basede connaissances pondérée et calculant à la fin la variable d'interêt, ceci à l'aide du langage Python et le solver *pycosa.*