Titouan Guerin - DAC

IAMSI TME1 COMPTE RENDU

Organisation d'un championnat:

Modélisation:

La formule du nombre total de variables propositionnelles est le produit du nombre total de jours nj et du nombre total d'équipes ne.
Cela garantit que chaque combinaison de chaque équipe et chaque match de la journée est représenté. nombre de variables = nj * ne**2

La fonction d'encoage permet de représenter de manière unique chaque combinaison de jour et de match entre deux équipes.
La fonction de décodage permet de retrouver les informations associées à une variable propositionnelle donnée (l'indice du jour et les deux équipes).

```
def codage(ne, nj, j, x, y):
    return j * ne**2 + x * ne + y + 1

def decodage(k, ne):
    y = (k - 1) % ne
    x = ((k - 1 - y) // ne) % ne
    j = (k - 1 - y - x * ne) // (ne * ne)
    return j, x, y
```

Test pour verifier si l'encodage et le decodage fonctionne bien:

```
print(f'Test codage/decodage: {decodage(encoded, ne) == (j,x,y)}')
```

Génération d'un planning de matchs:

Fonctions pour gérer les contraintes de cardinalités:
Les fonctions **au_moins_k** et **au_plus_k** sont des versions plus poussées de **au_moins_un_vrai** et **au_plus_un_vrai.**
au_plus_k: pour chaque nombre i de variables vraies, elle génère toutes les permutations de ces i variables, crée une clause pour chaque permutation et l'ajoute à la liste des clauses.
au_moins_k: elle convertit toutes les variables de var en leur négation, puis appelle la fonction au_plus_k avec cette liste de variables négatives et la valeur len(var) - k (pour obtenir au moins k variables parmi celles de var sont vraies)

```
def au_moins_un_vrai(variables):
    cl = [str(var) for var in variables] + ["0"]
    return [" ".join(cl)]

def au_plus_un_vrai(variables):
    clauses = []
    n = len(variables)
```

```
    for i in range(n):
        for j in range(i + 1, n):
            clauses.append(f"{-variables[i]} {-variables[j]} 0")
    return clauses
```

-La fonction **encoderC1** parcourt chaque équipe et chaque jour du championnat, pour generer les contraintes nécessaires à l'aide de la fonction **au_plus_un_vrai**, pour s'assurer qu'au plus un match est joué par l'équipe sur une journée donnée.

-La fonction **encoderC2** est une variante de encoderC1, car elle utilise et **au_moins_un_vrai** et aussi **au_plus_un_vrai**. Les contraintes assurent qu'au moins un match est joué entre les deux équipes et qu'au plus un match est joué entre elles.

-La fonction **encoderC3** calcule le nombre minimum de matchs à domicile et à l'extérieur en fonction des pourcentages donnés (par défault fixés a 50% et 40%), puis parcourt chaque paire d'équipes distinctes pour générer les contraintes nécessaires à l'aide de la fonction **au_moins_k**, pour être sur qu'au moins un pourcentage spécifié de matchs sont joués à domicile ou à l'extérieur les dimanches (considéré comme le deuxième jour).

-La fonction **encoderC4** parcourt chaque équipe (pour chaque jour) et génère les contraintes nécessaires à l'aide de la fonction **au_plus_k**, qui assure qu'au plus deux matchs consécutifs sont joués à domicile ou à l'extérieur.

```
def encoderC1(ne, nj):
    contraintes_C1 = []
    # Pour chaque équipe et chaque jour
    for jour in range(nj):
        for equipe in range(ne):
            home = [codage(ne, nj, jour, equipe, adversaire) for
adversaire in range(ne) if adversaire != equipe]
            ext = [codage(ne, nj, jour, adversaire, equipe) for adversaire
in range(ne) if adversaire != equipe]

            contraintes_C1.extend(au_plus_un_vrai(home + ext))
#dimacs_f(clause)
    return contraintes_C1
def encoderC2(ne, nj):
    clauses = []
    for x in range(ne):
        for y in range(ne):
            if x != y:
                list_match = [codage(ne, nj, j, x, y) for j in range(nj)]
                clauses.extend(au_moins_un_vrai(list_match))
                clauses.extend(au_plus_un_vrai(list_match))
```

```
    return clauses

def encoderC3(ne,nj,exterieur=0.5,domicile=0.4):
    contraintes = []
    nb_min_dom = nj * ne * domicile // 100
    nb_min_ext = ne * nj * exterieur // 100
    for i in range(ne):
        for j in range(ne):
            if i!=j:
                domicile_c = [codage(ne, nj, y, i, j) for y in range(1,
nj, 2)]
                contraintes.extend(au_moins_k(domicile_c,nb_min_dom))
                exterieur_c = [codage(ne, nj, y, j, i) for y in range(1,
nj, 2)]
                contraintes.extend(au_moins_k(exterieur_c,nb_min_ext))
    return contraintes

def encoderC4(ne,nj):
    contraintes = []
    for i in range(nj):
        for x in range(ne):
            domicile_c = [codage(ne, nj, i, x, y) for y in range(ne) if
y!=x]
            contraintes.extend(au_plus_k(domicile_c,2))
            exterieur_c = [codage(ne, nj, i, y, x) for y in range(ne) if
y!=x]
            contraintes.extend(au_plus_k(exterieur_c,2))
    return contraintes
```

Une fois toutes les contraintes spécifiées, on utilise une fonction **encoder** pour gérer la création des contraintes pour le planning.
La fonction prends comme argument optionnel ex_exo5 pour savoir si on prends en compte ou non les contraintes 3 et 4.

```
def encoder(ne, nj, ext_exo5=None):
    contraintes = encoderC1(ne, nj) + encoderC2(ne, nj)
    if ext_exo5 is not None:
        contraintes.extend(encoderC3(ne, nj)+ encoderC4(ne, nj))
    with open("championnat.cnf", "w") as f:
        f.write(f"p cnf {ne**2 * nj - 1} {len(contraintes)}\n")
        # Ecrire les contraintes
```

```
        for contrainte in contraintes:
            f.write(contrainte + "\n")
    return contraintes
```

Ensuite, la fonction **decoder** décode les variables SAT pour obtenir le planning des matchs. Elle prend notamment en entrée le fichier contenant la sortie de l'exécution de l'algorithme SAT, glucose, et optionnelement un fichier contenant les noms des équipes (pour ce projet les équipes seront des équipes de basket de la NBA).

```
def decoder(output_file, ne, nj, team_names_file=None):
    if team_names_file != None:
        with open(team_names_file, "r") as f:
            team_names = [line.strip() for line in f]
    else:
        team_names = None

    with open(output_file, "r") as f:
        output_lines = f.readlines()

    output = output_lines[0].split()

    if "UNSAT" in output: return "UNSAT"

    planning = {jour+1: [] for jour in range(nj)}
    for var in output:
        if int(var) > 0:
            j,x,y=decodage(int(var),ne)
            if team_names != None:
                planning[j+1].append((team_names[x], team_names[y]))
            else: planning[j+1].append((x,y))

    return planning
```

La fonction call_glucose permet d'exécuter le solver avec le temps limite spécifié (normalement 10 secondes par défault). On utilise le schéma try-except pour essayer de récuperer l'output de glucose, et on a deux except pour repérer les erreurs du au timeout ou a tout autre erreur qui pourrait se produire.

```
def call_glucose(glucose, timeout):
    try:
        # Run the command with a timeout
        return subprocess.run(glucose, shell=True, timeout=timeout,
capture_output=True)
    except subprocess.TimeoutExpired:
```

```
        print("glucose execution timed out")
        return None
    except Exception as e:
        print(f"Error while running glucose: {e}")
        return None
```

Une fonction additionelle a été écrite pour pouvoir mieux visualiser les équipes et les matchs pour chaque jour:

```
def joli_affichage(planning):
    print("Schedule:")
    for jour, matches in planning.items():
        print(f"Jour {jour}:")
        for match in matches:
            print(f"  Equipe {match[0]} vs Equipe {match[1]}")
```

Enfin, la fonction **optimisation** a pour but de trouver le plus petit nombre de jours nécessaire pour un certain nombre d'équipes, et qui satisfait toutes les contraintes (C1 à C4).
Elle itère sur le nombre de jours nj de nj_min à nj_max, en appelant la fonction encoder pour générer les contraintes correspondantes. Ensuite, elle utilise la fonction call_glucose pour exécuter le solveur SAT (avec un timeout de 10 secondes). Si une solution est trouvée, la fonction utilise decoder pour décoder la solution, et si cette solution est satisfiable, alors on affiche le planning des matchs et on renvoie le nombre de jours nécessaire pour cette solution.

```
def optimisation(ne,nj_min,nj_max,timeout, extension=None):
    nj = nj_min
    not_found = True
    while nj <=nj_max and not_found:
        contraintes = encoder(ne,nj,extension)
        print(f'execution of {nj} days -')
        output = call_glucose('./glucose championnat.cnf output.cnf',
timeout)
        if output is not None:
            print(output)
            planning = decoder('output.cnf', ne, nj, 'equipes.txt')
            if planning == "UNSAT":
                nj+=1
            else:
                print("FOUND CORRECT NJ", nj)
                joli_affichage(planning)
                not_found = False
                return nj
```

```
        else:
            nj+=1
    return False
```

Execution du programme

Voici une trace d'éxecution pour 8 équipes, un nombre minimum de jours de 10 et un maximum de 20.
Les equipes sont: Atlanta Hawks, Boston Celtics, Brooklyn Nets, Charlotte Hornets, Chicago Bulls, Cleveland Cavaliers, Dallas Mavericks, Denver Nuggets
Remarque, la contrainte C4 fait bugger le solver et je n'ai pas eu le temps de le debugger donc la trace d'éxecution se fait sur les contraintes C1, C2 et C3.
Ici, on trouve que le planning peut se faire en 14 jours avec 4 matchs par jour.

```
titouan@DESKTOP-GCQA3UV://home/titouan/glucose-main/glucose/simp$ python3
projet.py
execution of 10 days -
glucose execution timed out
execution of 11 days -
glucose execution timed out
execution of 12 days -
glucose execution timed out
execution of 13 days -
glucose execution timed out
execution of 14 days -
CompletedProcess(args='./glucose championnat.cnf output.cnf',
returncode=10, stdout=b'c\nc This is glucose 4.2.1 --  based on MiniSAT
(Many thanks to MiniSAT team)\nc\nc
========================================[ Problem Statistics
]=========================================\nc |
|\nc |  Number of variables:           895
|\nc |  Number of clauses:           15344
|\nc |  Parse time:                   0.00 s
|\nc |
|\nc | Preprocesing is fully done\nc |  Eliminated clauses:           0.00
Mb                                                                  |\nc |
Simplification time:          0.01 s
|\nc |
|\nc ========================================[ MAGIC CONSTANTS
]==========================================\nc | Constants are supposed
to work well together :-)
|\nc | however, if you find better choices, please let us known...
```

```
|\nc
|-------------------------------------------------------------------------
---------------------------|\nc | Adapt dynamically the solver after
100000 conflicts (restarts, reduction strategies...)               |\nc
|-------------------------------------------------------------------------
---------------------------|\nc |                                    |
|                                    |\nc | - Restarts:
| - Reduce Clause DB:            | - Minimize Asserting:               |\nc
|   * LBD Queue    :     50      |   * First     :   2000         |    *
size <  30                     |\nc |   * Trail  Queue :   5000      |   *
Inc       :    300         |    * lbd  <   6                     |\nc |   *
K            :   0.80      |   * Special   :   1000         |
|\nc |   * R            :   1.40      |   * Protected :  (lbd)< 30     |
|\nc |                                |                                |
|\nc ================================[ Search Statistics (every  10000
conflicts) ]=========================\nc |
|\nc |          RESTARTS           |          ORIGINAL         |
LEARNT              | Progress |\nc |       NB   Blocked  Avg Cfc |    Vars
Clauses Literals |   Red   Learnts    LBD2  Removed |          |\nc
=========================================================================
============================\nc last restart ## conflicts  :  98 113 \nc
=========================================================================
============================\nc restarts              : 3 (852 conflicts
in avg)\nc blocked restarts      : 0 (multiple: 0) \nc last block at
restart : 0\nc nb ReduceDB           : 1\nc nb removed Clauses    : 998\nc
average learnt size   : 54\nc nb learnts DL2        : 0\nc nb learnts size
2     : 0\nc nb learnts size 1     : 0\nc conflicts             : 2556
(41443 /sec)\nc decisions             : 4008          (0.00 % random)
(64986 /sec)\nc propagations          : 65529          (1062489 /sec)\nc nb
reduced Clauses    : 0\nc LCM                  : 373 / 730 \nc CPU time
: 0.061675 s\n\ns SATISFIABLE\nSAT\n', stderr=b'')
FOUND CORRECT NJ 14
Schedule:
Jour 1:
  Equipe Brooklyn Nets vs Equipe Dallas Mavericks
  Equipe Charlotte Hornets vs Equipe Boston Celtics
  Equipe Chicago Bulls vs Equipe Cleveland Cavaliers
  Equipe Denver Nuggets vs Equipe Atlanta Hawks
Jour 2:
  Equipe Atlanta Hawks vs Equipe Dallas Mavericks
  Equipe Boston Celtics vs Equipe Cleveland Cavaliers
  Equipe Brooklyn Nets vs Equipe Charlotte Hornets
  Equipe Chicago Bulls vs Equipe Denver Nuggets
```

```
Jour 3:
  Equipe Atlanta Hawks vs Equipe Charlotte Hornets
  Equipe Boston Celtics vs Equipe Dallas Mavericks
  Equipe Chicago Bulls vs Equipe Brooklyn Nets
  Equipe Denver Nuggets vs Equipe Cleveland Cavaliers
Jour 4:
  Equipe Brooklyn Nets vs Equipe Chicago Bulls
  Equipe Cleveland Cavaliers vs Equipe Boston Celtics
  Equipe Dallas Mavericks vs Equipe Atlanta Hawks
  Equipe Denver Nuggets vs Equipe Charlotte Hornets
Jour 5:
  Equipe Boston Celtics vs Equipe Atlanta Hawks
  Equipe Chicago Bulls vs Equipe Charlotte Hornets
  Equipe Cleveland Cavaliers vs Equipe Brooklyn Nets
  Equipe Dallas Mavericks vs Equipe Denver Nuggets
Jour 6:
  Equipe Atlanta Hawks vs Equipe Brooklyn Nets
  Equipe Charlotte Hornets vs Equipe Chicago Bulls
  Equipe Cleveland Cavaliers vs Equipe Denver Nuggets
  Equipe Dallas Mavericks vs Equipe Boston Celtics
Jour 7:
  Equipe Boston Celtics vs Equipe Charlotte Hornets
  Equipe Brooklyn Nets vs Equipe Denver Nuggets
  Equipe Chicago Bulls vs Equipe Dallas Mavericks
  Equipe Cleveland Cavaliers vs Equipe Atlanta Hawks
Jour 8:
  Equipe Charlotte Hornets vs Equipe Atlanta Hawks
  Equipe Chicago Bulls vs Equipe Boston Celtics
  Equipe Cleveland Cavaliers vs Equipe Dallas Mavericks
  Equipe Denver Nuggets vs Equipe Brooklyn Nets
Jour 9:
  Equipe Boston Celtics vs Equipe Chicago Bulls
  Equipe Brooklyn Nets vs Equipe Atlanta Hawks
  Equipe Charlotte Hornets vs Equipe Denver Nuggets
  Equipe Dallas Mavericks vs Equipe Cleveland Cavaliers
Jour 10:
  Equipe Atlanta Hawks vs Equipe Denver Nuggets
  Equipe Brooklyn Nets vs Equipe Boston Celtics
  Equipe Charlotte Hornets vs Equipe Dallas Mavericks
  Equipe Cleveland Cavaliers vs Equipe Chicago Bulls
Jour 11:
  Equipe Atlanta Hawks vs Equipe Chicago Bulls
  Equipe Brooklyn Nets vs Equipe Cleveland Cavaliers
```

```
  Equipe Dallas Mavericks vs Equipe Charlotte Hornets
  Equipe Denver Nuggets vs Equipe Boston Celtics
Jour 12:
  Equipe Atlanta Hawks vs Equipe Cleveland Cavaliers
  Equipe Boston Celtics vs Equipe Denver Nuggets
  Equipe Charlotte Hornets vs Equipe Brooklyn Nets
  Equipe Dallas Mavericks vs Equipe Chicago Bulls
Jour 13:
  Equipe Boston Celtics vs Equipe Brooklyn Nets
  Equipe Chicago Bulls vs Equipe Atlanta Hawks
  Equipe Cleveland Cavaliers vs Equipe Charlotte Hornets
  Equipe Denver Nuggets vs Equipe Dallas Mavericks
Jour 14:
  Equipe Atlanta Hawks vs Equipe Boston Celtics
  Equipe Charlotte Hornets vs Equipe Cleveland Cavaliers
  Equipe Dallas Mavericks vs Equipe Brooklyn Nets
  Equipe Denver Nuggets vs Equipe Chicago Bulls
pour ne=8, nj_min=10, nj_max=20, le nj optimal est 14
```