In [6]:

```
import numpy as np

class Variable :
    name = ""
    type = ""
    value = []
    parents = []
    proba = []

    def __init__(self, name, arg):
        self.name = name
        self.type = arg[1]
        self.value = arg[2:]

    def addDepProba(self, par, arg):
        self.parents = par
        if self.parents == []:
            for val in arg[0][1:]:
                self.proba = self.proba + [float(val)]
        else:
            for i in range(len(arg)):
                self.proba = self.proba + [[arg[i][:len(self.parents)]] + [list(np.float_(arg[i][len(self.parents):]))]]

    def trouver_proba(self, bool = None, bool_par = []):
        if len(bool_par) > len(self.parents):
            print("Erreur : la liste de probabilite conditionnelle entree est trop grande")
            return

        while len(bool_par) < len(self.parents):
            bool_par = bool_par + [self.parents[len(bool_par)].value[0]]

        if self.parents == []:
            if(bool==True):
                return self.proba[0]
            else:
                return self.proba[1]


        if self.parents != []:
            for p in self.proba:
                if p[0] == bool_par:
                    if bool == None:
                        return p[1]
                    elif bool:
                        return p[1][0]
                    else:
                        return p[1][1]

        print("Erreur : les valeurs parentes entrees ne correpondent a aucun cas")

    def __repr__(self):
        return self.name

    def info(self):
        np=""
        for p in self.parents:
            np=np+p.name+", "
        return "Nom : "+self.name+" | Parents : "+np


class Network:
    name = ""
    liste_var = []
    prob = []

    def __init__(self, name):
        self.name = name

    def addVar(self, var):
        self.liste_var = self.liste_var + [var]

    def addP(self, dep, arg):
        var = None
        par = []
        for v in self.liste_var:
            if v.name == dep[0]:
                var = v
            elif v.name in dep:
                par = par + [v]

        var.addDepProba(par, arg)

    def addProb(self, arg):
        res = []
        indexMax = -1
        for i in range(len(self.liste_var)):
            val = self.liste_var[i].name
            if val not in arg:
                res = res + [None]
            elif val == arg[0]:
                indexMax = max(indexMax, 0)
                res = res + [self.liste_var[i]]
            else:
                indexMax = max(indexMax, arg.index(val) + 1)
                res = res + [arg[arg.index(val) + 1]]

        self.prob = [res] + [arg[indexMax + 1:]]

    def vByName(self, n):
        for l in self.liste_var:
            if(l.name==n):
                return l

    def __repr__(self):
        np="Le nom du réseau est : "+self.name+"\nVoici les variables : \n"
        for p in self.liste_var:
            np=np+p.info()+"\n"
        return np

    def varibales_certaines_b(self):
        L=[]
        tmp = self.prob[0]
        for i in range(len(tmp)):
            if(tmp[i]=='True' or tmp[i]=='true' or tmp[i]=='1'):
                L.append((self.liste_var[i],True))
            if(tmp[i]=='False' or tmp[i]=='false' or tmp[i]=='0'):
                L.append((self.liste_var[i],False))
        return L

    def varibales_certaines(self):
        L=[]
        tmp = self.prob[0]
        for i in range(len(tmp)):
            if(tmp[i]=='True' or tmp[i]=='true' or tmp[i]=='1'):
                L.append(self.liste_var[i])
            if(tmp[i]=='False' or tmp[i]=='false' or tmp[i]=='0'):
                L.append(self.liste_var[i])
        return L

    def target(self):
        for v in self.prob[0]:
```

```
            if isinstance(v,Variable):
                #print(v)
                return v

    # Renvoi la liste des variables dans la couverture de markov
    def enfants_v(self,variable):
        enfants=[]
        for v in self.liste_var:
            if(variable in v.parents):
                enfants.append(v)
        return enfants

    def femmes_v(self,variable):
        femmes=[]
        for v in self.liste_var:
            if(v==variable):
                continue
            inter_enfants = list(set( self.enfants_v(variable) ) & set( self.enfants_v(v) ))
            if( len(inter_enfants) != 0 ):
                femmes.append(v)
        return femmes

    def couverture_markov(self,v):
        return self.enfants_v(v) + self.femmes_v(v) + v.parents

    def proba_v_couverture_markov(self,v,v_bool,echantillon):
        bool_l_parent=[]
        for e in v.parents:
                index = echantillon[0].index(e)
                bool_l_parent.append(echantillon[1][index])
        proba= v.trouver_proba(v_bool,bool_l_parent)
        for tmp in self.couverture_markov(v):
            bool_l_parent=[]
            for e in tmp.parents:
                if(e==v):
                    bool_l_parent.append(str(v_bool).lower())
                else:
                    index = echantillon[0].index(e)
                    bool_l_parent.append(echantillon[1][index])
            index = echantillon[0].index(tmp)
            tmp_bool_str = echantillon[1][index]
            if (tmp_bool_str=="true"):
                tmp_bool=True
            if (tmp_bool_str=="false"):
                tmp_bool=False
            if (tmp_bool_str!="false" and tmp_bool_str!="true"):
                return
            proba= proba * tmp.trouver_proba(tmp_bool,bool_l_parent)
        return proba
```

In [7]:

```
import random as rd

def sample(p) :
    r = rd.uniform(0,1)
    tmp = 0
    for i in range(len(p)):
        tmp += p[i]
        if r <= tmp :
            return i

#on passe en parametre le network
def enchantillon_rejet(N):
    echantillon_valide=False
    #exemple : [ [Burglary,JohnCalls] , ['True','False']] nb : de meme tailles les 2 sous listes !
    while(echantillon_valide==False):
        echantillon=[[],[]]
        #on parcoure le réseau baysien
        for v in N.liste_var:
            #on regarde pour chaque variable ses parents tiré (si existe) avec leur valeur boleen
            #inter_parents_existant = list(set(v.parents) & set(echantillon[0]))

            bool_l_parent=[]
            for e in v.parents:
                index = echantillon[0].index(e)
                bool_l_parent.append(echantillon[1][index])
            #on fait un tirage entre True et False en fonction des parents tiré (si existe)
            tirage = sample([v.trouver_proba(True,bool_l_parent),v.trouver_proba(False,bool_l_parent)])

            if(tirage==0):
                tirage_bool=True
            if(tirage==1):
                tirage_bool=False

            #on rejette l'echantillon
            if(v in N.varibales_certaines() and (v,tirage_bool) not in N.varibales_certaines_b()):
                # On rejette l'echantillon car on a pas tiré en fonction des valeurs boolen des variables du sachant que
                echantillon_valide=False
                break

            if(v in N.varibales_certaines() and (v,tirage_bool) in N.varibales_certaines_b()):
                #on valide échantillon et on le rajoute car bonne valeur boolen des variables du sachant que
                echantillon[0].append(v)
                echantillon[1].append(str(tirage_bool).lower())
                echantillon_valide=True

            if(v not in N.varibales_certaines()):
                #on valide échantillon et on le rajoute
                echantillon[0].append(v)
                echantillon[1].append(str(tirage_bool).lower())
                echantillon_valide=True

        if (echantillon_valide):
            return echantillon

def prob_rej(rb, it):
    occu = []
    var = None
    for e in rb.liste_var:
        if e in rb.prob[0]:
            var = e
            occu = [0] * len(var.value)

    #on compte les occurences de chacune des valeurs dans les echantillons tires
    for i in range(it):
        ech = enchantillon_rejet(rb)
        occu[var.value.index(ech[1][ech[0].index(var)])] += 1

    proba = list(map(lambda x: x/it, occu))
    return proba
```

In [8]:

```
def enchantillon_weighted_sampling(N):
    # exemple : [ [Burglary,JohnCalls] , ['True','False'] , poids] nb : de meme tailles les 2 sous listes !
    echantillon=[[],[],1]
    # On parcoure le réseau baysien
    for v in N.liste_var:
        # on regarde pour chaque variable ses parents tiré ( si existe ) avec leur valeur boleen
        #inter_parents_existant = list(set(v.parents) & set(echantillon[0]))
        bool_l_parent=[]
        for e in v.parents:
            index = echantillon[0].index(e)
```

```
            bool_l_parent.append(echantillon[1][index])

        if(v in N.varibales_certaines()):
            index = N.varibales_certaines().index(v)
            bool_v = N.varibales_certaines_b()[index][1]
            echantillon[0].append(v)
            echantillon[1].append(str(bool_v).lower())
            p = v.trouver_proba(bool_v,bool_l_parent)
            echantillon[2] = echantillon[2]*p

        if(v not in N.varibales_certaines()):
            tirage = sample([v.trouver_proba(True,bool_l_parent),v.trouver_proba(False,bool_l_parent)])
            if(tirage==0):
                tirage_bool=True
            if(tirage==1):
                tirage_bool=False
            if(tirage!=1 and tirage!=0 ) :
                print('erreur')
                print(tirage)
                print(v)
                print(inter_parents_existant)
                print(bool_l_parent)
                print(v.trouver_proba(True,bool_l_parent))
                print(v.trouver_proba(False,bool_l_parent))
            echantillon[0].append(v)
            echantillon[1].append(str(tirage_bool).lower())

    return echantillon

def prob_ws(rb, it):
    poids = []
    var = None
    tot = 0
    for e in rb.liste_var:
        if e in rb.prob[0]:
            var = e
            occu = [0] * len(var.value)

    #on somme les poids de chaque valeur
    for i in range(it):
        ech = enchantillon_weighted_sampling(rb)
        occu[var.value.index(ech[1][ech[0].index(var)])] += ech[2]
        tot += ech[2]

    proba = list(map(lambda x: x/tot, occu))
    return proba
```

In [9]:

```
def enchantillon_gibbs(N,nb):
    # pense à verifier pour l etat de la target avant le sachant que
    target = N.target()
    #print(target)
    echantillon=[[],[]] # premier 0 : nb de target true et le deuxième nb target faux
    compte=[0,0]
    #Initialisation
    for t in N.varibales_certaines_b():
        echantillon[0] = echantillon[0] + [t[0]]
        echantillon[1] = echantillon[1] + [str(t[1]).lower()]
    for t in N.liste_var:
        if t in N.varibales_certaines():
            continue
        else:
            echantillon[0] = echantillon[0] + [t]
            echantillon[1] = echantillon[1] + ['true']
    for i in range(nb):
        for v in N.liste_var:
            if v in N.varibales_certaines():
                continue
            p_v_vrai = N.proba_v_couverture_markov(v,True,echantillon)
            p_v_faux = N.proba_v_couverture_markov(v,False,echantillon)
            p_vv_vrai = p_v_vrai / ( p_v_vrai + p_v_faux )
            p_vv_faux = p_v_faux / ( p_v_vrai + p_v_faux )
            S=p_vv_vrai + p_vv_faux
            if (S != 1):        #parce que S = 0,9999999999
                p_vv_vrai = 1 - p_vv_faux
            tirage = sample([p_vv_vrai,p_vv_faux])
            if(tirage==0):
                 tirage_bool='true'
            if(tirage==1):
                tirage_bool='false'
            if(tirage!=1 and tirage!=0):
                print('ERREUR')
            index = echantillon[0].index(v)
            echantillon[1][index] = tirage_bool
            index = echantillon[0].index(target)
            booll = echantillon[1][index]
            if booll == 'true':
                compte[0]=compte[0]+1
            if booll == 'false':
                compte[1]=compte[1]+1
    somme = compte[0] + compte[1]
    compte[0]=compte[0]/somme
    compte[1]=compte[1]/somme
    return compte

def prob_gib(rb, it):
    return enchantillon_gibbs(rb,it)
```

In [ ]:

```
import matplotlib.pyplot as plt
import time

def lireBD(nom_bd):
    bd = open(nom_bd, "r")
    line = bd.readline().lower()
    rb = []

    while (line != ""):
        lineTab = line.translate(str.maketrans("{}[](),;|=\n", "           ")).split()

        if lineTab != []:
            if lineTab[0] == "network":
                rb = rb + [Network(lineTab[1])]

            elif lineTab[0] == "variable":
                line = bd.readline().lower()
                arg = line.translate(str.maketrans("{}[](),;|=\n", "           ")).split()

                var = Variable(lineTab[1], arg[1:])
                rb[len(rb)-1].addVar(var)

            elif lineTab[0] == "probability":
                dep = lineTab[1:]

                arg = []
                lineTab = [None]
                while lineTab != []:
                    line = bd.readline().lower()
                    lineTab = line.translate(str.maketrans("{}[](),;|=\n", "           ")).split()
                    if lineTab != []:
                        arg = arg + [lineTab]
```

```
                rb[len(rb)-1].addP(dep, arg)

            elif lineTab[0] == "prob":
                rb[len(rb)-1].addProb(lineTab[1:])

            else:
                print("Erreur de syntaxe dans la base de donnée")

        line = bd.readline().lower()
    bd.close()
    return rb

def comp_avec_rej(list_rb):
    err_rej = []
    exe_rej = []
    err_ws = []
    exe_ws = []
    err_gib = []
    exe_gib = []

    it = list(map(lambda x: (x + 1) * 1000, range(10)))

    for i in it:
        print(i/100-10, "%")

        res = err_exe_rej(list_rb, i)
        err_rej = err_rej + [res[0]]
        exe_rej = exe_rej + [res[1]]

        res = err_exe_ws(list_rb, i)
        err_ws = err_ws + [res[0]]
        exe_ws = exe_ws + [res[1]]

        res = err_exe_gib(list_rb, i)
        err_gib = err_gib + [res[0]]
        exe_gib = exe_gib + [res[1]]

    print("100.0 %")

    #on affiche les graphes respectivement pour l'erreur et pour le temps d'execution
    plt.plot(it, err_rej)
    plt.plot(it, err_ws)
    plt.plot(it, err_gib)
    plt.legend(["Rejet", "Weighted sampling", "Gibbs"])
    plt.xlabel("nb echantillon")
    plt.ylabel("erreur moyenne")
    plt.title("Erreur moyenne en fonction du nombre d'echantillons")
    plt.savefig("graphe\erreur_avec_rejet")
    plt.clf()

    plt.plot(it, exe_rej)
    plt.plot(it, exe_ws)
    plt.plot(it, exe_gib)
    plt.legend(["Rejet", "Weighted sampling", "Gibbs"])
    plt.xlabel("nb echantillon")
    plt.ylabel("temps moyen")
    plt.title("Temps moyen en fonction du nombre d'echantillons")
    plt.savefig("graphe\\temps_avec_rejet")
    plt.clf()

    print("Les graphiques avec rejet ont été générés et sauvegardés !\n")

def comp_sans_rej(list_rb):
    err_ws = []
    exe_ws = []
    err_gib = []
    exe_gib = []

    it = list(map(lambda x: (x + 1) * 10000, range(10)))

    for i in it:
        print(i/1000-10, "%")

        res = err_exe_ws(list_rb, i)
        err_ws = err_ws + [res[0]]
        exe_ws = exe_ws + [res[1]]

        res = err_exe_gib(list_rb, i)
        err_gib = err_gib + [res[0]]
        exe_gib = exe_gib + [res[1]]

    print("100.0 %")

    #on affiche les graphes respectivement pour l'erreur et pour le temps d'execution
    plt.plot(it, err_ws)
    plt.plot(it, err_gib)
    plt.legend(["Weighted sampling", "Gibbs"])
    plt.xlabel("nb echantillon")
    plt.ylabel("erreur moyenne")
    plt.title("Erreur moyenne en fonction du nombre d'echantillons")
    plt.savefig("graphe\erreur_sans_rejet")
    plt.clf()

    plt.plot(it, exe_ws)
    plt.plot(it, exe_gib)
    plt.legend(["Weighted sampling", "Gibbs"])
    plt.xlabel("nb echantillon")
    plt.ylabel("temps moyen")
    plt.title("Temps moyen en fonction du nombre d'echantillons")
    plt.savefig("graphe\\temps_sans_rejet")
    plt.clf()

    print("Les graphiques sans rejet ont été générés et sauvegardés !\n")

def err_exe_rej(list_rb, it):
    #premier de chaque liste est la somme des erreurs et le second la somme des temsp d'execution
    res = [0, 0]

    for rb in list_rb:
        start = time.time()
        sol = prob_rej(rb, it)
        end = time.time()
        res[0] += abs(float(rb.prob[1][0]) - sol[0])
        res[1] += end - start

    #on divise par le nb de rb dans la liste pour avoir les moyennes
    res = list(map(lambda x: x/len(list_rb), res))
    return res

def err_exe_ws(list_rb, it):
    #premier de chaque liste est la somme des erreurs et le second la somme des temsp d'execution
    res = [0, 0]

    for rb in list_rb:
        start = time.time()
        sol = prob_ws(rb, it)
        end = time.time()
        res[0] += abs(float(rb.prob[1][0]) - sol[0])
        res[1] += end - start

    #on divise par le nb de rb dans la liste pour avoir les moyennes
    res = list(map(lambda x: x/len(list_rb), res))
    return res
```

```
def err_exe_gib(list_rb, it):
    #premier de chaque liste est la somme des erreurs et le second la somme des temsp d'execution
    res = [0, 0]
    for rb in list_rb:
        start = time.time()
        sol = prob_gib(rb, it)
        end = time.time()
        res[0] += abs(float(rb.prob[1][0]) - sol[0])
        res[1] += end - start

    #on divise par le nb de rb dans la liste pour avoir les moyennes
    res = list(map(lambda x: x/len(list_rb), res))
    return res


#rb = lireBD("bn.bif")
#print(prob_gib(rb[0], 100000))

menu = True
while menu:
    print("Que voulez-vous faire ?")
    print("\t 0 : Approximer les probabilités d'un réseau bayésien binaire")
    print("\t 1 : Calculer l'erreur moyenne et le temps moyen d'exécution")
    print("\t 2 : Quitter")

    choix = int(input("Entrez 0, 1 ou 2 : "))
    while choix not in [0, 1, 2]:
        choix = int(input("Entrez 0, 1 ou 2 : "))

    if choix==0:
        print("\nEntrez le nom du fichier contenant le réseau bayésien binaire : ")
        fichier = (input(""))
        rb = lireBD(fichier)

        print("\nQuelle méthode voulre-vous utiliser ?")
        print("\t 0 : Rejet")
        print("\t 1 : Weighted sampling")
        print("\t 2 : Gibbs")

        choix_methode = int(input("Entrez 0, 1 ou 2 : "))
        while choix_methode not in [0, 1, 2]:
            choix_methode = int(input("Entrez 0, 1 ou 2 : "))

        it = int(input("\nNombre d'échantillon : "))

        if choix_methode == 0:
            start = time.time()
            res = prob_rej(rb[0], it)
            end = time.time()

        if choix_methode == 1:
            start = time.time()
            res = prob_ws(rb[0], it)
            end = time.time()

        if choix_methode == 2:
            start = time.time()
            res = prob_gib(rb[0], it)
            end = time.time()

        print("Erreur : ", abs(float(rb[0].prob[1][0]) - res[0]))
        print("Temps : ", end - start)
        print()

    if choix==1:
        print("\nEntrez le nom du fichier contenant les réseaux bayésiens binaires : ")
        fichier = input("")
        rb = lireBD(fichier)

        print("\nVoulez-vous utilisez le rejet (en plus du weighted sampling et de gibbs) ?")
        print("\t 0 : Avec rejet (Moyenne effectuée sur moins d'échantillons)")
        print("\t 1 : Sans rejet")

        choix_methode = int(input("Entrez 0 ou 1 : "))
        while choix_methode not in [0, 1]:
            choix_methode = int(input("Entrez 0 ou 1 : "))
        print()

        if choix_methode==0:
            comp_avec_rej(rb)

        if choix_methode==1:
            comp_sans_rej(rb)

    if choix==2:
        menu = False
```

```
Oue voulez-vous faire ?
        0 : Approximer les probabilités d'un réseau bayésien binaire
        1 : Calculer l'erreur moyenne et le temps moyen d'exécution
        2 : Quitter
```

In [ ]: