CLT_solver_metis_cust

DA lab.

UNIST

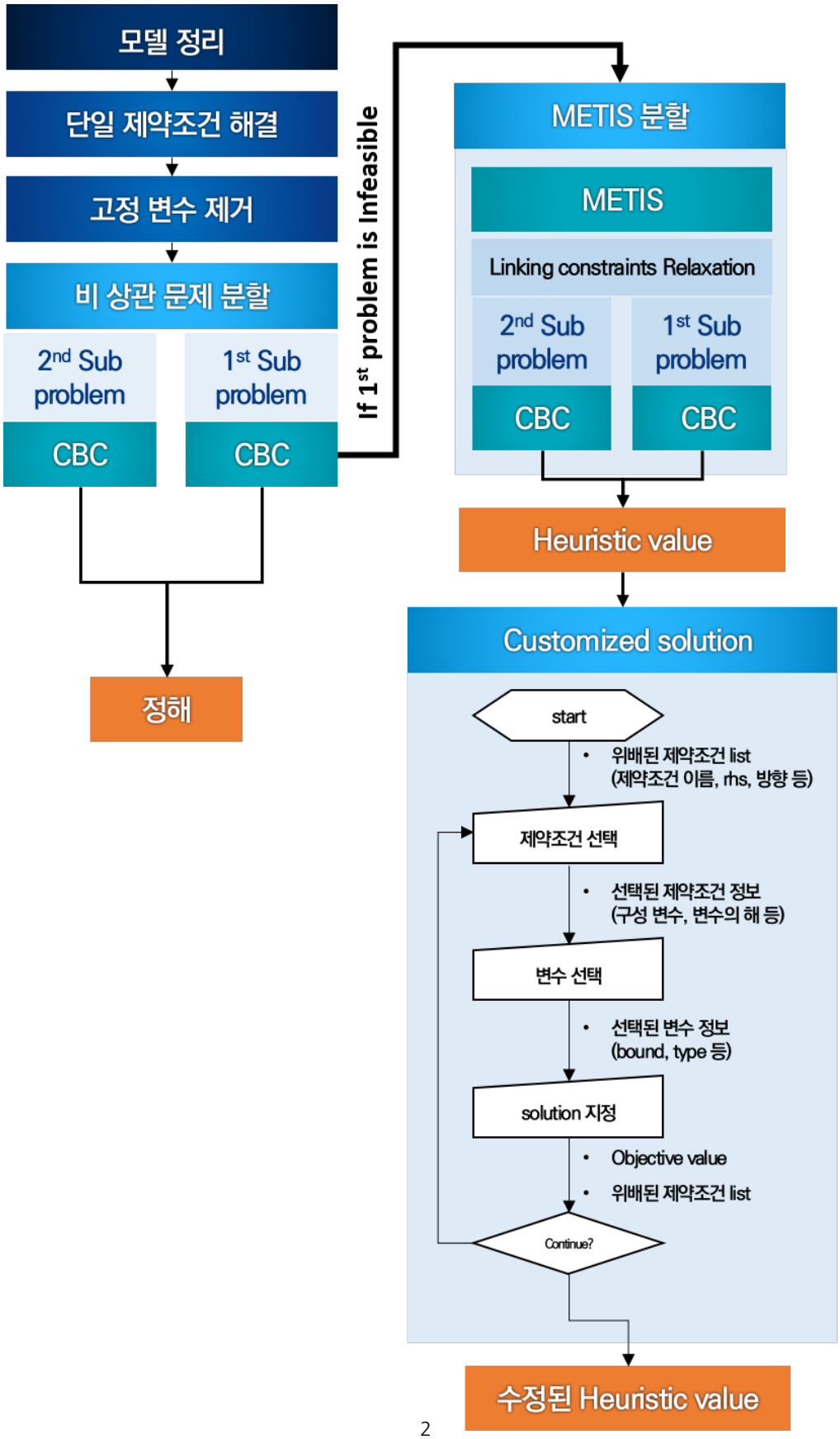
Input 형식

0. 데이터프레임 column 정보

1. 기본 문제 형태 (problem_input)

- xxx.mps 정보를 다음 데이터프레임 형태로 변형하여 사용

알고리즘 모식도


모델 정리
단일 제약조건 해결
고정 변수 제거
비 상관 문제 분할
2nd Sub problem
CBC
1st Sub problem
CBC
정해
If 1st problem is Infeasible
METIS 분할
METIS
Linking constraints Relaxation
2nd Sub problem
CBC
1st Sub problem
CBC
Heuristic value
Customized solution
start
• 위배된 제약조건 list (제약조건 이름, rhs, 방향 등)
제약조건 선택
• 선택된 제약조건 정보 (구성 변수, 변수의 해 등)
변수 선택
• 선택된 변수 정보 (bound, type 등)
solution 지정
• Objective value
• 위배된 제약조건 list
Continue?
수정된 Heuristic value

Package

```
import argparse
import os
import glob
import time
import math

import numpy as np
import numpy.lib.recfunctions as rfn

import networkx as nx
import metis

import cplex
from mip.model import *
from tqdm import tqdm
import pandas as pd
```

Function

0. cbc_solve : CBC solver

```
def cbc_solve(model, time_limit = TimeLimit, tol_limit = GAP, int_tol_limit = 0.01):
    model.store_search_progress_log = True
    model.max_gap = tol_limit #de
    model.integer_tol = int_tol_limit
    startTime = time.time()
    model.preprocess = True
    status = model.optimize(max_seconds=time_limit)
    endTime = time.time() - startTime
    print('Time: {} (s)'.format(endTime))
    print('Status: {}'.format(status))
    print('Objective Value: {}'.format(model.objective_value))
    return model
```

1. Functions for preprocessing

xxx.mps to numpy list for handling

```
def to_element_list(CONSTRAINTS_np):
    # Ignore when constraints have no variable # len(c['expr.ind'] == 0) #Meaningless Constraints
    con_df_dtype = np.dtype([('c.idx', int), ('ind', int), ('val', float),
                             ('sense', object), ('rhs', float)])
    CONSTRAINTS_df = np.concatenate([np.c_[[c['c.idx']]*len(c['expr.ind']),
                                           c['expr.ind'], c['expr.val'],
                                           [c['sense']]*len(c['expr.ind']),
```

```
                                            [c['rhs']]*len(c['expr.ind'])] for c in CONSTRAINTS_np])
    CONSTRAINTS_df = np.array([(int(c[0]), int(c[1]), float(c[2]),
                                str(c[3]), float(c[4])) for c in CONSTRAINTS_df], dtype = con_df_dtype)
    return CONSTRAINTS_df
```

Numpy list to mps format

```
def to_mps_format(CONSTRAINTS_df):
    con_np_dtype = np.dtype([('c.idx', int), ('expr.ind', list), ('expr.val', list),
                             ('sense', object), ('rhs', float)])
    con_group = np.split(CONSTRAINTS_df, np.cumsum(np.unique(CONSTRAINTS_df['c.idx'], return_counts=Tru
    CONSTRAINTS_df_np = np.array([(int(c['c.idx'][0]), list(c['ind']), list(c['val']),
                                   str(c['sense'][0]), float(c['rhs'][0]))
                                  for c in con_group], dtype = con_np_dtype)
    return CONSTRAINTS_df_np

def array_to_mps(VARIABLES_np, CONSTRAINTS_np):
    model = Model(solver_name="cbc")

    for v in tqdm(VARIABLES_np):
        model.add_var(name = VARIABLES_idx[int(v['v.idx'])],
                      obj = float(v['v.obj']),
                      ub = float(v['v.ub']),
                      lb = float(v['v.lb']),
                      var_type = str(v['v.var_type']))
    for c in tqdm(CONSTRAINTS_np):
        var_list = [model.var_by_name(VARIABLES_idx[ind]) for ind in c['expr.ind']]
        model.add_constr(LinExpr(variables = list(var_list),
                                 coeffs = list(c['expr.val']),
                                 const = float(c['rhs']),
                                 sense = str(c['sense'])),
                         name = CONSTRAINTS_idx[c['c.idx']],
                        )

    model.objective = xsum(v.obj * v for v in model.vars)
    print('\n model has {} vars, {} constraints and {} nzs'.format(model.num_cols, model.num_rows, model
    return (model)
```

Solution reflecting at the problem set

```
def update_model(VARIABLES_df_p, CONSTRAINTS_df_p, ANS):
    CONSTRAINTS_df_upd = CONSTRAINTS_df_p
    CONSTRAINTS_df_upd_ANS = np.empty(CONSTRAINTS_df_upd.shape,
                                      dtype = [('c.idx', int), ('ind', int), ('val', float),
                                               ('sense', object), ('rhs', float), ('ANS', float)])
    CONSTRAINTS_df_upd_ANS[['c.idx', 'ind', 'val', 'sense', 'rhs']] =
      CONSTRAINTS_df_upd[['c.idx', 'ind', 'val', 'sense', 'rhs']]

    for c in CONSTRAINTS_df_upd_ANS:
        if ANS[VARIABLES_idx[c['ind']]] != None:
```

```
            c['ANS'] = ANS[VARIABLES_idx[c['ind']]] * c['val']
        else:
            c['ANS'] = None

    con_group = np.split(CONSTRAINTS_df_upd_ANS, np.cumsum(np.unique(CONSTRAINTS_df_upd_ANS['c.idx'], r
    con_group_upd = []

    for c in con_group:
        c['rhs'] = c['rhs'][0] + sum(c['ANS'][np.logical_not(np.isnan(c['ANS']))])
        if sum((np.isnan(c['ANS']))) != 0:
            con_group_upd.append(c[np.isnan(c['ANS'])])

    CONSTRAINTS_df_p = np.concatenate(con_group_upd)
    CONSTRAINTS_df_p = CONSTRAINTS_df_p[['c.idx', 'ind', 'val', 'sense', 'rhs']]

    var_ANS = [VARIABLES_name[x] for x in ANS if ANS[x] is not None]
    del_var = np.array([np.where(VARIABLES_df_p['v.idx']==v)[0][0] for v in var_ANS
                        if len(np.where(VARIABLES_df_p['v.idx']==v)[0]) != 0])
    if len(del_var) != 0:
        VARIABLES_df_p = np.delete(VARIABLES_df_p, del_var, axis = 0)
    return (VARIABLES_df_p, CONSTRAINTS_df_p)
```

2. Functions for presolve

fixed_value_solver: 고정변수 solver (Fixed variable solver)

```
def fixed_value_solver(VARIABLES_df_p, ANS):
    fx_var = np.array([int(v['v.idx'])
                       for v in VARIABLES_df_p if v['v.ub'] == v['v.lb'] and v['v.var_type'] == str('I'
    for v in fx_var:
        ANS[VARIABLES_idx[v]] = VARIABLES_df_p[np.where(VARIABLES_df_p['v.idx']==v)]['v.ub'].item()
    cnt_f = len(fx_var)
    return(VARIABLES_df_p, ANS, cnt_f)
```

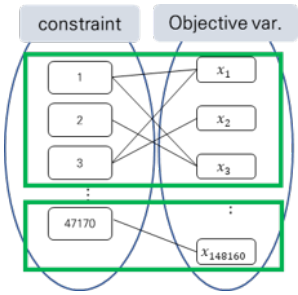
single_constraints_solver: 단순 방정식 solver (Linear Equation in One Variable solver)


```
def single_constraints_solver(VARIABLES_df_p, CONSTRAINTS_df_p, ANS):
    cnt_s = 0
    con_group = np.split(CONSTRAINTS_df_p, np.cumsum(np.unique(
      CONSTRAINTS_df_p['c.idx'],  return_counts=True)[1])[:-1])

    con_group_I = []
    con_group_upd =[]

    for c in con_group:
        if len(c) == 1:
            if c['sense'] ==  '=':
                ANS[VARIABLES_idx[int(c['ind'])]] = - float(c['rhs']) / float(c['val'])
                cnt_s = cnt_s + 1
                con_group_upd.append(c)
            elif VARIABLES_df_p[np.where(
              VARIABLES_df_p['v.idx']==c['ind'])]['v.var_type'].item() == 'I':
                 con_group_I.append(c)
            else:
                con_group_upd.append(c)
        else:
            con_group_upd.append(c)

    CONSTRAINTS_df_p = np.concatenate(con_group_upd)
    CONSTRAINTS_df_p = CONSTRAINTS_df_p[['c.idx', 'ind', 'val', 'sense', 'rhs']]

    for c in con_group_I:
        if c['sense'] == '<':
            ub_upd = max(VARIABLES_df_p['v.lb'][np.where(
              VARIABLES_df_p['v.idx'] == c['ind'])], math.floor(-c['rhs'] / c['val']))
            VARIABLES_df_p['v.ub'][np.where(VARIABLES_df_p['v.idx'] == c['ind'])] =
              min(VARIABLES_df_p['v.ub'][np.where(VARIABLES_df_p['v.idx'] == c['ind'])], ub_upd)
        elif c['sense'] == '>':
            lb_upd = min(VARIABLES_df_p['v.ub'][np.where(
              VARIABLES_df_p['v.idx'] ==  c['ind'])], math.ceil(-c['rhs'] / c['val']))
            VARIABLES_df_p['v.lb'][np.where(
```

```
                VARIABLES_df_p['v.idx'] == c['ind'])] =
                max(VARIABLES_df_p['v.lb'][np.where(
                  VARIABLES_df_p['v.idx'] == c['ind'])], -c['rhs'] / c['val'])

    cnt_i = len(con_group_I)
    return(VARIABLES_df_p, CONSTRAINTS_df_p, cnt_s, cnt_i)
```

3. Dicomp_solver : 문제 분할 solver (Dicomposition solver)

- 완전히 독립적인 문제 set으로 분할하는 알고리즘
- 분할 된 Matrix 중 크기가 큰 Matrix는 ind_problem_1, 작은 Matrix는 ind_problem_2




```
def decompose (CONSTRAINTS_df_p,VARIABLES_df_p):
    r = 'int'
    diff = -1
    while diff!=0:
        if r == 'int':
            ind=CONSTRAINTS_df_p['ind'][0]
            Cidx = CONSTRAINTS_df_p[np.isin(CONSTRAINTS_df_p['ind'], ind)]['c.idx']
            ind = CONSTRAINTS_df_p[np.isin(CONSTRAINTS_df_p['c.idx'], Cidx)]['ind']
            diff = len(CONSTRAINTS_df_p[np.isin(CONSTRAINTS_df_p['ind'], ind)])
            r = 'noint'
        else:
            step1 = CONSTRAINTS_df_p[np.isin(CONSTRAINTS_df_p['ind'], ind)]
            Cidx = CONSTRAINTS_df_p[np.isin(CONSTRAINTS_df_p['ind'], ind)]['c.idx']
            ind = CONSTRAINTS_df_p[np.isin(CONSTRAINTS_df_p['c.idx'], Cidx)]['ind']
            step2 = CONSTRAINTS_df_p[np.isin(CONSTRAINTS_df_p['ind'], ind)]
            diff = len(step1)-len(step2)

    ind1 = CONSTRAINTS_df_p[np.isin(CONSTRAINTS_df_p['ind'],ind)==True]
    ind2 = CONSTRAINTS_df_p[np.isin(CONSTRAINTS_df_p['ind'],ind)==False]
    ind1_var = VARIABLES_df_p[np.isin(VARIABLES_df_p['v.idx'],ind1['ind'])]
    ind2_var = VARIABLES_df_p[np.isin(VARIABLES_df_p['v.idx'],ind2['ind'])]

    return(ind1,ind2,ind1_var,ind2_var)
```

4. Metis_2way: METIS solver를 이용한 G(E,V) 이분할

- 목적함수 변수와 제약조건으로 이루어진 matrix를 그래프 G(E,V)로 상정
  - E: 목적함수 변수 set, V: 제약조건 set 인

```
def METIS_2way (CONSTRAINTS_LR):
    const=CONSTRAINTS_LR['c.idx']
    obj=CONSTRAINTS_LR['ind']
    con_LR_index = []
    obj_LR_index = []
    for i in range(len(const)):
        con_LR_index.append("c"+f'{const[i]:08}')
    for i in range(len(obj)) :
        obj_LR_index.append("o"+f'{obj[i]:08}')

    # Graph element setting
    col = obj_LR_index
    row = con_LR_index
    edge = list()
    for i in range(len(col)):
        edge.append((col[i],row[i]))

    # Graph
    B = nx.Graph()
    B.add_nodes_from(col, bipartite=0)
    B.add_nodes_from(row, bipartite=1)
    B.add_edges_from(edge)

    # Partitiong
    part=metis.part_graph(B, nparts=2)
    part0=np.array(B.node)[np.isin(part[1],0)]
    part1=np.array(B.node)[np.isin(part[1],1)]

    part_0_v=[]
    part_1_v=[]
    for i in range(len(part0)):
        if 'o' in part0[i]:
            part_0_v.append(int(part0[i].replace("o","")))
    for i in range(len(part1)):
        if 'o' in part1[i]:
            part_1_v.append(int(part1[i].replace("o","")))

    return(part_0_v,part_1_v)
```

5. Customizing Function

CONST_satisfy_function: 현재 해를 적용하였을 때, 위배되는 제약조건 list 반환

- 시간이 다소 소요

```
def CONST_satisfy_function(SOLUTION):
    const_stat = []
    lhs = []
    const_name = []

    for lc in con_names:

        linking_const_set = CONSTRAINTS_df[np.isin(CONSTRAINTS_df['c.idx'], CONSTRAINTS_idx2[lc])]
        linking_var=VARIABLES_df[np.isin(VARIABLES_df['v.idx'],linking_const_set['ind'])]['v.idx']
        const_name.append([CONSTRAINTS_idx[CONSTRAINTS_idx2[lc]],linking_const_set['sense'][0],
                           linking_const_set['rhs'][0]])

        linking_const_lhs = []

        for c in linking_var:
            linking_const_lhs.append(SOLUTION[VARIABLES_idx[c]]* linking_const_set[
              np.isin(linking_const_set['ind'], c)]['val'][0])
        if linking_const_set['sense'][0] == '<':
            const_stat.append(sum(linking_const_lhs)+linking_const_set['rhs'][0]<= 1e-1 )
            lhs.append(sum(linking_const_lhs))
        if linking_const_set['sense'][0] == '>':
            const_stat.append(sum(linking_const_lhs)+linking_const_set['rhs'][0] >= -1e-1 )
            lhs.append(sum(linking_const_lhs))
        if linking_const_set['sense'][0] == '=':
            const_stat.append(abs(
              sum(linking_const_lhs)+ np.array(linking_const_set['rhs'][0],
                                               dtype = float)) <= 1e-1 )
            lhs.append(sum(linking_const_lhs))

    linking_result = []
    for c in range(len(const_name)):
        linking_result.append(
          (CONSTRAINTS_idx2[const_name[c][0]],const_name[c][0],lhs[c],
           const_name[c][1],const_name[c][2], const_stat[c]))

    linking_result_dtype = np.dtype([('c.idx',int),('c.name', object),
                                     ('lhs', object), ('sense', object),
                                     ('rhs', float) ,('TF', object)])
    linking_result=np.array([(c[0], c[1], c[2], c[3], c[4],c[5]) for c in linking_result],
                            dtype = linking_result_dtype)
    linking_result_F=linking_result[np.isin(linking_result['TF'],False)]

    print('The number of unsatisfied constraint:{}'.format(len(const_stat)-sum(const_stat)))
    return(linking_result_F)
```

total_obj_calculator : 현재 해를 적용했을 때 목적함수 값 반환

```
def total_obj_calculator(SOLUTION):
    obj_value=[]
    for v in VARIABLES_df:
        obj_value.append(SOLUTION[VARIABLES_idx[v['v.idx']]]*v['v.obj'])
    return(np.nansum(obj_value))
```

check_const : 사용자가 check하고 싶은 제약조건 index 입력 시, 해당 제약조건의 구성 list 반환

- 구성 변수의 현재 해, 구성 변수의 coefficient, 구성 변수의 해와 coefficient 내적 값, 변수 index, 변수의 목적함수에서의 coefficient, 변수의 lower bound, upper bound, type, 이름
- 사용자는 일련의 정보를 바탕으로 해를 부여

```
def check_const(check_con_idx):
    var_set=CONSTRAINTS_df[np.isin(CONSTRAINTS_df['c.idx'],check_con_idx)]
    print(check_con_idx,var_set['sense'][0],var_set['rhs'][0])
    print('sol|var|sol*var|v.idx|v.obj|ub|lb|type|variable name')
    for v in var_set:
        print(SOLUTION[VARIABLES_idx[v['ind']]],
              v['val'],SOLUTION[VARIABLES_idx[v['ind']]]*v['val'],
              VARIABLES_df[np.isin(VARIABLES_df['v.idx'],v['ind'])],
             VARIABLES_idx[VARIABLES_df[np.isin(VARIABLES_df['v.idx'],
                                                v['ind'])]['v.idx'][0]])
```

sol_to_change : 사용자가 지정한 변수를 사용자가 지정한 해로 바꿔주는 함수

```
    def sol_to_change(sol_to_):
    sol_to_change = []
    for v in sol_to_:
        print('{} is to {} '.format(VARIABLES_idx[v[0]],v[1]))
        SOLUTION.update(zip([v[0]],[v[1]]))
```

Final code

1. Input

- File number : 싸이버로지텍 예제의 임의 파일 번호 부여
- Time Limit : cbc solver의 time limit (예: 300)
- GAP : cbc solver의 tolerance gap (예: 30)

2. Output

- SOLUTION_fin : 'Input 형식'의 solution 형태로 도출
- 최종 결과물은 싸이버로지텍에서 요구한 형식에 맞춰 xxx.sol로 저장됨
- 만족하지 못한 제약조건 list와 정보를 xxx.csv로 반환

3. 실행 방법

- (1) CMD or Terminal 실행
- (2) 다음의 코드 입력

```
Python CLT_solver_metis_cust.py 0 300 30
```

- (3) Customizing solution 옵션이 시작하면, 질문에 맞는 답 입력

4. Main Code

```
# input
parser = argparse.ArgumentParser()
parser.add_argument('TimeLimit', type = int, help="Time limit(s) is :" )
parser.add_argument('Tol', type = int, help = "Tolerance(%) is : ")
parser.add_argument('filenumber', type = int, help = "The file number is : ")
args=parser.parse_args()
TimeLimit = args.TimeLimit
Tol = args.Tol
filenumber = args.filenumber
GAP=Tol
Total_startTime = time.time()
filelist = glob.glob(str(os.getcwd())+'/mps/data/*.mps')
print(filelist)


filename = filelist[filenumber]
print(filename)


basename = os.path.basename(filename)
currentpath = filename[:-len(basename)]
currentpath = currentpath[:-5]
savepath_cplex = currentpath +'processing/CPLEX_file/'
```

```
savename_cplex = savepath_cplex + basename
savename_cplex = savename_cplex[:-4]+'_cplex.mps'


# CBC solver to translate readable file
## Solve the mps file with cbc solver
m = Model(name = basename, sense = 'MIN', solver_name = "cbc")
m.read(savename_cplex)
print('model has {} vars, {} constraints and {} nzs'.format(m.num_cols, m.num_rows, m.num_nz))
savepath_cbc = currentpath +'processing/CBC_file/'
savepath_cbc
savename_cbc = savepath_cbc + os.path.basename(filename)
savename_cbc = savename_cbc[:-4]+'_cbc.mps'
savename_cbc
m.write(savename_cbc)
model = Model(solver_name="cbc")
model.read(savename_cbc+'.mps')
print('model has {} vars, {} constraints and {} nzs'.format(m.num_cols, m.num_rows, m.num_nz))


#preprocessing
num_var = m.num_cols
var_index = [v.idx for v in m.vars]
var_names = [v.name for v in m.vars]
var_dtype = np.dtype([('v.idx', int), ('v.obj', float), ('v.ub', float),
('v.lb', float), ('v.var_type', object)])
VARIABLES = [(int(v.idx), float(v.obj), float(v.ub), float(v.lb), str(v.var_type)) for v in m.vars]

VARIABLES_np = np.array(VARIABLES, dtype=var_dtype)
VARIABLES_df = VARIABLES_np.copy()
VARIABLES_idx = dict(zip(var_index, var_names))
VARIABLES_name = dict(zip(var_names, var_index))

num_con = m.num_rows
con_index = [c.idx for c in m.constrs]
con_names = [c.name for c in m.constrs]
con_np_dtype = np.dtype([('c.idx', int), ('expr.ind', list), ('expr.val', list),
                         ('sense', object), ('rhs', float)])
CONSTRAINTS = [(int(c.idx),
                list([np.int(VARIABLES_name[ind.name]) for ind in c.expr.expr.keys()]),
                list([np.float(val) for val in c.expr.expr.values()]),
                str(c.expr.sense), float(c.expr.const)) for c in m.constrs]

CONSTRAINTS_np = np.array(CONSTRAINTS, dtype = con_np_dtype)
CONSTRAINTS_idx = dict(zip(con_index, con_names))
CONSTRAINTS_name = dict(zip(con_names, con_index))

ANS = dict.fromkeys(var_names, None)
OBJ = dict.fromkeys(var_names, None)

for v in m.vars:
    OBJ[v.name] = v.obj
```

```
# Define function for constraints table in mps format and element list
CONSTRAINTS_df = to_element_list(CONSTRAINTS_np)
CONSTRAINTS_df_np = to_mps_format(CONSTRAINTS_df)

# Define function for writing mps
model_copy = array_to_mps(VARIABLES_np, CONSTRAINTS_df_np)
savepath_copy = currentpath +'processing/Copy/'
savepath_copy

savename_copy = savepath_copy + os.path.basename(filename)
savename_copy = savename_copy[:-4]+'_copy.mps'
savename_copy


# Presolve
Presolve_startTime = time.time()
model_p = Model(name = basename, sense = 'MIN', solver_name = "cbc")
model_p.read(savename_copy+'.mps')
ANS = dict.fromkeys(var_names, None)
VARIABLES_df_p = VARIABLES_df.copy()
CONSTRAINTS_df_p = CONSTRAINTS_df.copy()



cnt_MC = 0
cnt_MOV = 0

# Remove Meaningless Constraint
cnt_MC = CONSTRAINTS_np.shape[0] - to_mps_format(CONSTRAINTS_df_p).shape[0]
print('Remove {} MCs'.format(cnt_MC))

# Remove Meaningless Objective Variables
s = set(np.unique(CONSTRAINTS_df_p['ind']))
v_not = np.array([int(v) for v in VARIABLES_df_p['v.idx'] if int(v) not in s])
for v in v_not:
    ANS[VARIABLES_idx[v]] = 0
cnt_MOV = len(v_not)
if len(v_not) != 0:
    VARIABLES_df_p = np.delete(VARIABLES_df_p, v_not, axis = 0)
print('Remove {} MOVs'.format(cnt_MOV))

cnt_f = -1
VARIABLES_df_p, ANS, cnt_f = fixed_value_solver(VARIABLES_df_p, ANS)
VARIABLES_df_p, CONSTRAINTS_df_p = update_model(VARIABLES_df_p, CONSTRAINTS_df_p, ANS)
print('Model has {} fixed value variables'.format(cnt_f))


cnt_s = -1
cnt_i = -1
while cnt_s != 0:
    VARIABLES_df_p, CONSTRAINTS_df_p, cnt_s, cnt_i = single_constraints_solver(VARIABLES_df_p, CONSTRAI
    VARIABLES_df_p, CONSTRAINTS_df_p = update_model(VARIABLES_df_p, CONSTRAINTS_df_p, ANS)
    print('Model has {} simple equations'.format(cnt_s))
```

```
    print('Model has {} integer bound-fixing variables'.format(cnt_i))


print(VARIABLES_df.shape, VARIABLES_df_p.shape)
print(CONSTRAINTS_df.shape, CONSTRAINTS_df_p.shape)



pre = []
for v in var_names:
    if ANS[v] != None:
        pre.append(ANS[v] * float(VARIABLES_df[np.where(VARIABLES_df['v.idx']==VARIABLES_name[v])]['v.ob
print(sum(pre))
CONSTRAINTS_df_p_np = to_mps_format(CONSTRAINTS_df_p)

print(filename)
Presolve_endTime = time.time() - Presolve_startTime
print('Presolve_Time: {} (s)'.format(Presolve_endTime))

Total_endTime = time.time() - Total_startTime
print('Total_Time: {} (s)'.format(Total_endTime))



CONSTRAINTS_df_p_cp = CONSTRAINTS_df_p.copy()
VARIABLES_df_p_cp = VARIABLES_df_p.copy()

# Decomposition
CONSTRAINTS_df_p_decomp1,CONSTRAINTS_df_p_decomp2,VARIABLES_df_p_decomp1,VARIABLES_df_p_decomp2 =
decompose(CONSTRAINTS_df_p_cp,VARIABLES_df_p_cp)
CONSTRAINTS_df_p_np_decomp1=to_mps_format(CONSTRAINTS_df_p_decomp1)
CONSTRAINTS_df_p_np_decomp2=to_mps_format(CONSTRAINTS_df_p_decomp2)

model_p1 = array_to_mps(VARIABLES_df_p_decomp1, CONSTRAINTS_df_p_np_decomp1)
savepath_copy = currentpath +'processing/Copy/'
savename_copy = savepath_copy + os.path.basename('model_p1')
savename_copy = savename_copy[:-4]+'_copy.mps'
model_p1.write(savename_copy)
model_p1 = Model(name = basename, sense = 'MIN', solver_name = "cbc")
model_p1.read(savename_copy+'.mps')
result1=cbc_solve(model_p1,  time_limit = 100)

model_p2 = array_to_mps(VARIABLES_df_p_decomp2, CONSTRAINTS_df_p_np_decomp2)
savepath_copy = currentpath +'processing/Copy/'
savename_copy = savepath_copy + os.path.basename('model_p2')
savename_copy = savename_copy[:-4]+'_copy.mps'
model_p2.write(savename_copy)
model_p2 = Model(name = basename, sense = 'MIN', solver_name = "cbc")
model_p2.read(savename_copy+'.mps')
result2=cbc_solve(model_p2,  time_limit = TimeLimit)

print('Presolve objective value : {} '.format(sum(pre)))
print('1st decomposition objective value : {}'.format(result1.objective_value))
```

```
print('2nd decomposition objective value : {}'.format(result2.objective_value))

# It will be work when there is Infesiable set

num_feasible = []
num_optimal = []
num_infeasible = []
totalvalue = []
for a in (result1, result2):
    if a.status.name!='INFEASIBLE' :
        num_feasible.append(a.num_cols)
        if a. status.name == "OPTIMAL" :
            num_optimal.append(a.num_cols)
        totalvalue.append(a.objective_value)
    else :
        num_infeasible.append(a.num_cols)
        savepath_write = currentpath +'processing/SYM/'
        savename_write = savepath_write + os.path.basename(filename)
        savename_write = savename_write[:-4]+'sym.mps'
        a.write(savename_write)
        a = Model(name = basename, sense = 'MIN', solver_name = "cbc")
        a.read(savename_write+'.mps')
        metis_file= savename_write
        os.path.basename(metis_file)
        basename = os.path.basename(metis_file)

        # CBC solver to translate readable file
        ## Solve the mps file with cbc solver
        m = Model(name = basename, sense = 'MIN', solver_name = "cbc")
        m.read(metis_file+'.mps')
        print('model has {} vars, {} constraints and {} nzs'.format(m.num_cols, m.num_rows, m.num_nz))

        #preprocessing
        num_var = m.num_cols
        var_index = [v.idx for v in m.vars]
        var_names = [v.name for v in m.vars]
        var_dtype = np.dtype([('v.idx', int), ('v.obj', float),
        ('v.ub', float), ('v.lb', float), ('v.var_type', object)])
        VARIABLES = [(int(v.idx), float(v.obj),
        float(v.ub), float(v.lb), str(v.var_type)) for v in m.vars]

        VARIABLES_np = np.array(VARIABLES, dtype=var_dtype)
        VARIABLES_df = VARIABLES_np.copy()
        VARIABLES_idx = dict(zip(var_index, var_names))
        VARIABLES_name = dict(zip(var_names, var_index))

        num_con = m.num_rows
        con_index = [c.idx for c in m.constrs]
        con_names = [c.name for c in m.constrs]
        con_np_dtype = np.dtype([('c.idx', int), ('expr.ind', list), ('expr.val', list),
                                 ('sense', object), ('rhs', float)])
        CONSTRAINTS = [(int(c.idx),
                        list([np.int(VARIABLES_name[ind.name]) for ind in c.expr.expr.keys()]),
```

```
                    list([np.float(val) for val in c.expr.expr.values()]),
                    str(c.expr.sense), float(c.expr.const)) for c in m.constrs]

CONSTRAINTS_np = np.array(CONSTRAINTS, dtype = con_np_dtype)
CONSTRAINTS_idx = dict(zip(con_index, con_names))
CONSTRAINTS_name = dict(zip(con_names, con_index))

ANS = dict.fromkeys(var_names, None)
OBJ = dict.fromkeys(var_names, None)

for v in m.vars:
    OBJ[v.name] = v.obj

# Define function for constraints table in mps format and element list
CONSTRAINTS_df = to_element_list(CONSTRAINTS_np)
CONSTRAINTS_df_np = to_mps_format(CONSTRAINTS_df)

Presolve_startTime = time.time()
ANS = dict.fromkeys(var_names, None)
VARIABLES_df_p = VARIABLES_df.copy()
CONSTRAINTS_df_p = CONSTRAINTS_df.copy()

# METIS decomposition
part_0_v, part_1_v=METIS_2way(CONSTRAINTS_df)

VARIABLES_LR=VARIABLES_df
VARIABLES_METIS_0=VARIABLES_LR[np.isin(VARIABLES_LR['v.idx'],part_0_v)]
VARIABLES_METIS_1=VARIABLES_LR[np.isin(VARIABLES_LR['v.idx'],part_1_v)]

CONSTRAINTS_LR=CONSTRAINTS_df
CONSTRAINTS_METIS_0=CONSTRAINTS_LR[np.isin(CONSTRAINTS_LR['ind'],part_0_v)]
CONSTRAINTS_METIS_1=CONSTRAINTS_LR[np.isin(CONSTRAINTS_LR['ind'],part_1_v)]

linking_const=list(set(CONSTRAINTS_METIS_0[np.isin(CONSTRAINTS_METIS_0['c.idx'],
CONSTRAINTS_METIS_1['c.idx'])]['c.idx']))

print(len(linking_const))
print(len(VARIABLES_METIS_0))
print(len(VARIABLES_METIS_1))
print(len(CONSTRAINTS_METIS_0))
print(len(CONSTRAINTS_METIS_1))

# linking constraint relaxation (remove)
CONSTRAINTS_METIS_0=CONSTRAINTS_METIS_0[np.isin(CONSTRAINTS_METIS_0['c.idx'],
linking_const)==False]
CONSTRAINTS_METIS_1=CONSTRAINTS_METIS_1[np.isin(CONSTRAINTS_METIS_1['c.idx'],
linking_const)==False]
print(len(CONSTRAINTS_METIS_0))
print(len(CONSTRAINTS_METIS_1))

# Solve the each decomposed problems
CONSTRAINTS_metis_0=to_mps_format(CONSTRAINTS_METIS_0)
CONSTRAINTS_metis_1=to_mps_format(CONSTRAINTS_METIS_1)
```

```
model_p1 = array_to_mps(VARIABLES_METIS_0, CONSTRAINTS_metis_0)
savepath_copy = currentpath +'processing/Copy/'
savename_copy = savepath_copy + os.path.basename('model_p1')
savename_copy = savename_copy[:-4]+'_copy.mps'
model_p1.write(savename_copy)
model_p1 = Model(name = basename, sense = 'MIN', solver_name = "cbc")
model_p1.read(savename_copy+'.mps')

model_p2 = array_to_mps(VARIABLES_METIS_1, CONSTRAINTS_metis_1)
savepath_copy = currentpath +'processing/Copy/'
savename_copy = savepath_copy + os.path.basename('model_p2')
savename_copy = savename_copy[:-4]+'_copy.mps'
model_p2.write(savename_copy)
model_p2 = Model(name = basename, sense = 'MIN', solver_name = "cbc")
model_p2.read(savename_copy+'.mps')
result4=cbc_solve(model_p2,  time_limit = TimeLimit)

result3=cbc_solve(model_p1,  time_limit = TimeLimit)
result4=cbc_solve(model_p2,  time_limit = TimeLimit)

# Customizing solution function
VARIABLES_idx2 = dict(zip(var_names,var_index))
sol_names = []
for v in result3.vars:
    sol_names.append(v.name)
for v in result4.vars:
    sol_names.append(v.name)

sol_index = []
for v in sol_names:
    sol_index.append(VARIABLES_idx2[v])

org_var_index = []
for v in var_names:
    org_var_index.append(VARIABLES_idx2[v])
org_var_index

no_solution=np.array(org_var_index)[np.isin(org_var_index, sol_index)==False]
no_solution_names=[]
for v in no_solution:
    no_solution_names.append(VARIABLES_idx[v])
no_solution_names
sol_names = []
for v in result3.vars:
    sol_names.append(v.name)
for v in result4.vars:
    sol_names.append(v.name)
for v in no_solution_names:
    sol_names.append(v)
sol_names

sol_val = []
for v in result3.vars:
```

```
        sol_val.append(v.x)
    for v in result4.vars:
        sol_val.append(v.x)
    for v in no_solution_names:
        sol_val.append(VARIABLES_df[np.isin(VARIABLES_df['v.idx'],VARIABLES_idx2[v])]['v.lb'][0])
    SOLUTION = dict(zip(sol_names, sol_val))

    const_stat = []
    const_name = []
    lhs = []

    for lc in linking_const:
        linking_const_set=CONSTRAINTS_df[np.isin(CONSTRAINTS_df['c.idx'], lc)]
        linking_var=VARIABLES_df[np.isin(VARIABLES_df['v.idx'],
        linking_const_set['ind'])]['v.idx']

        const_name.append([CONSTRAINTS_idx[lc],linking_const_set['sense'][0],
        linking_const_set['rhs'][0]])

        linking_const_lhs = []
        for c in linking_var:

            linking_const_lhs.append(SOLUTION[VARIABLES_idx[c]]*
            linking_const_set[np.isin(linking_const_set['ind'], c)]['val'][0])

        if linking_const_set['sense'][0] == '<':
            const_stat.append((sum(linking_const_lhs)+
            np.array(linking_const_set['rhs'][0], dtype = float)) <= 1e-1 )
            lhs.append(sum(linking_const_lhs))
        if linking_const_set['sense'][0] == '>':
            const_stat.append((sum(linking_const_lhs)+
            np.array(linking_const_set['rhs'][0], dtype = float)) >= -1e-1 )
            lhs.append(sum(linking_const_lhs))
        if linking_const_set['sense'][0] == '=':
            const_stat.append(abs(sum(linking_const_lhs)+
            np.array(linking_const_set['rhs'][0], dtype = float)) <= 1e-1 )
            lhs.append(sum(linking_const_lhs))

        const_satis_ratio=sum(const_stat)/len(linking_const)*100
        num_const_unsatis = len(linking_const)-sum(const_stat)

        CONSTRAINTS_idx2=dict(zip(con_names,con_index))
        VARIABLES_idx = dict(zip(var_index, var_names))


        linking_result = []
        for c in range(len(const_name)):
            linking_result.append((CONSTRAINTS_idx2[const_name[c][0]],
            const_name[c][0],lhs[c],const_name[c][1],const_name[c][2], const_stat[c]))

        linking_result_dtype = np.dtype([('c.idx',int),('c.name', object),
        ('lhs', object), ('sense', object),('rhs', float) ,('TF', object)])
        linking_result=np.array([(c[0], c[1], c[2], c[3], c[4],c[5]) for c in linking_result],
```

```
            dtype = linking_result_dtype)
            linking_result_F=linking_result[np.isin(linking_result['TF'],False)]



# Customizing function
CONST_satisfy_list_pre=linking_result_F
print('Do you want customizing the solution? ([y] or [n])')
key = input()
if key == 'y':
    end = 'y'
    while end =='y':
        print(CONST_satisfy_list_pre)
        print('Which constraint do you want to check? Enter the constraint index (only one)')
        const_idx_select = input()
        if const_idx_select!='n':
            print(check_const(int(const_idx_select)))
            print('Enter the variable index and solution which do you want. (multiple is avable, e.g. >
            var_idx_select=input()
            if var_idx_select != 'n':
                var_idx_select=var_idx_select.split(',')
                var_to=[]
                for v in var_idx_select:
                    var_to.append(int(v))
                print(VARIABLES_df[np.isin(VARIABLES_df['v.idx'],var_to)])
                print('Enter the solution')
                sol_select= input()
                sol_select=sol_select.split(',')
                sol=[]
                for v in sol_select:
                    sol.append(int(v))
                var_to_sol=[]
                for v in range(len(var_to)):
                    var_to_sol.append((var_to[v],sol[v]))
                sol_to_change(var_to_sol)
                CONST_satisfy_list=CONST_satisfy_function(SOLUTION)

                print('Do you want to customizint the solution more? ([y] or [n])')
                CONST_satisfy_list_pre=CONST_satisfy_list
                end = input()
            else: end = 'n'
        else: end = 'n'
    print('================================')
    print('     End the process')
    print('================================')

    SOLUTION_1st=[]
    for v in var_names:
        SOLUTION_1st.append((v,SOLUTION[v]))

else:
    SOLUTION_1st=[]
```

```
    for v in var_names:
        SOLUTION_1st.append((v,SOLUTION[v]))

print('The total objective value is {}'.format(total_obj_calculator(SOLUTION)))


# solution out
SOLUTION_1st_dtype = np.dtype([('v.name', object), ('sol', int)])
SOLUTION_1st_list = np.array([(c[0], c[1]) for c in SOLUTION_1st], dtype = SOLUTION_1st_dtype)

FINAL=pd.DataFrame(SOLUTION_1st_list)

savepath_copy = currentpath +'processing/Copy/'
savename_copy = savepath_copy + os.path.basename('model_p2')
savename_copy = savename_copy[:-4]+'_copy.mps'
resultname=filename.replace('data','processing/result_LP_sol')+'.csv'
FINAL.to_csv(resultname,header=True, index=False)


# Final summary

Total_endTime = time.time() - Total_startTime
solved_ratio=(sum(num_feasible)+len(pre))/(sum(num_feasible)+len(pre)+sum(num_infeasible))*100
optimal_ratio=(sum(num_optimal)+len(pre))/(sum(num_feasible)+len(pre)+sum(num_infeasible))*100
infeasible_ratio = 100-solved_ratio




import pandas as pd
if result1.status.name != "INFEASIBLE":
    total_value= sum(pre)+result1.objective_value+result2.objective_value
    FINAL = [[sum(pre),result1.objective_value, result2.objective_value, 0,0,
              total_value,
              result1.status.name,result2.status.name,'nan','nan',
              len(pre),result1.num_cols, result2.num_cols,0,0,
              solved_ratio, optimal_ratio,  infeasible_ratio,Total_endTime           ]]
    print('====================================================================')
    print('The summary of {}'.format(filename))
    print('Ratio of solved variables (%) is :{}'.format(solved_ratio))
    print('Ratio of optimal solved variables (%) is :{}'.format(solved_ratio))
    print('The toal value is :{}'.format((total_value)))
    print('Presolve objective value : {} '.format(sum(pre)))
    print('1st decomposition objective value : {}'.format(result1.objective_value))
    print('2nd decomposition objective value : {}'.format(result2.objective_value))
    print('Total_Time: {} (s)'.format(Total_endTime))
    print('====================================================================')

else:
    total_value= sum(pre)+result2.objective_value +
    result3.objective_value + result4.objective_value

    FINAL = [[sum(pre),result1.objective_value, result2.objective_value,
```

```
    result3.objective_value, result4.objective_value,
              total_value,
              result1.status.name,result2.status.name,result3.status.name,result4.status.name,
              len(pre),result1.num_cols, result2.num_cols,result3.num_cols,result4.num_cols,
              solved_ratio, optimal_ratio,  infeasible_ratio,
              Total_endTime,
              len(linking_const), num_const_unsatis, const_satis_ratio,(m.num_rows-num_const_unsatis)/m
    print('=====================================================================')
    print('The summary of {}'.format(filename))
    print('Ratio of optimal solved variables (%) is :{}'.format(solved_ratio))
    print('The toal value is :{}'.format((total_value)))
    print('Presolve objective value : {} '.format(sum(pre)))
    print('1st decomposition objective value : {}'.format(result1.objective_value))
    print('2nd decomposition objective value : {}'.format(result2.objective_value))
    print('1st part of METIS objective value : {}'.format(result3.objective_value))
    print('2nd part of METIS objective value : {}'.format(result4.objective_value))
    print('{} of {} linking constraints violate the constraints ({}% satisfaction at the linking constra
    print('{}% satisfaction at the total constraints'.format((m.num_rows-num_const_unsatis)/m.num_rows*
    print('Total_Time: {} (s)'.format(Total_endTime))
    print('=====================================================================')


FINAL=pd.DataFrame(FINAL)
resultname=filename.replace('data','processing/result')+'.csv'
FINAL.to_csv(resultname,header=True, index=False)

FINAL=pd.DataFrame(linking_result_F)
resultname=filename.replace('data','processing/unsatis_const')+'.csv'
FINAL.to_csv(resultname,header=True, index=False)
```

Result

- 결과 표 그림으로 삽입 or 표로..

Conclusion

- 장점
- 특징
- 한계점