The definition variables

```
onference_sessions = 40
slots = 7
papers_range = np.arange(3, 7)
max_parallel_sessions = 11
working_groups = 20
npMax = {1: 4, 2: 6, 3: 6, 4: 4, 5: 4, 6: 5, 7: 3}

# list of groups session groups
session_groups = [
    [1], [2], [3], [], [], [], [6], [7], [7, 8], [10], [8], [8, 11], [5,
8],
    [3, 8], [7], [13], [13], [14], [], [13], [16], [16], [20], [17], [13],
    [], [9], [11], [11, 12], [9], [6, 19], [], [], [18], [10], [5], [16],
    [4, 5], [8, 12], [7, 15]
]
```

function to generate unique identifier

```
def var_x(s, c, l):
    s_index = conference_sessions * slots * len(papers_range)
    c_index = slots * len(papers_range)
    l_index = len(papers_range)
    return s_index - (conference_sessions - s) * c_index - (slots - c) *
l_index - (len(papers_range) - l)

max_var_x = var_x(conference_sessions, slots, papers_range[-1])

def var_z(s, c):
    z_offset = max_var_x + 1  # Start after the last x variable
    return z_offset + (s - 1) * slots + c

max_var_z = var_z(conference_sessions, slots)

def var_y(s1, s2, c, g):


    y_offset = max_var_z + 1

    # Calculate the unique identifier for y variables
    unique_index = ((s1 - 1) * conference_sessions + (s2 - 1)) * slots *
working_groups + (c - 1) * working_groups + (g - 1)
    return y_offset + unique_index
```

The first constraint

1. At most one amount of papers chosen for a (session, slot) pair:

```
\sum_{l \in L} x_{(s,c,l)} \leq 1 \quad \forall (s,c) \in S \times C
```

implementation

```
for s in range(1, conference_sessions + 1):
    for c in range(1, slots + 1):
        vars_for_s_c = [var_x(s, c, l) for l in papers_range]
        # en peut le modifier
        amo_clause = CardEnc.atmost(lits=vars_for_s_c, bound=1,
encoding=EncType.pairwise)
        constraints.extend(amo_clause.clauses)
```

The second constraint

2 - The subdivision of a session into slots covers all the papers in the session: $$ \sum_{\substack{c\in C\ l\in L}} x_{(s,c,l)}*l = np(s) \qquad \forall s\in S$$

implementation

```
for s in range(1, conference_sessions + 1):
    aux_vars = []
    weights = []

    for c in range(1, slots + 1):
        for l in papers_range:
            aux_vars.append(var_x(s, c, l))
            weights.append(l)


    equals_clause = PBEnc.equals(lits=aux_vars, weights=weights,
bound=session_papers[s])
    constraints.extend(equals_clause.clauses)
```

The third constraint

3 - The subdivision respects the maximum length of each slot: %$$npMin(c) \leq x_{(s,c,l)}*l \leq npMax(c)\qquad \forall (s,c,l)\in S\times C\times L$$ $$ x_{(s,c,l)}*l \leq npMax(c)\qquad \forall (s,c,l)\in S\times C\times L$$

implementation

```
# 3 eme constraint

for s in range(1, conference_sessions + 1):
    for c in range(1, slots + 1):
        for l in papers_range:
            if l > npMax[c]:
               # j'ai travailler avec la conjunction des negation de x if
l>npMax(c)
                constraints.append([-var_x(s, c, l)])
```

Conversion of the Equivalence into Conjunctive Normal Form (CNF)

\ $$z_{(s,c)}\Leftrightarrow \bigwedge_{l\in L} \overline{x_{(s,c,l)}} \qquad \forall (s,c)\in S\times C$$

$$ \left(z_{(s,c)} \Longrightarrow \bigwedge_{l\in L} \overline{x_{(s,c,l)}} \right) \wedge \left(\bigwedge_{l\in L} \overline{x_{(s,c,l)}} \Longrightarrow z_{(s,c)} \right) \quad \forall (s,c)\in S\times C $$

$$\left(\overline{z_{(s,c)}} \vee \left(\bigwedge_{l\in L} \overline{x_{(s,c,l)}}\right) \right)\wedge \left(\overline{\left(\bigwedge_{l\in L} \overline{x_{(s,c,l)}}\right)} \vee z_{(s,c)} \right) $$ \textbf{The final CNF formula}\ $$\left(\overline{z_{(s,c)}} \vee \overline{x_{(s,c,l)}}\right) \wedge \left(\overline{z_{(s,c)}} \vee \overline{x_{(s,c,l)}}\right)\wedge ..... \wedge \left(\bigvee_{l\in L} x_{(s,c,l)} \vee z_{(s,c)}\right)$$

implementation

```
for s in range(1, conference_sessions + 1):
    for c in range(1, slots + 1):
        z_var = var_z(s, c)
        x_vars = [var_x(s, c, l) for l in papers_range]


        for x in x_vars:
            constraints.append([-z_var, -x])

        or_clause = [-x for x in x_vars] + [z_var]
        constraints.append(or_clause)
```

The 4'th constraint

4 - The number of parallel sessions is not exceeded (or is equal ?) for each slot: $$\sum_{\substack{s\in S\ l\in L}} x_{(s,c,l)} \leq n \qquad \forall c \in C$$

but in our code we implement it with z

This Rewriting can enable us to reduce the size of constraint 4 as follows :

$$\sum_{\substack{s\in S}} \overline{z_{(s,c)}} \leq n \qquad \forall c \in C$$

implementation

```
or c in range(1, slots + 1):
    neg_z_vars = [-var_z(s, c) for s in range(1, conference_sessions + 1)]
    atmost_clause = CardEnc.atmost(lits=neg_z_vars,
bound=max_parallel_sessions)
    constraints.extend(atmost_clause.clauses)
```

Soft constraint

We want to minimize the number of working-group conflicts in the schedule: $$max \sum_{\substack{(s_1,s_2,c,g)\in S\times S\times C\times G \ s1<s2 \ g\in WG(s_1)\cap WG(s_2)}} \overline{y_{(s_1,s_2,c,g)}}$$

implentation

```
for s1 in range(1, conference_sessions + 1):
    for s2 in range(s1 + 1, conference_sessions + 1):  # Ensure s1 < s2
        for c in range(1, slots + 1):
            common_groups = set(session_groups[s1 -
1]).intersection(session_groups[s2 - 1])
            for g in common_groups:
                y_var = var_y(s1, s2, c, g)
                constraints.append([-y_var], weight=1)
```