CLT_solver_LR

DA lab.

UNIST

Input 형식

0. 데이터프레임 column 정보

- **[obj.index]** objective variable index
- **[const.index]** constraint index
- **[obj.var]** objective variable coefficient
- **[dir]** direct of constraint
- **[rhs]** right hand side value for constraint
- **[type]** variable type
- **[obj.lo]** objective variable lowerbound
- **[obj.up]** objective variable upperbound

1. 기본 문제 형태 (problem_input)

- xxx.mps 정보를 다음 데이터프레임 형태로 변형하여 사용

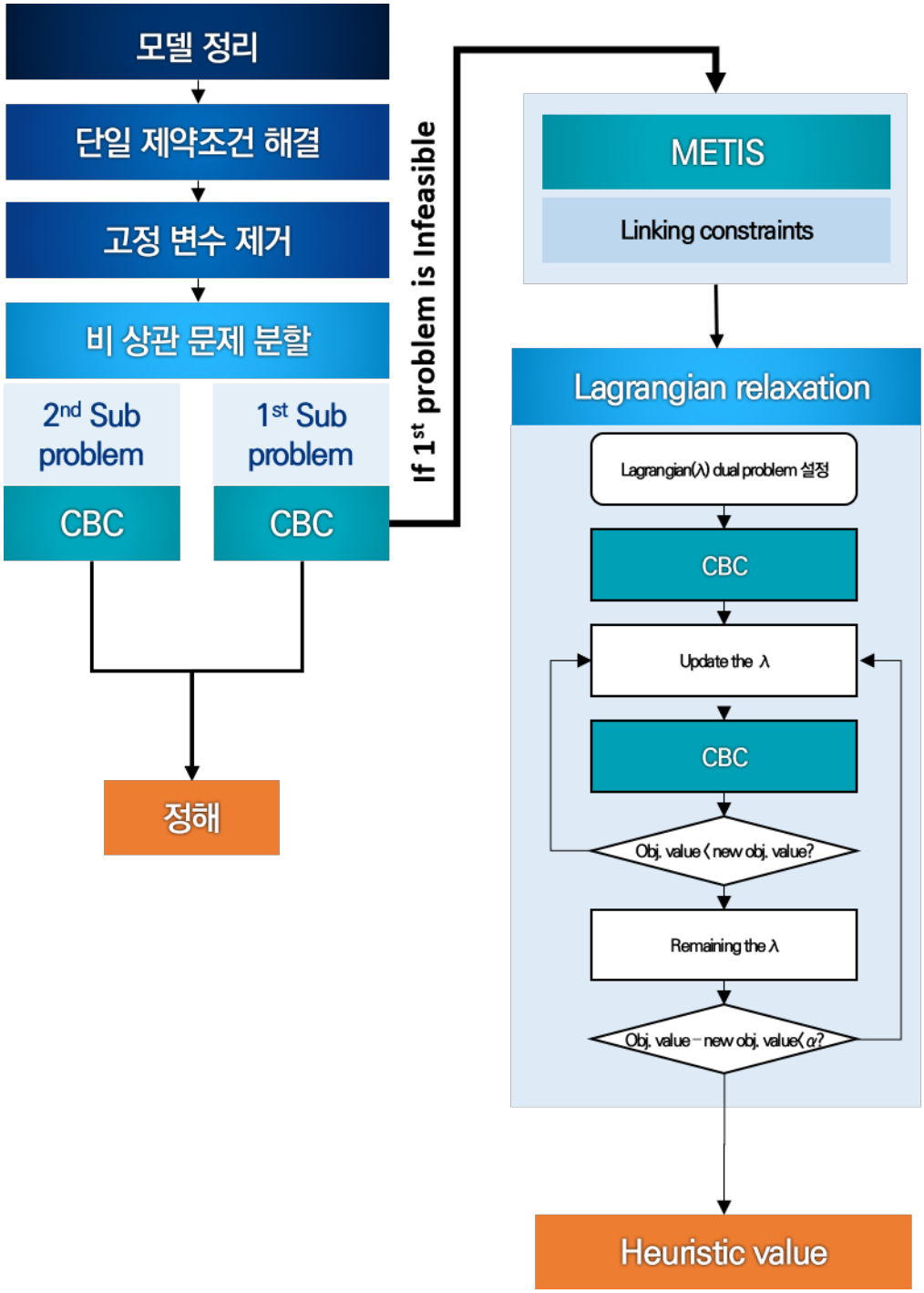
	obj.index	const.index	obj.var	dir	rhs	types	obj.lo	obj.up
1	1	21	-1	<=	1	I	0	100
2	2	22	-1	<=	1	I	0	100
3	3	23	-1	<=	1	I	0	100
4	4	24	-1	<=	1	I	0	100
5	5	25	-1	<=	1	I	0	100
6	6	26	-1	<=	1	I	0	100
7	7	27	-1	<=	1	I	0	100
8	8	28	-1	<=	1	I	0	100
9	9	29	-1	<=	1	I	0	100
10	10	30	-1	<=	1	I	0	100

2. solution 형태 (solution)

- objective variable의 해는 다음 데이터프레임 형태로 사용

	obj.index	obj.sol
1	8528	0
2	8645	0
3	8647	0
4	8649	0
5	18303	1
6	18304	1
7	18305	1
8	18306	1
9	18307	1
10	18308	1

알고리즘 모식도


모델 정리
단일 제약조건 해결
고정 변수 제거
비 상관 문제 분할
2nd Sub problem
1st Sub problem
CBC
CBC
정해
If 1st problem is Infeasible
METIS
Linking constraints
Lagrangian relaxation
Lagrangian(λ) dual problem 설정
CBC
Update the λ
CBC
Obj. value < new obj. value?
Remaining the λ
Obj. value − new obj. value< α?
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
```

Function

1.

```
def cbc_solve(model, time_limit = TimeLimit, tol_limit = Tol, 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
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
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

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

    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

## Check fixed value variables
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)

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=

    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.fl
            VARIABLES_df_p['v.ub'][np.where(VARIABLES_df_p['v.idx'] == c['ind'])] = min(VARIABLES_df_p[
        elif c['sense'] == '>':
            lb_upd = min(VARIABLES_df_p['v.ub'][np.where(VARIABLES_df_p['v.idx'] == c['ind'])], math.ce
            VARIABLES_df_p['v.lb'][np.where(VARIABLES_df_p['v.idx'] == c['ind'])] = max(VARIABLES_df_p[

    cnt_i = len(con_group_I)

    return(VARIABLES_df_p, CONSTRAINTS_df_p, cnt_s, cnt_i)


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)

def LR_lambda (linking_const,LAMBDA):
        const_list=[]
        sense_list=[]
        rhs_list=[]
        for i in range(len(linking_const)):
            sense_list.append(CONSTRAINTS_df[np.isin(CONSTRAINTS_df['c.idx'], linking_const[i])]['sense
            const_list.append(CONSTRAINTS_df[np.isin(CONSTRAINTS_df['c.idx'], linking_const[i])]['c.idx
            rhs_list.append(CONSTRAINTS_df[np.isin(CONSTRAINTS_df['c.idx'], linking_const[i])]['rhs'][1

        lam_df_dtype = np.dtype([('c.idx', int), ('lambda', float), ('rhs', float),('sense', object)])
        CONSTRAINTS_lambda=[(const_list[i],LAMBDA[i],rhs_list[i],sense_list[i]) for i in range(len(link
        CONSTRAINTS_lambda = np.array([(c[0], c[1], c[2], c[3]) for c in CONSTRAINTS_lambda], dtype = la
    #    CONSTRAINTS_lambda[np.where(CONSTRAINTS_lambda['sense']=='>')]['lambda']= -CONSTRAINTS_lambda[
        index=np.where(CONSTRAINTS_lambda['sense']=='<')[0]
        for i in index:
            CONSTRAINTS_lambda[i]['lambda']=-CONSTRAINTS_lambda[i]['lambda']
        return(CONSTRAINTS_lambda)


def LR_linked_variable (CONSTRAINTS_lambda):
    LR_var=[]
    for j in range(len(CONSTRAINTS_lambda)):
        LR_0=CONSTRAINTS_df[np.isin(CONSTRAINTS_df['c.idx'], CONSTRAINTS_lambda['c.idx'][j])]
        LR_value=(LR_0['val']*CONSTRAINTS_lambda['lambda'][j])
        LR_ind=LR_0['ind']
        LR_var.append([(LR_ind[i],LR_value[i]) for i in range(len(LR_value))])
    LR_df_dtype = np.dtype([('v.idx', int), ('LR', float)])
    VARIABLES_lambda = []
    for item in LR_var:
        for i in item:
            [VARIABLES_lambda.append(i)]

    VARIABLES_lambda = np.array([(c[0], c[1]) for c in VARIABLES_lambda], dtype = LR_df_dtype)
```

```
    VARIABLES_lambda
    #np.split(VARIABLES_lambda, np.cumsum(np.unique(VARIABLES_lambda['v.idx'], return_counts=True)[1])[
    unique_groups = np.unique(VARIABLES_lambda['v.idx'])
    sums = []
    for group in unique_groups:
        sums.append(VARIABLES_lambda[VARIABLES_lambda['v.idx'] == group]['LR'].sum())
    lam_var_df_dtype = np.dtype([('v.idx', int), ('LR', float)])
    LR_var_df=[(unique_groups[i],sums[i]) for i in range(len(sums))]
    LR_var_df = np.array([(c[0], c[1]) for c in LR_var_df], dtype = lam_var_df_dtype)
    return(LR_var_df)


def LR_solver(VARIABLES_df_p_decomp1,CONSTRAINTS_df_p_decomp1,LR_update_variable):
    CONSTRAINTS_LR_update=CONSTRAINTS_df_p_decomp1[np.isin(CONSTRAINTS_df_p_decomp1,linking_const)==Fals
    VARIABLES_LR_update=VARIABLES_df_p_decomp1.copy()
    METIS_LR_variable=VARIABLES_LR_update[np.isin(VARIABLES_LR_update['v.idx'], LR_update_variable['v.id
    for i in range(len(METIS_LR_variable['v.idx'])):
        index=np.where(VARIABLES_LR_update['v.idx']==METIS_LR_variable['v.idx'][i])[0][0]
        new_variable_LR=(VARIABLES_LR_update[index]['v.obj']-LR_update_variable[LR_update_variable['v.id
        VARIABLES_LR_update[index]['v.obj']=new_variable_LR

    CONSTRAINTS_df_p_np_decomp1=to_mps_format(CONSTRAINTS_LR_update)
    model_LR = array_to_mps(VARIABLES_LR_update, CONSTRAINTS_df_p_np_decomp1)
    model_LR.store_search_progress_log = True
    model_LR.max_gap = 0.05
    model_LR.integer_tol = 0.01

    startTime = time.time()
    model_p1.preprocess = True
    status = model_LR.optimize(max_seconds=60)
    endTime = time.time() - startTime

    cbc_solve(model_LR,  time_limit = 60)
    obj_value = []
    sol_list=[]
    if status.name=='INFEASIBLE':
        print('Infeasible')
    else:
        for v in model_LR.vars:
            obj_value.append(v.obj * v.x)
            sol_list.append((VARIABLES_name[v.name], v.x))
    lam_sol_dtype = np.dtype([('v.idx', int), ('sol', float)])
    sol_list = np.array([(c[0], c[1]) for c in sol_list], dtype = lam_sol_dtype)

    ZLK=sum(obj_value)+sum(CONSTRAINTS_lambda['lambda']*CONSTRAINTS_lambda['rhs'])
    return(ZLK, sol_list)

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)
```

Final code

1. Input

- FileName : mps 파일 이름 (예: R181204001_1.mps)
- MAXITER : GA 알고리즘에서 반복할 세대 수 (예: 100)
- POPSIZE : 한 세대에서 만드는 solution 개 수 (예: 100)
- option : 전처리 유무 선택 (예: 1)
    - 1: bound reduction 적용
    - 2: original GA)

2. Output

- SOLUTION_fin : 'Input 형식'의 solution 형태로 도출
- 최종 결과물은 싸이버로지텍에서 요구한 형식에 맞춰 xxx.sol로 저장됨

3. 실행 방법

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

```
Rscript "CLT_solver_GA.R" FileName MAXITER POPSIZE option
```

4. Main Code

```
#
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)
#file = cplex.Cplex(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'
#file.write(savename_cplex)
#model_cplex = cplex.Cplex(savename_cplex)

# 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'
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_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 = len(CONSTRAINTS_df_p[CONSTRAINTS_df_p['ind']==None])
#CONSTRAINTS_df_p = np.delete(CONSTRAINTS_df_p, np.where(CONSTRAINTS_df_p['ind']==None), axis = 0)
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()

CONSTRAINTS_df_p_decomp1,CONSTRAINTS_df_p_decomp2,VARIABLES_df_p_decomp1,VARIABLES_df_p_decomp2 = decomp
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 = TimeLimit)

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))


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.va
        VARIABLES = [(int(v.idx), float(v.obj), float(v.ub), float(v.lb), str(v.var_type)) for v in m.va

        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()

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_METI

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

CONSTRAINTS_METIS_0=CONSTRAINTS_METIS_0[np.isin(CONSTRAINTS_METIS_0['c.idx'],linking_const)==Fal
CONSTRAINTS_METIS_1=CONSTRAINTS_METIS_1[np.isin(CONSTRAINTS_METIS_1['c.idx'],linking_const)==Fal
print(len(CONSTRAINTS_METIS_0))
print(len(CONSTRAINTS_METIS_1))
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')
        #result3=cbc_solve(model_p1,  time_limit = TimeLimit)

        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)

        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

            linking_const_lhs = []
            for c in linking_var:

                linking_const_lhs.append(SOLUTION[VARIABLES_idx[c]]*linking_const_set[np.isin(linking_c

            if linking_const_set['sense'][0] == '<':
                const_stat.append((sum(linking_const_lhs)+np.array(linking_const_set['rhs'][0], dtype =
                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 =
                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], dtyp
                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)

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

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

            const_satis_ratio
            print(num_const_unsatis)
            print('The number of unsatisfied constraint:{}'.format(num_const_unsatis))

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, result
              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

- 결과 요약
- 한계점