CLT_solver_GA

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

Library

만약 패키지가 깔려 있지 않다면, 다음 코드를 실행해 주세요.

```
install.packages('GA')
install.packages('lpSolve')
install.packages('Rglpk')
install.packages('rcbc')
install.packages('dplyr')
install.packages('Matrix')
install.packages('Rsymphony')
install.packages('stringr')
install.packages('reshape2)
```

```
library('GA')
library('Rglpk')
library('lpSolve')
library('Rsymphony')
library('rcbc')
library('dplyr')
library('Matrix')
library('stringr')
library('reshape2')
```

Function

1. fitness : GA알고리즘 중 fitness function

- Deep et all.(2009)의 GA fitness function 활용
  - Deep, Kusum, et al. "A real coded genetic algorithm for solving integer and mixed integer optimization problems." Applied Mathematics and Computation 212.2 (2009): 505-518.
- fitness function은 다음과 같다.

$$f(n) = \begin{cases} f(X_i), & \text{if } X_i \text{ is feasible;} \\ f_{worst} + \sum_{j=1}^{m} |\phi_j(X_i)|, & \text{otherwise;} \end{cases}$$

```
f<-function(x) {
  x<-Matrix(x,ncol = 1,sparse = T)
  t(x)%*%Matrix(obj,sparse = T)
}

fitness<-function(x){
  S<-const_x_1%*%Matrix(x,sparse = T)
  const<-cbind(S, const_rhs)
  penalty<-sum(abs(const[which(const[,1]>const[,2]),][,1]))
  p<-ifelse(penalty==0, f(x), f(x)+penalty)
  return(-p)
}
```

2. solution_out

- 싸이버로지텍에서 요구한 solution 형식
  - 예: objective variable name^solution

```
solution_out<-function(i,soltuion){
  obj.val<-obj.val[order(obj.val$obj.index),]
  testset<-readLines(i)
  col.start<-which(testset=="COLUMNS")
  rhs.start<-which(testset=="RHS")
  col.set<-testset[(col.start+1):(rhs.start-1)]
  col.set.df<-data.frame(as.character(col.set))
  test<-str_trim(col.set.df[1:nrow(col.set.df),])

  if(length(grep("\t",test))==0){
    variable_0 <- colsplit(test, pattern=" ", names=c(1:3))$`1`
  }else{variable_0 <- colsplit(test, pattern="\t", names=c(1:3))$`1`}

  variable<-unique(variable_0[-(grep("MARK",variable_0))])
  solution<-obj.val$obj.sol


  sol<-(paste(variable,'^',solution,sep=''))
  filename<-paste(substr(i,1,(nchar(ExampleSet[i])-3)),'sol',sep = "")
  write(sol,filename )
  return(length(variable)==length(solution))
}
```

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_GA_solver.R" FileName MAXITER POPSIZE option
```

4. Main Code

```
# Step 0: read file
# Input
args <- commandArgs(TRUE)
FileName <- args[1]
MAXITER= as.numeric(args[2])
POPSIZE= as.numeric(args[3])
option = args[4]
x <- Rglpk_read_file( FileName, type = "MPS_free")

if (option == 1){
  obj <- as.matrix(x$objective)
  simple_triplet_matrix_sparse <- x$constraints[[1]]
  mat <- sparseMatrix(i=simple_triplet_matrix_sparse$i,
                      j=simple_triplet_matrix_sparse$j,
                      x=simple_triplet_matrix_sparse$v,
                      dims=c(simple_triplet_matrix_sparse$nrow,
                             simple_triplet_matrix_sparse$ncol))
  dir <- x$constraints[[2]]
  rhs <- x$constraints[[3]]
  bound<-x$bounds
  types<-x$types
  max <- x$maximum
```

```r
# Step 1: Bound reduction
bound<-list()
bound$lower$ind<-seq(1,dim(obj)[1])
bound$upper$ind<-seq(1,dim(obj)[1])
lower_bound<-x$bounds$lower$val
upper_bound<-x$bounds$upper$val
upper_bound[which(upper_bound==Inf)]<-10000000000

reduce_bound<-0.5
bound$lower$val<-lower_bound
bound$upper$val<-upper_bound

types<-rep("C",dim(obj)[1])
sol<-Rsymphony_solve_LP(obj=obj, mat=mat, dir=dir, types = types,
                        rhs=rhs,bound=bound,max=F,time_limit=10)
sol.2<-sol
sol$objval
sol$status

t<-which(upper_bound>10000)
for(i in 1:50){
  i<-i+1
  if(sol$status==0){ #feasible
    while (sol$status==0&(length(upper_bound[t])!=0)) {
      t<-which(upper_bound>10000)
      upper_bound[t]<-upper_bound[t]*reduce_bound
      upper_bound[t]
      bound$lower$val<-lower_bound
      bound$upper$val<-upper_bound

      types<-rep("C",dim(obj)[1])
      sol<-Rsymphony_solve_LP(obj=obj, mat=mat, dir=dir, types = types,
                              rhs=rhs,bound=bound,max=T,time_limit=10)
      sol.2<-sol
      sol$objval
      sol$status
      print("feasible")
    }
  }else{
    while(sol$status!=0){
      upper_bound[t]<-upper_bound[t]*(1+reduce_bound^2)
      upper_bound[t]
      bound$lower$val<-lower_bound
      bound$upper$val<-upper_bound

      types<-rep("C",dim(obj)[1])
      sol<-Rsymphony_solve_LP(obj=obj, mat=mat, dir=dir, types = types,
                              rhs=rhs,bound=bound,max=T,time_limit=10)
      sol.2<-sol
      sol$objval
      sol$status
      print("infeasible")
    }
```

```
  }
}

t<-which(upper_bound>1000)
for(i in 1:50){
  i<-i+1
  if(sol$status==0){ #feasible
    while (sol$status==0&(length(upper_bound[t])!=0)) {
      t<-which(upper_bound>1000)
      upper_bound[t]<-upper_bound[t]*reduce_bound
      upper_bound[t]
      bound$lower$val<-lower_bound
      bound$upper$val<-upper_bound

      types<-rep("C",dim(obj)[1])
      sol<-Rsymphony_solve_LP(obj=obj, mat=mat, dir=dir, types = types,
                              rhs=rhs,bound=bound,max=T,time_limit=10)
      sol.2<-sol
      sol$objval
      sol$status
      print("feasible")
    }
  }else{
    while(sol$status!=0){
      upper_bound[t]<-upper_bound[t]*(1+reduce_bound^2)
      upper_bound[t]
      bound$lower$val<-lower_bound
      bound$upper$val<-upper_bound

      types<-rep("C",dim(obj)[1])
      sol<-Rsymphony_solve_LP(obj=obj, mat=mat, dir=dir, types = types,
                              rhs=rhs,bound=bound,max=F,time_limit=10)
      sol.2<-sol
      sol$objval
      sol$status
      print("infeasible")
    }
  }
}

t<-which(upper_bound>100)
for(i in 1:50){
  i<-i+1
  if(sol$status==0){ #feasible
    while (sol$status==0&(length(upper_bound[t])!=0)) {
      t<-which(upper_bound>100)
      upper_bound[t]<-upper_bound[t]*reduce_bound
      upper_bound[t]
      bound$lower$val<-lower_bound
      bound$upper$val<-upper_bound

      types<-rep("C",dim(obj)[1])
      sol<-Rsymphony_solve_LP(obj=obj, mat=mat, dir=dir, types = types,
```

```
                              rhs=rhs,bound=bound,max=T,time_limit=10)
      sol.2<-sol
      sol$objval
      sol$status
      print('feasible')
    }
  }else{
    while(sol$status!=0){
      upper_bound[t]<-upper_bound[t]*(1+reduce_bound^2)
      upper_bound[t]
      bound$lower$val<-lower_bound
      bound$upper$val<-upper_bound

      types<-rep("C",dim(obj)[1])
      sol<-Rsymphony_solve_LP(obj=obj, mat=mat, dir=dir, types = types,
                              rhs=rhs,bound=bound,max=F,time_limit=10)
      sol.2<-sol
      sol$objval
      sol$status
      print('infeasible')
    }
  }
}


# Step 2: GA
#if there is binary, then the bound is fixed
binary.position<-which(x$types=="B")
integer.position<-which(x$types=="I")

#LP relaxation
types<-rep("C",dim(obj)[1])
sol<-Rsymphony_solve_LP(obj=obj, mat=mat, dir=dir, types = types,
                        rhs=rhs,bound=bound,max=T,time_limit=10)
sol.2<-sol #lp solve benchmark
suggestedSol<-matrix((sol.2$solution),nrow = 1)
constraints<-Matrix(mat, sparse=T)
lesser<-which(dir=="<=")
greater<-which(dir==">=")
equal<-which(dir=="==")

const_lesser<-constraints[lesser,]
const_greater<- (-constraints[greater,])
const_equal<- rbind(constraints[equal,],(-constraints[equal,]))

const_x_1<-rbind(const_lesser,const_greater,const_equal)
const_x_1<-Matrix(const_x_1,sparse = T)
#const_x_1<-as(z,"dgCMatrix")
const_rhs<-Matrix(c(rhs[lesser],rhs[greater],rhs[equal],
                    rhs[equal]),ncol = 1, sparse = T)


GA <- ga(type = "real-valued",
```

```
            selection=ga_tourSelection,
            crossover =  gareal_laplaceCrossover,
            mutation = gareal_powMutation,
            suggestions = suggestedSol,
            fitness = fitness, lower = bound$lower$val, upper = bound$upper$val,
            popSize = POPSIZE,
            elitism = max(1, round(POPSIZE*0.1)),
            maxiter = MAXITER)

} else if (option==2){
  #case 2 (without any bound recustion)
  obj <- as.matrix(x$objective)
  simple_triplet_matrix_sparse <-  (x$constraints[[1]])
  mat <- sparseMatrix(i=simple_triplet_matrix_sparse$i,
                      j=simple_triplet_matrix_sparse$j,
                      x=simple_triplet_matrix_sparse$v,
                      dims=c(simple_triplet_matrix_sparse$nrow,
                             simple_triplet_matrix_sparse$ncol))
  dir <- x$constraints[[2]]
  rhs <- x$constraints[[3]]
  bound<-x$bounds
  types<-x$types
  max <- x$maximum

  # Step 1
  bound<-list()
  bound$lower$ind<-seq(1,dim(obj)[1])
  bound$upper$ind<-seq(1,dim(obj)[1])
  lower_bound<-x$bounds$lower$val
  upper_bound<-x$bounds$upper$val
  upper_bound[which(upper_bound==Inf)]<-10000000000
  bound$lower$val<-lower_bound
  bound$upper$val<-upper_bound
  # Step 2: GA
  #if there is binary, then the bound is fixed
  binary.position<-which(x$types=="B")
  integer.position<-which(x$types=="I")

  #LP relaxation
  types<-rep("C",dim(obj)[1])
  sol<-Rsymphony_solve_LP(obj=obj, mat=mat, dir=dir, types = types,
                          rhs=rhs,bound=bound,max=T,time_limit=10)
  sol.2<-sol #lp solve benchmark
  suggestedSol<-matrix((sol.2$solution),nrow = 1)
  constraints<-Matrix(mat, sparse=T)
  lesser<-which(dir=="<=")
  greater<-which(dir==">=")
  equal<-which(dir=="==")

  const_lesser<-constraints[lesser,]
  const_greater<- (-constraints[greater,])
  const_equal<- rbind(constraints[equal,],(-constraints[equal,]))
```

```
  const_x_1<-rbind(const_lesser,const_greater,const_equal)
  const_x_1<-Matrix(const_x_1,sparse = T)
  const_rhs<-Matrix(c(rhs[lesser],rhs[greater],rhs[equal],rhs[equal]),ncol = 1, sparse = T)


  GA <- ga(type = "real-valued",
           selection=ga_tourSelection,
           crossover =  gareal_laplaceCrossover,
           mutation = gareal_powMutation,
           suggestions = suggestedSol,
           fitness = fitness, lower = bound$lower$val, upper = bound$upper$val,
           popSize = POPSIZE,
           elitism = base::max(1, round(POPSIZE*0.1)),
           maxiter = MAXITER)
} else{
  print('Please select the option number([1]: Bound reduction GA [2]: Orginal GA)')
  }

print(summary(GA))
print(sum(GA@solution[1,]*obj))
obj.index<-seq(1:nrow(obj))
obj.sol<-as.factor(GA@solution)
SOLUTION_fin<-data.frame(cbind(obj.index,obj.sol))
solution_out("R181204001_1.mps", SOLUTION_fin)
```