Rapport TP Data-mining MIV

Par : Mataoui Chakib Souleyman Le 14 Mai 2018 Pour : Mr.Necir Chaib Narimane

1-Introduction

Dans ce rapport nous allons présenter notre approche pour la résolution du problème d'extraction des items-set fréquents avec l'algorithme Close pour une base distribuée et notre outil implémentant cette méthode.

2-Développement

Choix du language

Nous avons opté pour l'utilisation du language C++ car nous avons préféré la performance physique du programme que permet l'utilisation du compilateur GNU GCC pour le C++ (g++) . Pour cela nous avons utilisé l'option d'optimisation du code objet maximal qu'offre le compilateur pour que le programme puisse avoir une vitesse optimale à l'exécution car dans ce genre de problème NP-Complet la performance temporelle est très cruciale.

Définition des méthodes outils

La première étape à consister à modéliser une classe qui contient une matrice de contexte d'extraction en représentation binaire en utilisant les valeurs booléennes qu'on extrait d'un fichier Csv qui contient des 1 et 0 qu'on transforme en vrai (1) ou faux (0).

```
context::context(char* context_file){
  string file_name(context_file);
  fstream filter(file_name);
  vector<string> grid;
  string line;
  //Lire les lignes du fichier
  while(getline(filter,line)){
    grid.push_back(line);
  }
  size_i = grid.size();
  //Déduire combien d'élements existent
  size_j = grid[0].length() - ((grid[0].length()-1)/2);
  //Extraction des elements
  for (std::size_t row = 0; row < size_i; ++row)
    {
      string res = grid[row];
      context_matrix.push_back(vector<bool>());
      for (std::size_t col = 0; col < size_j; ++col)
      {
        context_matrix[row].push_back(stoi(res.substr((col)*2,1)));
      }
    }
  return;
  //Affich part
  cout << "size i : " << size_i << " size_j : " << size_j << endl;
  for(int i = 0; i < size_i;++i){
    cout << i << " : " ;
    for(int j = 0; j < size_j;++j)
      cout << context_matrix[i][j] << " " ;
    cout << endl;
  }
}
```

Ensuite on a développé la méthode de calcul du support pour un item et en deux versions dont on n'en utilisera qu'une seule au final. (Le 2e été supposé etre utilisé pour la génération des règles d'association mais cela n'a pas abouti)

```
int context::support_simple(vector<int> cols){
  int nbr = 0;
  //On prend chaque element ou ils sont tous TRUE
  for(int i = 0; i < size_i; ++i){
    bool is_true = true;
    for(int j = 0; j < cols.size(); ++j){
      if(context_matrix[i][cols[j]] == false){
        is_true = false;
        break;
      }
    }
    if(is_true)
      nbr++;
  }
  //cout << nbr << "/" << size_i << endl;
  return nbr;
}

double context::support(vector<int> cols){
  int nbr = 0;
  //On prend chaque element ou ils sont tous TRUE
  for(int i = 0; i < size_i; ++i){
    bool is_true = true;
    for(int j = 0; j < cols.size(); ++j){
      if(context_matrix[i][cols[j]] == false){
        is_true = false;
        break;
      }
    }
    if(is_true)
      nbr++;
  }
  //cout << nbr << "/" << size_i << endl;
  return (double)nbr/size_i;
}
```

Algorithme Close

La prochaine étape fut de développer l'algorithme Close séquentiel qui a en entrée le "minsup" définie à l'appel du programme (voir "Programme Finale") et retourne un vecteur de vecteurs d'entiers ou chaque vecteur est un item et chaque element de ce vecteur est l'indice colonne d'un item dans le contexte d'extraction et cela sa passe par plusieurs étapes :

1-insérer les singletons dans le générateur

```
vector<vector<int>> context::close(double minsup){
  cout << "Demmarage close" << endl;
  //On demmare par les items singletons
  vector<vector<int>> generateurs;
  vector<int> ffc1;
  for(int j = 0; j < size_j;j++){
    generateurs.push_back(vector<int>());
    generateurs[j].push_back(j);
    ffc1.push_back(j);
  }
```

2-calculer le support de chaque item-set et éliminer ceux qui sont inférieurs au minsup . Car si un item-set et est infréquent tous les item-sets et qui le contient son inférquents donc il nous interesse pas par la suite.

```
  bool generer = false;
  vector<vector<int>> fermeture;
  cout << "Debut de génération " << endl;
  do{
    generer = false;

    for(int gen = 0; gen < generateurs.size(); ++gen){
      //calculer le sup
      vector<int> cols(generateurs[gen]);
      double sup = this->support(cols);
      if(sup < minsup){
        if(generateurs[gen].size() == 1){
          ffc1.erase(find(ffc1.begin(),ffc1.end(),generateurs[gen][0]));
        }

        generateurs.erase(generateurs.begin()+gen);
        gen--;
        continue;
      }
```

3-Puis on calcule la fermeture de chaque item-set restant et ont trie le résultat pour optimiser les recherches et comparaison entre vecteurs

```
      //calculer la fermeture
      vector<int> ferme(generateurs[gen]);
      for(int col = 0; col < size_j;++col){
        if(find(cols.begin(),cols.end(),col) != cols.end()){
          continue;
        }
        //On vérifie si on peut ajouter l'item a la fermeture ou non
        bool ajout = true;
        for(int i = 0; i < size_i;++i){
          bool verif = true;
          //On vérifie que les cases des items du meme generateurs sont 1
          for(int j = 0; j < cols.size();++j){
            if(context_matrix[i][cols[j]] == false){
              verif = false;
              break;
            }
          }
          //Verifier et continuer
          if(verif){
            if(context_matrix[i][col] == false)
              ajout = false;
          }

        }
        if(ajout)
          ferme.push_back(col);
      }
      //Trier le résultat par ordre alphabétique
      sort(ferme.begin(),ferme.end());
      if(find(fermeture.begin(),fermeture.end(),ferme) == fermeture.end())
        fermeture.push_back(ferme);
    }
```

4-On termine par créer les nouveaux générateurs en vérifiant que les item-set généré ne font pas partie la fermeture.

```
    //ajouter les nouveau generateur
    int fc_size = generateurs.size();
    for(int i = 0; i < fc_size;i++){
      //On copie le générateur
      vector<int> g(generateurs[i]);
      //On ajoute pour un générateur toutes les possibilités
      for(int j = 0; j < ffc1.size();j++){
        if(find(g.begin(),g.end(),ffc1[j]) != g.end())
          continue;
        g.push_back(ffc1[j]);
        sort(g.begin(),g.end());
        //On cherche si il n'existe pas déja dans une fermeture
        bool exist = false;
        for(int f = 0; f < fermeture.size();++f){
          bool found = true;
          for(int z = 0; z < g.size();++z)
            if(find(fermeture[f].begin(),fermeture[f].end(),g[z]) == fermeture[f].end())
              found = false;
          if(found)
            exist = true;
        }
        if(!exist){
          bool found = false;
          if(find(generateurs.begin(),generateurs.end(),g) == generateurs.end()){
            generateurs.push_back(g);
            generer = true;
          }
        }
        g.erase(find(g.begin(),g.end(),ffc1[j]));
      }
    }
    //On enleve le générateur qu'on a utiliser
    generateurs.erase(generateurs.begin(),generateurs.begin()+fc_size);
```

5-Tant qu'on a généré de nouvaux items on repart à l'étape 2

```
  }while(generer);

  return fermeture;
}
```

Méthode de distribution

Pour le développent de l'algorithme Close distribué nous avons fait en sorte que l'utilisateur décide du nombre de bases locales qui seront utilisés. Donc on effectue un découpage horizontal de la matrice équitablement suivant le nombre donné en entrée.

Pour cela nous avons créé une méthode qui permet de copier un certain intervalle de lignes d'un contexte dans un autre contexte.

```
context::context(context &mat,int deb,int fin){
  size_i = fin - deb ;
  size_j = mat.size_j;
  for(int i = deb; i < fin;++i)
    context_matrix.push_back(mat.context_matrix[i]);
}
```

Close Distribué

Pour l'algorithme Close distribué nous nous sommes inspirée de l'algorithme DFCIM (Distributed Frequent Closed Itemsets Mining)

Qui consiste à calculer la fermeture de chaque base locale avec l'algorithme Close puis de retourner le résultat à la base master qui contient le contexte global qui fait l'union de tous les items-set locaux et recalcule le support de chaque item-set mais cette fois par rapport à la matrice globale.

Pour l'implémentation de cette méthode nous avons considéré le multithreading ou chaque thread serait une base locale et le thread principal du programme la base master. Puis nous avons créé une méthode qui permet l'union de tous les items et ensuite calculer le support de chaque item générer et supprimer tous les items et dont le support global est inférieur au minsup.

```
//Thread qui calcul les close distribué locaux
void th(context c,int sup,vector<vector<int>> &res){
  res = c.close((double)sup/100);
}
```

```
//Creation des bases locals
for(int i = 0; i < NUM_THREAD;++i){
  vc.push_back(context(c,round(c.size_i/NUM_THREAD)*i,round(c.size_i/NUM_THREAD)*(i+1)));
  result.push_back(vector<vector<int>>());
}
//Lancement des thread
vector<thread> para;
for(int i = 0; i < NUM_THREAD;++i)
  para.push_back(thread(th,vc[i],atoi(argv[2]),ref(result[i])));

//Attente fin de thread
for(int i = 0; i < NUM_THREAD;++i)
  para[i].join();
//Affichage des fermetures locals
for(int t = 0; t < NUM_THREAD;++t){
  cout << "Liste fermetures locales : " << t << endl ;
  vector<vector<int>> fermeture(result[t]);
  for(int i = 0; i < fermeture.size();++i){
    sort(fermeture[i].begin(),fermeture[i].end());
    for(int j = 0; j < fermeture[i].size();++j)
      cout << (char)(fermeture[i][j]+65);
    cout << ",";
  }
  cout << endl;
}
```

```
//Union des résultats
vector<vector<int>> inter(result[0]);
for(int i = 1; i < NUM_THREAD;++i){
  inter = c.unif(inter,result[i]);
}

//Eleminer les support faibles
for(int i = 0; i < inter.size(); ++i){
  if(c.support(inter[i]) < (double)atoi(argv[2])/100){
    inter.erase(inter.begin()+i);
    i--;
  }
}
```

En ce qui concerne le multithreading étant donnée que chaque microprocesseur à des limites en matière de nombre de thread maximum qu'il peut utiliser. Dépasser le nombre de thread du processeur n'est plus un problème car l'architecture des systèmes d'exploitation moderne fait que c'est lui-même qui gère lorsqu'un programme exécute plus de thread que le processeur ne puisse en gérer et permet quand même de les exécuter parallèlement.

Programme finale

Le programme final consiste à être la base globale où l'on donne en entrée le fichier qui contient le contexte d'extraction en format .Csv et de donner le minsup en % et le nombre de bases locales qui vont être générées.

l'appel du programme sous linux se fait sous cette forme (le choix du système d'exploitation est arbitraire)

$ ./exe [CTX] [MINSUP] [BD_LCL]

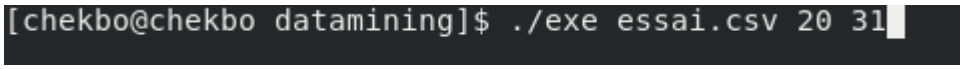
ou :

[CTX] est le nom du fichier .csv

[MINSUP] un chiffre ent11re 0 et 100

[BD_LCL] un chiffre qui varie entre 1 et le nombre maximal est le nombre de lignes du csv

Example :



L'affichage final affiche les items-set fréquents pour les deux méthodes séquentiel et distribué pour pouvoir être comparé :

```
,DEH,
Liste fermetures locales : 13
AE,BF,C,CD,E,F,G,FH,ABEFG,ACDE,AEF,AEG,BEFG,BFG,BEFGH,CF,CFH,CDF,EF,EG,
Liste fermetures locales : 14
A,B,C,D,E,F,EG,H,AD,AEFH,BD,CDE,CFH,CDEG,CH,DE,DEF,CDEH,EF,EH,FH,EGH,
Liste fermetures locales : 15
A,B,C,D,E,F,G,H,ACE,AD,AE,ADF,AEG,BCH,BDF,BH,CDF,CE,CG,CH,ADE,DF,DG,DH,EFH,EH,FH,BDFH,CDFH,CEH,DFH,
Liste fermetures locales : 16
AB,B,C,D,BE,BF,BFG,BCH,ABC,ABD,ABE,ABF,ABFG,BD,CD,BCE,BCF,BDE,BDF,BCEF,BCEH,BCFH,BCDE,
Liste fermetures locales : 17
ACF,B,C,DF,E,F,G,FH,ACDEFH,ACFG,BCG,BCDFG,BCEG,BF,BFH,CDF,CE,CG,DEFH,DFG,DFH,FG,DFGH,
Liste fermetures locales : 18
A,B,C,D,BEH,F,G,BH,ABEH,AC,ABDEH,AF,AG,BFG,BG,CD,CF,DF,ABDEFGH,BEFGH,BEGH,BGH,
Liste fermetures locales : 19
A,B,CD,D,E,EF,GH,H,ABCD,ACD,AE,AGH,AH,BCD,BD,BEH,BDEFH,BEGH,CDEGH,DEH,EGH,EH,AEH,
Liste fermetures locales : 20
A,B,CE,D,E,AF,G,ADH,AB,ACE,AE,AG,BD,ABE,ABDF,BG,ACDE,CEG,ADE,ADF,DG,AEF,EG,ADFH,AEG,ADEF,
Liste fermetures locales : 21
A,BF,AC,D,E,F,G,H,ABF,ADH,AE,AF,AG,AH,BEF,ABFG,BFH,ACE,ACG,DF,DG,EH,FG,FH,AGH,ABEF,ABFH,AEH,AFH,BEFH,
Liste fermetures locales : 22
AE,B,C,D,E,F,G,H,ABCE,ACE,ACDE,ACEG,BC,BDE,BE,BG,BH,CD,CDF,CG,BCGH,DE,DF,DG,DH,BEH,FG,
Liste fermetures locales : 23
A,B,C,D,E,F,H,AB,AF,BCH,BDH,BE,BF,BH,CD,CF,DEFH,DF,DH,EF,EH,FH,BEF,BEH,BFH,
Liste fermetures locales : 24
A,B,C,CD,CDEH,CF,G,H,AB,AC,AG,AH,BC,BCD,BCDFH,BCDG,BH,CG,CDFH,CDG,CFG,CDFGH,
Liste fermetures locales : 25
A,B,C,BD,E,EF,BG,BEH,AB,ABD,AE,AEF,ABEH,BC,BEFH,CEF,BDEH,BDG,BEGH,
Liste fermetures locales : 26
A,B,C,D,ACE,F,G,CGH,AB,AD,AF,AG,ACEGH,BC,ABCEF,BF,BCG,BCFGH,CD,CF,DF,DG,CDGH,FG,CFGH,ABCG,ADG,AFG,
Liste fermetures locales : 27
A,B,C,D,E,F,DGH,ABC,AC,AD,AE,AF,ADGH,BC,BD,BCDEF,BCF,BDGH,CDF,CF,CDFGH,DE,DEGH,ABCDF,ABCF,ACDF,ACF,ACDFGH,ADE,BCDFGH,
Liste fermetures locales : 28
A,B,C,D,E,F,G,H,ABE,AC,AD,AFG,AG,BDG,BE,BEF,BG,CD,CE,CF,CFG,DE,CDF,DG,EF,ABEG,FG,
Liste fermetures locales : 29
A,B,BCF,D,E,F,G,AH,AB,ABCF,ABCDF,AEF,AF,ABG,BD,BE,ABH,ABCEF,ABCDEFG,DE,DF,DEFG,EF,AFH,
Liste fermetures locales : 30
A,B,C,D,E,F,GH,H,ABCF,AEFH,AF,AGH,AH,BCF,BD,BE,BF,BFH,CD,CEH,CF,DE,BDF,DGH,EGH,EH,FGH,FH,
Demmarage close
Debut de génération
++++++++++++++++++Liste fermetures global :
A,AB,AC,AD,AE,AF,AG,AH,B,BC,BD,BE,BF,BG,BH,C,CD,CE,CF,CG,CH,D,DE,DF,DG,DH,E,EF,EG,EH,F,FG,FH,G,GH,H,
++++++++++++++++++Liste fermetures lfinal :
A,AB,AC,AD,AE,AF,AG,AH,B,BC,BD,BE,BF,BG,BH,C,CD,CE,CF,CG,CH,D,DE,DF,DG,DH,E,EF,EG,EH,F,FG,FH,G,GH,H,
```

Note : Liste fermetures « global « représente les fermetures calculées avec close séquentiel et « lfinal » ceux avec Close distribué

Critique

Après expérimentation nous nous sommes rendu compte des faiblaisse du découpage de cette méthode. Le nombre de bases locales qu'on peut diviser doit être précis il doit permettre la distribution totale et équitable de la base globale.

La diviser en 1 revienderai a utilisé Close séquentielle
La diviser au Max qui est que chaque ligne devient une sous base donneront des résultats erronées et diviser en interséquant 2 bases c'est-à-dire que 2 bases aient 1 ou plusieurs lignes en commun aussi à un impact sur l'erreur des résultats

Par example ici on a diviser par 310 qui est le maximum de lignes

```
++++++++++++++++++Liste fermetures global :
A,AB,AC,AD,AE,AF,AG,AH,B,BC,BD,BE,BF,BG,BH,C,CD,CE,CF,CG,CH,D,DE,DF,DG,DH,E,EF,EG,EH,F,FG,FH,G,GH,H,
++++++++++++++++++Liste fermetures lfinal :
A,AB,AC,AE,AF,AG,AH,B,BC,CD,CE,CF,CG,D,DG,DH,E,EH,
[chekbo@chekbo datamining]$ ./exe essai.csv 20 310
```

Note : Liste fermetures « global « représente les fermetures calculées avec close séquentiel et « lfinal » ceux avec Close distribué

Un bon découpage serait par exemple pour une base de 100 lignes de diviser par 10 ou une base de 310 lignes par 31 ou 10...etc. Tant que chaque sous-base ait un découpage complet et équitable

3-Conclusion

Cette méthode lorsqu'elle est bien executé permet de réduire le nombre d'items-set généré a calculer et le nombre d'itération a accomplir pour pouvoir obtenir la fermeture. Elle pourait encore etre améliorer pour etre utiliser dans un système distribué en réseau ou chaque machine pour des données extrememement grandes executerai récursivement le meme procédé. Cependant l'arrivée des processeurs quantiques va considérablement réduire le temps de calcule des fermetures au point ou le nombres de données a traiter va décupler.

Bibliographie

Chun Liu, Zheng Zheng, Kai-Yuan Cai, Shichao Zhang 2008. Distributed Frequent Closed Itemsets Mining. DOI: 10.1109/SITIS.2007.64 · Source: IEEE Xplore