AIMMS Language Reference - AIMMS Outer Approximation Algorithm for MINLP

Chapter 18

AIMMS Outer Approximation Algorithm for MINLP

Open solver approach

Outer approximation (see [Du86]) is a basic approach for solving Mixed-Integer NonLinear Programming (MINLP) models. The underlying algorithm is an interplay between two solvers, one for solving mixed-integer linear models and one for solving nonlinear models. Even though the standard outer approximation algorithm is provided with AIMMS, you as an algorithmic developer may want to customize the individual steps in order to obtain better performance and/or a better solution for your particular model.

The AIMMS Outer Approximation algorithm

The outer approximation algorithm in AIMMS is, therefore, provided as a customizable procedure written in the AIMMS language itself using functions and procedures provided by the GMP library (a white box solver), whereas most other outer approximation solvers are provided as a closed implementation (a black box solver). The outer approximation algorithm in AIMMS is implemented as a system module with the name `GMP Outer Approximation`. You can install this module using the **Install System Module** command in the AIMMS **Settings** menu. In the remainder of this chapter, we will refer to the outer approximation algorithm as AIMMS Outer Approximation (AOA).

Convex algorithm

Besides the basic algorithm, the AOA module also implements the Quesada-Grossmann algorithm (see [Qu92]) which is designed to solve convex MINLP models. The basic algorithm can also be used to solve convex models but the Quesada-Grossmann algorithm is often more efficient.

This chapter

In this chapter you find the description of, and the motivation behind, the open approach to solving MINLP models based on outer approximation. We continue with a brief introduction to the problem statement and the basic algorithm. Next we explain how the AOA algorithm can be setup, followed by a detailed explanation of the parameters inside the AOA module that can be used to control the outer approximation algorithm. We then describe the Quesada-Grossmann algorithm for convex MINLP models. Next we describe an initial implementation of the basic solution algorithm using procedures in the AIMMS language is described. These procedures use functions that are especially designed to support the open approach. The chapter concludes with suggestions on additional ways to vary the individual steps of the overall algorithm in order to obtain customized versions of the outer approximation algorithm.

18.1 Problem statement

MINLP

The mixed-integer nonlinear programming models to be solved can be expressed as follows.

Minimize:

$$f(x,y)$$

Subject to:

$$\begin{aligned}
h(x,y) &= 0 \\
g(x,y) &\leq 0 \\
Cy + Dx &\leq d \\
x \in \mathcal{X} &= \{x \in \mathbb{R}^n | x^L \leq x \leq x^U\} \\
y \in \mathcal{Y} &= \mathbb{Z}^m
\end{aligned}$$

Usual assumption

The usual assumption is that the nonlinear subproblem (i.e. the model in which all integer variables are fixed) is convex. This assumption is to guarantee that each locally optimal solution of the nonlinear subproblem is also a globally optimal solution. In practice this assumption does not always hold, but the algorithm can still be applied. Convergence to a global optimum of the MINLP using the outer approximation algorithm is then no longer guaranteed.

18.2 Basic algorithm

Algorithm in words

The algorithm solves an alternating sequence of mixed-integer linear models and nonlinear models.

1. First, the entire model is solved as a nonlinear program with all the integer variables relaxed as continuous variables between their bounds.
2. Then a linearization is carried out around the optimal solution, and the resulting constraints are added to the linear constraints that are already present. This new linear model is referred to as the master MIP model.
3. The master MIP problem is solved as an mixed-integer linear program.
4. The integer part of the resulting optimal solution is then temporarily fixed, and the original MINLP model with fixed integer variables is solved as a nonlinear subproblem.
5. Again, a linearization around the optimal solution is constructed and the new linear constraints are added to the master MIP problem. To prevent

cycling, one or more constraints are added to cut off the previously-found integer solution of the master problem.

6. Steps 3–5 are repeated until one of the termination criteria is satisfied.

A more detailed description of the general Outer Approximation algorithm can be found in [Du86].

Convexity and convergence

As linearizations are added to the master MIP problem, the model becomes an improved approximation of the original MINLP model. Using the usual convexity assumption regarding the nonlinear subproblem, convergence to a global optimum occurs when the objective function value of the master MIP problem is worse than the value associated with the NLP subproblem.

Termination . . .

Several termination criteria are used in practice. These criteria can be used in isolation or in some logical combination. Three of them are discussed in the following paragraphs.

. . . iteration limit

Perhaps the most frequently-used criterion is the iteration limit. One reason is that a good solution is usually found during the first few iterations. Another reason for using an iteration limit is that the size of the underlying master MIP problem tends to grow significantly each time linearization constraints are added, causing an increase in computation time.

. . . objective worsening

A second criterion is the worsening of the objective function value of two successive nonlinear subproblems. This worsening occurs quite frequently, even if the NLP subproblem is convex. The underlying reason is that the master MIP problem will not always select binary solutions that lead to successively improving NLPs. This criterion seems appropriate when the worsening is persistent over several iterations.

. . . crossover

A third termination criterion is insufficient improvement in the objective function value of the master MIP problem in relation to the objective function value of the previously solved NLP subproblem. The difference between these two values represents the optimality gap, since the master MIP problem represents an outer approximation (thus a relaxation) of the original MINLP model. When the gap is closed at crossover, the optimal solution has been found provided the NLP subproblem is convex.

Final solution

Upon termination of the algorithm, the known best solution (also referred to as the incumbent solution) is declared as the final solution. In many practical applications, this solution is not necessarily optimal due to termination based on an iteration limit. In addition, it is often not possible to verify that the NLP subproblem is convex.

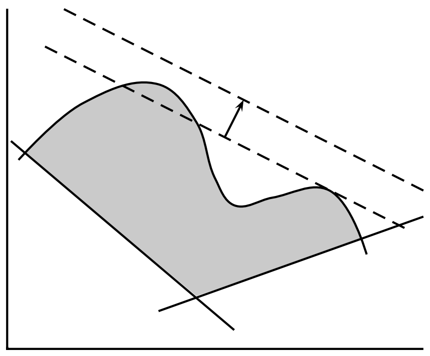

Figure 18.1: Effect of loosening a linearization

Linearizations

The term 'outer approximation' refers to the linear approximation of the convex nonlinear constraints at selected points along the boundary of the convex solution region. The accumulation of such inequality constraints forms an outer approximation of the solution region, and this approximation can be used in the optimization rather than the nonlinear constraints from which it was derived. The formula for the linearization of a scalar nonlinear inequality $g(x, y) \le 0$ around the point $(x, y) = (x^0, y^0)$ is as follows.

$$g(x^0, y^0) + \nabla g(x^0, y^0)^T \begin{bmatrix} x - x^0 \\ y - y^0 \end{bmatrix} \le 0$$

The nonconvex case

The linear approximation ceases to be an outer approximation if the solution region is not convex. In this situation there is the possibility that portions of the solution region are cut off as illustrated in Figure 18.1.

Loosening inequalities

In practical implementations of the outer approximation algorithm, the linearizations are allowed to move away from the feasible region. Such heuristic flexibility allows solutions to be found that would otherwise have been cut off. The implementation allows deviations through the use of artificial nonnegative variables and then penalizing them while solving the master problem.

Open solver approach

The basic outer approximation algorithm that is part of the AOA module has been completely implemented using functionality provided by the GMP library.

- From the math program instance representing the original MINLP model, a new math program instance representing the initial master MIP problem can be created using the function `GMP::Instance::CreateMasterMIP`.
- The functions from the `GMP::Linearization` namespace can be used to add linearizations of the nonlinear constraints of the original MINLP model to the master MIP, in a customizable manner.

- Using the `GMP::Instance::FixColumns` procedure, the integer columns of the nonlinear subproblem can fixed to the current integer solution of the master MIP.
- Using the `GMP::Instance::AddIntegerEliminationRows` procedure, prior integer solutions of the master MIP are excluded from subsequent solves.

18.3 Using the AOA algorithm

AOA module . . .

The basis GMP implementation of the AIMMS Outer Approximation (AOA) algorithm can be found in a single AIMMS module, called `GMP Outer Approximation`, that is provided as part of the AIMMS system. You can install this module using the **Install System Module** command in the AIMMS **Settings** menu.

Basic algorithm

The procedure `DoOuterApproximation` inside the module implements the basic algorithm from the previous section. The procedure `DoOuterApproximation` has one input argument, namely:

- `MyGMP`, an element parameter with range `AllGeneratedMathematicalPrograms`.

This procedure is called as follows:

```
generatedMP := GMP::Instance::Generate( SymbolicMP );

GMPOuterApprox::DoOuterApproximation( generatedMP );
```

Here `SymbolicMP` is the symbolic mathematical program containing the MINLP model, and `generatedMP` is an element parameter in the predefined set `AllGeneratedMathematicalPrograms`. `GMPOuterApprox` is the prefix of the AOA module. The implementation of this procedure will be discussed in Section 18.6.

Modifying the algorithm

Because the AIMMS Outer Approximation algorithm is completely implemented using functionality provided the GMP library, you have the complete freedom to modify the math program instances generated by the basic AOA algorithm using the matrix manipulation routines discussed in Section 16.3. Such problem-specific modifications to the basic algorithm may help you to find a better overall solution to your MINLP model, or to find a good solution faster.

18.4 Control parameters that influence the AOA algorithm

Control parameters

The multistart module defines several parameters that influence the outer approximation algorithm. These parameters have a similar functionality as options of a solver, e.g., CPLEX. The most important parameters, with their default setting, are shown in Table 18.1. The parameters that are not self-explanatory are explained in this section; the last column in the table refers to the subsection that discusses the corresponding parameter.

Parameter	Default	Range	Subsection
`IterationMax`	20	{0,maxint}	
`CreateStatusFile`	0	{0,1}	
`UsePresolver`	1	{0,1}	18.4.1
`UseMultistart`	0	{0,1}	18.4.2
`TerminateAfterFirstNLPIsInteger`	1	{0,1}	18.4.3
`IsConvex`	0	{0,1}	18.4.4
`NLPUseInitialValues`	1	{0,1}	18.4.5

Table 18.1: Control parameters in the outer approximation module

18.4.1 Using the AIMMS Presolver

Parameter *`UsePresolver`*

By default the outer approximation algorithm starts by applying the AIMMS Presolver to the MINLP model. By preprocessing the MINLP model, the model might become smaller and easier to solve. The parameter `UsePresolver` can be used to switch off the preprocessing step.

18.4.2 Combining outer approximation with multistart

Parameter *`UseMultistart`*

If the parameter `UseMultistart` is switched on then the outer approximation algorithm will use the multistart algorithm to solve the nonlinear subproblems. For non-convex models this can have a positive effect on the quality of the solution that is returned by the outer approximation algorithm. The multistart algorithm is described in section 17.2. The parameters `MultistartNumberOfSamplePoints` and `MultistartNumberOfSelectedSamplePoints` can be used to specify the number of sample and selected sample points, respecively, as used by the multistart algorithm.

Multistart module

To use the multistart algorithm, the system module `Multi Start` should be added to your project. You can install this module using the **Install System Module** command in the AIMMS **Settings** menu.

18.4.3 Terminate if solution of relaxed model is integer

Parameter *`TerminateAfterFirstNLPIsInteger`*

By default the outer approximation algorithm will terminate if it finds an integer solution for the initial NLP problem, which is obtained from the MINLP model by relaxing the integer variables. By switching off the parameter `TerminateAfterFirstNLPIsInteger` you can enforce the algorithm to continue.

18.4.4 Solving a convex model

Parameter IsConvex

The parameter IsConvex can be used to indicate that the model is convex. In that case the outer approximation algorithm will no longer stop after the iteration limit is hit, as specified by the parameter IterationMax. Instead, the algorithm will stop if the gap between the objective values of the master MIP problem and the nonlinear subproblem is sufficiently small, as controlled by the parameter RelativeOptimalityTolerance. Note that AIMMS cannot identify whether a model is convex or not.

18.4.5 Starting point strategy for NLP subproblems

Parameter NLPUseInitialValues

The parameter NLPUseInitialValues specifies the starting point strategy used for solving the NLP subproblems. For nonconvex nonlinear problems the starting point often has a big influence on the solution that the NLP solver will find. By default the AOA algorithm will use the initial values as provided by the user for all NLP subproblems that are solved. By setting this parameter to 0, the algorithm will use the solution of the previous master MIP problem as the starting point for the next NLP subproblem (and for the initial NLP it will use the initial values provided by the user). Note: if one of the parameters UseMultistart or IsConvex equals 1 then NLPUseInitialValues is automatically set to 0.

18.5 The Quesada-Grossmann algorithm

One MIP problem

Quesada and Grossmann ([Qu92]) noticed that the classic outer approximation algorithm often spends a large amount of time in solving the MIP problems in which a significant amount of rework is done. They proposed an algorithm in which only one MIP problem is solved. The algorithm implemented in AIMMS uses a callback procedure for lazy constraints which is supported by modern MIP solvers like CPLEX and GUROBI.

Convex models

The Quesada-Grossmann algorithm is designed to solve convex MINLP models. The basic outer approximation algorithm can also be used to solve convex models by using the parameter IsConvex, but the Quesada-Grossmann algorithm is often more efficient. The Quesada-Grossmann algorithm is also available in the GMP Outer Approximation module.

Calling procedure

The procedure DoConvexOuterApproximation inside the module implements the Quesada-Grossmann algorithm. This procedure is called in the same way as the DoOuterApproximation procedure of Section 18.3, which implements the basic algorithm. The following control parameters in Table 18.1 can be used to influence the Quesada-Grossmann algorithm: `CreateStatusFile` and `UsePresolver`.

18.6 A first and basic implementation

Calling the AOA algorithm

To call the AOA algorithm, the GMP library is used to generate a number of math program instances, and associated solver sessions, where `SymbolicMP` is the symbolic mathematical program containing the MINLP model.

```
! Generate the MINLP model.
GMINLP := GMP::Instance::Generate(SymbolicMP, FormatString("%e", SymbolicMP)) ;

! Create NLP subproblem.
GNLP := GMP::Instance::Copy( GMINLP, 'OA_NLP' ) ;
GMP::Instance::SetMathematicalProgrammingType( GNLP, 'RMINLP' ) ;
ssNLP := GMP::Instance::CreateSolverSession( GNLP ) ;

! Create Master MIP problem.
GMIP := GMP::Instance::CreateMasterMip( GMINLP, 'OA_MasterMIP' ) ;
ssMIP := GMP::Instance::CreateSolverSession( GMIP ) ;

BasicAlgorithm;
```

The basic algorithm outlined above is available in the GMP Outer Approximation module as the procedure `DoOuterApproximation`.

The basic algorithm

The basic algorithm is straightforward, and makes a call to five other procedures that execute the various algorithm steps. The naming convention is self-explanatory, and the following lines make up this first example of a main procedure. For the sake of brevity and clarity, the parts of the code used to create a status file and to customize the contents of the progress window have been left out. They can be found in the basic implementation of the AOA algorithm in the AOA module.

```
InitializeAlgorithm;
SolveRelaxedMINLP;

while ( not MINLPAlgorithmHasFinished ) do
    AddLinearizationsAndSolveMasterMIP;
    FixIntegerVariablesAndSolveNLP;
    TerminateOrPrepareForNextIteration;
endwhile;
```

Note that the scalar parameter `MINLPAlgorithmHasFinished` must be initially set to zero, and should only get a nonzero value when the algorithm is ready to terminate.

Initialize-Algorithm

The following procedure is used to set all algorithmic parameters and options, and to prepare the status file and progress window output.

```
IterationCount                := 0 ;
LinearizationCount            := 1 ;
EliminationCount              := 1 ;
IncumbentSolutionHasBeenFound := 0 ;
MINLPAlgorithmHasFinished     := 0 ;

if ( NLPUseInitialValues ) then
    GMP::Solution::RetrieveFromModel( GNLP, SolNumbInitialValues ) ;
endif;

if ( GMP::Instance::GetDirection( GMINLP ) = 'maximize' ) then
    MINLPOptimizationDirection := 1;
else
    MINLPOptimizationDirection := -1;
endif;

GMP::Solution::SetProgramStatus( GMINLP, SolNumb, 'ProgramNotSolved' ) ;
GMP::Solution::SetSolverStatus( GMINLP, SolNumb, 'Unknown' ) ;

! The marginals of the NLP solver are needed.
option always_store_marginals := 'On';
```

The algorithmic parameters are initially set such that the AOA algorithm will always select the original initial values (i.e. the values of the variables prior to starting the AOA algorithm) as the starting values for each NLP subproblem to be solved. This setting has found to work quite well in extensive tests performed using this algorithm.

MINLPTerminate

The following termination procedure is used in several of the procedures that are described later.

```
if ( IncumbentSolutionHasBeenFound ) then
    GMP::Solution::SetProgramStatus( GMINLP, SolNumb, 'LocallyOptimal' ) ;
    GMP::Solution::SetSolverStatus( GMINLP, SolNumb, 'NormalCompletion' ) ;
else
    GMP::Solution::SetProgramStatus( GMINLP, SolNumb, 'LocallyInfeasible' ) ;
    GMP::Solution::SetSolverStatus( GMINLP, SolNumb, 'NormalCompletion' ) ;
endif;

GMP::Solution::SendToModel( GMINLP, SolNumb ) ;
MINLPAlgorithmHasFinished := 1 ;
```

The parameter `IncumbentSolutionHasBeenFound` contains a value of one or zero depending on whether the AOA algorithm has received an incumbent solution to the original MINLP model. Such a solution may be found when solving the NLP subproblem, and this must then be communicated to the AOA algorithm. Note that you also need to set the program status and indicate when the MINLP algorithm has finished.

SolveRelaxed-MINLP

The first model that is solved during the algorithm is the relaxed MINLP model. All integer variables are relaxed to continuous variables. The following procedure implements this first solution step of the outer approximation algorithm.

```
SolveNLPSubProblem( 1 );
ProgramStatus := GMP::Solution::GetProgramStatus( GNLP, SolNumb ) ;

if ( ProgramStatus in NLPOptimalityStatus ) then
    ! Save NLP solution as MINLP solution if an integer solution has been found.

    if ( GMP::Solution::IsInteger( GNLP, SolNumb ) ) then

        ! Set incumbent solution for MINLP.
        GMP::Solution::RetrieveFromModel( GMINLP, SolNumb ) ;
        IncumbentSolutionHasBeenFound := 1 ;

        if ( TerminateAfterFirstNLPIsInteger ) then
            ! Terminate if an integer solution has been found.

            MINLPTerminate;
        endif;
    endif;
else
    ! Terminate if no linearization point has been found.

    SolverStatus := GMP::Solution::GetSolverStatus( GNLP, SolNumb ) ;

    if not ( SolverStatus in NLPContinuationStatus ) then
        MINLPTerminate;
    endif;
endif ;

IterationCount += 1 ;
GMP::Solution::SetIterationCount( GMINLP, SolNumb, IterationCount ) ;
```

When the procedure `SolveNLPSubProblem` has terminated, the AOA algorithm has typically found a point for the linearization step. The exception being when the NLP solver does not produce a solution at all (either feasible or infeasible). In such a situation the outer approximating algorithm should be terminated. Note that in the special event that the solution is feasible and has integer values for the integer variables, a locally optimal solution has been found and the AOA algorithm is instructed accordingly. Otherwise, the next step of the outer approximation algorithm can be executed.

Add-Linearizations-AndSolve-MasterMIP

If a termination flag has not been set, the following procedure adds linearizations to the master MIP problem prior to solving it. If this model becomes infeasible, the outer approximation algorithm will be terminated.

```
return when ( MINLPAlgorithmHasFinished );

GMP::Linearization::Add( GMIP, GNLP, SolNumb, AllNonLinearConstraints,
                         DeviationsPermitted, PenaltyMultiplier,
                         LinearizationCount, JacobianTolerance ) ;

LinearizationCount += 1 ;
```

```
GMP::SolverSession::Execute( ssMIP ) ;

GMP::Solution::RetrieveFromSolverSession( ssMIP, SolNumb ) ;
GMP::Solution::SendToModel( GMIP, SolNumb ) ;

ProgramStatus := GMP::Solution::GetProgramStatus( GMIP, SolNumb ) ;

if not ( ProgramStatus in MIPOptimalityStatus ) then
    MINLPTerminate;
endif ;
```

The AIMMS parameters `DeviationsPermitted` and `PenaltyMultiplier` are part of the AOA module. By default, deviations are allowed and are penalized with the value 1000 in the objective function of the master MIP.

FixInteger-VariablesAnd-SolveNLP

The following procedure implements the next major step of the outer approximation algorithm. First, the NLP subproblem is solved after fixing all the integer variables in the MINLP model using the values found from solving the previous master MIP problem. Then, if the combination of integer values and feasible NLP solution values improves the current MINLP incumbent solution, a new incumbent solution is set. When the NLP subproblem does not produce a solution (either feasible or infeasible), the outer approximation algorithm will be terminated.

```
return when ( MINLPAlgorithmHasFinished );

SolveNLPSubProblem( 0 );
ProgramStatus := GMP::Solution::GetProgramStatus( GNLP, SolNumb ) ;

if ( ProgramStatus in NLPOptimalityStatus ) then
    ! Save NLP solution as MINLP solution if no incumbent solution
    ! has been found yet, or if the NLP solution is better than
    ! the current incumbent.

    if ( not IncumbentSolutionHasBeenFound ) then
        ! Set incumbent solution for MINLP.

        GMP::Solution::RetrieveFromModel( GMINLP, SolNumb ) ;
        IncumbentSolutionHasBeenFound := 1 ;
    else

        NLPobjectiveValue   := GMP::Solution::GetObjective( GNLP  , SolNumb ) ;
        MINLPIncumbentValue := GMP::Solution::GetObjective( GMINLP, SolNumb ) ;

        if ( MINLPSolutionImprovement( NLPobjectiveValue, MINLPIncumbentValue ) ) then
            ! Set incumbent solution for MINLP.

            GMP::Solution::RetrieveFromModel( GMINLP, SolNumb ) ;
            IncumbentSolutionHasBeenFound := 1 ;
        endif;
    endif ;
else
    ! Terminate if no linearization point has been found.

    SolverStatus := GMP::Solution::GetSolverStatus( GNLP, SolNumb ) ;

    if not ( SolverStatus in NLPContinuationStatus ) then
```

```
        MINLPTerminate;
    endif;
endif ;
```

The AOA algorithm maintains the MINLP problem, the master MIP problem, the NLP subproblem, and the incumbent solution of the MINLP. As a result, direct access to the corresponding objective function values is available.

SolveNLPSub-Problem

The procedure `SolveNLPSubProblem` solves the NLP subproblem using various routines from the GMP library. The procedure has a single argument `initial-Solve` which indicates whether this is the solve of the initial relaxed MINLP problem. In that case some steps in the procedure are not necessary.

```
if ( NLPUseInitialValues ) then
    GMP::Solution::SendToModel( GNLP, SolNumbInitialValues ) ;
elseif ( not initialSolve ) then
    GMP::Solution::SendToModel( GMIP, SolNumb ) ;
endif;

GMP::Solution::RetrieveFromModel( GNLP, SolNumb ) ;
GMP::Solution::SendToSolverSession( ssNLP, SolNumb ) ;

if ( not initialSolve ) then
    GMP::Instance::FixColumns( GNLP, GMIP, SolNumb, AllIntegerVariables ) ;
endif;

GMP::SolverSession::Execute( ssNLP ) ;

GMP::Solution::RetrieveFromSolverSession( ssNLP, SolNumb ) ;
GMP::Solution::SendToModel( GNLP, SolNumb ) ;
```

TerminateOr-PrepareForNext-Iteration

The following procedure implements the final major step of the outer approximation algorithm. If a termination flag has not been set previously, and the maximum number of iterations has not yet been reached, then the previously found integer solution of the master MIP problem will be eliminated by adding the appropriate cuts. This will ensure that the next master MIP will have a new integer solution (or none at all).

```
return when ( MINLPAlgorithmHasFinished );

if ( IterationCount = IterationMax ) then
    MINLPTerminate;
else
    ! Prepare for next iteration

    IterationCount += 1 ;
    GMP::Solution::SetIterationCount( GMINLP, SolNumb, IterationCount ) ;
    GMP::Instance::AddIntegerEliminationRows( GMIP, SolNumb, EliminationCount ) ;
    EliminationCount += 1 ;
endif ;
```

Note that you are responsible for determining the appropriate iteration count for the overall outer approximation algorithm. As you are free to develop a solution algorithm in any way you desire, it is not always possible for the AOA algorithm to determine the correct setting of the MINLP iteration count.

18.7 Alternative uses of the open approach

Customize algorithm

Using the open outer approximation approach for solving MINLP models it is possible to add to the existing procedures or write alternative procedures to meet the needs of the final user. For instance, a user evaluating the performance of the algorithm may want to add certain performance measurements and print statements to the existing code. Some less trivial examples of modifications are provided in the next few paragraphs.

Solve more NLPs

Practical experience has shown that it is sometimes difficult to get a feasible solution to the initial relaxed NLP model. Based on the particular application, the user may specify how multiple starting values can be found, and then modify the algorithm to solve multiple NLPs to get a feasible and/or a better solution. While doing so, it is also possible to specify how the algorithm should switch between different solvers (using the predefined AIMMS identifier `CurrentSolver`). Such extensions could then also be applied to the NLP subproblem inside the `While` statement.

Retain integer solutions

It is possible to activate a MIP callback procedure whenever the MIP solver finds an integer solution. Even though these intermediate solutions are not optimal, the user may want to save the integer portion of these solutions for later evaluation. Once the main algorithm has terminated, all these integer solutions can be retrieved and evaluated by solving the corresponding nonlinear subproblem. In some instances, one of these extra solutions may be a better solution to the original MINLP model than the one produced by the main algorithm.

Adjust penalties

Setting the penalties for the deviations of the linear approximation constraints in the master MIP subproblem is a delicate manner, and has an effect on the solution quality when the nonlinear subproblems are nonconvex. The user can consider several problem-dependent strategies to adjust the penalty values, and implement them inside the basic AOA algorithm.

Example of modified procedure

The following procedure is a variant of the termination procedure provided in the previous section. Assuming that the two parameters that refer to the previous and current NLP objective function values have been properly set in the procedure that solves the NLP subproblem, then termination is invoked whenever there is insufficient progress between two subsequent NLP solutions, or between the objective values of the master MIP problem and the current NLP subproblem. The third termination criterion is the number of iterations reaching its maximum.

```
return when ( MINLPAlgorithmHasFinished );

if    (not MINLPSolutionImprovement( NLPCurrentObjectiveValue,
                                      NLPPreviousObjectiveValue ))
```

```
      or (not MINLPSolutionImprovement( GMP::Solution::GetObjective(GMINLP, SolNumb),
                                        NLPCurrentObjectiveValue ))
      or ( IterationCounter = IterationMax ) then

       MINLPTerminate;

   else
       ! Prepare for next iteration

       IterationCount += 1 ;
       GMP::Solution::SetIterationCount( GMINLP, SolNumb, IterationCount ) ;
       GMP::Instance::AddIntegerEliminationRows( GMIP, SolNumb, EliminationCount ) ;
       EliminationCount += 1 ;
   endif ;
```

Conclusion

The above paragraphs indicate just a few of the ways in which you can alter the basic implementation of the outer approximation algorithm in AIMMS. Of course, it is not necessary to develop your own variant. Whenever you need to solve a MINLP model using the AOA algorithm, you can simply call the basic implementation described in the previous section. As soon as you can see improved ways to solve a particular model, you can apply your own ideas by modifying the procedures as you see fit.

Bibliography

[Du86] M.A. Duran and I.E. Grossmann, *An outer-approximation algorithm for a class of mixed-integer nonlinear programs*, Mathematical Programming **36** (1986), 307–339.

[Qu92] I. Quesada and I.E. Grossmann, *An lp/nlp based branch and bound algorithm for bonvex minlp optimization problems*, Computers and Chemical Engineering **16** (1992), 937–947.