最適化・シミュレーション演習 第4回 分枝限定法

```
model knapsack-relax.mod ;
data knapsack-relax.dat ;
option display_round 6;              #小数点以下 6 桁表示
option solver cplex;                 #ソルバを CPLEX に指定


param maxsubproblems;                #子問題の最大生成数
param size_activenode;               #未探索頂点(子問題)の個数

param parent {k in 1..maxsubproblems};                    #子問題 k の親問題番号
param var_branching {k in 1..maxsubproblems} ;            #子問題 k において選択された分枝変数
param fix_binary {k in 1..maxsubproblems} binary;         #子問題 k における分枝変数の値(0-1)
param ACTIVE_NODELIST {k in 1..size_activenode+2};        #未探索頂点(子問題)のリスト
param total_node_number;                                  #生成された頂点(子問題)の数

param epsilon;                                            #整数変数が 0-1 条件を満たしているか判定係数
param ancestor;                                           #各子問題に対応する頂点の先祖

param tmpprofit;                                          #暫定目的関数値
param tmpx {i in 1..Nitem} >=0;                           #暫定解の値
```

```
###### 初期設定 #####
        let maxsubproblems :=1000;
        let total_node_number :=1;
        let size_activenode :=total_node_number;
        let ACTIVE_NODELIST[size_activenode] :=total_node_number;

        let epsilon :=1e-5;
        let tmpprofit := -1e10;
```

repeat while (size_activenode >0 and total_node_number < maxsubproblems)　#分枝限定法の主要部分
{

```
##### 未分枝頂点、生成された問題情報などの表示 #####
printf: "********************************* ¥n" >bb-knapsack.sol;
printf: "Total number of subproblems %d ¥n", total_node_number >bb-knapsack.sol;
printf: "Current Problem %d ¥n", ACTIVE_NODELIST[size_activenode]   >bb-knapsack.sol;
for{i in 1..size_activenode}
{
        printf: "ACTIVE_NODELIST[%d]=%d ¥n", i, ACTIVE_NODELIST[i] > bb-knapsack.sol;
}
display size_activenode > bb-knapsack.sol;
```

```
###### 当該問題のデータ作成 #####
let ancestor :=ACTIVE_NODELIST[size_activenode] ; #未分枝頂点リストの最終要素を選択
for{i in 1..Nitem}                                    #0-1 変数の制限を初期化
{
        let l[i]:=0;
        let u[i]:=1;
}
repeat while (ancestor >1)          #選択した子問題の親が根でない限り、0-1 変数の固定を行う
{
        let   l[var_branching[ancestor]]:= fix_binary[ancestor];
        let   u[var_branching[ancestor]]:= fix_binary[ancestor];

        let ancestor :=parent[ancestor];   #選択した子問題の先祖ノードを探索
}
```

```
##### 子問題を解き、未分枝頂点リストから除く #####
solve;                                          #問題をソルバで解く
display solve_result >bb-knapsack.sol;
if (solve_result="solved") then
{
    display Profit >bb-knapsack.sol;
    display l, u, x> bb-knapsack.sol;
}
let size_activenode :=size_activenode-1 ;       #未分枝頂点リストの長さを 1 つ減少
```

```
##### 実行可能で非整数解が得られ、限定操作が適用されない場合 子問題へ分枝 #####
if ( (exists {i in 1..Nitem} x[i]*(1-x[i]) >= epsilon) and
    (solve_result="solved") and (Profit > tmpprofit)   ) then
{
        for{ii in 1..Nitem}
        {
            if ( x[ii]*(1-x[ii]) > epsilon ) then
            {
                printf: "INTEGER CONSTRAINT VIOLATED ¥n" >bb-knapsack.sol;
                printf: "Subproblem %d and %d generated ¥n",
                        total_node_number+1, total_node_number+2 >bb-knapsack.sol;

                let parent[total_node_number+1]:=ACTIVE_NODELIST[size_activenode+1];
                let parent[total_node_number+2]:=ACTIVE_NODELIST[size_activenode+1];

                let var_branching[total_node_number+1]:=ii;
                let fix_binary[total_node_number+1]:=1;
                let ACTIVE_NODELIST[size_activenode+1] :=total_node_number+1;

                let var_branching[total_node_number+2]:=ii;
                let fix_binary[total_node_number+2]:=0;
                let ACTIVE_NODELIST[size_activenode+2] :=total_node_number+2;
                let total_node_number:=total_node_number+2;
                let size_activenode :=size_activenode+2 ;
                break;                                  #for 文から脱出
             }
        }
}
```

```
##### 暫定解表示 #####
printf: "TEMPORALLY objective value             %lf ¥n", tmpprofit >bb-knapsack.sol;
if ((forall {i in 1..Nitem} x[i]*(1-x[i]) < epsilon) and
   (solve_result="solved") and (Profit > tmpprofit)   ) then
{
      let tmpprofit:=Profit;
      let {i in 1..Nitem} tmpx[i]:=x[i];
      printf: "TEMPORALLY objective value refined %lf ¥n", tmpprofit >bb-knapsack.sol;
}
```

}　#分枝限定法の主要部分（終わり）

```
##### 最終結果表示 #####
printf: "########################################## ¥n" >bb-knapsack.sol;
printf: "Branch and Bound FINISHED ¥n"   >bb-knapsack.sol;
display tmpprofit > bb-knapsack.sol;
display tmpx > bb-knapsack.sol;
```

```
quit;
```