Erc. Ünv. Fen Bil. Derg., 10, 1-2 (1994) 73-89

# An Algorithm for Integer Programming Problems : ( With Mathematica Coding )

**Mehmet ÇINAR**
Erciyes Üniversitesi İktisadi ve İdari Bilimler Fakültesi İşletme Bölümü
**Mehmet Emin YÜKSEL**
Erciyes Üniversitesi Mühendislik Fakültesi Elektronik Bölümü
**Ömer TAHTASAKAL**
AppleCenter Mega Bilgisayar Ltd. Genel Müdür

## ABSTRACT

In this study, by using different redefined the algorithms of cutting-plane and branch-and-bound method both and added some heuristic sub-routines, a new algorithm with Mathematica coding under Machintosh is introduced for solving all the problems based on integer programming.

For this purpose, types of the integer programming problems and related solution methods are discussed first. Then, a new algoritlum is introduced in a detailed sample and also is tested utilizing the IBM test problems. The results show that the new algorithm is efficint, easy to run and needs less computational effort than the other algorithms discussed in this paper.

## ÖZET

Bu çalışmada, kesme düzeyi ve dal-sınır yöntemlerini değişik bir yorumla birlikte kullanarak ve bazı sezgisel sub-routine eklemeleriyle, her türlü tamsayılı programlama problemlerinin çözümüne yönelik, Machintosh ortamında kullanılan Mathematica dili ile yazılmış yeni bir algoritma ve ilişkin kodlaması tanıtılmaktadır.

Bu amaçla, önce tamsayılı programlama problemi, türleri ve çözüm yöntemleri tartışılmış, ardısıra sözü edilen algoritma örnek uygulamalarla tanıtılmış ve IBM tarafından tamsayılı programlar için geliştirilen 10 test örnek problem üzerindeki çözümleri verilmiştir. DOS ortamında kullanılan LINDO paket programına, MOS alternatifi olarak geliştirilen yeni algoritma, opsiyonları gereği etkileşimli ve çok amaçlı olarak kullanılabilmekte ve hız açısından doyurucu görülmektedir.

## INTRODUCTION

In the linear programming (LP) it is allowed both the decision variables and the slack variables to assume nonnegative fractional values, or nonnegative integer values, in the optimal solution. In many of the poblem situations, fractional values for the decision variables were obtained as an optimal solution was determined; and these fractional values were acceptable and appropriate in the context of the problem being considered. In general, it is quite possible to use fractional amount of a resource to produce a fractional amount of a product.

However, there are also a number of important problem situations in which fractional answers are neither practical nor very meaningful. For example, consider the following production planning situation [1]. A firm manufactures three types of corporate jets. The selling prices for these jets are as follows;

Model A - $ 1,750,000, Model B - $ 2,000,000, Model C - $ 1,900,000. Yearly poduction of these jets is constrained by available work force, available machine time and capital availability. The production conditions for the problem situation can be expressed as follows:

| Constraint | Model A | Model B | Model C | Resource |
|---|---|---|---|---|
| Work force | 40 Workers | 65 Workers | 50 Workers | 150 Workers |
| Machine time | 6,000 Hours | 10,000 Hours | 8,000 Hours | 30,000 Hours |
| Capital | $ 1,500,000 | $ 1,000,000 | $ 1,500,000 | $ 1,250,000 |

The standard LP formulation of this problem would be;

Maximize Z = 1750000 x1 + 2000000 x2 + 3500000 x3

subject to :

$$40\,x1 + 65\,x2 + 50\,x3 <= 150$$

$$6000\,x1 + 10000\,x2 + 8000\,x3 <= 30000$$

$$1500000\,x1 + 1000000\,x2 + 1250000\,x3 <= 3500000$$

with $x1 >= 0$ $x2 >= 0$ $x3 >= 0$.

The LP solution to this problem is x1 = 2.32 airplanes, x3 = 1.14 airplanes, and value is Z = $ 6,234,474. However, from the nature of the problem it is apperant that we could not allow to manufacture of a part of an airplane. In this production scheduling situation the decision variables have relevance only if they have integer values.

In some practical applications, an integer solution to a particular LP problem can be obtained by simply "rounding off " the fractional values that appear in the optimal solution. Unfortunately, this rounding procedure may cause two difficulties. First, this

integer solution may not be feasible, particularly if some of the coefficients in the constraint set are negative. Second, even if the rounding off solution is feasible, it may not be the optimal solution. To illustrate the first problem, suppose that the constraint set for the problem is:

$x1 + x2 <= 22/3$

$x1 - x2 <= 2/3$

and the application of the simplex method has resulted in an optimal (noninteger) solution of $x1 = 4$, $x2 = 10/3$. Observe that we cannot round of x2 either 3 or 4 and maintain feasibility. Indeed, we can round off x2 only if we also change the integer value of x1. Obviously, rounding off becomes even more impractical as the number of constraints and variables increases.

The second problem, as noted above, is that even though a feasible integer solution can be obtained by rounding off, it may not be the optimal integer solution. To illustrate, consider the following problem formulation [2];

Maximize $Z = x1 + 4\ x2$

subject to: $x1 + 6\ x2 <= 18$

$x1 <= 3$

with $x1, x2 >= 0$, and all integers.

The noninteger solution for this problem is $x1 = 3$, $x2 = 2.5$, $Z = 13$. To obtain an integer solution to this problem by rounding off, we would obtain $x1 = 3$, $x2 = 2$, $Z = 11$. However, the optimal integer solution is $x1 = 0$, $x2 = 3$, $Z = 12$.

Practical applications of integer programming (IP) have become much more common in recent years. Because of this fact, and problems associated with simply rounding off linear solutions, there has arisen a need for an efficient solution procedure for IP problems.

There are basically three types of IP problems [3]:

1. An all-integer programming problem. All the decision variables are constrained to integer values.

2. A mixed-integer programming problem. Some, but not all, of the decision variables are constrained to integer values.

3. A zero-one integer programming problem. All of the decision variables are constrained to the integer values zero or one.

All these types of IP problems are couched within the general format of what would otherwise be a LP problem. That is, the objective function and constraint set of the problem are linear functions, with only the additional restriction that the decision vari-

ables must be solved for as integers.

There are three approaches to IP, namely:

1. Cutting-plane algorithm,
2. Branch and bound technique,
3. Balas' additive 0-1 algorithm.

## THE ALGORITHM

An algorithm for solving all-integer and mixed integer programming problems has been developed by Ralph E. Gomory [4]. In using this algorithm, the integer requirement is first relaxed and the resulting LP problem is solved in the usual manner. If all decision variables have integer values, then this current solution is also the solution to the corresponding IP problem. However, if the current LP solution does not have integer values for the decision variables, the original LP problem will be modified by adding a new constraint that eliminates some noninteger solutions (including the previously optimal noninteger LP solution), but which does not eliminate any feasible integer solutions.

**At this stage, our new algorithm considers only the decision variable has maximum fractional part instead of all variables had as a new additional constraint.**

Utilizing the new algorithm, the optimal integer solution will eventually be obtained after all the noninteger to the problem have been cut away. The key idea in the process is what we are searching for a new LP problem whose set of feasible integer solutions coincides which the set of feasible solutions for the IP problem that we are attempting to solve. If this optimal solution (that obtained by using simplex algorithm) has all integer values for the decision variables, it is the integer solution to the problem. If it does not, we add another new constraint to the current modified problem, by using branch-and-bound technique, and repeat the procedure.

**In the case of mixed-integer programming problem (there is no need being integer for some variables), only the variables those are restricted to integer values are taken into consideration. This situation will be identified in the "t" set as an input information.**

If the added new constraint makes the solution infeasible, proposed algorithm will use the branch-and-bound technique for further consideration. By this purpose, current added constraint will be dropped, and the remaining modified constraints set will be considered. Subsequently, the current value of the objective function determined with simplex is the current bound. The next step in the solution is to partition the set of feasible solutions into many subsets by the same manner. Those subsets whose bounds violate the current bound or the new constraint that does not satisfy the model

solution, are said to be fathomed and are excluded from further consideration.

At this step of the solution process, one of the remaining subsets, namely the one with the smallest lowest bound in the minimizing case (largest upper bound in the maximization), is partitioned further into several subsets. This process is then repeated until a feasible solution is found such that the corresponding value of the objective function is no greater (less) than the lower (upper) bound for any subset.

Several alternative rules have been suggested for the branch-and-bound steps. The most popular branch rule for selecting a subset to partition is the best bound rule [5]. This rule says to select the subset having the most favorable bound (the smallest lower bound in the case of minimization) because this subset would seem to be most promising in terms of containing an optimal solution.

**We used the newest bound rule which selects the most recently created subset that has not been fathomed, breaking a tie between subsets created at the same time by selecting the one with the most favorable bound.**

Many IP problems have the added feature that all the integer variables are restricted to two values; zero or one. This frequently occurs as the integer decision variable is used to indicate whether possible action is to be undertaken (xi = 1) or not (xi = 0), where the level of activity represents a fixed allocation of resources. For example, portfolio selection, site selection, and assignment of certain jobs to certain machines, all have a zero-one integer structure.

**Utilizing the new algorithm, optimal solution will also be obtained as follows. If the case is a pure 0-1 IP problem, it will generate all feasible solution alternatives with zero and one. Then, selects the alternatives whose satisfy the original constraints set. After calculation of the results for the objective function, the algorithm selects the best one.**

**In the mixed zero-one IP problem case, all the same with the mixed-IP but, in the branching step a partial solution may be selected for partitioning, and if so, it is then partitioned into two new subsets by setting xi=1 and xi=0. The newest bound rule is used to make this selection.**

The new algorithm for solving IP maximization models can be summerized as follows:

**Step 1. Initialization.** Input data for objective function [**func**], constraints set [**cons**], all the decision variables [**vars**], variables to be integrized [**t**], variables to be equalized to zero or one [**z0**]. z0 is option for the mixed 0-1 IP models. If **t** is null, the algorithm accepts that all the variables are to be integrized. Solve the model without integer restrictions using standard LP model [**ConstrainedMax**].

**Step 2. Calculate the fractional parts of the variables to be integerized.** Call [**pref**]. If the optimal solution has all integer variables, stop. If not, set **aa1** to the value

of the objective function, **bb1** to minus infinity.

**Step 3. Partition the subset with the largest upper bound.** Select the noninteger variable with higher fractional part **[Branches]** from the LP solution to the subset of feasible solution **k** with the largest **aa** or **bb**.

**Step 4. Find an upper bound on each newly created subset.** Call **[Bound]**. The upper bound for each subset is found by solving the modified LP model.

**Step 5. Fathom subsets where possible.** For any subset **k**, fathom it from further consideration if;

1. aa <= bb, or

2. Subset k contains no feasible solutions, or

3. The best feasible integer solution in subset **k** has been found. If aa > bb, reset aa=bb and retain the solution as the best one found so far.

**Step 6. Stop the procedure.** When all subsets have been fathomed, the integer solution is the one that corresponds to the current maximum value of the solution set **[sevT]**. If all subsets have not been fathomed, return to Step 2 and continue the procedure.

In the minimization case, the new algorithm takes func= -1 func and bb1= + infinity.

For more details about the proposed algorithm can be seen in the program coding list at the Appendix.

## TESTING THE ALGORITHM

**Let's solve the problem below.**

Max[50x1 + 30x2]

s.t. 6x1 + 13x2 <= 70

5x1 + 2x2 <=33

x1, x2 = All integer and positive.

**In order to use the proposed algorithm, first, we have to define inputs as follows:**

func= 50x1 + 30x2; cons = {6x1 + 13x2 <= 70, 5x1 + 2x2 <=33}; vars = {x1, x2}; t = {1, 2}; sıra = 0; sevT = {};

**As the first step, we will find the linear solution for the problem.**

```
In[1] = : son=N[ConstrainedMax[func,cons,vars]];

ilk=son;sev=son;Print["linear solution=",son];
```

values= vars /. son[[2]];val1=values;

Print["values=",values]

**Out[1] = :** linear solution={358.679, {x1 -> 5.45283, x2 -> 2.86792}}

values={5.45283, 2.86792}

**Then, search for the fraction part of the variables.**

**In[2] = :** d=Pref[values];Print["d=",d];

If[Max[d]==0.,Print["solution=",Last[sevT]];Break[]];

**Out[2] = :** d={0.45283, 0.867925}

**Choose the bigger fraction which is beneficial if the variable is integerized.**

**In[3] = :** tt1=e;tt1=t[[tt1]]

**Out[3] = :** 2

**Branch the subset for chosen variable.**

**In[4] = :** Branche:[tt1,cons];

**Out[4] = :** OPT=350.

const={6 x1 + 13 x2 <= 70, 5 x1 + 2 x2 <= 33, x2 <= 2.}

x1=5.8

x2=2.

OPT=348.333

x1=5.16667

x2=3.

**Take the most promising one. At this stage, constraints set of the problem will be as follows.**

**In[5] = :** Bound[aa,bb];

**Out[5] = :** *aa*=350.

**Check, whether all the variables are integer or not.**

**In[6] = :** d=Pref[values];Print["d=",d];

If[Max[d]==0.,Print["solution=",Last[sevT]];Break[]];

**Out[6] = :** d={0.8, 0.}

**If the variables are not integerized, Return the first step.**

**In[7] = :** tt1=e;tt1=t[[tt1]]

**Out[7] = :** 1

**In[8] = :** Branches[tt1,cons];

**Out[8] = :** OPT=310.

const={6 x1 + 13 x2 <= 70, 5 x1 + 2 x2 <= 33, x2 <= 2., x1 <= 5.}

x1=5.

x2=2.

OPT=345.

x1=6.

x2=1.5

**In[9] = :** Bound[aa,bb];

**Out[9] = :** *bb*=345.

**In[10] = :** d=Pref[values];Print["d=",d];

If[Max[d]==0.,Print["solution=",Last[sevT]];Break[]];

**Out[10] = :** d={0., 0.5}

**In[11] = :** tt1=e;tt1=t[[tt1]]

**Out[11] = :** 2

**If the branch does not satisfy the constraints, fathom it.**

**In[12] = :** Branches[tt1,cons];

**Out[12] = :** OPT=340.

const={6 x1 + 13 x2 <= 70, 5 x1 + 2 x2 <= 33, x2 <= 2., x1 >= 6., x2 <= 1.}

x1=6.2

x2=1.

OPT=(FATHOMED)

x1=5.16667

x2=3.

**In[13] = :** Bound[aa,bb];

**Out[13] = :** *bb*=348.333

**In[14]** = : d=Pref[values];Print["d=",d];

If[Max[d]==0.,Print["solution=",Last[sevT]];Break[]];

**Out[14]** = : d={0.166667, 0.}

**In[15]** = : tt1=e;tt1=t[[tt1]]

**Out[16]** = : 1

**In[17]** = : Branches[tt1,cons];

**Out[17]** = : OPT=342.308

const={6 x1 + 13 x2 <= 70, 5 x1 + 2 x2 <= 33, x2 >= 3., x1 <= 5.}

x1=5.

x2=3.07692

OPT=(FATHOMED)

x1=5.

x2=3.07692

**In[18]** = : Bound[aa,bb];

**Out[19]** = : *bb*=342.308

**In[20]** = : d=Pref[values];Print["d=",d];

If[Max[d]==0.,Print["solution=",Last[sevT]];Break[]];

**Out[20]** = : d={0., 0.0769231}

**In[21]** = : tt1=e;tt1=t[[tt1]]

**Out[21]** = : 2

**In[22]** = : Branches[tt1,cons];

**Out[22]** = : OPT=340.

const={6 x1 + 13 x2 <= 70, 5 x1 + 2 x2 <= 33, x2 >= 3., x1 <= 5., x2 <= 3.}

x1=5.

x2=3.

OPT=270.

x1=3.

x2=4.

```
In[23] = : d=Pref[values];Print["d=",d];
If[Max[d]==0.,Print["solution=",Last[sevT]];Break[]];
Out[23] = : d={0., 0.}
solution={340., {x1 -> 5., x2 -> 3.}}
```

**At this moment, fesible solutions are;**

```
In[24] = : sevT
Out[24] = : {{270., {x1 -> 3., x2 -> 4.}}, {310., {x1 -> 5., x2 -> 2.}},
 {340., {x1 -> 5., x2 -> 3.}}}
```

**This process continues until the last subset is performed. Then, optimal solution will be obtained among the solution alternatives where they lie in sevT. If the solution set is null, consider the variable has second highest fraction, and repeat the algorithm. If it still null, print "optimal solution is not found".**

Solutions for the IBM Test Problems I to X :

```
In[25] = : IntegerMax[ x3+x4+x5,
{20x1+30x2+x3+2x4+2x5<=210,
30x1+20x2+2x3+x4+2x5<=210,
-90x1+x3<=0,
-90x2+x4<=0},{x1,x2,x3,x4,x5},{1,2,3,4,5}]
Out[25] = : {106., {x1 -> 1., x2 -> 1., x3 -> 54., x4 -> 52., x5 -> 0.}}
In[26] = : IntegerMax[ x3+x4+x5,
{20x1+30x2+x3+2x4+2x5<=180,
30x1+20x2+2x3+x4+2x5<=150,
-60x1+x3<=0,
-75x2+x4<=0},{x1,x2,x3,x4,x5},{1,2,3,4,5}]
Out[26] = : {76., {x1 -> 1., x2 -> 1., x3 -> 24., x4 -> 52., x5 -> 0.}}
In[27] = : IntegerMax[ x1+2x2+x3+2x4,
{x1+0.75x2<=12,
1.25x3+1.33x4<=14,
x1>=2.,x2>=3.,x3>=2.,x4>=3.},{x1,x2,x3,x4},{3,1,2,4}]
```

```
Out[27] = : {46., {x1 -> 2., x2 -> 13., x3 -> 2., x4 -> 8.}}

In[28] = : IntegerMax[ 50x1+30x2,
{6x1+13x2<=70,
5x1+2x2<=33},{x1,x2},{1,2}]

Out[28] = : {340., {x1 -> 5., x2 -> 3.}}

In[29] = : IntegerMax[ x7+x8+x9+x10+x11+x12,
{9x1+7x2+16x3+8x4+24x5+5x6+3x7+7x8+8x9+4x10+6x11+5x12<=110,
 12x1+6x2+6x3+2x4+20x5+8x6+4x7+6x8+3x9+1x10+5x11+5x12<= 95,
 15x1+5x2+12x3+4x4+4x5+5x6+5x7+5x8+6x9+2x10+1x11+8x12<= 80,
 18x1+4x2+4x3+18x4+28x5+x6+6x7+4x8+2x9+9x10+7x11+1x12<=100,
-12x1+x7 <= 0, -15x2+x8<= 0, -12x3+x9<= 0, -10x4+x10<= 0, -11x5+x11<= 0,
-11x6+x12 <= 0}, {x1,x2,x3,x4,x5,x6,x7,x8,x9,x10,x11,x12},{11,8,9,10,7,12}]

Out[29] = : {16., {x1 -> 0., x2 -> 0., x3 -> 0., x4 -> 0.19, x5 -> 0.63, x6 -> 0.63, x7 ->
0., x8 -> 0., x9 -> 0., x10 -> 2., x11 -> 7., x12 -> 7.}}

In[30] = : IntegerMax[ -2000x1-3500x2,{50000x1+30000x2>=900000,
      10x1+15x2>=300,x1<=22,x2<=25},{x1,x2},{2,1}]

Out[30] = : {-63000., {x1 -> 21., x2 -> 6.}}

In[31] = : IntegerMax[ 3x1+20x2,
{x1+8x2<=32,
2x1+x2<=14},{x1,x2},{1,2}]

Out[32] = : {80., {x1 -> 0., x2 -> 4.}}

In[33] = : IntegerMin[ 1600x1+3200x2+2300x3 ,
{ 3x1 + x2 + 6x3 == 2000,
2x1 + 5x2 +x3 >= 1000,
x1 + 2x2 + 4x3 <= 3000},{x1,x2,x3},{1,2,3}]

Out[33] = : {969400., {x1 -> 438., x2 -> 2., x3 -> 114.}}

In[34] = : IntegerMax[ -(4x1+5x2),
{3x1+x2-x3==2,
x1+4x2-x4==5,
```

```
3x1+2x2-x5==7},{x1,x2,x3,x4,x5},{1,2}]
Out[34] = : {-13., {x1 -> 2., x2 -> 1., x3 -> 5., x4 -> 1., x5 -> 1.}}
In[35] = : IntegerMin[ 13x1+15x2+14x3+11x4,
{4x1+5x2+3x3+6x4>=96,
20x1+21x2+17x3+12x4>=200,
11x1+12x2+12x3+7x4>=101},{x1,x2,x3,x4},{4,2,3,1}]
Out[35] = : {187., {x1 -> 0., x2 -> 0., x3 -> 0., x4 -> 17.}}
```

## CONCLUSION

In this study, we have presented a new algorithm to solve integer programming models. As it is known, most practical models involve many more variables and constraints. As such, computer codes are required to solve these models. Because of the integer restrictions on the variables, these codes are much less efficient than the simplex method. Usually, significant computer time is needed to solve these models. Consequently, applications to date have involved models with only a limited number of integer variables. Most codes are based on branch-and-bound method because the cutting plane method, which has met with only limited success so far, is too dependent on the model structure. Nonetheless, research is continuing in an effort to find more efficient methods based on these two approaches.

Another approach is to solve the model without integer restrictions using the simplex method and then round off the variables to the nearest integer value such that the constraints are not violated. For large models, this procedure may not be simple because of all the possible combinations of values for the decision variables. In addition, it cannot be assured that the optimal solution will be found.

Finally, a significance amount of research is being devoted to the development of heuristic procedures for solving IP models. These procedures cannot guarantee an optimal solution, but they promise to be superior to the use of inspection or rounding off and should take less computational effort than the cutting-plane and branch-and-bound methods.

The proposed heuristic algorithm coded by Mathematica [6,7,8,9] is tested utilizing the IBM test problems. The results show that the new algorithm is efficient, easy to run and needs less computational effort than most of the algorithms discussed in this paper.

## REFERENCES

[1]. R. E. Markland, **Topics in Management Science.** John Wiley and Sons Inc., New York (1981)

[2]. L. J. Krajewski and H. E. Thompson, **Management Science: Quantitative Methods in Context.** John Wiley and Sons Inc., Toronto (1981)

[3]. A. H. Taha, **Integer Programming: Theory, Applications, and Computations.** Academic Press Inc., New York (1975)

[4]. R. E. Gomory, **" An Algorithm for Integer Solutions to Linear Programs ", in** R. L. Graves and P. Wolfe **(eds.), Recent Advances in Mathematical Programming,** McGraw-Hill Book Company, New York (1963)

[5]. F. S. Hillier and G. J. Lieberman, **Introduction to Operation Research.** McGraw-Hill Book Company, New York (1990)

[6]. R. E. Crandall, **Mathematica for the Sciences.** Addison-Wesley Publishing Company Inc. (1991)

[7] R. Maeder, **Programming in Mathematica.** Addison-Wesley Publishing Company Inc. (1990)

[8]. S. Skiena, **Implementing Discrete Mathematics: Combinatorics and Graph Theory in Mathematica.** Addison-Wesley Publishing Company Inc. (1990)

[9]. S. Wolfram, **Mathematica : A System for Doing Mathematics by Computer. 2nd Ed.** Addison-Wesley Publishing Company Inc. (1991)

## Appendix. Program List of Integer Programming.

```
BeginPackage["IntegerProg`"]
IntegerMin::usage=""
IntegerMax::usage="GİRİŞ SERBEST SIRADA"
Begin["`Private`"]
Compile[IntegerMax[f_,c_,v_,t1_]:=
    Block[{func=f,cons=c,vars=v,t=t1,values,sev,sev1},
    Off[ConstrainedMax::nbdd];Off[ConstrainedMax::nanopt];
    Off[ConstrainedMax::nonl];Off[ConstrainedMax::nsat];
    Off[Replace::rep];sev={};const={};const1={};sevT={};const={};sev1={};lopt=0;
    st=1;con=cons;iter=0;a=1;le=0;sıra=0;lop=0;aa=1;bb=2;lkc=Length[cons];
    ll=Length[vars];
    If[t==Null,t={};tm=0;
    Do[tm=tm+1;t=Append[t,tm],{ll}]];
    t2=t;t=Sort[t];
    son=N[ConstrainedMax[func,cons,vars]];ilk=son;sev=son;
    values= vars /. son[[2]];val1=values;
    Bal[values,t];
    If[bk==2,Print[son];Break[]];
    If[bk==3,Print["Integer solution iş not found"];
    Break[]];
    For[i=1,i!=2,i++,
    If[lop>=ll+1,sev=Last[Sort[sevT]];Break[]];
    If[iter>=ll,sev=Last[Sort[sevT]];Break[]];
    If[aa==bb,i=i-1;values=val1;cons=con;a=a+1;le=0;sıra=0;st=2;];
    If[a>=ll+1,sev=Last[Sort[sevT]];Break[]];
    d=Pref[values];
    If[Max[d]==0.,Break[],i=i-1];
    If[st==2,
    Do[
    e=Position[d,Max[d]][[1,1]];
    d[[e]]=0;st=st+1,{a}];iter=iter+1];tt1=e;
    If[le==e,d[[e]]=0;tt1=Position[d,Max[d]][[1,1]]];le=tt1;tt1=t[[tt1]];
    Branches[tt1,cons];
    Bound[aa,bb]
    ];
    sev=Last[Sort[sevT]]
    ]]

Pref[data_]:=Block[
    {d=Map[(#-Floor[#])&,data]},
    If[ll > Length[t],dd={};
```

```
    For[i=1,i!=Length[t]+1,i++,
    dd1=Take[d,{t[[i]]}];dd=Append[dd,dd1]];d=dd];e=Position[d,Max[d]][[1,1]];d]

Branches[tt1_,cons8_] :=
    Block[{tt=tt1,cons=cons8}, m=values[[tt]]; m=Floor[m];n=m+1;
    const=Append[cons,vars[[tt]]<=m*1.]; sev=N[ConstrainedMax[func,const,vars]];
    aa=sev[[1]];ab=Length[sev];If[sev1=={},sev1=ilk];
    If[sıra==0,bir=sev];sıra=sıra+1;
    If[ab==3,bir=sev1];
    If[ab==3,sev=bir;aa=bir[[1]];sıra=0;
    const=Drop[const,-1];];
    values=vars /. sev[[2]]; d=Pref[values];
    If[Max[d]==0.,sevT=Append[sevT,sev];
    sevT=Union[sevT]; lop=Length[sevT];];
    const1=Append[cons,vars[[tt]]>=n*1.];
    sev1=N[ConstrainedMax[func,const1,vars]]; bb=sev1[[1]];ba=Length[sev1]
    If[ba==3,sev1=sev;bb=-100000000000;const1=Drop[const1,-1]; ];
    values1=vars /. sev1[[2]]; d=Pref[values1];
    If[Max[d]==0.,sevT=Append[sevT,sev1];
    sevT=Union[sevT]; lop=Length[sevT];];]

Bound[aa1_,bb1_] :=
    Block[{aa=aa1,bb=bb1},
    If[aa-bb > 0, cons=const; lc=Length[cons];
    If[lc==lkc+ll+2, i=i-1; values=vars /. ilk[[2]];
    st=2;cons=con;a=a+1;le=0;sıra=0], cons=const1;values=values1; sev=sev1;
    lc=Length[cons];
    If[lc==lkc+ll+2, i=i-1; values=vars /. ilk[[2]]; st=2;cons=con;a=a+1;le=0;sıra=0]]; ]

Bal[val_,t4_] :=
    Block[{values=val,t=t4},
    For[z=1,z!=Length[t]+1,z++,
    tam=values[[t[[z]]]];
    If[tam-Floor[tam] > 0,
    son=0]]; bk=Length[son] ]

IntegerMin[f5_,c5_,v5_,t5_]:=
    Block[{f=f5,c=c5,v=v5,t1=t5},
    solution=IntegerMax[-f,c,v,t1];solution[[1]]= (-1)*solution[[1]] /. solution[[1]];
    solution]

ZeroOneIntegerMax[f_,c_,v_]:=
    Block[{func=f,cons=c,vars=v,t=t1},
```

```
    optsol1={}; n=Length[vars]; values=rep[n];
    solutionalt=Append[MakeList[values]+MakeTable[Length[values]],values];
    allsolutions=Map[Apply[Rule,#]&,(Transpose[{vars,#}]& /@ solutionalt),{2}];
    solutions=Select[allsolutions,SatisfiedQ[cons,#]&];
    results=(func /. #)& /@ solutions;
    optsol1=Take[solutions,Position[results,Max[results]][[1]]];
    optsol1=Prepend[optsol1,func /. optsol1[[1]]]]

ZeroOneIntegerMin[f_,c_,v_]:=
    Block[{func=f,cons=c,vars=v,t=t1},
    optsol1={}; n=Length[vars]; values=rep[n];
    solutionalt=Append[MakeList[values]+MakeTable[Length[values]],values];
    allsolutions=Map[Apply[Rule,#]&,(Transpose[{vars,#}]& /@ solutionalt),{2}];
    solutions=Select[allsolutions,SatisfiedQ[cons,#]&];
    results=(func /. #)& /@ solutions;
    optsol1=Take[solutions,Position[results,Min[results]][[1]]];
    optsol1=Prepend[optsol1,func /. optsol1[[1]]]]

SatisfiedQ[constraints_, parameters_]:=
    Apply[And, constraints /. parameters]
MakeList[l_List]:= l& /@ Range[2^Length[l]-1]
MakeTable[len_Integer]:=Join[ Table[0,{len-Length[#]}], #]& /@
    (Digits[#,2]& /@ Range[2^len-1])
rep[n_] := Table[0,{n}]

MixZeroOneIntegerMax[f_,c_,v_,t1_,z1_]:=
    Block[{func=f,f1=f,cons=c,c1=c,vars=v,v7=v,t=t1,z0=z1,values,sev,sev1},
    sev={};const={};const1={};sevT={};const={};sev1={};lopt=0;
    st=1;con=cons;iter=0;a=1;le=0;sıra=0;abi[[1]]=100000000; aha=1;
    If[z0==Null,son=ZeroOneIntegerMax[f1,c1,v7];Print[son];Break[]];
    lkc=Length[cons];ll=Length[vars];
    If[t==Null,t={};tm=0;
    Do[tm=tm+1;t=Append[t,tm],{ll}]];t2=t;t=Sort[t];
    son=N[ConstrainedMax[func,cons,vars]];ilk=son;sev=son;
    values= vars /. son[[2]];val1=values;
    Bal[values,t];
    If[bk==2,son=ZeroOneIntegerMax[f1,c1,v7];Print[son];Break[]];
    If[bk==3,Print["Integer solution is not found"];
    Break[]];
    For[i=1,i!=2,i++,
    If[lopt==6,sev=Last[Sort[sevT]];Break[]];
    If[a==ll+1,sev=Last[Sort[sevT]];Break[]];
    If[iter==ll,sev=Last[Sort[sevT]];Break[]]; d=Pref[values];
```

```
    If[Max[d]==0.,Break[],i=i-1];
    If[st==2,
    Do[
    e=Position[d,Max[d]][[1,1]];
    d[[e]]=0;st=st+1,{a}]; iter=iter+1]; tt1=e;
    If[le==e,
    d[[e]]=0;tt1=Position[d,Max[d]][[1,1]]];
    le=tt1; tt1=t[[tt1]];
    If[aha==1,
    For[mh=1,mh!=Length[z0]+1,mh=mh+1,
    m=0;n=1;
    const=Append[cons,vars[[z0[[mh]]]]==m*1.];
    sev=N[ConstrainedMax[func,const,vars]]; aa=sev[[1]];ba=Length[sev];
    If[ba==3,sev=sev1;aa=-100000000000;const=Drop[const,-1]];
    const1=Append[cons,vars[[z0[[mh]]]]==n*1.];
    sev1=N[ConstrainedMax[func,const1,vars]]; bb=sev1[[1]];
    If[aa> bb,cons=const;
    values= vars /. sev[[2]],
    cons=const1;values= vars /. sev1[[2]]];
    ]];
    aha=2;
     d=Pref[values];
    If[Max[d]==0.,
    sevT=Append[sevT,sev];
    sevT=Append[sevT,sev1];Break[], i=i-1];
    Branches[tt1,cons];
    Bound[aa,bb]];
    sev=Last[Sort[sevT]]]

MixZeroOneIntegerMin[f5_,c5_,v5_,t5_,z5_]:=
    Block[{f=f5,c=c5,v=v5,t1=t5,z1=z5},
    solution=MixZeroOneIntegerMax[-f,c,v,t1,z1];
    solution[[1]]= (-1)*solution[[1]] /. solution[[1]];
    solution
]

End[]
EndPackage[]
```