The OS Dip Solver: A Generic Block-Angluar Decomposition Implementation

Horand Gassmann, Jun Ma, Kipp Martin

September 19, 2010

Abstract

In this document we describe how to use the Decomposition in Integer Programming (Dip) package with the Optimization Services (OS) package. The code for this example is contained in the folder `ApplicationTemplates/osDip`.

1 Building and Testing the OS-Dip Example

Currently, the Decomposition in Integer Programming (**Dip**) package is not a dependency of the Optimization Services (**OS**) package – **Dip** is not included in the **OS** Externals file. In order to run the OS Dip solver it is necessary to download both the **OS** and **Dip** projects. Download order is irrelevant. In the discussion that follows we assume that for both **OS** and **Dip** the user has successfully completed a `configure`, `make`, and `make install`. We also assume that the user is working with the trunk version of both **OS** and **Dip.**

The OS Dip solver C++ code is contained in `TemplateApplication/osDip`. The `configure` will create a `Makefile` in the `TemplateApplication/osDip` folder. The `Makefile` must be edited to reflect the location of the **Dip** project. The `Makefile` contains the line

```
DIPPATH = /Users/kmartin/coin/dip-trunk/vpath-debug/
```

This setting assumes that there is a **lib** directory:

```
/Users/kmartin/coin/dip-trunk/vpath-debug/lib
```

with the **Dip** library that results from `make install` and an `include` directory

```
/Users/kmartin/coin/dip-trunk/vpath/include
```

with the **Dip** header files generated by `make install`. The user should adjust

```
/Users/kmartin/coin/dip-trunk/vpath/
```

to a path containing the **Dip** `lib` and `include` directories. After building the executable by executing the `make` command run the `osdip` application using the command:

```
./osdip --param osdip.parm
```

This should produce the following output.

```
FINISH SOLVE
Status= 0 BestLB= 16.00000   BestUB= 16.00000   Nodes= 1
SetupCPU= 0.01 SolveCPU= 0.10 TotalCPU= 0.11 SetupReal= 0.08
SetupReal= 0.12 TotalReal= 0.16
Optimal Solution
-------------------------
Quality = 16.00
0       1.00
1       1.00
12      1.00
13      1.00
14      1.00
15      1.00
17      1.00
```

If you see this output, life is good and things are working. If this doesn't work, I almost certainly did something stupid and forget to fix it. The file `osdip.parm` is a parameter file. The use of the parameter file is explained in Section 7.

2 The OS Dip Solver – Code Description

The OS Dip Solver uses **Dip** to implement a Dantzig-Wofe decomposition algorithm for block-angular integer programs.

2.1 General Philosophy

$$z_{IP} \quad = \quad \min\{c^{\top}x \,|\, A'x \geq b',\ A''x \geq b'',\ x \in \mathbb{Z}^n\} \tag{1}$$

$$\mathcal{P} \quad = \quad \mathrm{conv}(\{\mathrm{x} \in \mathbb{Z}^{\mathrm{n}} \,|\, \mathrm{A'x} \geq \mathrm{b'}\}) \tag{2}$$

$$z_{LP} \quad = \quad \min\{c^{\top}x \,|\, A'x \geq b',\ A''x \geq b'',\ x \in \mathbb{R}^n\} \tag{3}$$

$$z_{D} \quad = \quad \min\{c^{\top}x \,|\, A'x \geq b',\ x \in \mathcal{P},\ x \in \mathbb{R}^n\} \tag{4}$$

The implementation of the OS Dip solver provides a virtual class `OSDipBlockSolver` with a pure virtual function `solve()`. The user is expected to provide a class that inherits from `OSDipBlockSolver` and implements the method `solve()`. The `solve()` method should optimize a linear objective function over $\mathcal{P}$. More details are provided below. The implementation is such so that the user only has to provide a class with a solve method. The user does not have to edit or alter any of the OS Dip Solver code. By using polymorphic factories the actual solver details are hidden from the OS Solver. A default solver, `OSDipBlockCoinSolver`, is provided. This default solver takes no advantage of special structure and simply calls COIN-OR bf Cbc.

2.2 The Code

Key classes

OSDipBlockSolver: This is a virtual class with a pure virtual function:

```
void solve(double *cost, std::vector<IndexValuePair*> *solIndexValPair,
double *optVal)
```

OSDipBlockSolverFactory: This is also virtual class with a pure virtual function:

```
OSDipBlockSolver* create()
```

This class also has the static method

```
OSDipBlockSolver* createOSDipBlockSolver(const string &solverName)
```

and a map

```
std::map<std::string, OSDipBlockSolverFactory*> factories;
```

Factory: This class inherits from the class **OSDipBlockSolverFactory**. Every sover class that inherits from the **OSDipBlockSolver** class should have a **Factory** class memember and since this **Factory** class member inherits from the **OSDipBlockSolverFactory** class it should implement a `create()` method that creates an optject in the class inheriting from **OSDipBlockSolver**.

OSDipFactoryInitializer: This class initializes the static map

```
OSDipBlockSolverFactory::factories
```

in the **OSDipBlockSolverFactory** class.

OSDipApp: This class inherits from the **Dip** class **DecompApp**. In **OSDipApp** we implement methods for creating the core (coupling) constraints, i.e. the constraints $A''x \geq b''$. This is done by implementing the `createModels()` method. Regardless, of the problem none of the relaxed or block constraints in $A'x \geq b'$ are created. These are treated implicitly in the solver class that inherits from the class **OSDipBlockSolver.** This class also implemnts a method that defines the variables that appear only in the blocks (**createModelMasterOnlys2**), and a method for generating an initial master (the method **generateInitVars()**).

Since the constraints $A'x \geq b'$ are treated explicitly by the Dip solver the `solveRelaxed()` method must be implemented. In our implementation we have the **OSDipApp** class data member:

```
std::vector<OSDipBlockSolver* > m_osDipBlockSolver;
```

when the `solveRelaxed()` method is called for block `whichBlock` in turn we make the call

```
m_osDipBlockSolver[whichBlock]->solve(cost, &solIndexValPair, &varRedCost);
```

and the appropriated solver in class **OSDipBlockSolver** is called. Finally, the **OSDipApp** class also initiates the reading of the OS option and instance files. How these files are used is discussed in Section 6. Based on option input data this class also creates the appropriate solver object for each block, i.e. it populates the `m_osDipBlockSolver` vector.

OSDipInterface: This class is used as an interface between the **OSDipApp** class and classes in the **OS** library. This provides a number of get methods to provide information to **OSDipApp** such as the coefficients in the A'' matrix, objective function coefficients, number of blocks etc. The **OSDipInterface** class reads the input OSiL and OSoL files and creates in-memory data structures based on these files.

OSDipBlockCoinSolver: This class inherits from the **OSDipBlockSolver** class. It is meant to illustrate how to create a solver class. This class solves each block by calling **Cbc**. Use of this class provides a generic black angular decomposition algorithm.

There is also **OSDip_Main.cpp:** which has the `main()` routine and is the entry point for the executable. It first creates a new price-branch-and-cut decomposition algorithm and then an Alps solver for which the `solve()` method is called.

3 User Requirements

The **OSDipBlockCoinSolver** class provides a solve method for optimizing a linear objective function over P given a linear objective function. However, this takes no advantage of the special structure available in the blocks. Therefore, the user may wish to implement his or her own solver class. In this case the user is required to do the following:

1. implement a class that inherits from the **OSDipBlockSolver** class and implements the solve method,
2. implement a class **Factory** that inherits from the class **OSDipBlockSolverFactory** and implements the `create()` method,
3. edit the file **OSDipFactoryInitializer.h** and add a line:

```
OSDipBlockSolverFactory::factories["MyBlockSolver"] = new
MyBlockSolver::Factory;
```

4. alter the Makefile to include the new source code.

4 Simple Plant/Lockbox Location Example

The problem minimizing the sum of the cost of capital due to float and the cost of operating the lock boxes is the problem.
Parameters:

$m-$ number of customers to be assigned a lock box

$n-$ number of potential lock box sites

$c_{ij}-$ annual cost of capital associated with serving customer j from lock box i

f_i- annual fixed cost of operating a lock box at location i

Variables:

$x_{ij}-$ a binary variable which is equal to 1 if customer j is assigned to lock box i and 0 if not

y_i- a binary variable which is equal to 1 if the lock box at location i is opened and 0 if not

The integer linear program for the lock box location problem is

$$
(LB) \quad
\begin{aligned}
\min \quad & \sum_{i=1}^{n}\sum_{j=1}^{m} c_{ij}x_{ij} + \sum_{i=1}^{n} f_i y_i && && (5)\\
& x_{ij} - y_i \le 0, && i = 1,\ldots,n,\ j = 1,\ldots,m && (6)\\
\text{s.t.} \quad & \sum_{i=1}^{n} x_{ij} = 1, && j = 1,\ldots,m && (7)\\
& x_{ij},\ y_i \in \{0,1\}, && i = 1,\ldots,n,\ j = 1,\ldots,m. && (8)
\end{aligned}
$$

The objective (5) is to minimize the sum of the cost of capital plus the fixed cost of operating the lock boxes. Constraints (6) are forcing constraints and require that a lock box be open if a customer is served by that lock box. For now, we consider these the $A'x \ge b'$ constraints. The requirement that every customer be assigned a lock box is modeled by constraints (7). For now, we consider these the $A''x \ge b''$ constraints.

Location Example 1: A three plant, five customer model.

Table 1: Data for a 3 plant, 5 customer problem

		CUSTOMER					FIXED COSTS
		1	2	3	4	5	
PLANT	1	2	3	4	5	7	2
	2	4	3	1	2	6	3
	3	5	4	2	1	3	3

$$\begin{aligned}\min \quad & 2x_{11} + 3x_{12} + 4x_{13} + 5x_{14} + 7x_{15} + 2y_1 + \\ & 4x_{21} + 3x_{22} + x_{23} + 2x_{24} + 6x_{25} + 3y_2 + \\ & 5x_{31} + 4x_{32} + 2x_{33} + x_{34} + 3x_{35} + 3y_3\end{aligned}$$

$$\begin{aligned}&x_{11} \leq y_1 \leq 1 \\ &x_{12} \leq y_1 \leq 1 \\ &x_{13} \leq y_1 \leq 1 \\ &x_{14} \leq y_1 \leq 1 \\ &x_{15} \leq y_1 \leq 1 \\ &x_{21} \leq y_2 \leq 1 \\ &x_{22} \leq y_2 \leq 1 \\ &x_{23} \leq y_2 \leq 1 \qquad A'x \geq b' \text{ constraints} \\ &x_{24} \leq y_2 \leq 1 \\ &x_{25} \leq y_2 \leq 1 \\ &x_{31} \leq y_3 \leq 1 \\ &x_{32} \leq y_3 \leq 1 \\ &x_{33} \leq y_3 \leq 1 \\ &x_{33} \leq y_3 \leq 1 \\ &x_{33} \leq y_3 \leq 1 \\ &x_{ij}, y_i \geq 0, \ i = 1, \ldots, n, \ j = 1, \ldots, m.\end{aligned}$$

$$\begin{aligned}\text{s.t.} \quad & x_{11} + x_{21} + x_{31} = 1 \\ & x_{12} + x_{22} + x_{32} = 1 \\ & x_{13} + x_{23} + x_{33} = 1 \qquad A''x \geq b'' \text{ constraints} x \geq 0 \text{ constraints} \\ & x_{14} + x_{24} + x_{34} = 1 \\ & x_{15} + x_{25} + x_{35} = 1\end{aligned}$$

Location Example 2 (SPL2): A three plant, three customer model.

$$\begin{aligned}\min \quad & 2x_{11} + x_{12} + x_{13} + y_1 + \\ & x_{21} + 2x_{22} + x_{23} + y_2 + \\ & x_{31} + x_{32} + 1x_{33} + +y_3\end{aligned}$$

Table 2: Data for a three plant, three customer problem

		CUSTOMER 1	2	3	FIXED COSTS
	1	2	1	1	1
PLANT	2	1	2	1	1
	3	1	1	2	1

$$\text{s.t.} \quad \begin{aligned} x_{11} + x_{21} + x_{31} &= 1 \\ x_{12} + x_{22} + x_{32} &= 1 \qquad Ax \geq b \text{ constraints} \\ x_{13} + x_{23} + x_{33} &= 1 \end{aligned}$$

$$\begin{aligned} x_{11} &\leq y_1 \leq 1 \\ x_{12} &\leq y_1 \leq 1 \\ x_{13} &\leq y_1 \leq 1 \\ x_{21} &\leq y_2 \leq 1 \\ x_{22} &\leq y_2 \leq 1 \qquad Bx \geq b \text{ constraints} \\ x_{23} &\leq y_2 \leq 1 \\ x_{31} &\leq y_3 \leq 1 \\ x_{32} &\leq y_3 \leq 1 \\ x_{33} &\leq y_3 \leq 1 \\ x_{ij}, y_i &\geq 0, \ i = 1, \ldots, n, \ j = 1, \ldots, m. \end{aligned}$$

5 Generalized Assignment Problem Example

A problem that plays a prominent role in vehicle routing is the *generalized assignment problem.* The problem is to assign each of n tasks to m servers without exceeding the resource capacity of the servers.

Parameters:

$n-$ number of required tasks

$m-$ number of servers

$f_{ij}-$ cost of assigning task i to server j

b_j- units of resource available to server j

$a_{ij}-$ units of server j resource required to perform task i

Variables:

$x_{ij}-$ a binary variable which is equal to 1 if task i is assigned to server j and 0 if not

The integer linear program for the generalized assignment problem is

$$
(GAP) \qquad \begin{aligned}
\min \quad & \sum_{i=1}^{n}\sum_{j=1}^{m} f_{ij}x_{ij} && (9)\\
\text{s.t.} \quad & \sum_{j=1}^{m} x_{ij} = 1, \qquad i = 1,\ldots,n && (10)\\
& \sum_{i=1}^{n} a_{ij}x_{ij} \le b_j, \qquad j = 1,\ldots,m && (11)\\
& x_{ij} \in \{0,1\},\ i = 1,\ldots,n,\ j = 1,\ldots,m. && (12)
\end{aligned}
$$

The objective function (9) is to minimize the total assignment cost. Constraint (10) requires that each task is assigned a server. The requirement that the server capacity not be exceeded is given in (11).

The test problem

```
 min     2 X11 + 11 X12 + 7 X21 + 7 X22 + 20 X31 +
2 X32 + 5 X41 + 5 X42
s.t.
X11 + X12 =    1
X21 + X22 =    1
X31 + X32 =    1
X41 + X42 =    1

3 X11 + 6 X21 + 5 X31 + 7 X41 <=   13
2 X12 + 4 X22 + 10 X32 + 4 X42 <=   10
```

6 Defining the Problem Instance and Blocks

Here we describe how to use the OSoption stuff and OSInstance. We illustrate with a simple plant location problem. Refer back to the example in Table 4 for a three-plant, five-customer problem. We treat the fixed charge constraints as the block constraints, i.e. we treat constraint set 6 as the set $A'x \ge b'$ constraints. These constraints naturally break into a block for each plant, i.e. there is a block of constraints:

$$x_{ij} \le y_i \tag{13}$$

In order to use the OS Dip solver it is necessary to: 1) define the set of variables in each block and 2) define the set of constraints that constitute the core or coupling constraints. This information is communicated to the OS Dip solver using Optimization Services option Language (OSoL). The OSoL input file for the example in Table 4 appears in Figures 1 and 2. See lines 32-55. There is an `<other>` option with `name="variableBlockSet"` for each block. Each block then lists the variables in the block. For example, the first block consists of variable indexed by 0, 1, 2, 3, 4, and 15. These correspond to variables x_{11}, x_{12}, x_{13}, x_{13}, x_{14}, and y_1. Likewise the second block corresponds to the variable for the second plant and the third block corresponds to variables for the third plant.

It is also necessary to convey which constraints constitute the core constraints. This is done in lines 58-64. The core constraints are indexed by 0, 1, 2, 3, 4. These constitute the demand

```
<?xml version="1.0" encoding="UTF-8"?>
<osol>
   <general>
      <instanceName>spl1 -- setup constraints are the blocks</instanceName>
   </general>
   <optimization>
      <variables numberOfOtherVariableOptions="6">
      <other name="initialCol" solver="Dip" numberOfVar="6" value="0">
         <var idx="0" value="1"/>
         <var idx="1" value="1"/>
         <var idx="2" value="1"/>
         <var idx="3" value="1"/>
         <var idx="4" value="1"/>
         <var idx="15" value="1"/>
      </other>
         <other name="initialCol" solver="Dip" numberOfVar="6" value="1">
            <var idx="5" value="1"/>
            <var idx="6" value="1"/>
            <var idx="7" value="1"/>
            <var idx="8" value="1"/>
            <var idx="9" value="1"/>
            <var idx="16" value="1"/>
         </other>
         <other name="initialCol" solver="Dip" numberOfVar="6" value="2">
            <var idx="10" value="1"/>
            <var idx="11" value="1"/>
            <var idx="12" value="1"/>
            <var idx="13" value="1"/>
            <var idx="14" value="1"/>
            <var idx="17" value="1"/>
         </other>
         <other name="variableBlockSet" solver="Dip" numberOfVar="6" value="MySolver1">
            <var idx="0"/>
            <var idx="1"/>
            <var idx="2"/>
            <var idx="3"/>
            <var idx="4"/>
            <var idx="15"/>
         </other>
         <other name="variableBlockSet" solver="Dip" numberOfVar="6" value="MySolver2">
            <var idx="5"/>
            <var idx="6"/>
            <var idx="7"/>
            <var idx="8"/>
            <var idx="9"/>
            <var idx="16"/>
         </other>
```

Figure 1: A sample OSoL file – SPL1.osol

constraints given in Equation (7). In the Optimization Services instance Language (OSiL file) file that corresponds to this problem the demand constraints are input first.

```
            <other name="variableBlockSet" solver="Dip" numberOfVar="6" value="MySolver3">
               <var idx="10"/>
               <var idx="11"/>
               <var idx="12"/>
               <var idx="13"/>
               <var idx="14"/>
               <var idx="17"/>
            </other>
         </variables>
         <constraints numberOfOtherConstraintOptions="1">
            <other name="constraintSet" solver="Dip" numberOfCon="5" type="Core">
               <con idx="0"/>
               <con idx="1"/>
               <con idx="2"/>
               <con idx="3"/>
               <con idx="4"/>
            </other>
         </constraints>
      </optimization>
   </osol>
```

Figure 2: A sample OSoL file – SPL1.osol (Continued)

7 The Dip Parameter File

Look at the osdip.parm file. You can see by commenting and uncommenting you can run one of three problems that will also get downloaded.

sp1.osil – a simple plant location problem spl2.osil – a second simple plant location problem genAssign.osil – a generalize assignment problem

The osol files (the option files) determine behavior. For example, if you use

osolFiles/spl1-b.osol

then the assingment constraints are the block constraints. If you use

osolFiles/spl1.osol

then the setup forcing constraints are the block constraints. This new example also exhibits the problems I filed ticked on.

8 Implementing A Block Solver

Describe the Factory Code

9 Issues to Fix

- Enhance solveRelaxed to allow parallel processing of blocks. See ticket 30.

- Does not work when there are 0 integer variables. See ticket 31.
- Be able to set options in C++ code. See ticket 41.
- Problem with Alps bounds at node 0. See ticket 43
- Figure out how to use BranchEnforceInMaster or BranchEnforceInSubProb so I don't get the large bonds on the variables. See ticket 47.