Optimization Services 1.0 User's Manual

Robert Fourer, Jun Ma, Kipp Martin

July 16, 2007

Abstract

This is the User's Manual for the Optimization Services (OS) project. The objective of (OS) is to provide a set of standards for representing optimization instances, results, solver options, and communication between clients and solvers in a distributed environment using Web Services. This COIN-OR project provides source code for libraries and executable programs that implement OS standards. See the Optimization Services (OS) Home Site `www.optimizationservices.org` and the COIN-OR Trac page `projects.coin-or.org` for more information.

Contents

List of Figures

List of Tables

1 The Optimization Services (OS) Project

The objective of Optimization Services (OS) is to provide a set of standards for representing optimization instances, results, solver options, and communication between clients and solvers in a distributed environment using Web Services. This COIN-OR project provides source code for libraries and executable programs that implement OS standards. See the Optimization Services (OS) Home Site `www.optimizationservices.org` and the COIN-OR Trac page `projects.coin-or.org` for more information. The OS project provides the following:

1. A set of XML based standards for representing optimization instances (OSiL), optimization results (OSrL), and optimization solver options (OSoL). There are other standards, but these are the main ones. The schemas for these standards are described in Section 5.

2. A robust solver and modeling language interface (API) for linear and nonlinear optimization problems. Corresponding to the OSiL problem instance representation there is an in-memory object, `OSInstance`, along with a set of `get()`, `set()`, and `calculate()` methods for accessing and creating problem instances. This is a very general API for linear, integer, and nonlinear programs. Any modeling language that can produce OSiL can easily communicate with any solver that uses the OSInstance API. The `OSInstance` object is described in more detail in Section 6. The nonlinear part of the API is based on the COIN project `projects.coin-or.org/CppAD` by Brad Bell but is written in a very general manner and could be used with other algorithmic differentiation packages. More detail on algorithmic differentiation is provided in Section 7.

3. A command line executable `OSSolverService` for reading problem instances (OSiL format, nl format, MPS format) and calling a solver either locally or on a remote server. This is described in Section 8.

4. Utilities that convert AMPL nl files into the OSiL XML format and MPS files into the OSiL XML format. This is described in Section 4.3.

5. Standards that facilitate the communication between clients and optimization solvers using Web Services. In Section 4.1 we describe the `OSAgent` part of the OS library that is used to create Web Services SOAP packages with OSiL instances and contact a server for solution.

6. An executable program `amplClient` that is designed to work with the AMPL modeling language. The `ampClient` appears as a "solver" to AMPL and, based on options given in AMPL, contact solvers either remotely or locally to solve instances created in AMPL. This is described in Section 10.1.

7. Server software that works with Apache Tomcat and Apache Axis. This software uses Web Services technology and acts a middleware between the client that creates the instance and solver on the server that optimizes the instance and returns the result. This is illustrated in Section 9

2 Download and Installation

OS is released as open source code under the Common Public License (CPL). This project was created by Robert Fourer, Jun Ma, and Kipp Martin. The code has been written primarily by Jun Ma, Kipp Martin, Robert Fourer, and Huanyuan Sheng. Jun Ma and Kipp Martin are the COIN project leaders for OS. Below we describe different methods for obtaining the C++ source code and binaries.

2.1 Obtaining the Source Code Subversion Repository (SVN)

The C++ source code can be obtained using Subversion. Users with Unix operating systems will most likely have an svn client. For Windows users wishing to obtain and SVN client we recommend TortoiseSVN. See `tortoisesvn.tigris.org`.

The OS project page with a Wiki is available at `projects.coin-or.org\OS`. Execute the following steps to get the source code using SVN.
Step 1: Connect to a directory where you want the OS project to go. The following command will download the project into the directory COIN-OS

```
svn co https://projects.coin-or.org/svn/OS/stable/1.0 COIN-OS
```

Step 2: Connect to the distribution root directory.

```
cd COIN-OS
```

Step 3: Run the configure script that will generate the makefiles.

```
./configure
```

For more information and options on the `./configure` script see `https://projects.coin-or.org/BuildTools/wiki/user-configure#PreparingtheCompilation`.
Step 4: Run the make files.

```
make
```

Step 5: Run the unitTest.

```
make test
```

Depending upon which third party software you have installed, the result of running the unitTest should look something like:

```
HERE ARE THE UNIT TEST RESULTS:

Solved problem avion2.osil with Ipopt
Solved problem HS071.osil with Ipopt
Solved problem rosenbrockmod.osil with Ipopt
Solved problem parincQuadratic.osil with Ipopt
Solved problem parincLinear.osil with Ipopt
Solved problem callBack.osil with Ipopt
Solved problem callBackRowMajor.osil with Ipopt
Solved problem parincLinear.osil with Clp
Solved problem p0033.osil with Cbc
Solved problem rosenbrockmod.osil with Knitro
Solved problem callBackTest.osil with Knitro
Solved problem parincQuadratic.osil with Knitro
Solved problem HS071_NLP.osil with Knitro
Solved problem p0033.osil with SYMPHONY
Solved problem parincLinear.osil with DyLP
Solved problem volumeTest.osil with Vol
Solved problem p0033.osil with GLPK
Solved problem lindoapiaddins.osil with Lindo
Solved problem rosenbrockmod.osil with Lindo
Solved problem parincQuadratic.osil with Lindo
Solved problem wayneQuadratic.osil with Lindo
Test the MPS -> OSiL converter on parinc.mps usig Cbc
Test the AMPL nl -> OSiL converter on hs71.nl using LINDO
Test a problem written in b64 and then converted to OSInstance
Successful test of OSiL parser on problem parincLinear.osil
Successful test of OSrL parser on problem parincLinear.osrl
Successful test of prefix and postfix conversion routines on problem rosenbrockmod.osil
Successful test of all of the nonlinear operators on file testOperators.osil
Successful test of AD gradient and Hessian calculations on problem CppADTestLag.osil


CONGRATULATIONS! YOU PASSED THE UNIT TEST
```

If you do not see

```
CONGRATULATIONS! YOU PASSED THE UNIT TEST
```

then you have not passed the unitTest and hopefully some semi-inteligble error message was given.

Step 6: Install the libraries.In addition you will have the following directories.

```
make install
```

This will install all of the libraries in the `lib` directory under the distribution root. In particuar, the main OS library `libOS` along with the libraries of the other COIN-OR project that download with the OS project will get installed in the `lib` directory. In addtion the `make install` command will install four executable programs in the `bin` directory. One of these binaries is `OSSolverService` which is main OS project executable. This is described in Section 8. In addition `clp`, `cbc`, `cbc-generic`, and `symphony` get installed in the `bin` directory.

2.2 Obtaining the Source Code From a Tarball or Zip File

The OS source code can also be obtained from either a tarball or zip file. This may be preferred for users who are not managing other COIN-OR projects wish to only work with periodic release versions of the code. In order to obtain the code from a Tarball or Zip file do the following.

Step 1: In a browser go the link `http://www.coin-or.org/Tarballs/OS/`. Listed at this page are files in the format:

```
OS-release_number.tgz
OS-release_number.zip
```

Step 2: Click on either the `tgz` or `zip` file and download to the desired directory.

Step 3: Upack the files. For `tgz` do the following at the command line:

```
gunzip OS-release_number.tgz
tar -xvf OS-release_number.tar
```

Windows users should be able to double click on the file `OS-release_number.zip` and have the directory unpacked.

Step 4: Rename `OS-release_number` to `COIN-OS`. Next follow Steps 2 - 6 outlined in Section 2.1.

2.3 Obtaining the Binaries

If the user does not wish to compile source code, the OS library, OSSolverService executable, and Tomcat server software configuration are available at `http://www.coin-or.org/Binaries/OS` in binary format.

kipp – discuss this more with with Jun

2.4 Obtaining and Installing a Visual Studio Project

In this section, we assume that the user has installed Microsoft Visual Studio on their computer.

Jun – fill in more.

2.5 Third Party Software

The default OS project is configured out-of-the-box with the COIN-OR projects `Cbc`, `Clp`, `Cgl`, `CoinUtils`, `CppAD`, `DyLP`, `SYMPHONY`, and `Vol`. However, the project is also designed to work with other COIN-OR projects and several other open source and commercial software projects.

In many of the header files there are `#include` statements inside `#ifdef` statements. For example,

```
#ifdef COIN_HAS_LINDO
#include "LindoSolver.h"
#endif
#ifdef COIN_HAS_IPOPT
#include "IpoptSolver.h"
#endif
```

In the `inc` subdirectory of the `OS` directory, there is a header file, `config_os.h` that defines the values of the

```
COIN_HAS_*****
```

variables. If the project is configured with the simple `./configure` command given in Step 3 with no arguments, then in the `config_os.h` these variables associated with the third-party software will be undefined. For example.

```
/* Define to 1 if the Cplex package is used */
/* #undef COIN_HAS_CPX */
```

unlike the configured COIN-OR projects that appear as

```
/* Define to 1 if the Clp package is used */
#define COIN_HAS_CLP 1
```

In the following subsections we describe how to incorporate various third-party packages into the OS project and see to it that the

```
COIN_HAS_*****
```

variable is defined in `config_os.h`.

2.5.1 AMPL

The OS library contains a class, `OSnl2osil` (see Section (4.3.2)) and `amplClient` (see Section (10.1)) that require the use of the AMPL ASL library. See `http://netlib.sandia.gov/ampl/` and . See Users with a Unix system should locate the `ASL` folder that is part of the distribution. The `ASL` folder is in the `ThirdParty` folder which is in the project root folder. Locate and execute the `get.ASL` script. Do this prior to running the `configure` script. The `configure` script will build the correct ASL library.

Microsoft Visual Studio users will have to build the ASL library separately and then link it with the OS lib in the OS project file. The necessary source files are at `http://netlib.sandia.gov/cgi-bin/netlib/netlibfiles.tar?filename=netlib/ampl/solvers`. After unpacking the distribution build the source code with the utility `nmake` which should be part of the Visual Studio distribution. The appropriate command is

```
nmake -f makefile.vc" .
```

If the OS project is properly configured with the ASL library, `config_os.h` will contain the lines

```
/* If defined, the Ampl Solver Library is available. */
#define COIN_HAS_ASL 1
```

At this point the reader may wish to view `https://projects.coin-or.org/BuildTools/wiki/user-configure#CommandLineArgumentsforconfigure` for more information on command line arguments that illustrated in the subsections below.

2.5.2 Cplex

Cplex is a linear, integer, and quadratic solver. See `http://www.ilog.com/products/cplex/?CFID=4534586&CFTOKEN=61400951`. Cplex does provide source code and you are can only download the platform dependent binaries. After installing the binaries and include files in an appropriate run configure to point to the include and library directory. An example is given below.

```
configure --with-cplex-lib="-L$(CPLEXDIR)/lib/$(SYSTEM)/$(LIBFORMAT)
-lcplex -lilocplex -lm -lpthread" --with-cplex-incdir= $(CPLEXDIR)/include
```

You may also need the following environment variables.

```
SYSTEM     =i86_linux2_glibc2.3_gcc3.2
LIBFORMAT  =static_pic
CPLEXDIR      =/usr/local/ilog/cplex90
OBJECTCPLEX=CplexContinuousRun.o
CPLEXLIBPATH= -L$(CPLEXDIR)/lib/$(SYSTEM)/$(LIBFORMAT)
CPLEXINCDIR = $(CPLEXDIR)/include
CPLEX_LIBS=-lcplex -lilocplex -lm -lpthread
```

2.5.3 GLPK

`GLPK` is a an open-source linear and integer-programming solver from the GNU organization. See `http://www.gnu.org/software/glpk/`. In order to use GLPK with OS first go to any GNU mirror site (`http://www.gnu.org/prep/ftp.html`) and download the GLPK software in the directory `/gnu/glpk/`. Follow the instructions in the `INSTALL` file. After installing GLPK, run the configure script with the path to the GLPK library and include directory. An example is given below.

```
./config --with-glpk-lib="-L/home/kmartin/files/code/glpk/linux/lib -lglpk "
--with-glpk-incdir=/home/kmartin/files/code/glpk/linux/include
```

2.5.4 Ipopt

Ipopt is a COIN-OR project. In order to use Ipopt with OS, follow the instructions at `projects.coin-or.org\Ipopt` and build the project. Then when running configure point to the include directory and library directory. An example is given below.

```
./configure --with-ipopt-lib="-L/home/kmartin/files/code/ipopt/linux/Ipopt-3.2.2_clean/lib -lipopt
‘cat /home/kmartin/files/code/ipopt/linux/Ipopt-3.2.2_clean/lib/ipopt_addlibs_cpp.txt‘"
--with-ipopt-incdir=/home/kmartin/files/code/ipopt/linux/Ipopt-3.2.2_clean/include/ipopt
```

2.5.5 Knitro

Knitro is a nonlinear solver. See `http://www.ziena.com/`. Knitro does provide source code and you are can only download the platform dependent binaries. After installing the binaries and include files in an appropriate run configure to point to the include and library directory. An example is given below.

```
./configure --with-knitro-lib="-L/home/kmartin/files/code/knitro/linux/lib -lknitro "
--with-knitro-incdir=/home/kmartin/files/code/knitro/linux/include
```

2.5.6 LINDO

LINDO is a commercial linear, integer, and nonlinear solver. See `www.lindo.com`. LINDO does provide source code and you are can only download the platform dependent binaries. After installing the binaries and include files in an appropriate run configure to point to the include and library directory. An example is given below.

```
--with-lindo-lib="-L/home/kmartin/files/code/lindo/linux/lib -llindo -lmosek"
--with-lindo-incdir=/home/kmartin/files/code/lindo/linux/include
```

2.5.7 MATLAB

Install MATLAB on the client machine and follow the instruction in Section 4.3.3.

2.5.8 Library Paths

After running `configure` as described above, on Unix systems, it will be necessary to set the `LD_LIBRARY_PATH` or `DYLD_LIBRARY_PATH` (on Mac OS X) to environment variables to point to the location of the installed third party libraries in the case that the libraries are dynamic and not static libraries.

2.6 Bug Reporting

Bug reporting is done through the project Trac page. This is at `http://projects.coin-or.org/OS`. To report a bug, you must be a registered user. For instructions on how to register go to `http://www.coin-or.org/usingTrac.html` After registering, log in and then file a trouble ticket by going to `http://projects.coin-or.org/OS/newticket`.

2.7 Documentation

If you have Doxygen www.doxygen.org available (the executable doxygen should be in the path command) then executing

```
make  doxydoc
```

in the project root directory will result in the Doxygen documentation being generated and stored in the doxydoc folder in the project root.

In order to view the documentation, open a browser and open the file

```
projectroot/doxydoc/html/index.html
```

Running Doxygen will generate documentation for only the OS project. Documentation will not be generated for the other COIN-OR projects in the project root. In the doxydoc folder is a configuration file doxygen.conf. This configuration for file contains for the EXCLUDE parameter

```
EXCLUDE =  Cbc \
   Cgl \
   Clp \
   CoinUtils \
   cppad \
   SYMPHONY \
   Vol \
   DyLP \
   ThirdParty \
   Osi \
   include
```

This file can be edited, and any project for which documentation is desired, can be deleted from the EXCLUDE list.

2.8 Obtaining the Server Software

Kipp – discuss with Jun

2.9 Platforms

The build process described in Section 2.1 has been tested on Linux, Mac OS X, and on Windows using MINGW/MSYS and CYGWIN. The gcc/g++ and Microsoft cl compiler have been tested. A number of solvers have also been tested with the OS library. For a list of tested solvers and platforms see Table 1. More detail on the platforms listed in Table 1 is given in Table 2.

3 The OS Project Components

The directories in the project root are outlined in Figure 1.

If you download the OS package, you get these additional COIN-OR projects. The links to the project home pages are provided below and give more information on these projects.

- BuildTools projects.coin-or.org\BuildTools
- Cbc projects.coin-or.org\Cbc

Table 1: Tested Platforms for Solvers

	Mac	Linux	Cyg-gcc	Msys-cl	Msys-gcc	MSVS
AMPL-Client	x	x		x		
MATLAB	x					
Cbc	x	x	x	x		
Clp	x	x	x	x		
Cplex	x	x				
DyLP	x	x	x	x		
Ipopt	x	x				
Knitro	x	x				
Lindo	x	x		x		
SYMPHONY	x	x	x	x		
Vol	x	x	x	x		

Table 2: Platform Description

	Operating System	**Compiler**	**Hardware**
Mac	Mac OS X 10.4.9	gcc 4.0.1	Power PC
Linux	Red Hat 3.4.6-8	gcc 3.4.6	Dell Intel 32 bit chip
Cyg-gcc	Windows 2003 Server	gcc 3.4.4	Dell Intel 32 bit chip
Msys-cl	Windows XP	Visual Studio 2003	Dell Intel 32 bit chip
Msys-gcc			
MSVS	Windows XP	Visual Studio 2003	Dell Intel 32 bit chip

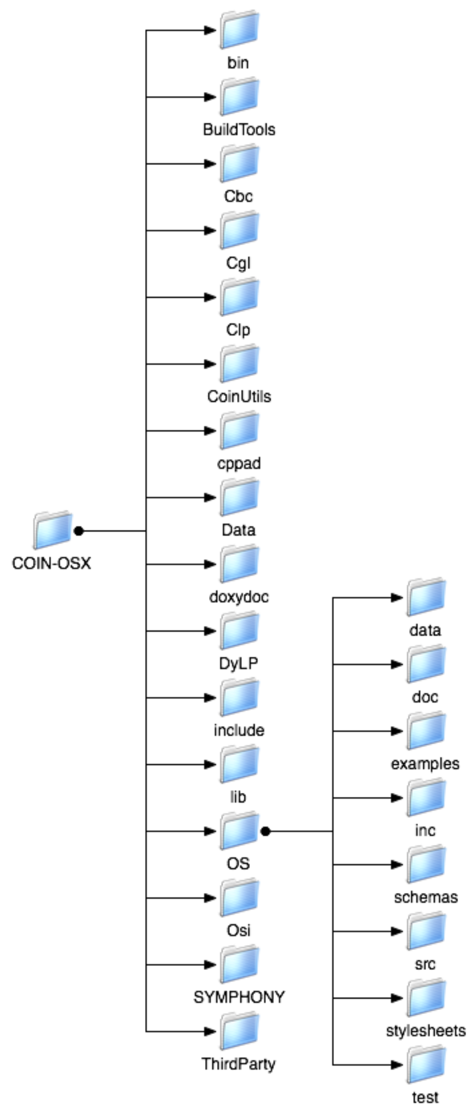

Figure 1: The OS project root directory.

- Cgl `projects.coin-or.org\Cgl`
- Clp `projects.coin-or.org\Clp`
- CppAD `projects.coin-or.org\CppAD`
- Dylp `projects.coin-or.org\Dylp`
- Osi `projects.coin-or.org\Osi`
- SYMPHONY `projects.coin-or.org\SYMPHONY`
- Vol `projects.coin-or.org\Vol`

The following directories are also in the project root.

- `bin` after executing `make install` the bin directory will contain `OSSolverService`, `clp`, `cbc`, `cbc-generic` and symphony.
- `Data` this directory contains numerous test problems that are used by some of the COIN-OR project's unitTest.
- `doxydoc` is a folder for documentation
- `include` is a directory for header files. If the user wishes to write code to link against any of the libraries in the `lib` directory, it may be necessary to include these header files.
- `lib` is a directory of libraries. After running `make install` the OS library along with all other COIN-OR libraries are installed in `lib`.
- `ThirdParty` is a directory for third party software. For example, if AMPL related software is used such as `amplClient` is used, then certain AMPL libraries need to be present. This should go into the `ASL` directory in `ThirdParty.`

The directories in the OS directory are outlined in Figure 2.
The OS directories include the following:

- `data` is a directory that holds test problems. These test problems are used by the `unitTest`. Many of these files are also used to illustrate how the `OSSovlerService` works. See Section 8.
- `doc` is the directory with documentation, include this *OS User's Manual.*
- `examples` is a directory with code examples that illustrate various aspects of the OS project. These are described in Section 10.
- `inc` is the directory with the config˙os.h file which has information about which projects are included in the distribution.
- `schemas` is the directory that contains the W3C XSD (see `www.w3c.org`) schemas that are behind the OS standards. These are described in more detail in Section 5.
- `src` is the directory with all of the source code for the OS Library and for the executable `OSSolverService`. The OS Library components are described in Section 4.

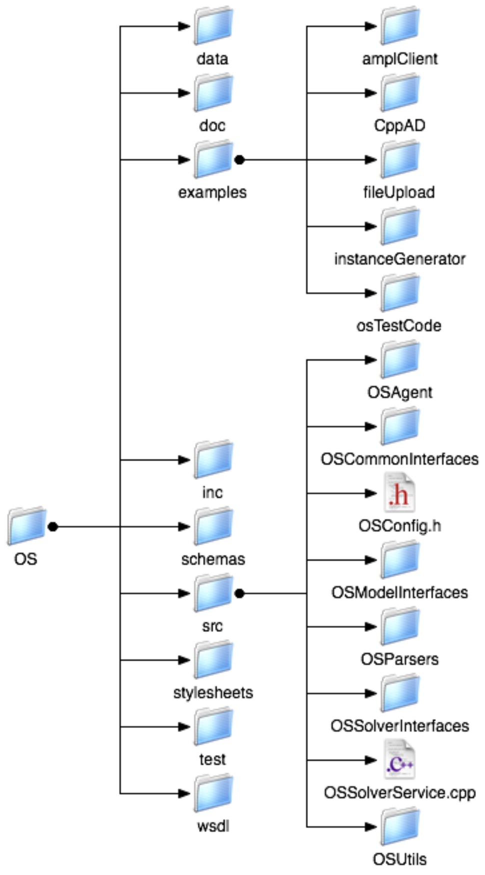

Figure 2: The OS directory.

- `stylesheets` this directory contains the XSLT stylesheet that is used to transform the solution instance in OSrL format into HTML so that it can be displayed in a browser.
- `test` this directory contains the `unitTest`.
- `wsdl` is a directory of WSDL (Web Services Discovery Language) files. These are used to specify the inputs and outputs for the methods provided by a Web service. The most relevant file for the current version of the OS project is `OShL.wsdl`. This describes the set of inputs and outputs for the methods implemented in the `OSSolverService`. See Section 8.

4 The OS Library Components

4.1 OSAgent

The `OSAgent` part of the library is used to facilitate communication with remote solvers. It is not used if the solver is invoked locally (i.e. on the same machine). There are two key classes in the `OSAgent` component of the OS library. The two classes are `OSSolverAgent` and `WSUtil`.

The `OSSolverAgent` class is used contact a remote solver service. For example, assume that `sOSiL` is a string with a problem instance and `sOSoL` is a string with solver options. Then the following code will call a solver service and invoke the `the solve` method.

```
OSSolverAgent *osagent;
string serviceLocation = http://128.135.130.17:8080/os/OSSolverService.jws
osagent = new OSSolverAgent(  serviceLocation );
osagent->solve(sOSiL, sOSoL);
```

Other methods in the `OSSolverAgent` class are `send`, `retrieve`, `getJobID`, `knock`, and `kill`. The use of these methods is described in Section 8.3.

The methods in the `OSSolverAgent` class call methods in the `WSUtil` class that perform such tasks and creating and parsing SOAP messages and making low level socket calls to the server running the solver service. The average user will not use methods in the `WSUtil` class, but they are available to anyone wanting to make socket calls or create SOAP messages.

There is also a method, `fileUpload` in the OSAgentClass that is used to upload files from the hard drive of a client to the server. It is very fast and does not involve SOAP or Web Services. The `fileUpload` method is illustrated and described in the example code `fileUpload.cpp` described in Section 10.3.

4.2 OSCommonInterfaces

The classes in the OSCommonInterfaces component of the OS library are used to read and write files and strings in the OSiL and OSrL protocols. See Section 5 for more detail on OSiL, OSrL, and other OS protocols. For a complete listing of all of the files in `OSCommonInterfaces` see the Doxygen documentation in the `doxydoc` folder (see Section 5). Below we highlight some key classes.

4.2.1 The OSInstance Class

The OSInstance class is the in-memory representation of an optimization instance and is a key class for users of the OS project. This class has an API defined by a collection of `get()` methods for extracting various components (such as bounds and coefficients) from a problem instance, a collection of `set()` methods for modifying or generating an optimization instance, and a collection of `calculate()` methods for function, gradient, and Hessian evaluations. See Section 6. We now describe how to create an `OSInstance` object and the close relationship between the OSiL schema and the `OSInstance` class.

4.2.2 Creating an `OSInstance` Object

The OSCommonInterfaces component contains an `OSiLReader` class for reading an instance in an OSiL string and creating an in-memory `OSInstance` object. Assume that `sOSiL` is a string with an instance in OSiL format. Creating an `OSInstance` object is illustrated in Figure 3.

```
OSiLReader *osilreader = NULL;
OSInstance *osinstance = NULL;
osilreader = new OSiLReader();
osinstance = osilreader->readOSiL( sOSiL);
```

Figure 3: Creating an `OSInstance` Object

4.2.3 Mapping Rules

The `OSInstance` class has two member classes, `InstanceHeader` and `InstanceData`. These correspond to the OSiL schema's complexTypes `instanceHeader` and `instanceData`, and to the XML elements `<instanceHeader>` and `<instanceData>`.

Moving down one level, Figure 5 shows that the `InstanceData` class has in turn the member classes `Variables`, `Objectives`, `Constraints`, `LinearConstraintCoefficients`, `QuadraticCoefficients`, and `NonlinearExpressions`, corresponding to the respective elements in the OSiL schema with the same name.

```
class OSInstance{
public:
     OSInstance();
     InstanceHeader *instanceHeader;
     InstanceData *instanceData;
}; //class OSInstance
```

Figure 4: The `OSInstance` class

Figure 6 uses the `Variables` class to provide a closer look at the correspondence between schema and class. On the right, the `Variables` class contains a `number` data member and a sequence of `var` objects of class `Variable`. The `Variable` class has `lb` (double), `ub` (double), `name` (string), `init` (double), and `type` (char) data members. On the left the corresponding XML complexTypes are shown, with arrows indicating the correspondences. The following rules describe the mapping between the OSiL schema and the `OSInstance` class.

- ▷ Each complexType in an OSiL schema corresponds to a class in `OSInstance.` Thus the OSiL schema's complexType `Variable` corresponds to `OSInstance`'s class `Variable`. Elements in

```
class InstanceData{
public:
     InstanceData();
     Variables *variables;
     Objectives *objectives;
     Constraints *constraints;
     LinearConstraintCoefficients *linearConstraintCoefficients;
     QuadraticCoefficients *quadraticCoefficients;
     NonlinearExpressions *nonlinearExpressions;
}; // class InstanceData
```

Figure 5: The `InstanceData` class

an actual XML file then correspond to objects in `OSInstance`; for example, the `<var>` element that is of type `Variable` in an OSiL file corresponds to a `var` object in class `Variable` of `OSInstance`.

▷ An attribute or element used in the definition of a `complexType` is a member of the corresponding `OSInstance` class, and the type of the attribute or element matches the type of the member. In Figure 6, for example, `lb` is an attribute of the OSiL `complexType` named `Variable`, and `lb` is a member of the `OSInstance` class `Variable`; both have type `double`. Similarly, `var` is an element in the definition of the OSiL `complexType` named `Variables`, and `var` is a member of the `OSInstance` class `Variables`; the `var` element has type `Variable` and the `var` member is a `Variable` object.

▷ A schema sequence corresponds to an array. For example, in Figure 6 the complexType `Variables` has a sequence of `<var>` elements that are of type `Variable`, and the corresponding `Variables` class has a member that is an array of type `Variable`.

General nonlinear terms are stored in the data structure as `OSExpressionTree` objects, which are the subject of the next section.

The `OSInstance` class has a set of `get()` , `set()`, and `calculate()` methods that act as an API for the optimization instance and described in Section 6.

4.2.4 The OSExpressionTree OSnLNode Classes

The `OSExpressionTree` class provides the in-memory representation of the nonlinear terms. Our design goal is to allow for efficient parsing of OSiL instances, while providing an API that meets the needs of diverse solvers. Conceptually, any nonlinear expression in the objective or constraints is represented by a tree. The expression tree for the nonlinear part of the objective function (7), for example, has the form illustrated in Figure 7. The choice of a data structure to store such a tree — along with the associated methods of an API — is a key aspect in the design of the `OSInstance` class.

A base abstract class `OSnLNode` is defined and all of an OSiL file's operator and operand elements used in defining a nonlinear expression are extensions of the base element type `OSnLNode`. There is an element type `OSnLNodePlus`, for example, that extends `OSnLNode`; then in an OSiL instance file, there are `<plus>` elements that are of type `OSnLNodePlus`. Each `OSExpressionTree` object contains a pointer to an `OSnLNode` object that is the root of the corresponding expression tree. To every element that extends the `OSnLNode` type in an OSiL instance file, there corresponds a class that derives from the `OSnLNode` class in an `OSInstance` data structure. Thus we can construct an expression tree of homogenous nodes, and methods that operate on the expression tree to calculate function values, derivatives, postfix notation, and the like do not require switches or complicated logic.

The `OSInstance` class has a variety of `calculate()` methods, based on two pure virtual functions in the `OSInstance` class. The first of these, `calculateFunction()`, takes an array of `double` values corresponding to decision variables, and evaluates the expression tree for those values. Every class that extends `OSnLNode` must implement this method. As an example, the `calculateFunction` method for the `OSnLNodePlus` class is shown in Figure 8. Because the OSiL instance file must be validated against its schema, and in the schema each `<OSnLNodePlus>` element is specified to have exactly two child elements, this `calculateFunction` method can assume that there are exactly

Schema complexType **In-memory class**

```
<xs:complexType name="Variables">  <----------------------------------------------->  class Variables{
                                                                                       public:
  <xs:sequence>                                                                          Variables();
    <xs:element name="var" type="Variable" maxOccurs="unbounded"/>  <------------->      Variable *var;
  </xs:sequence>
  <xs:attribute name="number" type="xs:positiveInteger"  use="required"/>  <------>      int number;
</xs:complexType>                                                                      }; // class Variables


<xs:complexType name="Variable">  <------------------------------------------------>  class Variable{
                                                                                       public:
                                                                                         Variable();
  <xs:attribute name="name" type="xs:string" use="optional"/>  <------------------>      string name;
  <xs:attribute name="init" type="xs:double" use="optional"/>  <------------------>      double init;
  <xs:attribute name="initString" type="xs:string" use="optional"/>  <------------>      string initString;
  <xs:attribute name="type" use="optional" default="C">  <------------------------>      char type;
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="C"/>
        <xs:enumeration value="B"/>
        <xs:enumeration value="I"/>
        <xs:enumeration value="S"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
  <xs:attribute name="lb" type="xs:double" use="optional" default="0"/>  <-------->      double lb;
  <xs:attribute name="ub" type="xs:double" use="optional" default="INF"/>  <------>      double ub;
</xs:complexType>                                                                      }; // class Variable
```

OSiL elements **In-memory objects**

```
<variables number="2">                             OSInstance osinstance;
   <var lb="0" name="x0" type="C"/>                osinstance.instanceData.variables.number=2;
   <var lb="0" name="x1" type="C"/>                osinstance.instanceData.variables.var=new Var[2];
</variables>                                       osinstance.instanceData.variables.var[0].lb=0;
                                                   osinstance.instanceData.variables.var[0].name=x0;
                                                   osinstance.instanceData.variables.var[0].type=C;
                                                   osinstance.instanceData.variables.var[1].lb=0;
                                                   osinstance.instanceData.variables.var[1].name=x1;
                                                   osinstance.instanceData.variables.var[1].type=C;
```

Figure 6: The `<variables>` element as an `OSInstance` object

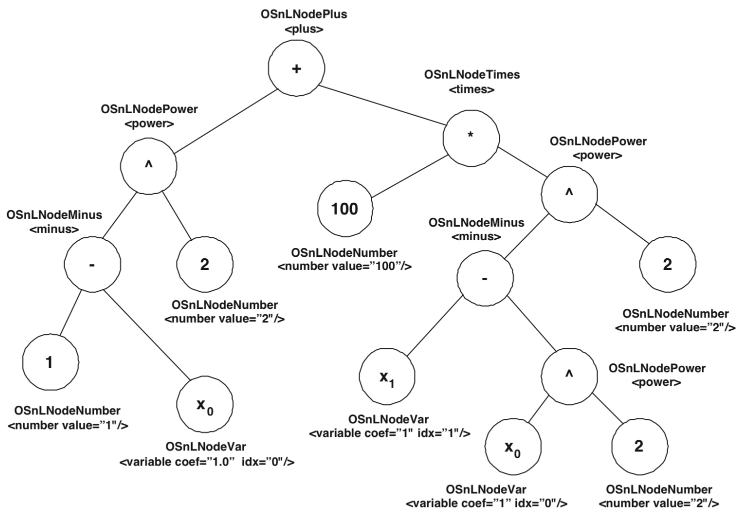

Figure 7: Conceptual expression tree for the nonlinear part of the objective (7).

```
double OSnLNodePlus::calculateFunction(double *x){
   m_dFunctionValue =
      m_mChildren[0]->calculateFunction(x) +
      m_mChildren[1]->calculateFunction(x);
   return m_dFunctionValue;
} //calculateFunction
```

Figure 8: The function calculation method for the "plus" node class with polymorphism

two children of the node that it is operating on. Thus through the use of polymorphism and recursion the need for switches like those in Figure **??** is eliminated. This design makes adding new operator elements easy; it is simply a matter of adding a new class and implementing the `calculateFunction()` method for it.

The following `OSnLNode` classes are currently supported.

- OSnLNodeVariable
- OSnLNodeTimes
- OSnLNodePlus
- OSnLNodeSum
- OSnLNodeMinus
- OSnLNodeNegate
- OSnLNodeDivide
- OSnLNodePower
- OSnLNodeProduct
- OSnLNodeLn

- OSnLNodeSqrt
- OSnLNodeSquare
- OSnLNodeSin
- OSnLNodeCos
- OSnLNodeExp
- OSnLNodeif
- OSnLNodeAbs
- OSnLNodeMax
- OSnLNodeMin
- OSnLNodeE
- OSnLNodePI
- OSnLNodeAllDiff

4.3 OSModelInterfaces

This part of the OS library is designed to help integrate the OS standards with other standards and modeling systems.

4.3.1 Converting MPS Files

The MPS standard is still a popular format for representing linear and integer programming problems. In `OSModelInterfaces,` there is a class `OSmps2osil` that can be used to convert files in MPS format into the OSiL standard. It is used as follows.

```
OSmps2osil *mps2osil = NULL;
DefaultSolver *solver  = NULL;
solver = new CoinSolver();
solver->sSolverName = "cbc";
mps2osil = new OSmps2osil(  mpsFileName);
mps2osil->createOSInstance() ;
solver->osinstance = mps2osil->osinstance;
solver->solve();
```

The `OSmps2osil` class constructor takes a string which should be the file name of the instance in MPS format. The constructor then uses the CoinUtils library to read and parse the MPS file. The class method `createOSInstance` then builds an in-memory `osintance` object that can be used by a solver.

4.3.2 Converting AMPL nl Files

AMPL is a popular modeling language that saves model instances in the AMPL nl format. The `OSModelInterfaces` library provides a class, `OSnl2osil` for reading in an nl file and creating a corresponding in-memory osinstance object. It is used as follows.

```
OSnl2osil *nl2osil = NULL;
DefaultSolver *solver  = NULL;
solver = new LindoSolver();
nl2osil = new OSnl2osil( nlFileName);
nl2osil->createOSInstance() ;
solver->osinstance = nl2osil->osinstance;
solver->solve();
```

The `OSnl2osil` class works much like the `OSmps2osil` class. The `OSnl2osil` class constructor takes a string which should be the file name of the instance in nl format. The constructor then uses the AMPL ASL library routines to read and parse the nl file. The class method `createOSInstance` then builds an in-memory `osintance` object that can be used by a solver.

In Section 10.1 we describe the `amplClient` executable that acts a "solver" for AMPL. The `amplClient` uses the `OSnl2osil` class to convert the instance in nl format to OSiL format before calling a solver either locally or remotely.

4.3.3 Using MATLAB

Linear, integer, and quadratic problems can be formulated in MATLAB and then optimized either locally or over the network using the OS Library. The `OSMatlab` class functions much like `OSnl2osil` and `OSmps2osil` and takes MATLAB arrays and creates and OSiL instance. This class is part of the OS library. In order to use the OS library with MATLAB the user should do the following. In order to use the `OSMatlab` class it is necessary to compile `matlabSolver.cpp` into a MATLAB Executable file. The `matlabSolver.cpp` file is in the `OSModelInterfaces` directory even though it is not part of the OS library. The following steps should be followed.

Step 1: In the project root run `make install.`

Step 2: Either leave `matlabSolver.cpp` in the the `OSModelInterfaces` or copy it to another desired directory.

Step 3: Edit the MATLAB **mexopts.sh** (UNIX) or `mexopts.bat` so that the `CXXFLAGS` option includes the header files in the `cppad` directory and the `include` directory in the project root. For example, it should look like:

```
CXXFLAGS='-fno-common -no-cpp-precomp -fexceptions
    -I/Users/kmartin/Documents/files/code/cpp/OScpp/COIN-OSX/
    -I/Users/kmartin/Documents/files/code/cpp/OScpp/COIN-OSX/include'
```

Next edit the `CXXLIBS` flag so that the OS and supporting libraries are included. For example, it should look like:

```
CXXLIBS="$MLIBS -lstdc++
    -L/Users/kmartin/Documents/files/code/ipopt/macosx/Ipopt-3.2.2/lib
    -L/Users/kmartin/Documents/files/code/cpp/OScpp/COIN-OSX/lib
    -lOS  -lIpopt -lOsiCbc -lOsiClp -lCbc -lCgl -lOsi -lClp -lCoinUtils  -lm"
```

For a UNIX system the `mexopts.sh` file will usually be found in a directory with the release name in `~/.matlab`. For example, `~/.matlab/R14SP3`.

On a Windows system, the `mexopts.bat` file will usually be in a directory with the release name in `C:\Documents and Settings\Username\Application Data\Mathworks\MATLAB`

Step 4: Build the MATLAB executable file. Start MATLAB and in the MATLAB command window connect to the directory containing the file `matlabSolver.cpp`. Execute the command:

```
mex -v matlabSolver.cpp
```

On a MAC OS X the resulting executable will be named `matlabSolver.mexmac`. On the Windows system the file we named `matlabSolver.mexw32`.

Step 5: Set the MATLAB path to include the directory with the `matlabSolver` executable. Also, put the $m-file$ `callMatlabSolver.m` in a directory which is on a MATLAB path. The `callMatlabSolver.m` m-file is in the OSModelInterfaces directory.

To use the `matlabSolver` it is necessary to put the coefficients from a linear, integer, or quadratic problem into MATLAB arrays.

$$\begin{aligned} \text{Minimize} \quad & 10x_1 + 9x_2 && (1)\\ \text{Subject to} \quad & .7x_1 + x_2 \leq 630 && (2)\\ & .5x_1 + (5/6)x_2 \leq 600 && (3)\\ & x_1 + (2/3)x_2 \leq 708 && (4)\\ & .1x_1 + .25x_2 \leq 135 && (5)\\ & x_1, x_2 \geq 0 && (6) \end{aligned}$$

The MATLAB representation of this problem in MATLAB arrays is

```
% the number of constraints
numCon = 4;
% the number of variables
numVar = 2;
% variable types
VarType='CC';
% constraint types
A = [.7  1; .5  5/6; 1   2/3  ; .1   .25];
BU = [630 600  708  135];
BL = [];
OBJ = [10  9];
VL = [-inf -inf];
VU = [];
ObjType = 1;
% leave Q empty if there are no quadratic terms
Q = [];
prob_name = 'ParInc Example'
password = 'chicagoesmuyFRIO';
```

```
%
%
%the solver
solverName = 'lindo';
%the remote service service address
%if left empty we solve locally
serviceAddress='http://128.135.130.17:8080/os/OSSolverService.jws';
% now solve
callMatlabSolver( numVar, numCon, A, BL, BU, OBJ, VL, VU, ObjType, ...
    VarType, Q, prob_name, password, solverName, serviceAddress)
```

This example m-file is in the `data` directory and is file `parincLinear.m`. Note that in addition to the problem formulation we can specify which solver to use through the `solverName` variable. If solution with a remote solver is desired this can be specified with the `serviceAddress` variable. If the `serviceAddress` is left empty, i.e.

```
serviceAddress='';
```

then a local solver is used. In this case it is crucial that the appropriate solver is linked in with the `matlabSolver` executable using the `CXXLIBS` option.

The data directory also contains the m-file `template.m` which contains extensive comments about how to formulate the problems in MATLAB. A second example which is a quadratic problem is given in the Appendix. The appropriate m-file is `markowitz.m`.

4.4 OSParsers

Mention reentrant. Mention that we still need OSoL.

4.5 OSSolverInterfaces

The `OSSolverInterfaces` library is designed to facilitate linking the OS library with various solver APIs. We first describe how to take a problem instance in OSiL format and connect to a solver that has a COIN-OR OSI interface. See the OSI project `www.projects.coin-or.org/Osi`. We then describe hooking to the COIN-OR nonlinear code `Ipopt.` See `www.projects.coin-or.org/Ipopt`. Finally we describe hooking to two commercial solvers KNITRO and LINDO.

The OS library has been tested with the following solvers using the Osi Interface.

- Cbc
- Clp
- Cplex
- DyLP
- Glpk
- SYMPHONY
- Vol

In the `OSSolverInterfaces` library there is an abstract class `DefaultSolver` that has the following key members:

```
std::string osil;
std::string osol;
std::string osrl;
OSInstance *osinstance;
OSResult  *osresult;
```

and the pure virtual function

```
virtual void solve() = 0 ;
```

In order to use a solver through the COIN-OR `Osi` interface it is necessary to an object in the `CoinSolver` class which inherits from the `DefaultSolver` class and implements the appropriate `solve()` function. We illustrate with the Clp solver.

```
DefaultSolver *solver  = NULL;
solver = new CoinSolver();
solver->m_sSolverName = "clp";
```

Assume that the data file containing the problem has been read into the string `osil` and the solver options are in the string `osol`. Then the `Clp` solver is invoked as follows.

```
solver->osil = osil;
solver->osol = osol;
solver->solve();
```

Finally, get the solution in `OSrL` format as follows

```
cout << solver->osrl << endl;
```

Even though LINDO and KNITRO are commercial solvers and do not have a COIN-OR `Osi` interface these solvers are used in exactly the same manner as a COIN-OR solver. For example, to invoke the LINDO solver we do the following.

```
solver = new LindoSolver();
```

Similarly for KNITRO and Ipopt. In the case of the KNITRO, the `KnitroSolver` class inherits from both `DefaultSolver` class and the KNITRO `NlpProblemDef` class. See `Kipp--putinKnitromanuallink` for more information on the KNITRO solver C++ implementation and the `NlpProblemDef` class. Similarly, for Ipopt the `IpoptSolver` class inherits from both the `DefaultSolver` class and the Ipopt `TNLP` class. See `Kipp--putinIpoptmanuallink` for more information on the Ipopt solver C++ implementation and the `TNLP` calss.

In the examples above the problem instance was assumed to be read from a file into the string `osil` and then into the class member `solver->osil.` However, everything can be done entirely in memory. For example, it is possible to use the `OSInstance` class to create an in-memory problem representation and give this representation directly to a solver class that inherits from `DefaultSolver`. The class member to use is `osinstance.` This is illustrated in the example given in Section 10.4.

4.6 OSUtils

5 OS Protocols

The objective of (OS) is to provide a set of standards for representing optimization instances, results, solver options, and communication between clients and solvers in a distributed environment using Web Services. These standards are specified by W3C XSD schemas. The schemas for the OS project are contained in the `schemas` folder under the `OS` root. There are numerous schemas in this directory that are part of the OS standard. For a full description of all the schemas see Ma [4]. We briefly discuss the standards most relevant to the current version of the OS project.

OSiL (Optimization Services instance Language): an XML-based language for representing instances of large-scale optimization problems including linear programs, mixed-integer programs, quadratic programs, and very general nonlinear programs.

OSiL, stores optimization problem instances as XML files. Consider the following problem instance that is a modification of an example of Rosenbrock [5]:

$$\text{Minimize} \quad (1 - x_0)^2 + 100(x_1 - x_0^2)^2 + 9x_1 \tag{7}$$

$$\text{s.t.} \quad x_0 + 10.5x_0^2 + 11.7x_1^2 + 3x_0x_1 \leq 25 \tag{8}$$

$$\ln(x_0x_1) + 7.5x_0 + 5.25x_1 \geq 10 \tag{9}$$

$$x_0, x_1 \geq 0 \tag{10}$$

There are two continuous variables, x_0 and x_1, in this instance, each with a lower bound of 0. Figure 9 shows how we represent this information in an XML-based OSiL file. Like all XML files, this is a text file that contains both *markup* and *data.* In this case there are two types of markup, *elements* (or *tags*) and *attributes* that describe the elements. Specifically, there are a `<variables>` element and two `<var>` elements. Each `<var>` element has attributes `lb`, `name`, and `type` that describe properties of a decision variable: its lower bound, "name", and domain type.

To be useful for communication between solvers and modeling languages, OSiL instance files must conform to a standard. An XML-based representation standard is imposed through the use of a *W3C XML Schema.* The W3C, or World Wide Web Consortium (`www.w3.org`), promotes standards for the evolution of the web and for interoperability between web products. XML Schema (`www.w3.org/XML/Schema`) is one such standard. A schema specifies the elements and attributes that define a specific XML vocabulary. The W3C XML Schema is thus a schema for schemas; it specifies the elements and attributes for a schema that in turn specifies elements and attributes for an XML vocabulary such as OSiL. An XML file that conforms to a schema is called *valid* for that schema.

```
<variables numberOfVariables="2">
    <var lb="0" name="x0" type="C"/>
    <var lb="0" name="x1" type="C"/>
</variables>
```

Figure 9: The `<variables>` element for the example (1)–(4).

By analogy to object-oriented programming, a schema is akin to a header file in C++ that defines the members and methods in a class. Just as a class in C++ very explicitly describes member and method names and properties, a schema explicitly describes element and attribute names and properties.

Figure 10 is a piece of our schema for OSiL. In W3C XML Schema jargon, it defines a *complexType,* whose purpose is to specify elements and attributes that are allowed to appear in a valid XML instance file such as the one excerpted in Figure 9. In particular, Figure 10 defines the complexType named `Variables`, which comprises an element named `<var>` and an attribute named `numberOfVariables`. The `numberOfVariables` attribute is of a standard type `positiveInteger`, whereas the `<var>` element is a user-defined complexType named `Variable`. Thus the complexType `Variables` contains a sequence of `<var>` elements that are of complexType `Variable`. OSiL's schema must also provide a specification for the `Variable` complexType, which is shown in Figure 11.

In OSiL the linear part of the problem is stored in the `<linearConstraintCoefficients>` element, which stores the coefficient matrix using three arrays as proposed in the earlier LPFML schema [2]. There is a child element of `<linearConstraintCoefficients>` to represent each array: `<value>` for an array of nonzero coefficients, `<rowIdx>` or `<colIdx>` for a

```
<xs:complexType name="Variables">
    <xs:sequence>
        <xs:element name="var" type="Variable" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="numberOfVariables"
            type="xs:positiveInteger" use="required"/>
</xs:complexType>
```

Figure 10: The `Variables` complexType in the OSiL schema.

```
<xs:complexType name="Variable">
    <xs:attribute name="name" type="xs:string" use="optional"/>
    <xs:attribute name="init" type="xs:string" use="optional"/>
    <xs:attribute name="type" use="optional" default="C">
        <xs:simpleType>
            <xs:restriction base="xs:string">
                <xs:enumeration value="C"/>
                <xs:enumeration value="B"/>
                <xs:enumeration value="I"/>
                <xs:enumeration value="S"/>
            </xs:restriction>
        </xs:simpleType>
    </xs:attribute>
    <xs:attribute name="lb" type="xs:double" use="optional" default="0"/>
    <xs:attribute name="ub" type="xs:double" use="optional" default="INF"/>
</xs:complexType>
```

Figure 11: The `Variable` complexType in the OSiL schema.

corresponding array of row indices or column indices, and `<start>` for an array that indicates where each row or column begins in the previous two arrays.

```
<linearConstraintCoefficients numberOfValues="3">
    <start>
        <el>0</el><el>2</el><el>3</el>
    </start>
    <rowIdx>
        <el>0</el><el>1</el><el>1</el>
    </rowIdx>
    <value>
        <el>1.</el><el>7.5</el><el>5.25</el>
    </value>
</linearConstraintCoefficients>
```

Figure 12: The `<linearConstraintCoefficients>` element for constraints (8) and (9).

The quadratic part of the problem is represented as follows.

```
<quadraticCoefficients numberOfQuadraticTerms="3">
     <qTerm idx="0" idxOne="0" idxTwo="0" coef="10.5"/>
     <qTerm idx="0" idxOne="1" idxTwo="1" coef="11.7"/>
     <qTerm idx="0" idxOne="0" idxTwo="1" coef="3."/>
</quadraticCoefficients>
```

Figure 13: The `<quadraticCoefficients>` element for constraint (8).

The nonlinear part of the problem is given in Figure 14.

The complete OSiL representation is given in the Appendix.

OSrL (Optimization Services result Language): an XML-based language for representing the solution of large-scale optimization problems including linear programs, mixed-integer programs, quadratic programs, and very general nonlinear programs. As example solution (for the problem given in (7)–(10)) in OSrL format is given below.

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type = "text/xsl"
    href = "/Users/kmartin/Documents/files/code/cpp/OScpp/COIN-OSX/OS/stylesheets/OSrL.xsl
<osrl xmlns="os.optimizationservices.org" xmlns:xsi="http://www.w3.org/2001/XMLSchema-inst
    xsi:schemaLocation="os.optimizationservices.org http://www.optimizationservices.org/sc
    <resultHeader>
        <generalStatus type="success"/>
        <serviceName>Solved using a LINDO service</serviceName>
        <instanceName>Modified Rosenbrock</instanceName>
    </resultHeader>
    <resultData>
```

```
<nl idx="-1">
     <plus>
          <power>
               <minus>
                    <number value="1.0"/>
                    <variable coef="1.0" idx="0"/>
               </minus>
               <number value="2.0"/>
          </power>
          <times>
               <power>
                    <minus>
                         <variable coef="1.0" idx="0"/>
                         <power>
                              <variable coef="1.0" idx="1"/>
                              <number value="2.0"/>
                         </power>
                    </minus>
                    <number value="2.0"/>
               </power>
               <number value="100"/>
          </times>
     </plus>
</nl>
```

Figure 14: The `<nl>` element for the nonlinear part of the objective (7).

```
<optimization numberOfSolutions="1" numberOfVariables="2" numberOfConstraints="2"
    numberOfObjectives="1">
    <solution objectiveIdx="-1">
        <status type="optimal"/>
        <variables>
            <values>
                <var idx="0">0.87243</var>
                <var idx="1">0.741417</var>
            </values>
            <other name="reduced costs" description="the variable reduced costs">
                <var idx="0">-4.06909e-08</var>
                <var idx="1">0</var>
            </other>
        </variables>
        <objectives>
            <values>
                <obj idx="-1">6.7279</obj>
            </values>
        </objectives>
        <constraints>
            <dualValues>
                <con idx="0">0</con>
```

```
                        <con idx="1">0.766294</con>
                    </dualValues>
                </constraints>
            </solution>
        </optimization>
```

OSoL (Optimization Services option Language): an XML-based language for representing options that get passed to an optimization solver.

OSnL (Optimization Services nonlinear Language): The OSnL schema is imported by the OSiL schema and is used to represent the nonlinear part of an optimization instane. This is explained in greater detail in Section 4.2.4. Also refer to Figue 14 for an illustration of elements from the OSnL standard.

OSpL (Optimization Services process Language): is a standard for dynamic process information that is kept by the Optimization Services registry. It is the result of a `knock` operation. See the example given in Section 8.3.5.

6 The OSInstance API

The OSInstance API can be used to:

- get information about model parameters, or convert the `OSExrpressionTree` into a prefix or postfix representation through a set of `get` methods,
- modify, or even create and instance from scratch, using a set of `set` methods,
- provide information to solvers that require function evaluations, Jacobian and Hessian sparsity patters, function gradient evaluations, and Hessian evaluations.

6.1 Get Methods

The `get()` methods are used by other classes to access data in an existing object or create an instance in postfix or prefix format. Assume `osinstance` is an object in the `OSInstance` class created as illustrated in Figure 3. Then, for example,

```
osinstance->getVariableNumber();
```

will return an integer which is the number of variables in the problem,

```
osintance->getVariableTypes();
```

will return a `char` pointer to the variable types (`C` for continuous, `B` for binary, and `I` for general integer),

```
getVariableLowerBounds();
```

will return a `double` pointer to the lower bound on each variable. There are similar `get` methods for the constraints. There are numerous `get` methods for the data in the `<linearConstraintCoefficients>` element, the `<quadraticCoefficients>` element, and the `<nonlinearExpressions>` element.

When an `osinstance` object is created, it is stored as in expression tree in an `OSExpressionTree` object. However, some solver APIs (e.g. LINDO) may take the data in a different format such as postfix and prefix. There are methods to return the data in either postfix or prefix format.

First define a `vector` of pointers to `OSnLNode` objects.

```
std::vector<OSnLNode*> postfixVec;
```

then get the expression tree for the objective function (index = -1) as a postfix vector of nodes.

```
postfixVec = osinstance->getNonlinearExpressionTreeInPostfix( -1);
```

If, for example, the `osinstance` object was the in-memory representation of the instance illustrated in Section 11.2 then the code

```
for (i = 0 ; i < n; i++){
cout << postfixVec[i]->snodeName << endl;
}
```

will produce

```
number
variable
minus
number
power
number
variable
variable
number
power
minus
number
power
times
plus
```

The method, `processNonlinearExpressions()` in the `LindoSolver` class in the `OSSolverInterfaces` library component illustrates using a postfix vector of `OSnLNode` objects to build a Lindo model instance.

6.2 Set Methods

The `set` methods can be used to build an in-memory `OSInstance` object. A code example of how to do this is in Section 10.4.

6.3 Calculate Methods

The calculate methods are described in Section 7.

7 Hooking to An Algorithmic Differentiation Package

The OS library provides a set of `calculate` methods for calculating function values, gradients, and Hessians. The `calculate` methods are part of the `OSInstance` class and are designed to work with solver APIs.

7.1 Algorithmic Differentiation: Brief Review

Here we provide a brief review of algorithmic differentiation. For an excellent reference on algorithmic differentiation see Griewank [3]. The OS package uses the COIN-OR package CppAD which is also excellent resource with extensive documentation and information about algorithmic differentiation. See the documentation written by Brad Bell [1]. The development here is from the CppAD documentation.

Consider the function $f : X \rightarrow Y$ from $\mathbb{R}^n$ to $\mathbb{R}^m$.

Express the input vector as scalar function of t by

$$X(t) = x^{(0)} + x^{(1)}t + x^{(2)}t^2 \tag{11}$$

Then

$$\begin{aligned} X(0) &= x^{(0)} \\ X'(0) &= x^{(1)} \\ X''(0) &= 2x^{(2)} \end{aligned}$$

and in general the $x^{(k)}$ correspond to the $k'th$ order Taylor coefficient, i.e.

$$x^{(k)} = \frac{1}{k!}X^{(k)}(0)$$

Then $Y(t) = f(X(t))$ is a function from $\mathbb{R}^1$ to $\mathbb{R}^m$ and it is expressed in terms of its Taylor series expansion as

$$Y(t) = y^{(0)} + y^{(1)}t + y^{(2)}t^2 + o(t^3) \tag{12}$$

where

$$y^{(k)} = \frac{1}{k!}Y^{(k)}(0) \tag{13}$$

It is shown by Bell `http://www.coin-or.org/CppAD/` that:

$$y^{(0)} = f(x^{(0)}) \tag{14}$$

Let $e^{(i)}$ denote the $i'th$ unit vector.

If $x^{(1)} = e^{(i)}$ then

$$y^{(1)} = \frac{\partial f}{\partial x_i}(x^{(0)}) \tag{15}$$

If $x^{(1)} = e^{(i)}$ and $x^{(2)} = 0$ then for function $f_k(x)$,

$$y_k^{(2)} = \frac{1}{2}\frac{\partial^2 f_k(x^{(0)})}{\partial x_i \partial x_i} \tag{16}$$

If $x^{(1)} = e^{(i)} + e^{(j)}$ and $x^{(2)} = 0$ then for function $f_k(x)$,

$$y_k^{(2)} = \frac{1}{2}\left(\frac{\partial^2 f_k(x^{(0)})}{\partial x_i \partial x_i} + \frac{\partial^2 f_k(x^{(0)})}{\partial x_i \partial x_j} + \frac{\partial^2 f_k(x^{(0)})}{\partial x_j \partial x_i} + \frac{\partial^2 f_k(x^{(0)})}{\partial x_j \partial x_j}\right) \tag{17}$$

or, expressed in terms of the mixed partials,

$$\frac{\partial^2 f_k(x^{(0)})}{\partial x_i \partial x_j} = y_k^{(2)} - \frac{1}{2}\left(\frac{\partial^2 f_k(x^{(0)})}{\partial x_i \partial x_i} + \frac{\partial^2 f_k(x^{(0)})}{\partial x_j \partial x_j}\right) \tag{18}$$

7.2 Using OSInstance Methods: Low Level Calls

We work with the following example. The code snippets fused for this are from `CppADTest.cpp` in the `CppADTest` folder in the `examples` folder.

$$\begin{aligned}
&\text{Minimize} && x_0^2 + 9x_1 && (19)\\
&\text{s.t.} && 33 - 105 + 1.37x_1 + 2x_3 + 5x_1 \leq 10 && (20)\\
& && \ln(x_0 x_3) + 7x_2 \geq 10 && (21)\\
& && x_0, x_1, x_2, x_3 \geq 0 && (22)
\end{aligned}$$

The OSiL representation of the instance (19)-(22) is given in Appendix 11.3. This example is designed to illustrate several features of OSiL. Note that in equation (20) the constant 33 appears in the `<con>` element corresponding to this constraint and the constant 105 appears as a `<number>` OSnL node in the `<nonlinearExpressions>` section. Although variable x_1 does not appear in any nonlinear expression the $5x_1$ term in equation (20) is expressed in the `<linearConstraintCoefficients>` section and the $1.37x_1$ term in equation (20) is expressed in the `<nonlinearExpressions>` section. Hence, in the OSInstance API, variable x_1 is treated as a nonlinear variable for purposes of algorithmic differentiation. However, variable x_2 never appears in the `<nonlinearExpressions>` section and is therefore treated as a linear variable and not used in algorithmic differentiation calculations.

Ignoring the nonnegativity constraints, instance (19)-(22) defines the following function $f : X \rightarrow Y$ from $\mathbb{R}^4$ to $\mathbb{R}^3$.

$$f(x) = \begin{bmatrix} f_1(x) \\ f_2(x) \\ f_3(x) \end{bmatrix} = \begin{bmatrix} x_0^2 + 9x_1 \\ 33 - 105 + 1.37x_1 + 2x_3 + 5x_1 \\ \ln(x_0 x_3) + 7x_2 \end{bmatrix} \quad (23)$$

The OSiL representation for the instance in (19)-(**??**) is read into an in-memory OSInstance object as follows

```
fileUtil = new FileUtil();
osil = fileUtil->getFileAsString( &osilFileName[0]);
osilreader = new OSiLReader();
osinstance = osilreader->readOSiL( &osil);
```

There is a method in the `OSInstance` class, `initForAlgDiff()` that is used to initialize the nonlinear data structures. A call to this method

```
osinstance->initForAlgDiff( );
```

will generate a map of the indices of the nonlinear variables. This is critical because the algorithmic differentiation only operates on variables that appear in the `<nonlinearExpressions>` section. An example of this map follows.

```
std::map<int, int> varIndexMap;
std::map<int, int>::iterator posVarIndexMap;
varIndexMap = osinstance->getAllNonlinearVariablesIndexMap( );
for(posVarIndexMap = varIndexMap.begin(); posVarIndexMap
!= varIndexMap.end(); ++posVarIndexMap){
std::cout <<  "Variable Index = "   << posVarIndexMap->first  << std::endl ;
}
```

The variable indices listed are 0, 1, and 3 since variable 2 does not appear in the `<nonlinearExpressions>` section.

Once the nonlinear structures are initialized it is possible to take derivatives using algorithmic differentiation. Algorithmic differentiation is done using either a forward or reverse sweep through an expression tree (or operation sequence) representation of f. The two key algorithmic differentiation `public` methods in the `OSInstance` class are `forwardAD` and `reverseAD`. These are actually generic "wrappers" around the corresponding CppAD methods with the same signature. This keeps the OS API public methods independent of any underlying algorithmic differentiation package.

The `forwardAD` signature is

```
std::vector<double> forwardAD(int p, std::vector<double> vdX);
```

where p is the highest order Taylor coefficient of f to be calculated, `vdX` is vector of doubles in $\mathbb{R}^n$, and the function return is a vector of doubles in $\mathbb{R}^m$. For example, by result in Equation (14) the following call will evaluate each component function defined in 23.

```
funVals = osinstance->forwardAD(0, x0);
```

Since there are three components in the vector defined by 23, the return value `funVals` will have three components. For an input vector,

```
x0[0] = 1; // the value for variable x0
x0[1] = 5; // the value for variable x1
x0[2] = 5; // the value for variable x3
```

the values returned by `osinstance->forwardAD(0, x0)` are 1, -63.15, and 1.6094, respectively. The Jacobian of the example in (23) is

$$J = \left[\begin{array}{rrrr} 2x_0 & 9.00 & 0.00 & 0.00 \\ 0.00 & 6.37 & 0.00 & 2.00 \\ 1/x_0 & 0.00 & 7.00 & 1/x_3 \end{array} \right] \tag{24}$$

when $x_0 = 1$, $x_1 = 5$, $x_2 = 10$, and $x_3 = 5$ the Jacobian is

$$J = \left[\begin{array}{rrrr} 2.00 & 9.00 & 0.00 & 0.00 \\ 0.00 & 6.37 & 0.00 & 2.00 \\ 1.00 & 0.00 & 7.00 & 0.20 \end{array} \right] \tag{25}$$

A forward sweep will calculate the Jacobian column-wise. See (15). The following code will return column 4 of the Jacobian (25) which corresponds to the third nonlinear variable.

```
x1[0] = 0;
x1[1] = 0;
x1[2] = 1;
fosinstance->forwardAD(1, x1);
```

Now calculate second derivatives. To illustrate we use the results in (16)-(18) and calculate

$$\frac{\partial^2 f_k(x^{(0)})}{\partial x_0 \partial x_3} \quad k = 1, 2, 3.$$

Variables x_0 and x_3 are the first and third nonlinear variables so by (17) the $x^{(1)}$ should be the sum of the $e^{(1)}$ and e^3 unit vectors and used in first-order forward sweep calculation.

```
x1[0] = 1;
x1[1] = 0;
x1[2] = 1;
osinstance->forwardAD(1, x1);
```

Next set $x^{(0)} = 0$ and do a second-order forward sweep.

```
std::vector<double> x2( n);
x2[0] = 0;
x2[1] = 0;
x2[2] = 0;
osinstance->forwardAD(2, x2);
```

This call returns the vector of values

$$y_1^{(2)} = 1, \quad y_2^{(2)} = 0, \quad y_3^{(2)} = -.52$$

By inspection,

$$\begin{aligned}
\frac{\partial^2 f_1(x^{(0)})}{\partial x_0 \partial x_0} &= 2 \\
\frac{\partial^2 f_2(x^{(0)})}{\partial x_0 \partial x_0} &= 0 \\
\frac{\partial^2 f_3(x^{(0)})}{\partial x_0 \partial x_0} &= -1 \\
\frac{\partial^2 f_1(x^{(0)})}{\partial x_3 \partial x_3} &= 0 \\
\frac{\partial^2 f_2(x^{(0)})}{\partial x_3 \partial x_3} &= 0 \\
\frac{\partial^2 f_3(x^{(0)})}{\partial x_3 \partial x_3} &= -.04
\end{aligned}$$

Then by (18),

$$\begin{aligned}
\frac{\partial^2 f_1(x^{(0)})}{\partial x_0 \partial x_3} &= y_1^{(2)} - \frac{1}{2}\left(\frac{\partial^2 f_1(x^{(0)})}{\partial x_0 \partial x_0} + \frac{\partial^2 f_k(x^{(0)})}{\partial x_3 \partial x_3}\right) = 1 - \frac{1}{2}(2+0) = 0 \\
\frac{\partial^2 f_2(x^{(0)})}{\partial x_0 \partial x_3} &= y_2^{(2)} - \frac{1}{2}\left(\frac{\partial^2 f_2(x^{(0)})}{\partial x_0 \partial x_0} + \frac{\partial^2 f_k(x^{(0)})}{\partial x_3 \partial x_3}\right) = 0 - \frac{1}{2}(0+0) = 0 \\
\frac{\partial^2 f_3(x^{(0)})}{\partial x_0 \partial x_3} &= y_3^{(2)} - \frac{1}{2}\left(\frac{\partial^2 f_3(x^{(0)})}{\partial x_0 \partial x_0} + \frac{\partial^2 f_k(x^{(0)})}{\partial x_3 \partial x_3}\right) = -52 - \frac{1}{2}(-1 - .04) = 0
\end{aligned}$$

Making all of the first and second derivative calculations using forward sweeps is most effective when the number of rows exceeds the number of variables.

The `reverseAD` signature is

```
std::vector<double> reverseAD(int p, std::vector<double> vdlambda);
```

where `vdlambda` is a vector of Lagrange multipliers. This method returns a vector in the range space. If a reverse sweep of order p is called, a forward sweep of order at $p-1$ must have been made prior to the call.

7.2.1 First Derivative Reverse Sweep Calculations

In order to calculate first derivatives execute the following sequence of calls.

```
std::vector<double> vlambda(3);
vlambda[0] = 0;
vlambda[1] = 0;
vlambda[2] = 1;
osinstance->forwardAD(0, x0);
osinstance->reverseAD(1, vlambda);
```

Since the `vlambda` only includes the third function $f_1(x)$ the sequence of calls will produce the third row of the Jacobian, i.e.

$$\frac{\partial f_3(x^{(0)})}{\partial x_0} = 1, \quad \frac{\partial f_3(x^{(0)})}{\partial x_1} = 0, \quad \frac{\partial f_3(x^{(0)})}{\partial x_3} = .2$$

7.2.2 Second Derivative Reverse Sweep Calculations

In order to calculate second derivatives using `reverseAD` forward sweeps of order 0 and 1 must be assume. The call to `reverseAD(2, vlambda)` will return a vector of dimension $2n$ where n is the number of variables. If the first-order sweep is `forwardAD(1, x1)` where $x1 = e^{(i)}$ then the return vector $z = forwardAD(1, x1)$ is

$$z[2j-2] = \frac{\partial L(x^{(0)}, \lambda^{(0)})}{\partial x_j}, \quad j = 1, \ldots, n \tag{26}$$

$$z[2j-1] = \frac{\partial^2 L(x^{(0)}, \lambda^{(0)})}{\partial x_i \partial x_j}, \quad j = 1, \ldots, n \tag{27}$$

where

$$L(x, \lambda) = \sum_{k=1}^{m} \lambda_k f_k(x) \tag{28}$$

For example, the following calls will calculate the third row (column) of the Hessian of the Lagrangian.

```
x0[0] = 1;
x0[1] = 5;
x0[2] = 5;
osinstance->forwardAD(0, x0);
x1[0] = 0;
x1[1] = 0;
x1[2] = 1;
osinstance->forwardAD(1, x1);
vlambda[0] = 1;
vlambda[1] = 2;
vlambda[2] = 1;
osinstance->reverseAD(2, vlambda);
```

This returns

$$\frac{\partial L(x^{(0)},\lambda^{(0)})}{\partial x_1} = 3, \quad \frac{\partial L(x^{(0)},\lambda^{(0)})}{\partial x_2} = 12.74, \quad \frac{\partial L(x^{(0)},\lambda^{(0)})}{\partial x_3} = 4.2$$

$$\frac{\partial^2 L(x^{(0)},\lambda^{(0)})}{\partial x_3 \partial x_0} = 0, \quad \frac{\partial^2 L(x^{(0)},\lambda^{(0)})}{\partial x_3 \partial x_1} = 0, \quad \frac{\partial^2 L(x^{(0)},\lambda^{(0)})}{\partial x_3 \partial x_3} = -.04$$

7.3 Using OSInstance Methods: High Level Calls

The methods `forwardAD` and `reverseAD` are low level calls and are not designed to work directly with solver APIs. Other methods are available that will probably be easier to work with. We describe these now.

7.3.1 Sparsity Methods

Many solvers such as Ipopt `projects.coin-or.org/Ipopt` or Knitro `www.ziena.com` require the sparsity pattern of the Jacobian of the constraint matrix and the Hessian of the Lagrangian function. The following code illustrates how to get the sparsity pattern of the constraint Jacobian matrix

```
SparseJacobianMatrix *sparseJac;
sparseJac = osinstance->getJacobianSparsityPattern();
for(idx = 0; idx < osinstance->getConstraintNumber(); idx++){
    std::cout << "number constant terms in constraint "   <<  idx << " is "
    << *(sparseJac->conVals + idx)  << std::endl;
    for(k = *(sparseJac->starts + idx); k < *(sparseJac->starts + idx + 1); k++){
        std::cout << "row idx = " << idx <<  "
        col idx = "<< *(sparseJac->indexes + k) << std::endl;
    }
}
```

For the example problem this will produce

```
JACOBIAN SPARSITY PATTERN
number constant terms in constraint 0 is 0
row idx = 0  col idx = 1
row idx = 0  col idx = 3
number constant terms in constraint 1 is 1
row idx = 1  col idx = 2
row idx = 1  col idx = 0
row idx = 1  col idx = 3
```

The `SparseJacobianMatrix` object has a data member `starts` which is the index of the start of each constraint row. The `int` data member `indexes` is the variable index of a potential nonzero derivative. There is also a `double` data member `values` that will the value of the partial derivative of the corresponding index at each iteration. Finally, there is an `int` data member `conVals` which is the number of constant terms in each gradient. A constant term is a partial derivative that cannot change at an iteration. A variable is considered a constant variable if it appears in the `<linearConstraintCoefficients>` section but not in the `nonlinearExpressions`. For a row indexed by `idx` the variable indices are in the `indexes` array between the elements `sparseJac->starts + idx` and `sparseJac->starts + idx + 1`. The first `sparseJac->conVals + idx` variables listed are indices of constant variables. In this example, when `idx` is 1, there is one constant variable and it is variable x_2. The constant variables never appear in the AD evaluation.

The following code illustrates how to get the sparsity pattern of the Hessian of the Lagrangian.

```
SparseHessianMatrix *sparseHessian;
sparseHessian = osinstance->getLagrangianHessianSparsityPattern( );
for(idx = 0; idx < sparseHessian->hessDimension; idx++){
std::cout <<  "Row Index = " << *(sparseHessian->hessRowIdx + idx) ;
std::cout <<  "  Column Index = " << *(sparseHessian->hessColIdx + idx);
}
```

The `SparseHessianMatrix` class has the `int` data members `hessRowIdx` and `hessColIdx` for indexing potential nonzero elements in the Hessian matrix. The `double` data member `hessValues` holds the value of the respective second derivative at each iteration. If `numVars` is the number of nonlinear variables, each array in `sparseHessian` is of size

$$numVars * (numVars + 1)/2;$$

All mixed partials of nonlinear terms are considered to be potential nonzeros. Hopefully, a future implementation will be more robust in preserving sparsity.

7.3.2 Function Evaluation Methods

There are several overloaded methods for calculating objective and constraint values. The method

```
double *calculateAllConstraintFunctionValues(double* x, bool new_x)
```

will return a `double` pointer to an array of constraint function values evaluated at `x`. If the value of `x` has not changed since the last function call, then `new_x` should be set to `false` and the most recent function values will be returned. When using this method, with this signature, all function evaluations are made in `double` using an `OSExpressionTree` object.

A second signature for the `calculateAllConstraintFunctionValues` is

```
double *calculateAllConstraintFunctionValues(double* x, double *objLambda,
    double *conLambda, bool new_x, int highestOrder)
```

In this signature, `x` is a pointer to the current primal values, `objLambda` is a vector of dual multipliers, `conLambda` is a vector of dual multipliers on the constraints, `new_x` is true if any components of `x` have changed since the last evaluation, and `highestOrder` is the highest order of derivative to be calculated at this iteration. The following code snippet illustrates defining a set of variable values for the example we are using and then the function call.

```
double* x = new double[4]; //primal variables
double* z = new double[2]; //Lagrange multipliers on constraints
double* w = new double[1]; //Lagrange multiplier on objective
x[ 0] = 1;    // primal variable 0
x[ 1] = 5;    // primal variable 1
x[ 2] = 10;   // primal variable 2
x[ 3] = 5;    // primal variable 3
z[ 0] = 2;    // Lagrange multiplier on constraint 0
z[ 1] = 1;    // Lagrange multiplier on constraint 1
w[ 0] = 1;    // Lagrange multiplier on the objective function
calculateAllConstraintFunctionValues(x, w, z,  true, 0);
```

When making all high level calls for function, gradient, and Hessian evaluations we use pass all the primal variables in the `x` argument, not just the nonlinear variables. Underneath the call, the nonlinear variables are identified and used in AD function calls.

The use of the parameters `new_x` and `highestOrder` is important and requires further explanation. The parameter `highestOrder` is an integer variable that will take on the value 0, 1, or 2 (actually higher values if we want third derivatives etc.). The value of this variable is the highest order derivative that is required of the current iterate. For example, if a callback requires a function evaluation and **highestOrder = 0** then only the function is evaluated at the current iterate. However, if `highsetOrder = 2` then the function call

```
calculateAllConstraintFunctionValues(x, w, z, true, 2)
```

will trigger first and second derivative evaluations in addition to the function evaluations.

In the `OSInstance` class code, every time a forward (`forwardAD`) or reverse sweep (`reverseAD`) is executed a private member, `m_iHighestOrderEvaluated` is set to the order of the sweep. For example, `forwardAD(1, x)` would result in `m_iHighestOrderEvaluated = 1`. Just knowing the value of **new˙x** alone is not sufficient. It is also necessary to know **highestOrder** and compare it with `m_iHighestOrderEvaluated`. For example, if **new˙x** is false, but `m_iHighestOrderEvaluated = 0`, and the callback requires a Hessian calculation, then it is necessary to calculate the first and second derivatives at the current iterate.

There are *exactly two* conditions that require a new function or derivative calculation. A new evaluation is required if and only if

1. The value of **new˙x** is true

–OR–

2. For the callback function the value of the input parameter **highestOrder** is strictly greater than the current value of **m˙iHhighestOrderEvaluated.**

It is not really necessary for all callback functions to have the above arguments. However, for an efficient implementation of AD it is important to be able to get the Lagrange multipliers and highest order derivative that is required from inside *any* callback – not just the Hessian evaluation callback. For example, in **Ipopt,** if `eval_g` or `eval_f` are called, and for the current iterate, `eval_jac` and `eval_hess` are also going to be called, then a more efficient AD implementation is possible if the Lagrange multipliers are available for `eval_g` and `eval_f`.

Currently, whenever `new_x = true` in the underlying AD implementation we do not retape the function. This is because we currently throw an exception if there are any logical operators involved in the AD calculations. This may change in a future implementation.

There are also similar methods for objective function evaluations. There is also a method

```
double calculateFunctionValue(int idx, double* x, bool new_x);
```

that will return the value of any constraint or objective function indexed by `idx`. This method works strictly with `double` data using an `OSExpressionTree` object.

There is also a public variable, `bUseExpTreeForFunEval` that, if set to `true`, will cause the method

```
calculateAllConstraintFunctionValues(x, objLambda,  conLambda, true, highestOrder)
```

to also use the OS expression tree for function evaluations when `highestOrder = 0` rather than use the operator overloading in the CppAD tape.

7.3.3 Gradient Evaluation Methods

One `OSInstance` method for gradient calculations is

```
SparseJacobianMatrix *calculateAllConstraintFunctionGradients(double* x, double *objLambda,
     double *conLambda, bool new_x, int highestOrder)
```

If a call has been placed to `calculateAllConstraintFunctionValues` with `highestOrder = 0`, then the appropriate call to get gradient evaluations is

```
calculateAllConstraintFunctionGradients( x, NULL, NULL,  false, 1);
```

Note that in this function call `new_x = false`. This prevents a call to `forwardAD()` with order 0 to get the function values.

If, at the current iterate, the Hessian of the Lagrangian function is also desired then an appropriate call is

```
calculateAllConstraintFunctionGradients(objLambda, conLambda, false, 2);
```

In this case, if there was a prior call

```
calculateAllConstraintFunctionValues(x, w, z,  true, 0);
```

then only first and second derivatives are calculated, not function values.

When calculating the gradients, if the number of nonlinear variables exceeds or is equal to the number of rows, a `forwardAD(0, x)` sweep is used to get the function values, and a `reverseAD(1,` e^k`)` sweep for each unit vector e^k in the row space is used to get the vector of first order partials for each row in the constraint Jacobian. If the number of nonlinear variables is less then the number of rows then a `forwardAD(0, x)` sweep is used to get the function values and a `forwardAD(1,` e^i`)` sweep or each unit vector e^i in the column space is used to get the vector of first order partials for each column in the constraint Jacobian.

Two other gradient methods are

```
SparseVector *calculateConstraintFunctionGradient(double* x,
    double *objLambda, double *conLambda,
    int idx, bool new_x, int highestOrder);
```

and

```
SparseVector *calculateConstraintFunctionGradient(double* x, int idx,
    bool new_x );
```

Similar methods are available for the objective function, however the objective function gradient methods treat the gradient of each objective function as a dense vector.

7.3.4 Hessian Evaluation Methods

There are two methods for Hessian calculations. The first method has the signature

```
SparseHessianMatrix *calculateLagrangianHessian( double* x,
    double *objLambda, double *conLambda, bool new_x, int highestOrder);
```

so if either function or first derivatives have been calculated an appropriate call is

```
calculateLagrangianHessian( x, w, z, false, 2);
```

If the Hessian of a single row or objective function is desired the following method is available

```
SparseHessianMatrix *calculateHessian( double* x, int idx, bool new_x);
```

8 The OSSolverService

The `OSSolverService` is a command line executable designed to pass problem instances in either OSiL, AMPL nl, or MPS format to solvers and get the optimization result back to be displayed either to standard output or a specified browser. The `OSSovlerService` can be used to invoke a solver locally or on a remote server. It can work either synchronously or asynchronously.

8.1 OSSolverService Input Parameters

At present, the `OSSolverService` takes the following parameters. The order of the parameters is irrelevant. Not all the parameters are required. However, if the `solve` or `send` service methods are invoked a problem instance location must be specified.

-osil xxx.osil this is the name of the file that contains the optimization instance in OSiL format. It is assumed that this file is available in a directory on the machine that is running `OSSolverService`. If this option is not specified then the instance location must be specified in the OSoL solver options file.

-osol xxx.osol this is the name of the file that contains the solver options. It is assumed that this file is available in a directory on the machine that is running `OSSolverService`. It is not necessary to specify this option.

-osrl xxx.osrl this is the name of the file that contains the solver solution. A valid file path must be given on the machine that is running `OSSolverService`. It is not necessary to specify this option.

-serviceLocation is the URL of the solver service. This is not required, and if not specified it is assumed that the problem is solved locally.

-serviceMethod method this is the solver service required. The options are `solve`, `send`,`kill`,`knock`, `getJobID`, and `retrieve`. The use of these options is illustrated in the examples below. This option is not required, and the default value is `solve.`

-mps xxx.mps this is the name of the mps file if the problem instance is in mps format. It is assumed that this file is available in a directory on the machine that is running `OSSolverService`. The default file format is OSiL so this option is not required.

-nl xxx.nl this is the name of the AMPL nl file if the problem instance is in AMPL nl format. It is assumed that this file is available in a directory on the machine that is running `OSSolverService`. The default file format is OSiL so this option is not required.

-solver solverName Possible values for default OS installation are tt clp (COIN-OR Clp), `cbc` (COIN-OR Cbc), `dylp` (COIN-OR DyLP), and `symphony` (COIN-OR SYMPHONY). Other solvers supported (if the necessary libraries are present) are `cplex` (Cplex through COIN-OR Osi), `glpk` (glpk through COIN-OR Osi), `ipopt` (COIN-OR Ipopt), `knitro` (Knitro), and `lindo LINDO`. If no value is specified for this parameter, then `cbc` is the default value of this parameter if the the `solve` or `send` service methods are used.

-browser browserName this paramater is a path to the browser on the local machine. If this optional parameter is specified then the solver result in OSrL format is transformed using XSLT into HTML and displayed in the browser.

-config pathToConfigureFile this parameter specifies a path on the local machine to a text file containing values for the input parameters. This is convenient for the user not wishing to constantly retype parameter values.

The input parameters to the `OSSolverService` may be given entirely in the command line or in a configuration file. We first illustrate giving all the parameters in the command line. The following command will invoke the `Clp` solver on the local machine to solve the problem instance `parincLinear.osil`.

```
OSSolverService -solver clp -osil ../data/osilFiles/parincLinear.osil
```

Alternatively, these parameters can be put into a configuration file. Assume that the configuration file of interest is `testlocalclp.config`. It would contain the two lines of information

```
-osil ../data/osilFiles/parincLinear.osil
-solver clp
```

Then the command line is

```
OSSolverService -config ../data/configFiles/testlocalclp.config
```

Some Rules:

1. When using the `send()` or `solve()` methods a problem instance file location *must* be specified either at the command line, in the configuration file, or in the `<instanceLocation>` element in the OSoL options file file.

2. The default `serviceMethod` is `solve` if another service method is not specified. The service method cannot be specified in the OSoL options file.

3. If the `solver` option is not specified, the COIN-OR solver `Cbc` is the default solver used. In this case an error is thrown if the problem instance has quadratic or other nonlinear terms.

4. If the options `send`, `kill`, `knock`, `getJobID`, or `retrieve` are specified, a `serviceLocation` must be specified.

Parameters specified in the configure file are overridden by parameters specified at the command line. This is convenient if a user has a base configure file and wishes to override only a few options. For example,

```
OSSolverService -config ../data/configFiles/testlocalclp.config -solver lindo
```

or

```
OSSolverService -solver lindo -config ../data/configFiles/testlocalclp.config
```

will result in the LINDO solver being used even though Clp is specified in the `testlocalclp` configure file.

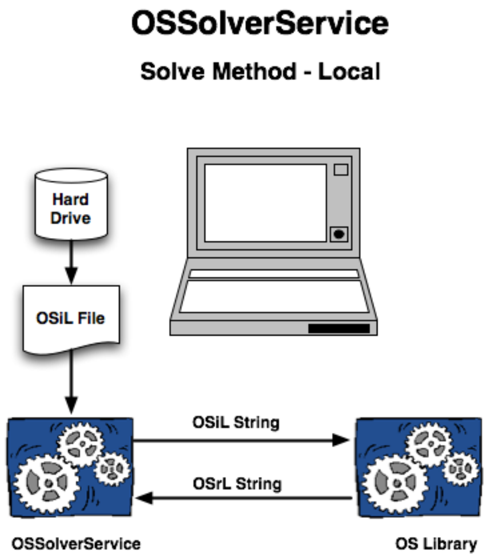

Figure 15: A local call to `solve`.

8.2 Solving Problems Locally

Generally, when solving a problem locally the user will use the `solve` service method. The `solve` method is invoked synchronously and waits for the solver to return the result. This is illustrated in Figure 16. As illustrated, the `OSSolverService` reads a file on the hard drive with the optimization instance, usually in OSiL format. The optimization instance is parsed into a string which is passed to the `OSLibrary` which is linked with various solvers. The result of the optimization is passed back to the `OSSolverService` as a string in OSrL format.

Here is an example of using a configure file, `testlocal.config`, to invoke `Ipopt` locally using the `solve` command.

```
-osil ../data/osilFiles/parincQuadratic.osil
-solver ipopt
-serviceMethod solve
-browser /Applications/Firefox.app/Contents/MacOS/firefox
-osrl /Users/kmartin/temp/test.osrl
```

The first line of `testlocal.config` gives the local location of the OSiL file, `parincQuadratic.osil`, that contains the problem instance. The second parameter, `-solver ipopt`, is the solver to be invoked, in this case COIN-OR Ipopt. The third parameter `-serviceMethod solve` is not really needed, but included only for illustration. The default solver service is `solve.` The fourth parameter is the location of the browser on the local machine. It will read the OSrL file on the local machine using the path specified by the value of the `osrl` parameter, in this case `/Users/kmartin/temp/test.osrl.`

Parameters may also be contained in an XML-file in OSoL format. In the configuration file `testlocalosol.config` we illustrate specifying the instance location in an OSoL file.

```
-osol ../data/osolFiles/demo.osol
-solver clp
```

The file `demo.osol` is

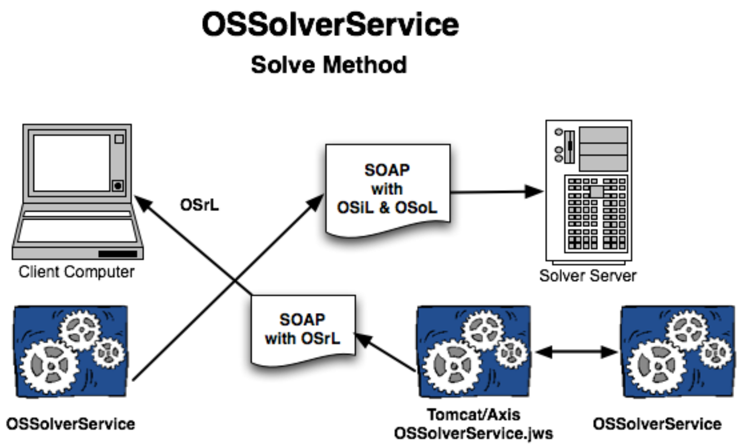

Figure 16: A remote call to `solve`.

```
<?xml version="1.0" encoding="UTF-8"?>
<osol xmlns="os.optimizationservices.org">
    <general>
        <instanceLocation locationType="local">
            ../data/osilFiles/parincLinear.osil
        </instanceLocation>
    </general>
</osol>
```

8.3 Solving Problems Remotely with Web Services

In many cases the client machine may be a "weak client" and using a more powerful machine to solve a hard optimization instance is required. Indeed, one of the major purposes of Optimization Services is to facilitate optimization in a distributed environment. We now provide examples that illustrate using the `OSSolverService` executable to call a remote solver service. By remote solver service we mean a solver service that is called using Web Services. The OS implementation of the solver service uses Apache Tomcat. See `tomcat.apache.org`. The Web Service running on the server is a Java program based on Apache Axis. See `ws.apache.org/axis`. This is described in greater detail in Section 9. This Web Service is called `OSSolverService.jws`. It is not necessary to use the Tomcat/Axis combination.

See Figure 16 for an illustration of this process. The client machine uses `OSSolverService` executable to call one of the six service methods, e.g. `solve`. The information such as the problem instance in OSiL format and solver options in OSoL format are packaged into a SOAP envelope and sent to the server. The server is running the Java Web Service `OSSolverService.jws`. This Java program running in the Tomcat Java Servlet container implements the six service methods. If a `solve` or `send` request is sent to the server from the client, an optimization problem must be solved. The Java solver service solves the optimization instance by calling the OSSolverService on the server. So there is an OSSolverService on the client that calls the Web Service `OSSolverService.jws` that in turn calls the executable `OSSovlerService` on the server. The Java solver service passes options to the local `OSSolverService` such as where the OSiL file is located and where to write the solution result.

In the following sections we illustrate each of the six service methods.

8.3.1 The `solve` Service Method

First we illustrate a simple call to `OSSolverService.jws` and request a solution using the COIN-OR `Clp` solver. The call on the client machine is

```
OSSolverService -config ../data/configFiles/testremote.config
```

where the `testremote.config` file is

```
-osil ../data/osilFiles/parincLinear.osil
-serviceLocation http://128.135.130.17:8080/os/OSSolverService.jws
```

No solver is specified so by default the `Cbc` solver will be used on the server.
Now use an OSoL options file

```
OSSolverService -osol ../data/osolFiles/remoteSolve1.osol -osil ../data/parincLinear.osil
```

where `remoteSolve1.osol` is

```
<?xml version="1.0" encoding="UTF-8"?>
<osol xmlns="os.optimizationservices.org">
    <general>
        <serviceURI>http://128.135.130.17:8080/os/OSSolverService.jws</serviceURI>
    </general>
    <optimization>
     <other name="os_solver">clp</other>
    </optimization>
</osol>
```

In this case we specify a sover to use, name `Clp`.
Next we illustrate a call to the remote SolverService and specify an OSiL instance that is on the remote machine.

```
OSSolverService -osol ../data/osolFiles/remoteSolve2.osol
```

where the `remoteSolve2.osol` file is

```
<?xml version="1.0" encoding="UTF-8"?>
<osol xmlns="os.optimizationservices.org">
    <general>
        <serviceURI>http://128.135.130.17:8080/os/OSSolverService.jws</serviceURI>
         <instanceLocation locationType="local">
     /home/kmartin/files/code/OSRepository/linear/continuous/pilot.osil
     </instanceLocation>
    </general>
    <optimization>
        <other name="os_solver">clp</other>
    </optimization>
</osol>
```

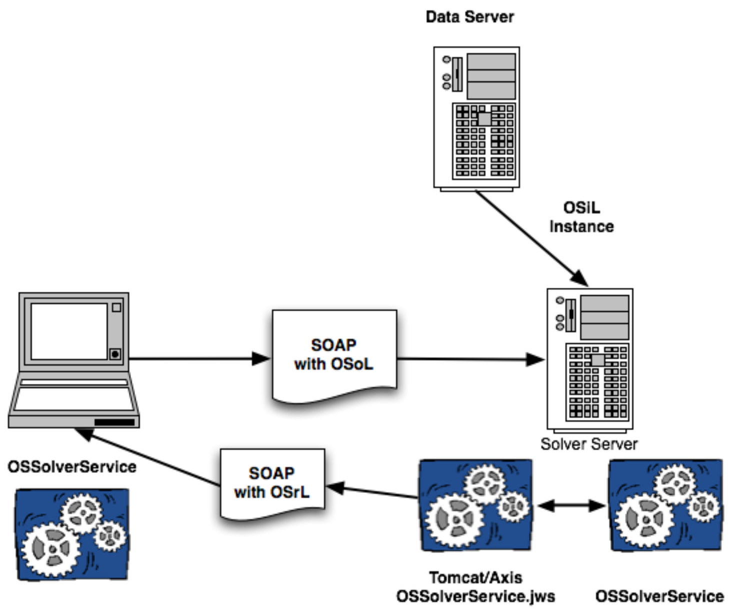

Figure 17: Downloading the instance from a remote source.

If we were to change to the `locationType` attribute in the `<instanceLocation>` element to `http` then we could specify the intance location to on yet another machine. This is illustrated below for `remoteSovle3.osol`. The scenario is depicted in Figure 17. The OSiL string passed from the client to the solver service is empty. However, the OSoL element `<instanceLocation>` has an attribute `locationType` equal to `http`. In this case, the text of the `<instanceLoction>` element contains the URL of a third machine which has the problem intance `parincLinear.osil`. The solver service will contact the machine with URL `gsbkip.chicagogsb.edu` and download this test problem.

```
<?xml version="1.0" encoding="UTF-8"?>
<osol xmlns="os.optimizationservices.org">
    <general>
        <serviceURI>http://128.135.130.17:8080/os/OSSolverService.jws</serviceURI>
         <instanceLocation locationType="http">
     http://gsbkip.chicagogsb.edu/testproblems/parincLinear.osil
     </instanceLocation>
    </general>
    <optimization>
        <other name="os_solver">clp</other>
    </optimization>
</osol>
```

8.3.2 The send Service Method

When the `solve` service method is used, the `OSSolverService` does not finish execution until the solution is returned from the remote solver service. The `solve` method communicates synchronously with the remote solver service. This may not be desirable for large problems when the user does not want to wait for a response. The `send` service method should be used when asynchronous communication is desired. When the send method is used the instance is communicated to the remote service and the `OSSolverService` terminates after submission. An example of this is

```
OSSolverService -config ../data/configFiles/testremoteSend.config
```

where the `testremoteSend.config` file is

```
-nl ../data/amplFiles/hs71.nl
-serviceLocation http://128.135.130.17:8080/os/OSSolverService.jws
-serviceMethod send
```

In this example the COIN-OR `Ipopt` solver is specified. The input file `hs71.nl` is in AMPL format. Before sending this to the remote solver service the `OSSolverService` executable converts the nl format into the OSiL XML format and packages this into the SOAP envelope used by Web Services.

Since the `send` method involves asynchronous communication the remote solver service must keep track of jobs. The send methd requires a `JobID`. In the above example no `JobID` was specified. When no `JobID` is specified the `OSSolverService` method first invokes the `getJobID` service method to get a `JobID` and then puts this information into a created OSoL file and send the information to the server. More information on the `getJobID` service method is provided in Section 8.3.4. The `OSSolverService` prints the OSoL file to standard output before termination. This is illustrated below,

```
<?xml version="1.0" encoding="UTF-8"?>
<osol xmlns="os.optimizationservices.org">
    <general>
        <jobID>
        gsbrkm4__127.0.0.1__2007-06-16T15.46.46.075-05.00149771253
        </jobID>
    </general>
    <optimization>
        <other name="os_solver">ipopt</other>
    </optimization>
</osol>
```

The `JobID` is one that is randomly generated by the server and passed back to the `OSSolverService.` The user can also provide a `JobID` in their OSoL file. For example, below is a user-provided OSoL file that could be specified in a configuration file or on the command line.

```
<?xml version="1.0" encoding="UTF-8"?>
<osol xmlns="os.optimizationservices.org">
    <general>
        <jobID>123456abcd</jobID>
    </general>
    <optimization>
        <other name="os_solver">ipopt</other>
    </optimization>
</osol>
```

In order to be of any use, it is necesary to get the result of the optimization. This is described in Section 8.3.3. Before proceeding to this section, we describe two ways for knowing when the optimization is complete. One feature of the standard OS remote SolverService is the ability to send an email when the job is complete. Below is an example of the OSoL that uses the email feature.

```
<?xml version="1.0" encoding="UTF-8"?>
<osol xmlns="os.optimizationservices.org">
    <general>
        <jobID>123456abcd</jobID>
        <contact transportType="smtp">
            kipp.martin@chicagogsb.edu
        </contact>
    </general>
    <optimization>
        <other name="os_solver">lindo</other>
    </optimization>
</osol>
```

The remote Solver Service will send an email to the above address when the job is complete. A second option for knowing when a job is complete is to use the knock method.

Note that in all of these examples we provided a value for the `name` attribute in the `<other>` element. The remote solver service will use Cbc if another solver is not specified.

8.3.3 The `retrieve` Service Method

The `retrieve` has a single string argument which is an OSoL instance. Here is an example of using the `retrieve` method with `OSSolverService`.

```
OSSolverService -config ../data/configFiles/testremoteRetrieve.config
```

The `testremoteRetrieve.config` file is

```
-serviceLocation http://128.135.130.17:8080/os/OSSolverService.jws
-osol ../data/osolFiles/retrieve.osol
-serviceMethod retrieve
-osrl /home/kmartin/temp/test.osrl
```

and the `retrieve.osol` file is

```
<?xml version="1.0" encoding="UTF-8"?>
<osol xmlns="os.optimizationservices.org">
    <general>
        <jobID>123456abcd</jobID>
    </general>
</osol>
```

The OSoL file `retrieve.osol` contains a tag `<jobID>` that is communicated to the remote service. The remove service locates the result returns it as a string. The string that is returned is an OSrL instance.

8.3.4 The `getJobID` Service Method

Before submitting a job with the `send` method a `JobID` is required. The `OSSolverService` can get a `JobID` as follows

```
-serviceLocation http://128.135.130.17:8080/os/OSSolverService.jws
-serviceMethod getJobID
```

Note that no OSoL input file is specified. In this case, the `OSSolverService` sends an empty string. A string is returned with the `JobID`. This `JobID` is then put into a `<jobID>` element in an OSoL string that would be used by the `send` method.

8.3.5 The `knock` Service Method

The OSSolverService terminates after executing the `send` method. Therefore, it is necessary to know when the job is completed on the remote server. One way is to include an email address in the `<contact>` element with the attribute `transportType` set to `smtp`. This was illustrated in Section 8.3.1. A second way to check on the status of a job is to use the `knock` service method. For example, assume a user wants to know if the job with `JobID 123456abcd` is complete. A user would make the request

```
OSSolverService -config ../data/configFiles/testRemoteKnock.config
```

where the `testRemoteKnock.config` file is

```
-serviceLocation http://128.135.130.17:8080/os/OSSolverService.jws
-osplInput ../data/osolFiles/demo.ospl
-osol ../data/osolFiles/retrieve.osol
-serviceMethod knock
```

the `demo.ospl` file is

```
<?xml version="1.0" encoding="UTF-8"?>
<ospl xmlns="os.optimizationservices.org">
<processHeader>
<request action="getAll"/>
</processHeader>
<processData/>
</ospl>
```

and the `retrieve.osol` file is

```
<?xml version="1.0" encoding="UTF-8"?>
<osol xmlns="os.optimizationservices.org">
  <general>
  <jobID>123456abcd</jobID>
</general>
</osol>
```

The result of this request is a string in OSrL format. Part of the return format is illustrated below.

```
<jobs>
    <job jobID="123456abcd">
        <state>finished</state>
        <serviceURI>http://128.135.130.17:8080/ipopt/IPOPTSolverService.jws</serviceURI>
        <submitTime>2007-06-16T14:57:36.678-05:00</submitTime>
        <startTime>2007-06-16T14:57:36.678-05:00</startTime>
        <endTime>2007-06-16T14:57:39.404-05:00</endTime>
        <duration>2.726</duration>
 </job>
</jobs>
```

Notice the `<state>` element indicating that the job is finished.

When making a `knock` request, the OSoL string can be empty. In this example, if the OSoL string had been empty the status of all jobs kept in the file ospl.xml is reported. In our default solver service implementation, there is a configuration file `OSParameter` that has a parameter `MAX_JOBIDS_TO_KEEP` . The current default setting is 100. In a large-scale or commercial implementation it might be wise to keep problem results and statistics in a database. Also, there are values other than `getAll` for the OSpL `action` attribute in the `<request>` tag. For example, the `action` can be set to a value of `ping` if the user just wants to check if the remote solver service is up and running.

8.3.6 The `kill` Service Method

If the user submits a job that is taking too long or is a mistake it is possible to kill the job on the remote server using the `kill` service method. For example to kill job `123456abcd` . At the command line type

```
OSSolverService -config  ../data/confgFiles/kill.config
```

where the configure file `kill.config` is

```
-osol ../data/osolFiles/kill.osol
-serviceLocation http://128.135.130.17:8080/os/OSSolverService.jws
-serviceMethod kill
```

and the `kill.osol` file is

```
<?xml version="1.0" encoding="UTF-8"?>
<osol xmlns="os.optimizationservices.org">
    <general>
        <jobID>123456abcd</jobID>
    </general>
</osol>
```

8.3.7 Summary

Below is a summary of the inputs and outputs of the six service methods. See also Figures 18 and 19.

- `solve(osil, osol)`:

  - Inputs: a string with the instance in OSiL format and a string with the solver options in OSoL format
  - Returns: a string with the solver solution in OSrL format
  - Synchronous call, blocking request/response

- `send(osil, osol)`
  - Inputs: a string with the instance in OSiL format and a string with the solver options in OSoL format
  - Returns: a boolean, true if the problem was successfully submitted, false otherwise
  - Has the same signature as `solve`
  - Asynchronous (server side), non-blocking call
  - The `osol` string should have a `JobID` in the `<jobID>` element

- `getJobID( osol)`
  - Inputs: a string with the solver options in OSoL format (in this case, the string may be empty because no options are required to get the JobID)
  - Returns: a string which is the unique job id generated by the solver service
  - Used to maintain session and state on a distributed system

- `knock(ospl, osol)`
  - Inputs: a string in OSpL format and a string with the solver options in OSoL format
  - Returns: process and job status information from the remote server in OSpL format

- `retrieve( osol)`
  - Inputs: a string with the solver options in OSoL format
  - Returns: a string with the solver solution in OSrL format
  - The `osol` string should have a `JobID` in the `<jobID>` element

- `kill( osol)`
  - Inputs: a string with the solver options in OSoL format
  - Returns: process and job status information from the remote server in OSpL format
  - Critical in long running optimization jobs

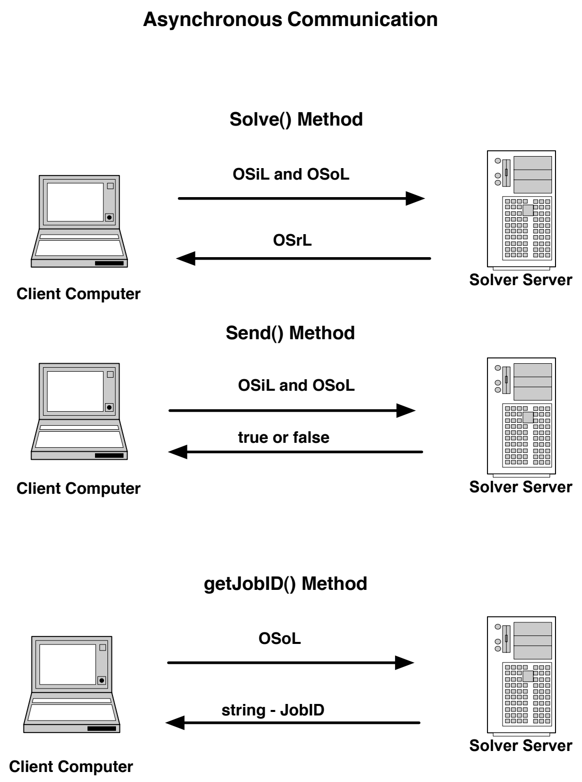

Figure 18: Input and output for `solve`, `send`, and `getJobID` methods.

9 Setting up a Solver Service with Tomcat

10 Examples

10.1 AMPL Client: Hooking AMPL to Solvers

The `amplClient` executable is designed to work with the AMPL program. See `www.ampl.com`. The `amplClient` acts like an AMPL "solver." The `amplClient` is linked with the OS library and can be used to solve problems either remotely. In both cases the `amplClient` uses the `OSnl2osil` class to convert the AMPL generated nl file (which represents the problem instance) into the corresponding instance representation in the OSiL format.

For example, assume that there is a problem instance, `hs71.mod` in AMPL model format. To solve this problem locally by calling the `amplClient` from AMPL first start AMPL and then execute the following commands. In this case we are assuming that the local solver used is `Ipopt.`

```
# take in problem 71 in Hock and Schittkowski
# assume the problem is in the AMPL directory
model hs71.mod;
# tell AMPL that the solve is amplClient
option solver amplClient;
# now tell amplClient to use Ipopt
option amplClient_options "solver ipopt";
# the name of the nl file (this is optional)
```

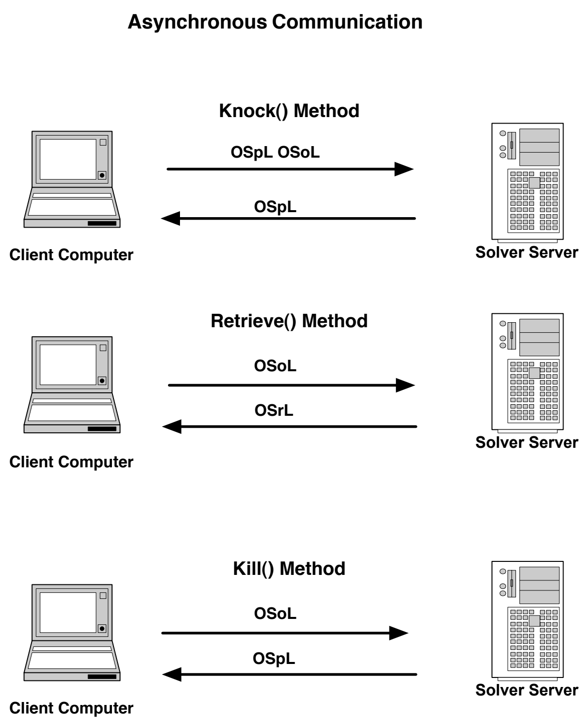

Figure 19: Input and output for `knock`, `retrieve`, and `kill` methods.

```
write gtestfile;
# now solve the problem
solve;
```

This will invoke `Ipopt` locally and the result in OSrL format will be displayed on the screen. In order to call a remote solver service, after the command

```
option amplClient_options "solver ipopt";
```

provide an option which has the address of the remote solver service.

```
option ipopt_options "http://128.135.130.17:8080/os/OSSolverService.jws";
```

10.2 CppAD: Using the CppAD Algorithmic Differentiation Package

10.3 File Upload: Using a File Upload Package

When the `OSAgent` class methods `solve` and `send` are used, the problem instance in OSiL format is packaged into a SOAP envelope and communication with the server is done using Web Services (for example Tomcat Axis). However, packing an XML file into a SOAP envelope may add considerably to the size of the file (each $<$ is replaced with `&lt;` and each $>$ is replaced with `&gt;`). Also, communicating with a Web Services servlet can also slow down the communication process. This could be a problem for large instances. An alternative approach is to use the `fileUpload` executable on the client end and the Java servlet `OSFileUpload` on the server end. The `OSFileUpload` Java servlet code is in the ******** directory. Jun — where should we put the Java server code for the File Uploader???

The `fileUpload` client executable takes one argument on the command line which is the location of the file on the local directory to upload to the server. For example,

```
fileUpload ../../data/osilFiles/parincQuadratic.osil
\end{verbartim}
The {\tt fileUpload} executable first creates an {\tt OSAgent} object.
\begin{verbatim}
OSSolverAgent* osagent = NULL;
osagent = new OSSolverAgent("http://128.135.130.17:8080/fileupload/servlet/OSFileUpload");
```

The `OSAgent` has a method `fileUpload` with the signature

```
std::string fileUpload(std::string osilFileName, std::string osil);
```

where `osilFileName` is the name of the OSiL problem instance to be written on the server and `osil` is the string with the actual instance. Then

```
osagent->fileUpload(osilFileName, osil);
```

will place a call to the server, upload the problem instance in the `osil` string, and cause the server to write a file on its hard drive named `osilFileName`.

Once the file is on the server, invoke the local `OSSolverService` by

```
OSSolverService -config ../data/configFiles/testremote.config
```

where the `config` file is

```
-osol ../data/osolFiles/remoteSolve2.osol
-serviceLocation http://128.135.130.17:8080/os/OSSolverService.jws
-serviceMethod solve
```

and the `osol` file is

```
<osol>
    <general>
         <instanceLocation locationType="local">
          /home/kmartin/temp/parincQuadratic.osil
         </instanceLocation>
    </general>
    <optimization>
     <other name="os_solver">ipopt</other>
    </optimization>
</osol>
```

10.4 Instance Generator: Using the OSInstance API to Generate Instances

This example is found in the `instanceGenerator` folder in the `examples` folder. This example illustrates how to build a complete in-memory model instance using the `OSInstance` API. See the code `instanceGenerator.cpp` for the complete example. Here provide a few highlights to illustrate the power of the API.

The first step is to create an `OSInstance` object.

```
tt
OSInstance *osinstance;
osinstance = new OSInstance();
```

Assume that the instance has two variables, x_0 and x_1. Variable x_0 is a continuous variable with lower bound of -100 and upper bound of 100. Variable x_1 is a binary variable. First declare the instance to have two variables.

```
osinstance->setVariableNumber( 2);
```

Next, add each variable. There is an `addVariable` method with the signature

```
addVariable(int index, string name, double lowerBound, double upperBound,
char type, double init, string initString);
```

Then the calls for these two variables are

```
osinstance->addVariable(0, "x0", -100, 100, 'C', OSNAN, "");
osinstance->addVariable(1, "x1", 0, 1, 'B', OSNAN, "");
```

There is also a method `setVariables` for adding more than one variable simultaneously. The objective function(s) and constraints are added through similar calls.

Nonlinear terms are also easily added. The following code illustrates how to add a nonlinear term $x_0 * x_1$ in the `<nonlinearExpressions>` section of OSiL.

```
osinstance->instanceData->nonlinearExpressions->nl[ 1] = new Nl();
osinstance->instanceData->nonlinearExpressions->nl[ 1]->idx = 1;
osinstance->instanceData->nonlinearExpressions->nl[ 1]->osExpressionTree =
new OSExpressionTree();
// create a variable nl node for x0
nlNodeVariablePoint = new OSnLNodeVariable();
nlNodeVariablePoint->idx=0;
nlNodeVec.push_back( nlNodeVariablePoint);
// create the nl node for x1
nlNodeVariablePoint = new OSnLNodeVariable();
nlNodeVariablePoint->idx=1;
nlNodeVec.push_back( nlNodeVariablePoint);
// create the nl node for *
nlNodePoint = new OSnLNodeTimes();
nlNodeVec.push_back( nlNodePoint);
// the vectors are in postfix format
// now the expression tree
osinstance->instanceData->nonlinearExpressions->nl[ 1]->osExpressionTree->m_treeRoot =
nlNodeVec[ 0]->createExpressionTreeFromPostfix( nlNodeVec);
```

11 Appendix

11.1 Building a Model in MATLAB

We illustrate how to build a simple Markowitz portfolio optimization problem (a quadratic programming problem) from **template.m**. First copy **template.m** to **markowitz.m**.

Assume that there are three stocks (variables) and two constraints (do not count the upper limit investment of .75 on the variables.).

```
% the number of constraints
numCon = 2;
% the number of variables
numVar = 3;
```

All the variables are continuous

```
VarType='CCC';
```

Next define the constraint upper and lower bounds. There are two constraints. A unity constraint (an =) and a lower bound on portfolio return of .15 (an $\geq$). These two constraints are expressed as

```
BU = [1  inf];
BL = [1   .15];
```

The variables are nonnegative and have upper limits of .75 (no stock can comprise more than 75% of the portfolio). This is written as

```
VL = [];
VU = [.75 .75 .75];
```

There are no nonzero linear coefficients in the objective function, but the objective function vector must always be defined and the number of components of this vector is the number of variables.

```
OBJ = [0 0 0 ]
```

Now the linear constraints. In the model the two linear constraints are

$$\begin{aligned} 0.3221x_1 + 0.0963x_2 + 0.1187x_3 &\geq .15 \\ x_1 + x_2 + x_3 &= 1 \end{aligned}$$

These are expressed as

```
 A = [ 1 1 1  ;
  0.3221   0.0963   0.1187 ];
```

Now for the quadratic terms. The only quadratic terms are in the objective function. The objective function is

$$\begin{aligned} \min 0.4253x_1^2 + 0.4458x_2^2 + 0.2314x_3^2 + 2 \times 0.1852x_1x_2 \\ +2 \times 0.1393x_1x_3 + 2 \times 0.1388x_2x_3 \end{aligned}$$

The quadratic matrix Q has 4 rows and a column for each quadratic term. In this example there are six quadratic terms. The first row of Q is the row index where the terms appear. By convention, the objective function has index -1 and we count constraints starting at 0. The first row of Q is

```
 -1 -1 -1 -1 -1 -1
```

The second row of Q is the index of the first variable in the quadratic term. We use zero based counting. Variable x_1 has index, variable x_2 has index 1, and variable x_3 has index 2. Therefore, the second row of Q is

```
0 1 2 0 0 1
```

The third row of Q is the index of the second variable in the quadratic term. Therefore, the third row of Q is

```
0 1 2  1 2 2
```

The last (fourth) row is the coefficient. Therefore, the fourth row is

```
 .425349654  .445784443   0.231430983
 .370437388  .27862509 .27763384
```

The quadratic matrix is

```
Q = [ -1 -1 -1 -1 -1 -1;
0 1 2 0 0 1 ;
0 1 2 1 2 2;
.425349654  .445784443   0.231430983   ...
.370437388  .27862509 .27763384];
```

Finally, name the problem, specify the solver (in this case `ipopt`), the service address (and password if required by the service), and call the solver.

```
prob_name = 'Markowitz Example from Anderson, Sweeney, Williams, and Martin'
password = 'chicagoesmuyFRIO';
%
%the solver
solverName = 'ipopt';
%the remote service service address
%if left empty we solve locally
serviceAddress='http://128.135.130.17:8080/os/OSSolverService.jws';
% now solve
callMatlabSolver( numVar, numCon, A, BL, BU, OBJ, VL, VU, ObjType, VarType, ...
     Q, prob_name, password, solverName, serviceAddress)
```

11.2 OSiL representation for problem given in (7)–(10)

```
<?xml version="1.0" encoding="UTF-8"?>
<osil xmlns="os.optimizationservices.org">
     <instanceHeader>
          <name>Modified Rosenbrock</name>
          <source>Computing Journal 3:175-184, 1960</source>
          <description>Rosenbrock problem with constraints</description>
     </instanceHeader>
     <instanceData>
          <variables numberOfVariables="2">
               <var lb="0" name="x0" type="C"/>
               <var lb="0" name="x1" type="C"/>
          </variables>
          <objectives numberOfObjectives="1">
               <obj maxOrMin="min" name="minCost" numberOfObjCoef="1">
                    <coef idx="1">9.0</coef>
               </obj>
          </objectives>
          <constraints numberOfConstraints="2">
               <con ub="25.0"/>
               <con lb="10.0"/>
          </constraints>
          <linearConstraintCoefficients numberOfValues="3">
               <start>
                    <el>0</el><el>2</el><el>3</el>
               </start>
               <rowIdx>
                    <el>0</el><el>1</el><el>1</el>
               </rowIdx>
               <value>
                    <el>1.</el><el>7.5</el><el>5.25</el>
               </value>
          </linearConstraintCoefficients>
          <quadraticCoefficients numberOfQuadraticTerms="3">
               <qTerm idx="0" idxOne="0" idxTwo="0" coef="10.5"/>
               <qTerm idx="0" idxOne="1" idxTwo="1" coef="11.7"/>
               <qTerm idx="0" idxOne="0" idxTwo="1" coef="3."/>
          </quadraticCoefficients>
          <nonlinearExpressions numberOfNonlinearExpressions="2">
               <nl idx="-1">
                    <plus>
                         <power>
                              <minus>
                                   <number type="real" value="1.0"/>
                                   <variable coef="1.0" idx="0"/>
                              </minus>
                              <number type="real" value="2.0"/>
                         </power>
                         <times>
                              <power>
                                   <minus>
                                        <variable coef="1.0" idx="0"/>
                                        <power>
                                             <variable coef="1.0" idx="1"/>
                                             <number type="real" value="2.0"/>
                                        </power>
                                   </minus>
                                   <number type="real" value="2.0"/>
                              </power>
                              <number type="real" value="100"/>
                         </times>
                    </plus>
               </nl>
               <nl idx="1">
                    <ln>
                         <times>
                              <variable coef="1.0" idx="0"/>
                              <variable coef="1.0" idx="1"/>
                         </times>
                    </ln>
               </nl>
          </nonlinearExpressions>
     </instanceData>
</osil>
```

11.3 OSiL representation for problem given in (19)–(??)

```
<?xml version="1.0" encoding="UTF-8"?>
<osil xmlns="os.optimizationservices.org" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="os.optimizationservices.org http://www.optimizationservices.org/sc
```

```
<instanceHeader>
        <description>A test problem for Algorithmic Differentiation</description>
</instanceHeader>
<instanceData>
        <variables numberOfVariables="4">
                <var lb="0" name="x0" type="C"/>
                <var lb="0" name="x1" type="C"/>
                <var lb="0" name="x2" type="C"/>
                <var lb="0" name="x3" type="C"/>
        </variables>
        <objectives numberOfObjectives=" 1">
                <obj maxOrMin="min" name="minCost" numberOfObjCoef="1">
                        <coef idx="1">9.0</coef>
                </obj>
        </objectives>
        <constraints numberOfConstraints="2">
                <con ub="10.0" constant="33"/>
                <con lb="10.0"/>
        </constraints>
        <linearConstraintCoefficients numberOfValues="2">
                <start>
                        <el>0</el>
                        <el>0</el>
                        <el>1</el>
                        <el>1</el>
                        <el>2</el>
                </start>
                <rowIdx>
                        <el>0</el>
                        <el>1</el>
                </rowIdx>
                <value>
                        <el>5</el>
                        <el>7</el>
                </value>
        </linearConstraintCoefficients>
        <nonlinearExpressions numberOfNonlinearExpressions="3">
                <nl idx="1">
                        <ln>
                                <times>
                                        <variable coef="1.0" idx="0"/>
                                        <variable coef="1.0" idx="2"/>
                                </times>
                        </ln>
                </nl>
                <nl idx="0">
                        <sum>
                                <number type="real" value="-105"/>
                                <variable coef="1.37" idx="1"/>
                                <variable coef="2" idx="2"/>
                        </sum>
                </nl>
                <nl idx="-1">
                        <power>
                                <variable coef="1.0" idx="0"/>
                                <number type="real" value="2.0"/>
                        </power>
                </nl>
        </nonlinearExpressions>
    </instanceData>
</osil>
```

References

[1] Bradley Bell. CppAD Documentation, 2007. `http://www.coin-or.org/CppAD/Doc/cppad.xml`.

[2] R. Fourer, L. Lopes, and K. Martin. LPFML: A W3C XML schema for linear and integer programming. *INFORMS Journal on Computing*, 17:139–158, 2005.

[3] Andreas Griewank. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. SIAM, Philadelphia, PA, 2000.

[4] J. Ma. Optimization services (OS), a general framework for optimization modeling systems, 2005. Ph.D. Dissertation, Department of Industrial Engineering & Management Sciences, Northwestern University, Evanston, IL.

[5] H.H. Rosenbrock. An automatic method for finding the greatest or least value of a function. *Comp. J.*, 3:175–184, 1960.