AIMMS Language Reference - Automatic Benders Decomposition

This file contains only one chapter of the book. For a free download of the complete book in pdf format, please visit www.aimms.com.

AIMMS B.V.
Diakenhuisweg 29-35
2033 AP Haarlem
The Netherlands
Tel.: +31 23 5511512

AIMMS Inc.
11711 SE 8th Street
Suite 303
Bellevue, WA 98005
USA
Tel.: +1 425 458 4024

AIMMS Pte. Ltd.
55 Market Street #10-00
Singapore 048941
Tel.: +65 6521 2827

AIMMS
SOHO Fuxing Plaza No.388
Building D-71, Level 3
Madang Road, Huangpu District
Shanghai 200025
China
Tel.: ++86 21 5309 8733

Email: info@aimms.com
WWW: www.aimms.com

Chapter 21

Automatic Benders' Decomposition

Important note

The solver CPLEX has its own implementation of the Benders' decomposition algorithm. An important difference is that the algorithm in CPLEX supports multiple subproblems. CPLEX allows you to specify the decomposition by assigning the variables to the master problem or a subproblem by using the procedure `GMP::Benders::SetDecomposition`. For more information see the CPLEX option `Benders_strategy`.

Introduction

Benders' decomposition, introduced by Jacques F. Benders in 1962 ([Be62]), is an algorithm that decomposes a problem into two simpler parts. The first part is called the master problem and solves a relaxed version of the problem to obtain values for a subset of the variables. The second part is often called the subproblem (or slave problem or auxiliary problem) and finds values for the remaining variables fixing the variables of the master problem. If the problem contains integer variables then typically they become part of the master problem while the continuous variables become part of the subproblem.

Cuts

The solution of the subproblem is used to cut off the solution of the master problem by adding one or more constraints ("cuts") to the master problem. This process of iteratively solving master problems and subproblems is repeated until no more cuts can be generated. The combination of the variables found in the last master problem and subproblem iteration forms the solution to the original problem.

Reduced solution times possible

For particular optimization problems, Benders' decomposition may lead to a good, or even the optimal, solution in relatively few iterations. In such cases, employing Benders' decomposition results in drastically reduced solution times compared to solving the original problem. For other problems, however, the progress per iteration is so small that there is no positive, or even an adversary, effect by applying Benders' decomposition. Upfront, it is hard predict whether or not there will be positive effects for your particular model.

Hard to implement manually

Implementing Benders' decomposition from scratch for a particular problem is a non-trivial and error-prone task. Because duality theory plays an important role, the process often involves explicitly working out the dual formulation of the subproblem—and keeping it up-to-date when you make changes to the original problem. Given the uncertainty whether Benders' decomposition will lead to an improvement in solution times at all, a manual implementation may not be a prospect to look forward to.

Automatic Benders' decomposition in AIMMS

For AIMMS, on the other hand, generating the master and slave problems in an automated fashion is a fairly straightforward task, given a generated mathematical program and the collection of variables that should go into the master problem. With such an automated scheme, verifying whether your particular model will benefit from Benders' decomposition becomes completely trivial. With just a few lines of code, and simply re-solving your model you will get immediate insight into the benefits of Benders' decomposition for your model.

Classical versus modern

The Benders' decomposition module in AIMMS implements both the classical Benders' decomposition algorithm and a modern version. By the classical approach we mean the algorithm described above that solves an alternating sequence of master problems and subproblems, and that, in principle, will work for any problem type. The modern approach will only work for problems containing integer variables. In the modern approach, the algorithm will solve only a *single* master MIP problem, where subproblems are solved whenever the MIP solver finds a solution for the master problem, using callbacks provided by modern MIP solvers.

Several algorithms

Besides the classical and the modern algorithm, the Benders' decomposition module in AIMMS also implements a two phase algorithm that solves a relaxed problem in the first phase and the original problem in the second phase. In addition, the module offers you the flexibility to solve the subproblem as a primal or dual problem, to normalize the subproblem to get better feasibility cuts, and so on.

Limitations of current implementation

Benders' decomposition in AIMMS can be used for solving Mixed-Integer Programming (MIP) problems and Linear Programming (LP) problems. Currently it cannot be used to solve nonlinear problems. Also, the current implementation does not support multiple subproblems which could be efficient in case the subproblem has a block diagonal structure. This implies that the current implementation cannot be used to solve (two stage) stochastic programming problems with a subproblem for each scenario.

Open algorithm

Benders' decomposition in AIMMS is implemented as a system module with the name GMP Benders Decomposition. You can install this module using the **Install System Module** command in the AIMMS **Settings** menu. The Benders' decomposition algorithms are implemented in the AIMMS language. Some supporting functions that are computationally difficult, or hard to express in the AIMMS language, have been added to the GMP library in support of the Benders' decomposition algorithm. Besides this small number of fixed subtasks, the implementation is an open algorithm; 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 problem.

This chapter

This chapter starts with a quick start for using Benders' decomposition in AIMMS for those already familiar with Benders' decomposition. Following a brief introduction to the problem statement, we discuss the Benders' decomposition algorithm as it can be found in several textbooks. Next we describe the implementation of the classic Benders' decomposition algorithm using procedures in the AIMMS language that are especially designed to support the open approach. This section is important for users that want to modify the algorithm. Next, we discuss in detail the parameters inside the Benders' module that can be used to control the Benders' decomposition algorithm. We continue by describing the implementation of a modern Benders' decomposition algorithm. The chapter ends by introducing a two phase algorithm that solves a problem by using information gathered while solving a relaxed version of the problem, and we also describe its implementation.

21.1 Quick start to using Benders' decomposition

System module

The system module with the name GMP Benders Decomposition implements the Benders' decomposition algorithm. You can add this module to your project using the **Install System Module** command in the AIMMS **Settings** menu. This module contains three procedures that can be called, each implementing a different algorithm.

Classic algorithm

The procedure DoBendersDecompositionClassic inside this module implements the classic version of the Benders' decomposition algorithm, in which the master problem and the subproblem are solved in an alternating sequence.

Modern algorithm

The procedure DoBendersDecompositionSingleMIP inside the module implements the modern approach for MIP problems which solves only a single MIP problem; the subproblem is solved whenever the MIP solver finds a solution for the master problem (using callbacks).

Two phase algorithm

The procedure DoBendersDecompositionTwoPhase inside the module implements a two phase algorithm for MIP problems. In the first phase it solves the relaxed problem (in which the integer variables become continuous) using the classic Benders decomposition algorithm. The Benders' cuts found in the first phase are then added to the master problem in the second phase after which the MIP problem is solved using either the classic or modern approach of the Benders decomposition algorithm.

The procedure DoBenders-Decomposition-Classic

The procedure DoBendersDecompositionClassic has two input arguments:

1. MyGMP, an element parameter with range AllGeneratedMathematicalPrograms, and
2. MyMasterVariables, a subset of the predefined set AllVariables defining the variables in the master problem.

For a MIP problem, the integer variables typically become the variables of the master problem, although it is possible to also include continuous variables in the set of master problem variables. The DoBendersDecompositionClassic procedure is called as follows:

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

GMPBenders::DoBendersDecompositionClassic( generatedMP, AllIntegerVariables );
```

Here SymbolicMP is the symbolic mathematical program containing the MIP model, and generatedMP is an element parameter in the predefined set AllGeneratedMathematicalPrograms. GMPBenders is the prefix of the Benders' module. The implementation of this procedure will be discussed in Section 21.3.

The procedure DoBenders-Decomposition-SingleMIP

The procedure DoBendersDecompositionSingleMIP has the same input arguments as the procedure DoBendersDecompositionClassic. The DoBendersDecomposition-SingleMIP procedure is called as follows:

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

GMPBenders::DoBendersDecompositionSingleMIP( generatedMP, AllIntegerVariables );
```

This procedure can only be used if the original problem contains some integer variables. The implementation of this procedure will be discussed in Section 21.6.

The procedure DoBenders-Decomposition-TwoPhase

The procedure DoBendersDecompositionTwoPhase has one additional argument compared to the procedure DoBendersDecompositionClassic. Namely, the third argument UseSingleMIP is used to indicate whether the second phase should use the classic algorithm (value 0) or the modern algorithm (value 1). The procedure is called as follows if the modern algorithm should be used:

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

GMPBenders::DoBendersDecompositionTwoPhase( generatedMP, AllIntegerVariables, 1 );
```

This procedure should only be used if the original problem contains some integer variables. The implementation of this procedure will be discussed in Section 21.7.

Combining procedure

To make it easier for you to switch between the three algorithms, the module also implements the procedure `DoBendersDecomposition` that calls one of the three procedures above based on the Benders' mode. The first two arguments of this procedure are the same as before, namely `MyGMP` and `MyMasterVariables`. The third argument, `BendersMode`, is an element parameter that defines the Benders' mode and can take value `'Classic'`, `'Modern'`, `'TwoPhaseClassic'` or `'TwoPhaseModern'`. The procedure is called as follows if the two phase algorithm should be used with the modern algorithm for the second phase:

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

GMPBenders::DoBendersDecomposition( generatedMP, AllIntegerVariables,
                                    'TwoPhaseModern' );
```

If the problem contains no integer variables then only mode `'Classic'` can be used.

Control parameters

The Benders' module defines several parameters that influence the Benders' decomposition 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 21.1. The parameters that are not self-

Parameter	Default	Range	Subsection
BendersOptimalityTolerance	1e-6	[0,1]	
IterationLimit	1e7	{1,maxint}	
TimeLimit	1e9	[0,inf)	
CreateStatusFile	0	{0,1}	
UseDual	0	{0,1}	21.5.1
FeasibilityOnly	1	{0,1}	21.5.2
NormalizationType	1	{0,1}	21.5.3
UseMinMaxForFeasibilityProblem	1	{0,1}	21.5.4
AddTighteningConstraints	1	{0,1}	21.5.5
UseStartingPointForMaster	0	{0,1}	21.5.6
UsePresolver	0	{0,1}	21.5.7

Table 21.1: Control parameters in the Benders' module

explanatory are explained in Section 21.5; the last column in the table refers to the subsection that discusses the corresponding parameter.

Optimality tolerance

The optimality tolerance, as controlled by the parameter `BendersOptimalityTolerance`, guarantees that a solution returned by the Benders' decomposition algorithm lies within a certain percentage of the optimal solution.

Solver options

The parameters `BendersOptimalityTolerance`, `IterationLimit` and `TimeLimit` are used by the classic algorithm (and the first phase of the two phase algorithm). For the modern algorithm, the corresponding general solver options, `MIP_relative_optimality_tolerance`, `iteration_limit` and `time_limit` respectively, are used.

21.2 Problem statement

MIP

We consider the following generic mixed-integer programming model to explain Benders' Decomposition. (The notation used is similar to [Fi10].)

Minimize:

$$c^T x + d^T y$$

Subject to:

$$\begin{aligned} Ax &\leq b \\ Tx + Qy &\leq r \\ x &\in \mathbb{Z}_+^n \\ y &\in \mathbb{R}_+^m \end{aligned}$$

The variable x is integer and the variable y is continuous. The matrix T may contain empty constraints but we assume that the matrix Q does not contain any empty constraint. So, the model can have constraints that only contain continuous variables.

Limitation

Benders' Decomposition cannot be used if the model contains only integer variables. If the number of continuous variable is small compared to the number of integer variables then Benders' Decomposition will very likely be inefficient.

21.3 Benders' decomposition - Textbook algorithm

Master problem

The basic Benders' decomposition algorithm as explained in several textbooks (e.g., [Ne88], [Ma99]) works as follows. After introducing an artificial variable $\eta = d^T y$, the master problem relaxation becomes:

Minimize:

$$c^T x + \eta$$

Subject to:

$$\begin{aligned} Ax &\leq b \\ \eta &\geq \overline{\eta} \\ x &\in \mathbb{Z}_+^n \end{aligned}$$

Here $\overline{\eta}$ is a lower bound on the variable η that AIMMS will automatically derive. For example, if the vector d is nonnegative then we know that 0 is a lower bound on $d^T y$ since we assumed that the variable y is nonnegative, and therefore we can take $\overline{\eta} = 0$. We assume that the master problem is bounded.

Subproblem

After solving the master problem we obtain an optimal solution, denoted by (x^*, η^*) with x^* integer. This solution is fixed in the subproblem which we denote by $PS(x^*)$:

Minimize:

$$d^T y$$

Subject to:

$$Qy \le r - Tx^*$$
$$y \in \mathbb{R}^m_+$$

Note that this subproblem is a linear programming problem in which the continuous variable y is the only variable.

Dual subproblem

Textbooks that explain Benders' decomposition often use the dual of this subproblem because duality theory plays an important role, and the Benders' optimality and feasibility cuts can be expressed using the variables of the dual problem. The dual of the subproblem $PS(x^*)$ is given by:

Maximize:

$$r - \pi^T (Tx^*)$$

Subject to:

$$\pi^T Q \ge d^T$$
$$\pi \ge 0$$

We denote this problem by $DS(x^*)$.

Optimality cut

If this subproblem is feasible, let z^* denote the optimal objective value and $\overline{\pi}$ an optimal solution of $DS(x^*)$. If $z^* \le \eta^*$ then the current solution (x^*, η^*) is a feasible and optimal solution of our original problem, and the Benders' decomposition algorithm only needs to solve $PS(x^*)$ to obtain optimal values for variable y. If $z^* > \eta^*$ then the Benders' optimality cut $\eta \ge \overline{\pi}^T (r - Tx)$ is added to the master problem and the algorithm continues by solving the master problem again.

Feasibility cut

If the dual subproblem is unbounded, implying that the primal subproblem is infeasible, then an unbounded extreme ray $\overline{\pi}$ is selected and the Benders' feasibility cut $\overline{\pi}^T (r - Tx) \le 0$ is added to the master problem. Modern solvers like CPLEX and GUROBI can provide an unbounded extreme ray in case a LP problem is unbounded. After adding the feasibility cut the Benders' decomposition algorithm continues by solving the master problem.

21.4 Implementation of the classic algorithm

The procedure DoBenders-Decomposition

In this section we show the implementation of the classic Benders' decomposition algorithm. It follows the classic approach of solving the master problem and the subproblem in an alternating sequence. The procedure DoBenders-Decomposition, introduced in Section 21.1, implements the classic algorithm.

Focus on textbook algorithm

The Benders' cuts can be generated in several ways; in this section we focus on the approach used in the textbook algorithm of the previous section (Section 21.3). The textbook algorithm uses the dual formulation of the subproblem and can add both Benders' optimality and feasibility cuts.

Calling DoBendersDecom-positionClassic

We have to change some of the control parameters of Table 21.1 to let the DoBendersDecomposition procedure execute the textbook algorithm. The relevant changes are listed below.

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

! Settings needed to run textbook algorithm:
GMPBenders::FeasibilityOnly := 0;
GMPBenders::AddTighteningConstraints := 0;
GMPBenders::UseDual := 1;
GMPBenders::NormalizationType := 0;

GMPBenders::DoBendersDecompositionClassic( generatedMP, AllIntegerVariables );
```

Implementation of DoBendersDecom-positionClassic

The DoBendersDecompositionClassic procedure starts by making copies of its input arguments. Next the master problem and the subproblem are created. For the subproblem we also create a solver session which gives us more flexibility passing and retrieving subproblem related information. The parameters for the number of optimality and feasibility cuts are reset. Finally, the procedure calls another procedure, namely BendersAlgorithm, and finishes by deleting the master problem and the subproblem. For the sake of brevity and clarity, we leave out parts of the code that handle details like creating a status file; this will also be the case for the other pieces of code shown in this chapter.

```
OriginalGMP := MyGMP ;
VariablesMasterProblem := MyMasterVariables ;

! Create (Relaxed) Master problem.
gmpM := GMP::Benders::CreateMasterProblem( OriginalGMP, VariablesMasterProblem,
                        'BendersMasterProblem',
                        feasibilityOnly : FeasibilityOnly,
                        addConstraints : AddTighteningConstraints ) ;

! Create Subproblem.
gmpS := GMP::Benders::CreateSubProblem( OriginalGMP, gmpM, 'BendersSubProblem',
                        useDual : UseDual,
                        normalizationType : NormalizationType );
```

```
solsesS := GMP::Instance::CreateSolverSession( gmpS ) ;

NumberOfOptimalityCuts  := 0;
NumberOfFeasibilityCuts := 0;

! Start the actual Benders' decomposition algorithm.
BendersAlgorithm;

GMP::Instance::Delete( gmpM );
GMP::Instance::Delete( gmpS );
```

The procedure BendersAlgorithm

The BendersAlgorithm procedure implements the actual Benders' decomposition algorithm. It initializes the algorithm by resetting the parameters for the number of iterations, etc. Next the master problem is solved. The separation step solves the subproblem and checks whether the current solution is optimal. If it is not optimal then the algorithm creates a constraint ("cut") that separates the current solution from the set of feasible solutions. This constraint is added to the master problem enforcing that the current solution of the master problem will not be found again if we solve the master problem once again. This alternating sequence of solving master problems and subproblems is repeated until a stopping criterion is met.

```
InitializeAlgorithm;

while ( not BendersAlgorithmFinished ) do

    NumberOfIterations += 1;

    SolveMasterProblem;

    if ( UseDual ) then
        if ( FeasibilityOnly ) then
            SeparationFeasibilityOnlyDual;
        else
            SeparationOptimalityAndFeasibilityDual;
        endif;
    else
        if ( FeasibilityOnly ) then
            SeparationFeasibilityOnly;
        else
            SeparationOptimalityAndFeasibility;
        endif;
    endif;

endwhile;
```

Separation

The code above shows four possible ways of performing the separation step. The textbook algorithm uses the procedure SeparationOptimalityAndFeasibilityDual which we will discuss below. The other three separation procedures are discussed in Appendix B.

The procedure SolveMasterProblem

The implementation of the SolveMasterProblem procedure is straightforward. This procedure solves the Benders' master problem and retrieves its objective value after checking the program status. If the program status is infeasible or unbounded then the algorithm terminates.

```
GMP::Instance::Solve( gmpM );

ProgramStatus := GMP::Solution::GetProgramStatus( gmpM, 1 ) ;

if ( ProgramStatus = 'Infeasible' ) then
    return AlgorithmTerminate( 'Infeasible' );
elseif ( ProgramStatus = 'Unbounded' ) then
    return AlgorithmTerminate( 'ProgramNotSolved' );
endif;

ObjectiveMaster := GMP::Instance::GetObjective( gmpM );
```

The procedure SeparationOptimalityAndFeasibilityDual

The procedure SeparationOptimalityAndFeasibilityDual is called by the Benders' decomposition algorithm in case the dual of the Benders' subproblem is used and if both optimality and feasibility cuts can be generated by the algorithm (we will discuss in Section 21.5 the case in which only feasibility cuts are generated). This procedure updates the dual subproblem and solves it. If the dual subproblem is unbounded then a feasibility cut is added to the master problem (using an unbounded extreme ray; see the next paragraph). If the subproblem is bounded and optimal then the objective value of the subproblem is compared to the objective value of the master problem to check whether the algorithm has found an optimal solution for the original problem. If the solution is not optimal yet then an optimality cut is added to the master problem, using the level values of the variables in the solution of the dual subproblem.

```
return when ( BendersAlgorithmFinished );

GMP::Benders::UpdateSubProblem( gmpS, gmpM, 1, round : 1 );

GMP::SolverSession::Execute( solsesS ) ;
GMP::Solution::RetrieveFromSolverSession( solsesS, 1 ) ;

ProgramStatus := GMP::Solution::GetProgramStatus( gmpS, 1 ) ;

if ( ProgramStatus = 'Unbounded' ) then

    ! Add feasibility cut to the Master problem.
    NumberOfFeasibilityCuts += 1;
    GMP::Benders::AddFeasibilityCut( gmpM, gmpS, 1, NumberOfFeasibilityCuts );

else

    ! Check whether optimality condition is satisfied.
    ObjectiveSubProblem := GMP::SolverSession::GetObjective( solsesS );

    if ( SolutionImprovement( ObjectiveSubProblem, BestObjective ) ) then
        BestObjective := ObjectiveSubProblem;
    endif;
```

```
    if ( SolutionIsOptimal( ObjectiveSubProblem, ObjectiveMaster ) ) then
        return AlgorithmTerminate( 'Optimal' );
    endif;

    ! Add optimality cut to the Master problem.
    NumberOfOptimalityCuts += 1;
    GMP::Benders::AddOptimalityCut( gmpM, gmpS, 1, NumberOfOptimalityCuts );

endif;
```

Unbounded extreme ray

In textbooks, if the dual subproblem is unbounded then an unbounded extreme ray is chosen and used to generate a feasibility cut. Choosing such an unbounded extreme ray is not trivial but luckily modern solvers like CPLEX and GUROBI can compute an unbounded extreme ray upon request. It is stored in the `.Level` suffix of the variables. The downside is that preprocessing by CPLEX or GUROBI has to be switched off which can have a negative impact on the performance. So, if the textbook algorithm is selected in which the dual subproblem is used and both optimality and feasibility cuts can be generated by the algorithm, the solver options for switching on the calculation of unbounded extreme ray and for switching off the preprocessor are set during the initialization of the Benders' decomposition algorithm:

```
if ( UseDual and ( not FeasibilityOnly ) ) then
    rval := GMP::SolverSession::SetOptionValue( solsesS, 'unbounded ray', 1 );
    if ( rval = 0 ) then
        halt with "Solver must support unbounded extreme rays.";
        return;
    endif;

    rval := GMP::SolverSession::SetOptionValue( solsesS, 'presolve', 0 );
    if ( rval = 0 ) then
        halt with "Switching off the solver option 'presolve' failed.";
        return;
    endif;
endif;
```

If the solver does not support unbounded extreme rays then the textbook algorithm cannot be used.

The procedure `AlgorithmTerminate`

The procedure `AlgorithmTerminate` is called whenever the Benders' decomposition algorithm is finished. Appropriate values are assigned to the program and solver status of the original problem. If the algorithm has found an optimal solution then the solutions of the last master problem and last subproblem are combined into an optimal solution for the original problem. In the code below, the uncommon situation in which the algorithm terminates after hitting the iteration limit has been omitted.

```
BendersAlgorithmFinished := 1;

if ( ProgrStatus = 'Optimal' ) then
    GMP::Solution::SetProgramStatus( OriginalGMP, 1, 'Optimal' ) ;
    GMP::Solution::SetSolverStatus( OriginalGMP, 1, 'NormalCompletion' ) ;

    GMP::Solution::SendToModel( gmpS, 1 ) ;

    GMP::Solution::SendToModelSelection( gmpM, 1, VariablesMasterProblem,
                                         AllSuffixNames );
    GMP::Solution::RetrieveFromModel( OriginalGMP, 1 );

    GMP::Solution::SetObjective( OriginalGMP, 1, BestObjective );
    GMP::Solution::SendToModel( OriginalGMP, 1 );
elseif ( ProgrStatus = 'Infeasible' ) then
    GMP::Solution::SetProgramStatus( OriginalGMP, 1, 'Infeasible' ) ;
    GMP::Solution::SetSolverStatus( OriginalGMP, 1, 'NormalCompletion' ) ;
elseif ( ProgrStatus = 'Unbounded' ) then
    GMP::Solution::SetProgramStatus( OriginalGMP, 1, 'Unbounded' ) ;
    GMP::Solution::SetSolverStatus( OriginalGMP, 1, 'NormalCompletion' ) ;
else
    GMP::Solution::SetProgramStatus( OriginalGMP, 1, 'ProgramNotSolved' ) ;
    GMP::Solution::SetSolverStatus( OriginalGMP, 1, 'SetupFailure' ) ;
endif;
```

21.5 Control parameters that influence the algorithm

This section

Some of the control parameters of Table 21.1 can be used to influence the behavior of the Benders' decomposition algorithm. We discuss these parameters in this section.

21.5.1 Primal versus dual subproblem

Parameter UseDual

In the textbook algorithm the dual of the subproblem is used. It is also possible to use the primal of the subproblem instead. This is controlled by the parameter UseDual. By default the Benders' decomposition algorithm uses the primal subproblem.

Dual solution

If the primal subproblem is solved and it appears to be feasible then the dual solution is used to construct an optimality cut. By the dual solution we mean the shadow prices of the constraints and the reduced costs of the variables in the primal subproblem.

Feasibility problem

If the primal subproblem is infeasible then another problem is solved to find a solution of minimum infeasibility (according to some measurement). The *feasibility problem* of $PS(x^*)$ (see Section 21.3) is denoted by $PFS(x^*)$ and defined by:

Minimize:

$$z$$

Subject to:

$$Qy - z \leq r - Tx^*$$
$$y \in \mathbb{R}^m_+$$
$$z \in \mathbb{R}$$

Here z is a scalar variable. The dual solution of this feasibility problem is used to create a feasibility cut which is added to the master problem.

Alternative feasibility problem

The feasibility problem above minimizes the maximum infeasibility among all constraints. It is also possible to minimize the sum of infeasibilities over all constraints; this is controlled by the parameter `UseMinMaxForFeasibilityProblem` which we discuss in Subsection 21.5.4. Also the parameter `NormalizationType` influences the formulation of the feasibility problem; see Subsection 21.5.3. Note that the feasibility problem is always feasible and bounded. Note further that if the optimal objective value of the feasibility problem is 0 or negative then the corresponding subproblem is feasible.

Relationship with parameter `FeasibilityOnly`

In the next subsection we discuss the parameter `FeasibilityOnly`. This parameter has a big influence on how the subproblem is created, for both the primal and dual subproblem. In some cases the subproblem can become a pure feasibility problem.

21.5.2 Subproblem as pure feasibility problem

Until now

By so far we assumed that the Benders' decomposition algorithm first tries to solve the subproblem to optimality to either conclude that the combined solution of the master problem and subproblem forms an optimal solution for the original problem, or to create an optimality cut that is added to the master problem. If the primal or dual subproblem appears to be infeasible or unbounded respectively, then a feasibility problem is solved (if we used the primal subproblem) or an unbounded extreme ray is calculated (if we used the dual subproblem) to create a feasibility cut.

Benders' subproblem always infeasible

For some problems the Benders' subproblem will (almost) always be infeasible unless an optimal solution of the original problem is found. For example, assume that the variables that become part of the subproblem have no objective coefficients. (In the MIP problem of Section 21.2 this is equivalent to the vector d being equal to 0.) In that case the Benders' decomposition algorithm tries to find a solution for the master problem that remains feasible if we also consider the part of the model that became the subproblem. The algorithm is finished if such a solution is found. Until then all subproblems will be infeasible. In

that case it is useless to try to solve the subproblem to optimality (which will always fail) but instead directly solve a feasibility problem for the subproblem.

Reformulation

It is possible to let the AIMMS automatically reformulate the original problem such that the variables that become part of the subproblem have no longer objective coefficients. (This reformulation exists only temporary while the function `GMP::Benders::CreateMasterProblem` is executed; the user will not notice anything inside his project.) For the MIP problem of Section 21.2 the reformulated problem becomes:

Minimize:

$$c^T x + \eta$$

Subject to:

$$\begin{aligned} d^T y - \eta &\le 0 \\ Ax &\le b \\ Tx + Qy &\le r \\ x &\in \mathbb{Z}_+^n \\ y &\in \mathbb{R}_+^m \\ \eta &\in \mathbb{R} \end{aligned}$$

If we assign the new continuous variable η, together with the integer variable x, to the master problem then the subproblem variables no longer have objective coefficients. As a consequence, the subproblem will always be infeasible (unless an optimal solution is found).

Parameter `FeasibilityOnly`

The parameter `FeasibilityOnly` can be used to control whether AIMMS should reformulate the original problem as explained above. AIMMS will do so if the value of this parameter equals 1, which is the default value. Also, if parameter `FeasibilityOnly` equals 1 then the Benders' decomposition algorithm will no longer solve the primal subproblem before solving the feasibility problem. Instead it will directly solve the feasibility problem.

Primal subproblem

After reformulating the original problem, the primal of the subproblem will be different from $PS(x^*)$ of Section 21.3, namely:

Minimize:

$$0$$

Subject to:

$$\begin{aligned} d^T y &\le \eta^* \\ Qy &\le r - Tx^* \\ y &\in \mathbb{R}_+^m \end{aligned}$$

We denote this primal subproblem by $PS'(x^*, \eta^*)$. The feasibility problem will also become slightly different, as compared to $PFS(x^*)$ of Subsection 21.5.1, namely:

Minimize:

$$z$$

Subject to:

$$\begin{aligned} d^T y - z &\leq \eta^* \\ Qy - z &\leq r - Tx^* \\ y &\in \mathbb{R}^m_+ \\ z &\in \mathbb{R} \end{aligned}$$

We denote this feasibility problem by $PFS'(x^*, \eta^*)$. If the optimal objective value of this feasibility problem is 0 or negative then we have found an optimal solution for the original problem, and the Benders' decomposition algorithm terminates. Otherwise the dual solution of the feasibility problem is used to add a feasibility cut to the master problem, and the algorithm continues by solving the master problem.

Dual subproblem

We have seen before that if we use the dual of the subproblem and parameter `FeasibilityOnly` equals 0 then the Benders' decomposition algorithm will first solve the dual subproblem and, if that subproblem is infeasible, use an unbounded extreme ray to create a feasibility cut. If parameter `FeasibilityOnly` equals 1 then the algorithm follows a different route. Consider the dual formulation of the above problem, the feasibility problem for $PS'(x^*, \eta^*)$:

Maximize:

$$\pi^T(r - Tx^*) + \pi_0\eta^*$$

Subject to:

$$\begin{aligned} \pi^T Q + \pi_0 d^T &\geq 0 \\ 1^T\pi + \pi_0 &= 1 \\ \pi, \pi_0 &\geq 0 \end{aligned}$$

Here 1^T denotes a vector of all 1's. We denote this problem by $DS'(x^*, \eta^*)$. This problem is always feasible and bounded. The Benders' decomposition algorithm uses this problem as the (dual) subproblem if the parameters `FeasibilityOnly` and `UseDual` equal 1. If the optimal objective value of this problem is 0 or negative then we have found an optimal solution for the original problem, and the Benders' decomposition algorithm terminates. Otherwise the solution of this problem is used to add a feasibility cut to the master problem, and the algorithm continues by solving the master problem.

Disadvantage

A serious disadvantage of reformulating the problem, as done in this section, is that a first feasible solution (which will be optimal) for the original problem will be found just before the Benders' decomposition algorithm terminates. This means that the "gap" between the lower and upper bound on the objective value is meaningless, and therefore this measurement of progress toward finding and proving optimality by the algorithm is not available. However, this

disadvantage only occurs when using the classic Benders' decomposition algorithm. For the modern approach in which only a single MIP problem is solved, see Section 21.6, the algorithm finds feasible solutions for the original problem during the solution process and therefore the "gap" exists.

21.5.3 Normalization of feasibility problem

Normalization

In the previous subsection we introduced the dual subproblem $DS'(x^*, \eta^*)$ which contains the normalization condition

$$1^T \pi + \pi_0 = 1. \tag{NC1}$$

In order to obtain better feasibility cuts, Fischetti et al. (in [Fi10]) proposed another normalization condition. The matrix T often contains null constraints which correspond to constraints that do not depend on x. These are "static" conditions in the subproblem that are always active. According to Fischetti et al. there is no reason to penalize the corresponding dual multiplier π_i. The new normalization condition then becomes

$$\sum_{i \in I(T)} \pi_i + \pi_0 = 1 \tag{NC2}$$

where $I(T)$ indexes the nonzero constraints of matrix T.

Parameter NormalizationType

The parameter `NormalizationType` controls which normalization condition is used. If it equals 0 then normalization condition (NC1) is used, else (NC2). The Benders' decomposition algorithm uses (NC2) by default because various computational experiments showed a better performance with this normalization condition.

Translation to primal subproblem

We can apply the normalization rule of Fischetti et al. also if we use the primal subproblem. In the corresponding feasibility problem, we then only add variable z for the nonzero rows of T. The relevant constraints in $PFS'(x^*, \eta^*)$ then become:

$$\begin{aligned} (Qy)_i - z_i &\le r_i - (Tx^*)_i \qquad && i \in I(T) \\ (Qy)_i &\le r_i && i \notin I(T) \end{aligned}$$

The feasibility problem can be normalized in this way regardless of the setting of parameter `FeasibilityOnly`.

Exception

In case the parameter `UseDual` equals 1 and the parameter `FeasibilityOnly` equals 0 then no feasibility problem is solved to derive a feasibility cut. Instead an unbounded extreme ray for the unbounded dual subproblem is used. Therefore, in that case the parameter `NormalizationType` is ignored.

21.5.4 Feasibility problem mode

Parameter UseMinMaxForFeasibilityProblem

The parameter UseMinMaxForFeasibilityProblem determines what kind of infeasibility is minimized: the maximum infeasibility among all constraints (value 1, the default) or the sum of infeasibilities over all constraints (value 0). If the sum of the infeasibilities over all constraints is used then also the normalization rule of Fischetti et al. can be used, as controlled by the parameter NormalizationType. This parameter is ignored if the parameter UseDual equals 1.

21.5.5 Tightening constraints

Illustrative example

If the Benders' master problem is created, using the function GMP::Benders::CreateMasterProblem, then AIMMS can try to automatically add valid constraints to the master problem that will cut off some infeasible solutions. This is best illustrated by the following MIP example.

Minimize:

$$\sum_i x_i$$

Subject to:

$$\begin{aligned} y_i &\le u_i x_i \qquad \forall i \\ \sum_i y_i &\ge b \\ x &\in \{0,1\} \\ y &\ge 0 \end{aligned}$$

We assume that u and b are strictly positive parameters. The binary variable x is assigned to the master problem and the continuous variable y to the subproblem. For this example, the initial master problem has no constraints (besides the integrality restriction on x) and therefore $x = 0$ is the optimal solution of the initial master problem. Clearly, for $x = 0$ our MIP example has no solution. Adding the constraint

$$\sum_i u_i x_i \ge b$$

to the master problem cuts off the $x = 0$ solution. Note that this constraint is redundant in the original MIP example. By adding these kind of master-problem-tightening constraints we hope that the Benders' decomposition algorithm requires less iterations to find an optimal solution.

Parameter AddTightening-Constraints

Adding tightening constraints to the master problem is controlled by the parameter AddTighteningConstraints. If this parameter equals 1, its default, then AIMMS will try to find and add tightening constraints. Computational experiments indicate that in general the Benders' decomposition algorithm benefits from adding these tightening constraints.

21.5.6 Using a starting point

Parameter UseStarting-PointForMaster

The parameter UseStartingPointForMaster can be used to let the classic Benders' decomposition algorithm start from a "good" solution. This solution can be obtained from a heuristic and must be a feasible solution for the master problem. The solution should be copied into the level suffix of the problem variables before the Benders' decomposition algorithm is called. If this parameter is set to 1 then the algorithm will skip the solve of the first master problem. Instead, the master problem variable x^* will be fixed in the subproblem $PS(x^*)$ according to the starting point, and the algorithm will continue by solving the subproblem.

21.5.7 Using the AIMMS Presolver

Parameter UsePresolver

The Benders' decomposition algorithm can use the AIMMS Presolver at the start. In that case the algorithm will use the preprocessed model instead of the original model. By preprocessing the model it might become smaller and easier to solve. The parameter UsePresolver can be used to switch on the preprocessing step.

21.6 Implementation of the modern algorithm

Single MIP

When solving a MIP problem, the classic Benders' decomposition algorithm often spends a large amount of time in solving the master MIP problems in which a significant amount of rework is done. In the modern approach only one single master MIP problem is solved. Whenever the solver finds a feasible integer solution for the master problem, the subproblem $PS(x^*)$ is solved after fixing the master problem variable x^* according to this integer solution.

Callbacks

Modern MIP solvers like CPLEX and GUROBI allow the user control over the solution process by so-called *callbacks*. Callbacks allow user code in AIMMS to be executed regularly during an optimization process. If the solver finds a new candidate integer solution then the user has the possibility to let the solver call one or more callback procedures. One of these callbacks is the callback for lazy constraints; that callback is used in the modern Benders' decomposition algorithm.

Feasible solutions

If no violated Benders' cut can be generated, after solving the subproblem, then we have found a feasible solution for the original problem and we can accept the current feasible integer solution as a "correct" solution for the master MIP problem. In the classic algorithm we would now be finished because we would know that no better solution of the original problem exists. In the modern algorithm we have to continue solving the master MIP problem because there might still exist a solution to the master MIP problem that results in a better solution for the original problem.

Lazy constraints

If a Benders' optimality or feasibility cut is found then this will be added as a so-called *lazy constraint* to the master MIP problem. Lazy constraints are constraints that represent one part of the model; without them the model would be incomplete. In this case the actual model that we want to solve is the original problem but we are solving the master MIP problem instead. The Benders' cuts represent the subproblem part of the model and we add them whenever we find one that is violated.

Implementation of DoBenders-Decomposition-SingleMIP

In the remainder of this section we show the implementation of the modern Benders' decomposition algorithm as implemented by the procedure DoBendersDecompositionSingleMIP which was introduced in Section 21.1. Similar to the procedure DoBendersDecompositionClassic, the procedure DoBendersDecompositionSingleMIP starts by making copies of its input arguments. Next the master problem and the subproblem are created. The parameters for the number of optimality and feasibility cuts are reset. Finally, the procedure calls another procedure, namely BendersAlgorithmSingleMIP, and finishes by deleting the master problem and the subproblem. As before we leave out parts of the code that handle details like creating a status file, for the sake of brevity and clarity.

```
OriginalGMP := MyGMP ;
VariablesMasterProblem := MyMasterVariables ;

! Create (Relaxed) Master problem.
gmpM := GMP::Benders::CreateMasterProblem( OriginalGMP, VariablesMasterProblem,
                        'BendersMasterProblem',
                        feasibilityOnly : FeasibilityOnly,
                        addConstraints : AddTighteningConstraints ) ;

! Create Subproblem.
gmpS := GMP::Benders::CreateSubProblem( OriginalGMP, gmpM, 'BendersSubProblem',
                        useDual : UseDual,
                        normalizationType : NormalizationType );

solsesS := GMP::Instance::CreateSolverSession( gmpS ) ;

NumberOfOptimalityCuts  := 0;
NumberOfFeasibilityCuts := 0;

! Start the actual Benders' decomposition algorithm.
BendersAlgorithmSingleMIP;
```

```
GMP::Instance::Delete( gmpM );
GMP::Instance::Delete( gmpS );
```

The procedure Benders-Algorithm-SingleMIP

The BendersAlgorithmSingleMIP procedure initializes the algorithm by resetting the parameters for the number of iterations, etc. Then it calls the procedure SolveMasterMIP which does the actual work.

```
InitializeAlgorithmSingleMIP;

SolveMasterMIP;
```

The procedure SolveMasterMIP

The SolveMasterMIP procedure implements the actual Benders' decomposition algorithm using the modern approach. It first installs a lazy constraint callback for which the module implements four different versions. We assume that the control parameters have their default settings (see Table 21.1) in which case the procedure BendersCallbackLazyFeasOnlySingleMIP is installed. Next the master problem is solved and if a feasible solution is found, the subproblem is solved one last time to obtain a combined optimal solution for the original problem. Finally the algorithm terminates.

```
if ( UseDual ) then
    if ( FeasibilityOnly ) then
        GMP::Instance::SetCallbackAddLazyConstraint( gmpM,
                        'GMPBenders::BendersCallbackLazyFeasOnlyDualSingleMIP' );
    else
        GMP::Instance::SetCallbackAddLazyConstraint( gmpM,
                        'GMPBenders::BendersCallbackLazyOptAndFeasDualSingleMIP' );
    endif;
else
    if ( FeasibilityOnly ) then
        GMP::Instance::SetCallbackAddLazyConstraint( gmpM,
                        'GMPBenders::BendersCallbackLazyFeasOnlySingleMIP' );
    else
        GMP::Instance::SetCallbackAddLazyConstraint( gmpM,
                        'GMPBenders::BendersCallbackLazyOptAndFeasSingleMIP' );
    endif;
endif;

GMP::Instance::Solve( gmpM );

ProgramStatus := GMP::Solution::GetProgramStatus( gmpM, 1 ) ;

if ( ProgramStatus = 'Infeasible' ) then
    AlgorithmTerminateSingleMIP( 'Infeasible' );
else
    if ( FeasibilityOnly and not UseDual ) then
        ! Solve feasibility problem fixing the optimal solution of the
        ! Master problem.
        GMP::Solution::SendToModel( gmpM, 1 );

        ! Update feasibility problem and solve it.
        GMP::Benders::UpdateSubProblem( gmpF, gmpM, 1, round : 1 );
        GMP::Instance::Solve( gmpF );
```

```
    else
        ! Solve Subproblem fixing the optimal solution of the Master problem.
        GMP::Solution::SendToModel( gmpM, 1 );

        ! Update Subproblem and solve it.
        GMP::Benders::UpdateSubProblem( gmpS, gmpM, 1, round : 1 );
        GMP::Instance::Solve( gmpS );
    endif;

    AlgorithmTerminateSingleMIP( 'Optimal' );
endif;
```

The procedure BendersCallbackLazyFeasOnlySingleMIP

The callback procedure BendersCallbackLazyFeasOnlySingleMIP is called by the MIP solver whenever it finds a candidate integer solution for the master problem. This procedure retrieves the candidate integer solution from the MIP solver. Then it creates the feasibility problem for the (primal) subproblem if it does not exist yet. The feasibility problem is updated and solved. If its optimal objective value is larger than 0, indicating that the subproblem would have been infeasible, we add a feasibility cut as a lazy constraint to the master MIP. The MIP solver will not treat this candidate integer solution as a real solution. If the optimal objective value equals 0 (or is negative) then we do not add a lazy constraint in which case the MIP solver accepts the candidate solution as a real solution. Finally, the callback procedure returns 1 such that the solution process of the master MIP problem continues.

```
! Get MIP incumbent solution.
GMP::Solution::RetrieveFromSolverSession( ThisSession, 1 );
GMP::Solution::SendToModel( gmpM, 1 );

! Create feasibility problem corresponding to Subproblem (if it does not exist yet).
if ( not FeasibilityProblemCreated ) then
    gmpF := GMP::Instance::CreateFeasibility( gmpS, "FeasProb",
                            useMinMax : UseMinMaxForFeasibilityProblem );
    solsesF := GMP::Instance::CreateSolverSession( gmpF ) ;
    FeasibilityProblemCreated := 1;
endif;

! Update feasibility problem corresponding to Subproblem and solve it.
GMP::Benders::UpdateSubProblem( gmpF, gmpM, 1, round : 1 );

GMP::SolverSession::Execute( solsesF ) ;
GMP::Solution::RetrieveFromSolverSession( solsesF, 1 ) ;

! Check whether objective is 0 in which case optimality condition is satisfied.
ObjectiveFeasProblem := GMP::SolverSession::GetObjective( solsesF );

if ( ObjectiveFeasProblem <= BendersOptimalityTolerance ) then
    return 1;
endif;

! Add feasibility cut to the Master problem.
NumberOfFeasibilityCuts += 1;
GMP::SolverSession::AddBendersFeasibilityCut( ThisSession, gmpF, 1 );

return 1;
```

The procedure AlgorithmTerminateSingleMIP

The procedure AlgorithmTerminateSingleMIP is called to finish the Benders' decomposition algorithm. This procedure is called directly after the master MIP problem is solved. Its implementation is similar to that of the procedure AlgorithmTerminate of Section 21.4 and therefore omitted.

21.7 Implementation of the two phase algorithm

First phase

The Benders' module also implements a two phase algorithm for MIP problems. In the first phase it solves the relaxed problem in which the integer variables become continuous. The resulting relaxed MIP problem is then solved using the classic Benders' decomposition algorithm in order to find Benders' cuts.

Second phase

The second phase solves the original MIP problem. The master problem created in the first phase is also used in the second phase but without relaxing the integer variables. The Benders' cuts that were added during the first phase are not removed; these cuts are still valid. In general the relaxed MIP problem can be solved more efficiently than the MIP problem using Benders' decomposition, and the hope is that by adding the Benders' cuts found during the first phase, the Benders' decomposition algorithm needs considerably less iterations in the second phase to solve the original MIP problem.

Implementation of DoBendersDecompositionTwoPhase

The procedure DoBendersDecompositionTwoPhase implements the two phase algorithm. It starts by making copies of its first two input arguments. Next the master problem and the subproblem are created. The parameters for the number of optimality and feasibility cuts are reset. The problem type of the master problem is changed from 'MIP' to 'RMIP' which basically changes the integer variables into continuous variables. The procedure BendersAlgorithm then solves the relaxed problem using the classic Benders' decomposition algorithm; see Section 21.4 for its implementation. After checking the program status of the relaxed master problem the algorithm continues by switching the problem type of the master problem back to 'MIP'. Next the original problem is solved using either procedure BendersAlgorithmSingleMIP (Section 21.6) or BendersAlgorithm (Section 21.4). The algorithm ends by deleting the master problem and the subproblem. As before we leave out parts of the code that handle details like creating a status file, for the sake of brevity and clarity.

```
OriginalGMP := MyGMP ;
VariablesMasterProblem := MyMasterVariables ;

! Create (Relaxed) Master problem.
gmpM := GMP::Benders::CreateMasterProblem( OriginalGMP, VariablesMasterProblem,
                        'BendersMasterProblem',
                        feasibilityOnly : FeasibilityOnly,
                        addConstraints : AddTighteningConstraints ) ;

! Create Subproblem.
gmpS := GMP::Benders::CreateSubProblem( OriginalGMP, gmpM, 'BendersSubProblem',
```

```
                        useDual : UseDual,
                        normalizationType : NormalizationType );

solsesS := GMP::Instance::CreateSolverSession( gmpS ) ;

NumberOfOptimalityCuts  := 0;
NumberOfFeasibilityCuts := 0;

! Start the classic Benders' decomposition algorithm for the relaxed Master
! MIP problem.
GMP::Instance::SetMathematicalProgrammingType( gmpM, 'RMIP' );

IterationLimit := IterationLimitPhaseSingle;

BendersAlgorithm;

ProgramStatus := GMP::Solution::GetProgramStatus( OriginalGMP, 1 );

if ( ProgramStatus = 'Infeasible' or
     ProgramStatus = 'Unbounded'  ) then
    DoPhaseTwo := 0;
endif;

if ( DoPhaseTwo ) then
    ! Switch back math program type.
    GMP::Instance::SetMathematicalProgrammingType( gmpM, 'MIP' );

    if ( UseSingleMIP ) then
        ! Start the Single MIP Tree Benders' decomposition algorithm.
        BendersAlgorithmSingleMIP;
    else
        IterationLimit := IterationLimitPhaseTwo;

        BendersAlgorithm;
    endif;
endif;

GMP::Instance::Delete( gmpM );
GMP::Instance::Delete( gmpS );
```

Iteration and time limit

The section in the Benders' module for the two phase algorithm contains two extra control parameters for setting the iteration and time limit used by the classic Benders' decomposition algorithm in the second phase. These parameters are IterationLimitPhaseTwo and TimeLimitPhaseTwo respectively. The parameters IterationLimit and TimeLimit are used in the first phase. In some cases it might be a good strategy to limit the number of iterations (or the running time) during the first phase. The two phase algorithm will then still find a global optimal solution of the original problem as long as the second phase terminates normally. If the modern approach (with a single MIP tree) is used in the second phase then the general solver options iteration_limit and time_limit are used for the second phase.

Bibliography

[Be62] J.F. Benders, *Partitioning procedures for solving mixed-variables programming problems*, Numerische Mathematic **4** (1962), 238–252.

[Fi10] M. Fischetti, D. Salvagni, and A. Zanette, *A note on the selection of benders' cuts*, Mathematical Programming B **124** (2010), 175–182.

[Ma99] R.K. Martin, *Large scale linear and integer optimization: A unified approach*, Kluwer Academic Publishers, Norwell, 1999.

[Ne88] G.L. Nemhauser and L.A. Wolsey, *Integer and combinatorial optimization*, John Wiley & Sons, New York, 1988.