REALIZZAZIONE DI UNA CLASSE WRAPPER C# PER IL SOLVER OPEN SOURCE COIN-OR

Elaborato per il corso di Progettazione di GIS

Anno accademico 2008-2009

Realizzato da

Andrea Pierantoni

SOMMARIO

1. INTRODUZIONE

Scopo di questo progetto era la realizzazione di una classe wrapper che permettesse l'interoperabilità fra la libreria CoinMP.dll, realizzata in linguaggio C++ unmanaged, ed una qualsiasi applicazione scritta in C#.
La libreria CoinMP.dll fa parte del progetto open source Computational Infrastructure for Operations Research (COIN-OR) e fornisce la maggior parte delle funzionalità dei progetti COIN-OR LP, COIN-OR Branch-and-Cut e Cut Generation Library.
Per maggiori informazioni riguardo COIN-OR si consiglia di visitare il sito ufficiale all'indirizzo http://www.coin-or.org.
Ritornando all'elaborato qui svolto, l'obiettivo è stato raggiunto minimizzando l'utilizzo del Marshalling e quindi preferendo a questo tipo di approccio la creazione di blocchi unsafe; questo ha permesso innanzitutto di gestire in maniera esplicita i puntatori, oltreché di creare dei blocchi fixed, consentendo di evitare problemi di gestione della memoria, dovuti all'incompatibilità fra le politiche del garbage collector del framework .NET e i metodi di allocazione tradizionali.
La classe wrapper così realizzata consente inoltre la totale interoperabilità con la libreria originaria, permettendo di richiamare tutti i metodi esportati da quest'ultima.
Si passerà dunque a descrivere, nel prossimo capitolo, come includere ed utilizzare il wrapper nella propria applicazione C#.

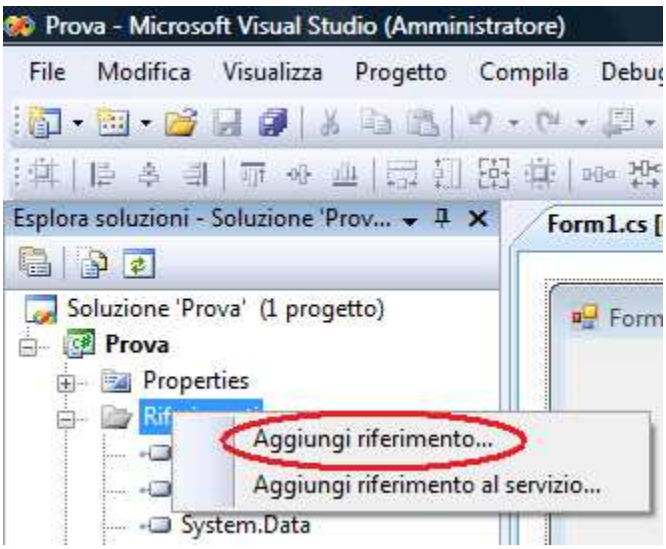
2. UTILIZZO DELLA CLASSE WRAPPER

In questo capitolo verrà descritto in primo luogo come creare un nuovo progetto che includa la libreria CoinWrapper.dll, mentre in seguito si mostrerà un esempio di codice, atto a chiarificare in che modo inizializzare e risolvere un problema di ottimizzazione con la classe Wrapper.

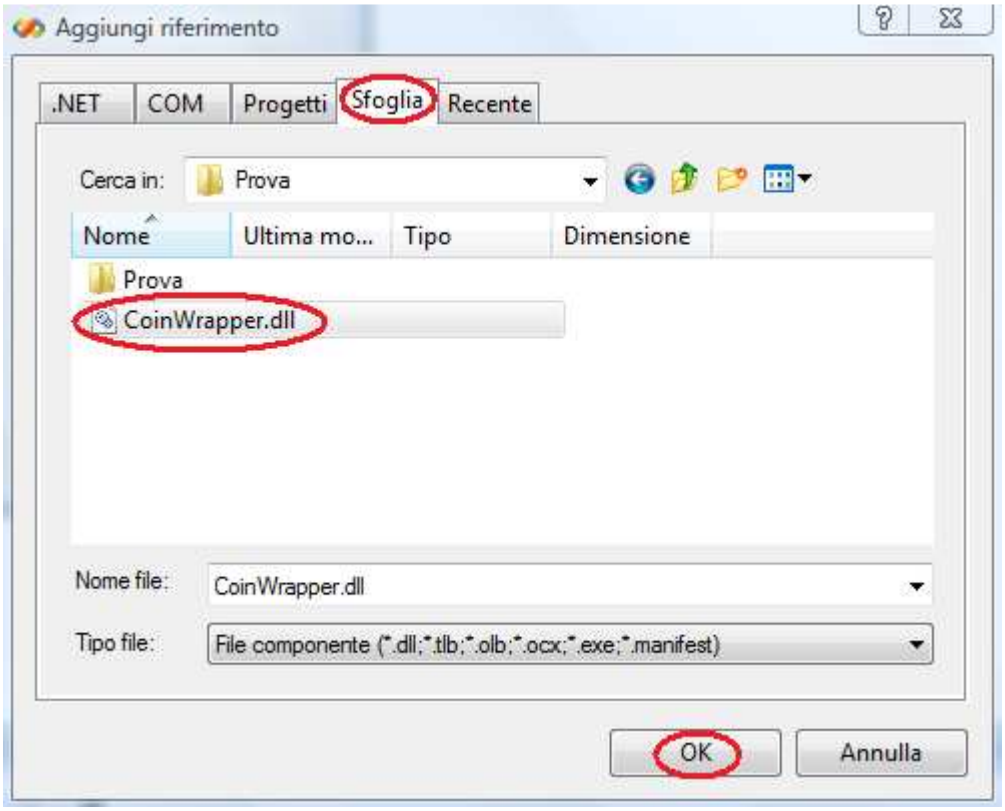
2.a. Inclusione del wrapper nella propria applicazione

Dopo aver aperto il proprio progetto con Visual Studio, per prima cosa è necessario aggiungere la libreria CoinWrapper.dll fra i riferimenti. Per fare questo occorre:

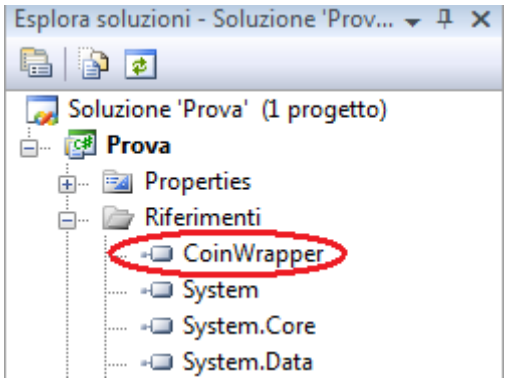
1) Cliccare con il pulsante destro del mouse su **Riferimenti** nella finestra **Esplora soluzioni** e poi scegliere nel menu **Aggiungi riferimento…**



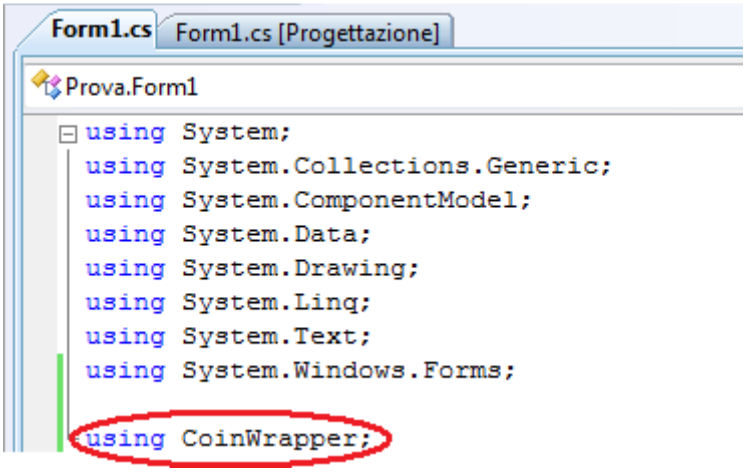
2) Nella finestra che si aprirà di seguito spostarsi nel tab **Sfoglia**, cercare il file **CoinWrapper.dll** (anche se non è strettamente necessario, si consiglia di mettere il file nella stessa cartella della soluzione), e cliccare su **OK**.



A questo punto fra i riferimenti dovrebbe essere presente **CoinWrapper**.



3) L'ultimo passo che resta da fare consiste nell'aggiungere nel file in cui s'intende utilizzare il wrapper il namespace di quest'ultimo, scrivendo nella posizione opportuna `using CoinWrapper;` (vedi figura seguente).



A questo punto, sarà possibile inizializzare nel progetto un oggetto della classe Wrapper.

2.b. Esempio di utilizzo

Le operazioni rese pubbliche dalla classe Wrapper hanno tutte il prefisso 'W_' , seguito dal nome originale del metodo importato dalla libreria CoinMP.dll, per questo motivo si rimanda alla documentazione di COIN-OR per la descrizione approfondita del funzionamento dei singoli metodi; ciò che invece s'intende chiarire in questo paragrafo, attraverso un esempio, è come inizializzare e risolvere un problema di ottimizzazione utilizzando la classe Wrapper.
A questo scopo, si consideri il problema caratterizzato dalla seguente formulazione matematica, in forma canonica:

$$\text{obj} = Max \ \sum_{i=1}^{8} x_i$$

$$\begin{aligned}
3x_1 + x_2 \quad - 2x_4 - x_5 \quad - x_8 &\leq 14 \\
2x_2 + 1.1x_3 &\leq 80 \\
x_3 \quad + x_6 &\leq 50 \\
2.8x_4 \quad - 1.2x_7 &\leq 50 \\
5.6x_1 \quad + x_5 \quad + 1.9x_8 &\leq 50
\end{aligned}$$

$$0 \leq x_i \leq 1000000 \qquad \forall i = 1,..,8$$

Per poter risolvere il dato problema è necessario innanzitutto inizializzare il numero di colonne (senza aggiungere le variabili di scarto) e di righe del tableau, che rappresenteremo con due variabili intere ***colCount*** e ***rowCount*** e che dunque, in questo caso, assumeranno rispettivamente i valori 8 e 5. Fatto questo, sarà necessario inizializzare una variabile, che chiameremo ***nonZeroCount***, nella quale memorizzeremo il numero di elementi diversi da 0 (in questo caso 14).
Ora è possibile inizializzare una stringa (***objectName***) nella quale scrivere il nome della funzione obiettivo ("obj"), e una variabile intera (***objectSense***) che specifica se il problema da risolvere è di massimo (-1, come in questo caso) o di minimo (1).
Dopo aver impostato questi valori, è possibile cominciare ad inserire i dati del problema nelle opportune strutture; serviranno dunque:

- ***objectCoeffs:*** vettore di double contenente i coefficienti della funzione obiettivo (in questo caso tutti a 1);
- ***lowerBounds:*** eventuali lower bound per le variabili (in questo caso tutti a 0);
- ***upperBounds:*** eventuali upper bound per le variabili (in questo caso tutti a 1000000), se il valore che le variabili possono assumere non è limitato superiormente è comunque necessario inizializzare questo vettore con il valore massimo rappresentabile dal tipo di dato utilizzato (nell'esempio il *double*);

- ***rowType***: una stringa che ha tanti caratteri quante sono le righe, questo perché ogni suo carattere esprime se il vincolo cui fa riferimento è di tipo "minore o uguale" ('L'), "uguale" ('E'), oppure "maggiore o uguale" ('G') (in questo caso abbiamo cinque vincoli tutti di tipo minore o uguale e quindi *rowType* sarà una stringa composta da cinque L);
- ***rhsValues***: un vettore che contiene i valori *right hand side* (nell'esempio 14, 80, 50, 50, 50);
- ***matrixBegin, matrixCount, matrixIndex, matrixValues***: questi quattro vettori sono il frutto della rappresentazione della matrice dei coefficienti compressa per colonne;
- ***colNames, rowNames:*** due vettori di stringhe che contengono rispettivamente i nomi delle colonne e delle righe del tableau.

Il codice per inizializzare le variabili finora descritte sarà dunque il seguente:

```
String problemName = "CoinTest";
int colCount = 8;
int rowCount = 5;
int nonZeroCount = 14;

String objectName = "obj";
int objectSense = -1;
double[] objectCoeffs = new double[8] { 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0};
double[] lowerBounds = new double[8] { 0, 0, 0, 0, 0, 0, 0, 0 };
double[] upperBounds = new double[8] { 1000000.0, 1000000.0, 1000000.0,
                                       1000000.0, 1000000.0, 1000000.0,
                                       1000000.0, 1000000.0 };

String rowType = "LLLLL";
double[] rhsValues = new double[5] { 14.0, 80.0, 50.0, 50.0, 50.0 };

int[] matrixBegin = new int[8 + 1] { 0, 2, 4, 6, 8, 10, 11, 12, 14 };
int[] matrixCount = new int[8] { 2, 2, 2, 2, 2, 1, 1, 2 };
int[] matrixIndex = new int[14] { 0, 4, 0, 1, 1, 2, 0, 3, 0, 4, 2, 3, 0, 4 };
double[] matrixValues = new double[14] { 3.0, 5.6, 1.0, 2.0, 1.1, 1.0, -2.0,
                                         2.8, -1.0, 1.0, 1.0, -1.2, -1.0, 1.9};

String[] colNames = new String[8] { "c1", "c2", "c3", "c4", "c5", "c6", "c7",
                                    "c8" };
String[] rowNames = new String[5] { "r1", "r2", "r3", "r4", "r5" };
```

Giunti a questo punto, è possibile richiamare, secondo le proprie necessità, i metodi implementati nella classe wrapper, oppure, richiamare la funzione W_RunTestProblem che permette di risolvere un problema di programmazione lineare (sia intera che non) ottenendo come risultato il puntatore alla struttura che contiene i dati del problema appena risolto.

Vale dunque la pena dare un'occhiata all'intestazione del metodo W_RunTestProblem, per meglio comprendere come richiamarlo:

```
public IntPtr W_RunTestProblem
(String problemName, double optimalValue, int colCount, int rowCount, int
nonZeroCount, int rangeCount, int objectSense, double objectConst, double[]
objectCoeffs, double[] lowerBounds, double[] upperBounds, String rowType,
double[] rhsValues, double[] rangeValues, int[] matrixBegin, int[]
matrixCount, int[] matrixIndex, double[] matrixValues, String[] colNames,
String[] rowNames, String objectName, double[] initValues, String columnType,
int LoadNamesType, StreamWriter logFile, bool writeMPS).
```

La stragrande maggioranza dei parametri di input di W_RunTestProblem sono quelli visti precedentemente; fra gli altri, gli unici degni di nota sono lo StreamWriter, che, se diverso da null, permette di salvare l'output della soluzione in un file di log, il booleano writeMPS che, se settato a *true*, permette di scrivere un file .MPS che contiene i dati del problema, e optimalValue che consente di passare il valore della soluzione ottima del problema – se già nota – per verificare se i dati sono stati inizializzati correttamente.
Quando un parametro richiesto da W_RunTestProblem non è necessario per rappresentare il problema che si sta tentando di risolvere, è sufficiente passare:

- **""**, se il parametro è una stringa;
- **0**, se il parametro è un int;
- **0.0**, se il parametro è un double;
- **null**, se il parametro è un array.

Si ricorda, inoltre, che è sempre necessario, per un corretto funzionamento, che le due librerie CoinMP.dll e CoinWrapper.dll si trovino nella stessa cartella della propria applicazione.

Infine, si faccia attenzione alla modalità di compilazione, in quanto, se si sta realizzando un'applicazione per un'architettura a 64 bit, sarà necessario ricompilare a 64 bit sia CoinMp che CoinWrapper, e si dovrà dunque includere nel proprio progetto la libreria così ottenuta.

APPENDICE A

Descrizione dei metodi esportati dalla classe wrapper

- `public int` **`W_CoinInitSolver`**`(String LicenseString)`
  inizializza il solver;

- `public int` **`W_CoinFreeSolver`**`()`
  funzione per liberare la memoria del risolutore dopo l'esecuzione;

- `public int` **`W_CoinGetSolverName`**`(out String SolverName, int buflen)`
  inserisce nella stringa, passata come riferimento, il nome del solver;

- `public int` **`W_CoinGetVersionStr`**`(out String VersionStr, int buflen)`
  inserisce la versione del solver nella stringa passata come riferimento;

- `public double` **`W_CoinGetVersion`**`()`
  ritorna la versione del solver come double;

- `public int` **`W_CoinGetFeatures`**`()`
  restituisce le funzionalità implementate dalla libreria;

- `public int` **`W_CoinGetMethods`**`()`
  restituisce i metodi risolutivi implementati;

- `public double` **`W_CoinGetInfinity`**`()`
  ritorna un double tendente a +infinito;

- `public IntPtr` **`W_CoinCreateProblem`**`(String ProblemName)`
  alloca la memoria della struttura dati che contiene il problema e ne ritorna il puntatore;

- `public int` **`W_CoinLoadProblem(`**`IntPtr hProb, int ColCount,`
  `int RowCount, int NZCount, int RangeCount, int ObjectSense,`
  `double ObjectConst, double[] ObjectCoeffs,`
  `double[] LowerBounds, double[] UpperBounds, String RowType,`
  `double[] RHSValues, double[] RangeValues, int[] MatrixBegin,`
  `int[] MatrixCount, int[] MatrixIndex, double[] MatrixValues,`
  `String []ColNames, String []RowNames,String ObjectName)`
  inserisce nella struttura opportuna i dati del problema di programmazione lineare, passati in input;

- `public int` **`W_CoinLoadInitValues`**`(IntPtr hProb, double[] InitValues)`
  inserisce nella struttura del problema il vettore degli initValues passato in input;

- `public int` **`W_CoinLoadInteger`**`(IntPtr hProb, String ColType)`
  carica nella struttura del problema il tipo dei vincoli interi passati in input con la stringa ColType;

- `public int` **`W_CoinLoadPriority(`**`IntPtr hProb, int PriorCount,`
  `int[] PriorIndex, int[] PriorValues, int[] BranchDir)`
  carica, per i problemi che lo richiedono, i dati relativi alla priorità;

- `public int` **`W_CoinLoadSos`**`(IntPtr hProb, int SosCount, int SosNZCount,`
  `int[] SosType, int[] SosPrior, int[] SosBegin, int[] SosIndex,`
  `double[] SosRef)`
  carica, per i problemi che lo richiedono, i dati Sos;

- `public int` **`W_CoinLoadQuadratic`**`(IntPtr hProb, int[] QuadBegin,`
  `int[]QuadCount, int[] QuadIndex, double[] QuadValues)`
  carica, per i problemi che lo richiedono, i dati quadratic ;

- `public int` **`W_CoinLoadNonLinear`**`(IntPtr hProb, int NlpTreeCount,`
  `int qNlpLineCount, int[] NlpBegin, int[] NlpOper,`
  `int[] NlpArg1, int[] NlpArg2, int[] NlpIndex1, int[] NlpIndex2,`
  `double[] NlpValue1, double[] NlpValue2)`
  carica nella struttura opportuna i dati del problema di programmazione non lineare, passati in input;

- `public int` **`W_CoinUnloadProblem`**`(IntPtr hProb)`
  libera la memoria occupata dalla struttura del problema;

- `public int` **`W_CoinCheckProblem`**`(IntPtr hProb)`
  verifica che i dati del problema siano stati inizializzati coerentemente;

- `public int` **`W_CoinSetLoadNamesType`**`(IntPtr hProb, int LoadNamesType)`
  …;

- `public int` **`W_CoinGetProblemName`**`(IntPtr hProb, out String ProbName,`
  `int buflen)`
  inserisce il nome del problema nella stringa passata come riferimento;

- `public int` **`W_CoinGetColCount`**`(IntPtr hProb)`
  ritorna il numero delle variabili colonna;

- `public int` **`W_CoinGetRowCount`**`(IntPtr hProb)`
  ritorna il numero delle variabili riga;

- `public int` **`W_CoinGetColName(`**`IntPtr hProb, int col, out String ColName, int buflen)`
  inserisce il nome della colonna che si trova all'indice dato, nella stringa passata come riferimento;

- `public int` **`W_CoinGetRowName`**`(IntPtr hProb, int row, out String RowName, int buflen)`
  inserisce il nome della riga che si trova all'indice dato, nella stringa passata come riferimento;

- `public int` **`W_CoinOptimizeProblem`**`(IntPtr hProb, int Method)`
  risolve il problema;

- `public int` **`W_CoinGetSolutionStatus`**`(IntPtr hProb)`
  ritorna il valore intero che descrive lo stato della soluzione;

- `public int` **`W_CoinGetSolutionText`**`(IntPtr hProb, int SolutionStatus, out String SolutionText, int buflen)`
  inserisce nella stringa passata come riferimento il testo che descrive lo stato della soluzione (es.: "Optimal solution found");

- `public double` **`W_CoinGetObjectValue`**`(IntPtr hProb)`
  ritorna il valore della soluzione;

- `public double` **`W_CoinGetMipBestBound`**`(IntPtr hProb)`
  …;

- `public int` **`W_CoinGetIterCount`**`(IntPtr hProb)`
  ritorna il contatore dell'iterazione corrente;

- `public int` **`W_CoinGetMipNodeCount`**`(IntPtr hProb)`
  …;

- `public int` **`W_CoinGetSolutionValues`**`(IntPtr hProb, double[] Activity, double[] ReducedCost, double[] SlackValues, double[] ShadowPrice)`
  inserisce i valori assunti dalle variabili, quando è stata trovata la soluzione ottima, nei vettori passati in input;

- `public int` **`W_CoinGetSolutionRanges`**`(IntPtr hProb, double[] ObjLoRange, double[] ObjUpRange, double[] RhsLoRange, double[]RhsUpRange)`
  …;

- `public int **W_CoinGetSolutionBasis**(IntPtr hProb, int[] ColStatus, double[] RowStatus)`
  …;

- `public int **W_CoinReadFile**(IntPtr hProb, int FileType, String ReadFilename)`
  carica un problema da file MPS;

- `public int W_CoinWriteFile(IntPtr hProb, int FileType, String WriteFilename)`
  scrive i dati del problema in un file MPS;

- `public int **W_CoinOpenLogFile**(IntPtr hProb, String logFilename)`
  non ancora implementato in CoinMP;

- `public int **W_CoinCloseLogFile**(IntPtr hProb)`
  non ancora implementato in CoinMP;

- `public int **W_CoinGetOptionCount**(IntPtr hProb)`
  ritorna il numero di opzioni disponibili;

- `public int W_CoinGetOptionInfo(IntPtr hProb, int OptionNr, out int OptionID, out int GroupType, out int OptionType, out String OptionName, out String ShortName, int buflen)`
  inserisce nei parametri passati in input come riferimento le informazioni relative ad un'opzione specificata;

- `public int **W_CoinGetIntOptionMinMax**(IntPtr hProb, int OptionNr, out int MinValue, out int MaxValue)`
  inserisce nei parametri passati in input come riferimento il valore minimo e quello massimo (interi) che può assumere una determinata opzione;

- `public int **W_CoinGetRealOptionMinMax**(IntPtr hProb, int OptionNr, out double MinValue, out double MaxValue)`
  inserisce nei parametri passati in input come riferimento il valore minimo e quello massimo (real) che può assumere una determinata opzione;

- `public int **W_CoinGetOptionChanged**(IntPtr hProb, int OptionID)`
  restituisce un intero che esplica se il valore dell'opzione è cambiato;

- `public int **W_CoinGetIntOption**(IntPtr hProb, int OptionID)`
  restituisce il valore di una data opzione solo se è di tipo intero;

- `public int` **`W_CoinSetIntOption`**`(IntPtr hProb, int OptionID, int IntValue)`
  cambia il valore di una data opzione con l'intero passato in input;

- `public double` **`W_CoinGetRealOption`**`(IntPtr hProb, int OptionID)`
  restituisce il valore di una data opzione solo se è di tipo double;

- `public int` **`W_CoinSetRealOption`**`(IntPtr hProb, int OptionID, double RealValue)`
  cambia il valore di una data opzione con il double passato in input;

- `public int` **`W_CoinGetStringOption`**`(IntPtr hProb, int OptionID, out String StringValue, int buflen)`
  non ancora implementata in CoinMP;

- `public int` **`W_CoinSetStringOption`**`(IntPtr hProb, int OptionID, String StringValue)`
  non ancora implementata in CoinMP;

- `public IntPtr` **`W_RunTestProblem`**`(String problemName, double optimalValue, int colCount, int rowCount, int nonZeroCount, int rangeCount, int objectSense, double objectConst, double[] objectCoeffs, double[] lowerBounds, double[] upperBounds, String rowType, double[] rhsValues, double[] rangeValues, int[] matrixBegin, int[] matrixCount, int[] matrixIndex, double[] matrixValues, String[] colNames, String[] rowNames, String objectName, double[] initValues, String columnType, int LoadNamesType, StreamWriter logFile, bool writeMPS)`
  metodo originariamente non presente in CoinMP. Fa tutte le chiamate necessarie per la risoluzione di un dato problema di programmazione lineare;

- `public IntPtr` **`W_RunSosTestProblem`**`(String problemName, double optimalValue, int colCount, int rowCount, int nonZeroCount, int rangeCount, int objectSense, double objectConst, double[] objectCoeffs, double[] lowerBounds, double[] upperBounds, String rowType, double[] rhsValues, double[] rangeValues, int[] matrixBegin, int[] matrixCount, int[] matrixIndex, double[] matrixValues, String[] colNames, String[] rowNames, String objectName, double[] initValues, String columnType, int LoadNamesType, int sosCount, int sosNZCount, int[] sosType, int[] sosPrior, int[] sosBegin, int[] sosIndex, double[] sosRef, StreamWriter logFile, bool writeMPS)`
  come W_RunTestProblem, ma per problemi con Sos.