BRANCH-AND-BOUND METHODS FOR AN MINLP MODEL WITH SEMI-CONTINUOUS VARIABLES

ERWIN KALVELAGEN

Abstract. This document describes several branch-and-bound methods to solve a convex Mixed-Integer Nonlinear Programming (MINLP) problem with GAMS. A straightforward MINLP formulation is compared with a piecewise linear approximation. In addition we include a branch-and-bound algorithm implemented in GAMS.

1. Introduction

The following question was posed:

```
Does anybody know an algorithm or mathematical description to solve
the following problem.

maximize sum(i=1 to n)    c_i +(d_i-c_i)/(1+exp(-(x_i-a_i)/b_i))   (=
the sum of n logistic functions), with a_i < 0 and b_i > 0 and c_i <
d_i
s.t.
sum(i=1 to n)   x_i <= Budget
When x_i >0  then x_i>= Budget_{min}

I know that I can describe the last restriction as
x_i <= M*y_i
x_i >= Budget_{min}*y_i
with y_i binary, but I don't want to use binary variables.
It would be best to have only linear restrictions.
Does anybody know another way to solve this problem. Perhaps a simple
algorithm is enough to find the optimal solution because all n
logistic functions are convex for x_i >=0 (because a_i < 0)
```

The mathematical model looks like:

$$
\begin{aligned}
&\max \sum_i c_i + \frac{d_i - c_i}{1+\exp\left(-\frac{x_i-a_i}{b_i}\right)} \\
&\sum_i x_i \le B \\
&x_i \in \{0\} \cup [L,\infty)
\end{aligned} \tag{1}
$$

Indeed the functions

$$
f_i(x) = c_i + \frac{d_i - c_i}{1+\exp\left(-\frac{x-a_i}{b_i}\right)} \tag{2}
$$

are convex, as can be seen from the figure 1.

However, the non-convexity introduced by x being not a continuous variable will cause that this model can not be solved using an NLP solver directly.

Date: April, 2003.

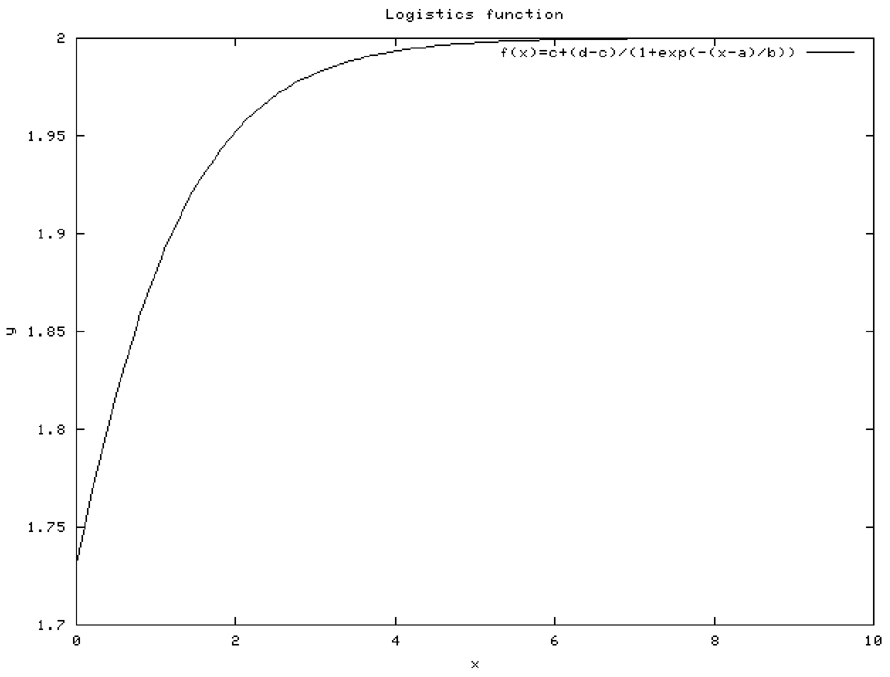

Figure 1. Logistics function

2. MINLP formulation

The simplest approach is to use an MINLP solver directly. The variables x_i are of a special type called "semi-continuous." A semi-continuous variable x is either zero or between its bounds $[L, U]$. This type of variable is often used to prevent very small values. E.g. in portfolio models, this restriction is sometimes added to prevent a lot of very small holdings which cause extra management and transaction costs and are thus undesirable.

Semi-continuous variables can be implemented in GAMS using the semicont variable type:

```
    semicont variables x;
```

The complete model looks like:

```
set i /i1*i10/;

parameters
    a(i) '<0'
    b(i) '>0'
    c(i) 'c(i) < d(i)'
    d(i)
;

a(i) = -ord(i);
b(i) = ord(i);
c(i) = ord(i);
d(i) = c(i)+ord(i);
display a,b,c,d;

scalar
```

```
  budget_avail / 10 /
  minx         / 0.5/
;



*--------------------------------------------------------------
* MINLP model with semi-continuous variables
*--------------------------------------------------------------
variables
    x1(i)
    z1
;
semicont variable x1;

equations
    logistic1
    budget1
;

logistic1.. z1 =e= sum(i, c(i) +(d(i)-c(i))/(1+exp(-(x1(i)-a(i))/b(i))));
budget1..   sum(i, x1(i)) =l= budget_avail;

x1.lo(i) = minx;

model m1 /logistic1,budget1/
option minlp=sbb;
option optcr=0;
solve m1 maximizing z1 using minlp;
```

Semi-continuous variables can be easily converted to a formulation with binary variables. This can be important if the solver does not support semi-continuous variables. The reformulation looks like:

$$
\begin{aligned}
&x \le Uy\\
&x \ge Ly\\
&x \in [0, U]\\
&y \in \{0, 1\}
\end{aligned}
\tag{3}
$$

It is needed that a tight upper bound U on x is available. In our case we know that $\sum_i x_i \le B$ and thus $x_i \le B$.

```
*--------------------------------------------------------------
* MINLP model with binary variables
*--------------------------------------------------------------

variables
    x2(i)
    y2(i)
    z2
;
binary variable y2;
positive variable x2;

equations
    logistic2
    budget2
    bigm2a(i)
    bigm2b(i)
;

scalar bigM;
bigM = budget_avail;

logistic2.. z2 =e= sum(i, c(i) +(d(i)-c(i))/(1+exp(-(x2(i)-a(i))/b(i))));
budget2..   sum(i, x2(i)) =l= budget_avail;
bigm2a(i)..  x2(i) =l= bigM*y2(i);
bigm2b(i)..  x2(i) =g= minx*y2(i);
```

```
model m2 /logistic2,budget2, bigm2a, bigm2b/
option minlp=sbb;
option optcr=0;
solve m2 maximizing z2 using minlp;
```

The convexity will assure that the subproblems of the branch-and-solver SBB[3] can be solved to optimality, and thus that we find a global optimum solution.

3. Piecewise linear approximation

The above model can be tackled by a MIP solver using a piecewise linear formulation. As the objective function is a separable function, i.e.:

$$\max \sum_i f_i(x_i) \tag{4}$$

we can use a standard interpolation scheme using so-called SOS2 variables (special ordered sets of type two).

Suppose we have tabular data for a scalar function $y = f(x)$ for a given interval $x \in [x^{lo}, x^{up}]$. Let's denote this data by $(\bar{x}_k, \bar{y}_k)$. We introduce variables λ_k with:

$$X = \sum_k \lambda_k \bar{x}_k \tag{5}$$

$$Y = \sum_k \lambda_k \bar{y}_k \tag{6}$$

$$\sum_k \lambda_k = 1 \tag{7}$$

$$\text{max 2 adjacent } \lambda\text{'s nonzero} \tag{8}$$

$$\lambda_k \geq 0 \tag{9}$$

$$x^{lo} \leq X \leq x^{up} \tag{10}$$

Equation (5) is called the *reference row*. Equation (6) is known as the *function row*, and equation (7) is the *convexity row*.

The λ_k variables form a Special Order Set of Type 2. In a SOS2 set only two adjacent variables of the set can assume non-zero values.

SOS2 variables can be implemented in GAMS using the SOS2 variable type:

```
   sos2 variables lambda(k);
```

The exact definition of SOS2 variables is solver specific, especially in special cases where nonzero lower bounds are used. In the above case we have added $\lambda_k \geq 0$, which only leaves the "max two adjacent λ's are nonzero" which is shared by all solvers that have SOS2 facilities.

```
*--------------------------------------------------------------
* MIP model with piecewise linear objective using SOS2 variables
*--------------------------------------------------------------


*
* set up grid (simply equidistant, but with logistic function
* it would be better to use wider grid for larger x)
*
set k /k0*k200/;
scalar stepsize;
stepsize = budget_avail/(card(k)-1);
parameter xbar(i,k), ybar(i,k);
xbar(i,k) = (ord(k)-1)*stepsize;
ybar(i,k) = c(i) +(d(i)-c(i))/(1+exp(-(xbar(i,k)-a(i))/b(i)));
```

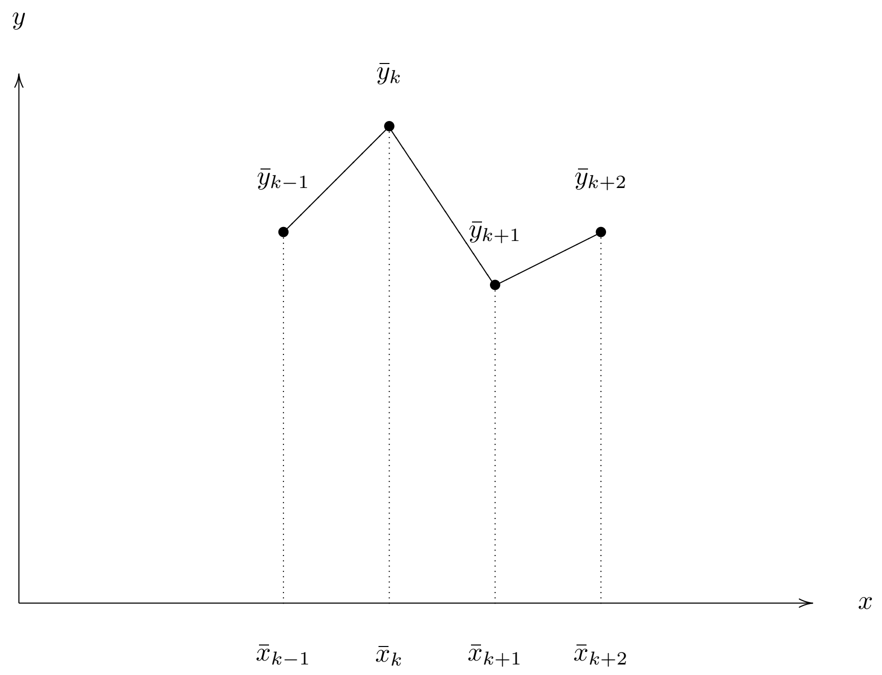

FIGURE 2. Linear interpolation

```
sos2 variables lambda3(i,k);
semicont variables x3(i);
variables z3,y3(i);

x3.lo(i) = minx;
x3.up(i) = budget_avail;

equations
   refrow3(i)
   funcrow3(i)
   convexity3(i)
   budget3
   obj3
;

refrow3(i)..  x3(i) =e= sum(k, lambda3(i,k)*xbar(i,k));
funcrow3(i).. y3(i) =e= sum(k, lambda3(i,k)*ybar(i,k));
convexity3(i).. sum(k,lambda3(i,k)) =e= 1;
budget3..     sum(i, x3(i)) =l= budget_avail;
obj3..        z3 =e= sum(i, y3(i));


model m3 /refrow3, funcrow3, convexity3, budget3, obj3/
option mip=cplex;
option optcr=0;
solve m3 maximizing z3 using mip;
```

It is noted that we have set up an equidistant grid. In this case the logistics function is the most "non-linear" at the beginning while it is almost linear for larger values of x. It could be beneficial to exploit this property, and space out the grid more for larger values of x.

As we deal with a convex objective, we can actually drop the SOS2 variables and consider them to be continuous:

```
*-------------------------------------------------------------
* MIP model with piecewise linear objective exploiting convexity
* SOS2 variables can be relaxed to positive variables.
*-------------------------------------------------------------

positive variables lambda4(i,k);
semicont variables x4(i);
variables z4,y4(i);

x4.lo(i) = minx;
x4.up(i) = budget_avail;

equations
   refrow4(i)
   funcrow4(i)
   convexity4(i)
   budget4
   obj4
;

refrow4(i)..  x4(i) =e= sum(k, lambda4(i,k)*xbar(i,k));
funcrow4(i).. y4(i) =e= sum(k, lambda4(i,k)*ybar(i,k));
convexity4(i).. sum(k,lambda4(i,k)) =e= 1;
budget4..     sum(i, x4(i)) =l= budget_avail;
obj4..        z4 =e= sum(i, y4(i));


model m4 /refrow4, funcrow4, convexity4, budget4, obj4/
option mip=cplex;
option optcr=0;
solve m4 maximizing z4 using mip;
```

4. A nonlinear branch-and-bound algorithm

The GAMS language is powerful enough to implement reasonable complex algorithms. Examples include Lagrangian Relaxation with subgradient optimization [7], Benders Decomposition [5] and Column Generation algorithms [6].

In this section we implement a branch-and-bound algorithm for the above model.

A standard branch-and-bound algorithm for MINLP problems with integer variables can look like [2]:

```
{Initialization}
$LB := -\infty$
$UB := \infty$
Store root node in waiting node list
**while** waiting node list is not empty **do**
   {Node selection}
   Choose a node from the waiting node list
   Remove it from the waiting node list
   Solve subproblem
   **if** infeasible **then**
      Node is fathomed
   **else if** optimal **then**
      **if** integer solution **then**
         **if** $obj > LB$ **then**
            {Better integer solution found}
            $LB := obj$
```

```
              Remove nodes $j$ from list with $UB_j < LB$
            end if
          else
            {Variable selection}
            Find variable $x_k$ with factional value $v$
            Create node $j_{new}$ with bound $x_k \leq \lfloor v \rfloor$
            $UB_{j_{new}} := obj$
            Store node $j_{new}$ in waiting node list
            Create node $j_{new}$ with bound $x_k \geq \lceil v \rceil$
            $UB_{j_{new}} := obj$
            Store node $j_{new}$ in waiting node list
          end if
        else
          Stop: problem in solving subproblem
        end if
        $UB = \max_j UB_j$
      end while
```

The root node is the problem with the integer restrictions relaxed. For more details consult [1, 4, 8].

When dealing with semi-continuous variables, we need to change just a few things. A variable x_k is considered fractional if $0 < x_k < x_k^{lo}$. Secondly the bounds we add to the newly created nodes are $x_k \leq 0$ and $x_k \geq x_k^{lo}$.

```
*-------------------------------------------------------------
* simple branch and bound algorithm
*-------------------------------------------------------------

variables
    x5(i)
    z5
;
positive variable x5;

equations
    logistic5
    budget5
;

logistic5.. z5 =e= sum(i, c(i) +(d(i)-c(i))/(1+exp(-(x5(i)-a(i))/b(i))));
budget5..   sum(i, x5(i)) =l= budget_avail;



model m5 /logistic5,budget5/
option nlp=conopt;


*---------------------------------------------------------
* datastructure for node pool
* each node has variables x(i) which is either
* still free, or fixed to zero or with a lowerbound
* of minx.
*---------------------------------------------------------
set node                 'maximum size of the node pool'    /node1*node1000/;
parameter bound(node)    'node n will have an obj <= bound(n)';
set fixed(node,i)        'variables x(i) are fixed to zero in this node';
set lowerbound(node,i)   'variables x(i)>=minx in this node';
scalar bestfound         'lowerbound in B&B tree'   /-INF/;
scalar bestpossible      'upperbound in B&B tree'   /+INF/;
set newnode(node)        'new node (singleton)';
set waiting(node)        'waiting node list';
set current(node)        'current node (singleton except exceptions)';
```

```
parameter log(node,*)   'logging information';
scalar done             'terminate'             /0/;
scalar first            'controller for loop';
scalar first2           'controller for loop';
scalar obj              'objective of subproblem';
scalar maxx;
set w(node);
parameter nodenumber(node);


nodenumber(node) = ord(node);

fixed(node,i) = no;
lowerbound(node,i) = no;

set j(i);
alias (n,node);


*
* add root node to the waiting node list
*
waiting('node1') = yes;
current('node1') = yes;
newnode('node1') = yes;
bound('node1') =INF;




loop(node$(not done),

*
* node selection
*
     bestpossible = smax(waiting(n), bound(n));
     current(n) = no;
     current(waiting(n))$(bound(n) = bestpossible)  = yes;
     first = 1;
*
* only consider first in set current
*
     loop(current$first,
          first = 0;

          log(node,'node') = nodenumber(current);
          log(node,'ub') = bestpossible;

*
* remove current node from waiting list
*
          waiting(current) = no;


*
* clear bounds
*
          x5.lo(i) = 0;
          x5.up(i) = budget_avail;

*
* set appropriate bounds
*
          j(i) = lowerbound(current,i);
          x5.lo(j) = minx;
          j(i) = fixed(current,i);
          x5.up(j) = 0;

*
* solve subproblem
*
          solve m5 maximizing z5 using nlp;
```

```
*
* check for optimal solution
*
          log(node,'solvestat') = m5.solvestat;
          log(node,'modelstat') = m5.modelstat;

          abort$(m5.solvestat <> 1) "Solver did not return ok";
          if (m5.modelstat = 1 or m5.modelstat = 2,


*
* get objective
*
              obj = z5.l;
              log(node,'obj') = obj;

*
*  check "integrality"
*
              maxx = smax(i, min(x5.l(i), max(minx-x5.l(i),0)));
              if (maxx = 0,

*
* found a new "integer" solution
*
                  log(node,'integer') = 1;
                  if (obj > bestfound,

*
* improved the best found solution
*
                        log(node,'best') = 1;
                        bestfound = obj;

*
* remove all waiting nodes with bound < bestfound
*
                        w(n) = no; w(waiting) = yes;
                        waiting(w)$(bound(w) < bestfound) = no;


                  );

              else

*
* "fractional" solution
*

                  j(i) = no;
                  j(i)$(min(x5.l(i), max(minx-x5.l(i),0))=maxx) = yes;
                  first2 = 1;
                  loop(j$first2,
                      first2 = 0;

*
* create 2 new nodes
*
                      newnode(n) = newnode(n-1);
                      fixed(newnode,i) = fixed(current,i);
                      lowerbound(newnode,i) = lowerbound(newnode,i);
                      bound(newnode) = obj;
                      waiting(newnode) = yes;

                      fixed(newnode,j) = yes;

                      newnode(n) = newnode(n-1);
                      fixed(newnode,i) = fixed(current,i);
                      lowerbound(newnode,i) = lowerbound(newnode,i);
                      bound(newnode) = obj;
                      waiting(newnode) = yes;
```

```
                       lowerbound(newnode,j) = yes;

                   );

               );


           else

*
* if subproblem as infeasible this node is fathomed.
* otherwise it is an error.
*
              abort$(m5.modelstat <> 4 and m5.modelstat <> 5) "Solver did not solve subproblem";

           );

           log(node,'waiting') = card(waiting);

      );

*
* are there new waiting nodes?
*
      done$(card(waiting) = 0) = 1;


*
* display log
*
      display log;



);
```

The log of this algorithm show it finds an optimal solution very quickly:

```
----    379 PARAMETER log  logging information

             node        ub solvestat modelstat       obj   integer      best       waiting

node1       1.000      +INF     1.000     2.000    97.090                             2.000
node2       2.000    97.090     1.000     2.000    97.089                             3.000
node3       3.000    97.090     1.000     2.000    97.085                             4.000
node4       4.000    97.089     1.000     2.000    97.085     1.000     1.000         1.000
node5       5.000    97.089     1.000     2.000    97.088     1.000     1.000
```

The complete model can be downloaded from `http://www.amsterdamoptimization.com/models/bb/bb.gms`.

References

1. B. Borchers and J. E. Mitchell, *An improved branch and bound algorithm for mixed integer nonlinear programming*, Computers and Operations Research **21** (1994), 359–367.
2. Brian Borchers, *(MINLP) Branch and Bound Methods*, Encyclopedia of Optimization, vol. 3, Kluwer, 2001, pp. 331–335.
3. GAMS Development Corp., *SBB*, `http://www.gams.com/docs/solver/sbb.pdf`, April 2002.
4. O. K. Gupta and V. Ravindran, *Branch and bound experiments in convex nonlinear integer programming*, Management Science **31** (1985), no. 12, 1533–1546.
5. Erwin Kalvelagen, *Benders decomposition with GAMS*, `http://www.amsterdamoptimization.com/pdf/benders.pdf`, December 2002.
6. ———, *Column generation with GAMS*, `http://www.amsterdamoptimization.com/pdf/colgen.pdf`, December 2002.
7. ———, *Lagrangian relaxation with GAMS*, `http://www.amsterdamoptimization.com/pdf/lagrange.pdf`, December 2002.

8. Sven Leyffer, *Integrating SQP and branch-and-bound for mixed integer nonlinear programming*, Computational Optimization and Applications **18** (2001), 295–309.

Amsterdam Optimization Modeling Group LLC, Washington D.C./The Hague
E-mail address: `erwin@amsterdamoptimization.com`