Títol: *Un entorn de modelització i optimització per a problemes de gran escala amb estructura angular per blocs*

Autor: *Xavier Jiménez Marimón*

Data: *17 de Desembre de 2012*

Director: *Jordi Castro Pérez*

Departament del director: *EIO*

Titulació: *Enginyeria en Informàtica*

Centre: *Facultat d'Informàtica de Barcelona (FIB)*

Universitat: *Universitat Politècnica de Catalunya (UPC) BarcelonaTech*

DADES DEL PROJECTE

Títol del Projecte: *Un entorn de modelització i optimització per a problemes de gran escala amb estructura angular per blocs*

Nom de l'estudiant: *Xavier Jiménez Marimón*

Titulació: *Enginyeria en Informàtica*

Crèdits: *37.5*

Director/Ponent: *Jordi Castro Pérez*

Departament: *EIO*

MEMBRES DEL TRIBUNAL *(nom i signatura)*

President: *Francisco Javier Heredia Cervera*

Vocal: *Rubén Tous Liesa*

Secretari: *Jordi Castro Pérez*

QUALIFICACIÓ

Qualificació numèrica:

Qualificació descriptiva:

Data:

Índex

1. Introducció

El principal objectiu d'aquest projecte és fer més senzill, de cara a l'usuari, el procés de modelització i resolució de problemes de gran escala amb estructura angular per blocs, oferint una plataforma senzilla, fàcil d'utilitzar i amb uns requeriments d'aprenentatge molt baixos degut a que l'entrada de dades de l'usuari es basa en adaptacions de formats àmpliament coneguts i utilitzats en l'àmbit de l'optimització matemàtica.

Per resoldre els problemes es poden utilitzar diferents solvers. En aquest treball únicament el solver BlockIP [11] treu profit de l'estructura per blocs que tenen els problemes d'entrada. I és al voltant d'aquest solver amb les seves característiques i necessitats especifiques on ha girat tot el treball a realitzar. De fet, amb la resta de solvers el problema d'entrada es tracta com si fos un problema sense estructura per blocs.

A banda de l'usuari final que vol resoldre problemes reals, aquesta plataforma també està pensada per poder realitzar benchmarks de BlockIP en comparació amb altres solvers, d'aquí que s'hagi buscat la facilitat de poder resoldre el mateix problema amb diferents solvers sense que l'usuari hagi de fer cap esforç addicional ni modificar l'entrada de dades segons el solver.

En concret s'ha implementat una plataforma que pot ser utilitzada de tres maneres diferents depenent de les necessitats i coneixements de l'usuari. Existeix una interfície gràfica, un programa per línia de comandes i una llibreria en C++.

Degut a que en Enginyeria en Informàtica no necessàriament s'aprèn que és la programació matemàtica ni que és un solver, en la secció "2. Programació matemàtica", s'expliquen aquests conceptes per tal de fer la lectura d'aquest projecte més entenedora.

El propòsit d'aquesta memòria no és entrar en el detall de com s'ha fet cadascuna de les parts, sinó que s'ha optat per una lectura més clara i entenedora. Entrar en el detall de cada apartat suposaria estendre aquest document molt considerablement i es perdria una visió global del treball realitzat.

Per posar un exemple, quan es tracta la comunicació entre SML i AMPL a través de fitxers .nl. Per entendre el treball que comporta fer l'extensió realitzada, s'hauria d'explicar quina és l'estructura d'aquest fitxer, de quines formes concretes es pot obrir i que permet cadascuna, quines són les funcions que es poden utilitzar, quines estructures de dades retorna AMPL i com s'han de tractar aquestes estructures de dades.

El mateix passa amb la comunicació entre Osi i els solvers, i molts altres aspectes de la plataforma. Per tant, per una visió més detallada del treball realitzat, el millor és veure les referències indicades per entendre el funcionament de les tecnologies externes i directament el codi font de la plataforma per veure els detalls d'implementació.

1.1 Objectius

Tal i com es diu al principi de la introducció, l'objectiu principal és crear una plataforma per poder resoldre problemes amb unes característiques determinades. Per aconseguir-ho es van crear els següents objectius implicats:

- Aprendre que és un problema amb estructura angular per blocs així com conceptes bàsics de modelització, optimització i eines principals per resoldre els problemes.
- Cercar les millors maneres d'entrar dades còmodament i funcionalitats rellevants per a l'usuari final.
- Implementar les entrades de dades i les funcionalitats, incloent-les en la mesura del possible dintre de la llibreria del propi solver BlockIP.
- Implementar opcions addicionals com per exemple connectar l'entrada de dades a altres solvers.
- Unir el treball realitzat en una llibreria prou abstracta, intuïtiva i potent de cara a l'usuari final.
- Creació de l'aplicació per línia de comandes.
- Creació de la interfície gràfica.
- Test de tota la plataforma.

Aquests objectius han sigut assolits i al llarg de la memòria s'explica com ha sigut possible resoldre els problemes trobats i quin ha sigut tant el procediment seguit com el resultat final.

1.2 Entorn de treball

L'entorn principal de treball ha sigut un servidor amb openSUSE 11.4 [18] i el software desenvolupat s'ha escrit amb C++ [12] sense utilitzar cap IDE.

En la creació del software s'ha tingut en compte que fos multiplataforma, així s'ha elegit codi fet per terceres parts que també ho fos.

Per a les versions en Windows s'ha utilitzat l'IDE Microsoft Visual Studio 2010 [17] ja que he trobat que era el més compatible amb les versions Windows del software de terceres parts.

Més endavant s'explica amb més detalls quins són aquests softwares de terceres parts així com altres tecnologies utilitzades.

1.3 Resum dels continguts

La resta del document es divideix en les següents seccions:

- Programació matemàtica: En aquesta secció s'explica que és la programació matemàtica, quins tipus de problemes existeixen, quin tipus de problemes es poden resoldre amb la plataforma creada i de quina manera s'han manipulat aquests problemes abans de poder-los resoldre.
- Tecnologies utilitzades: Part del treball realitzat ha sigut unir diferents peces ja creades, en aquesta secció s'explica quines són aquestes peces i com en alguns casos s'han modificat per a que fessin el que es necessitava.
- BlockIP Platform: En aquesta secció es descriu tota la plataforma creada, tant el seu funcionament extern com alguns detalls interns i d'implementació.
- Demostració: Conté alguns exemples d'ús de les aplicacions.
- Planificació i costos: Es presenta la planificació del projecte detallada així com els costos relacionats.

- Tasques futures: Tot i que s'han assolit els objectius proposats la plataforma no està finalitzada, aquí s'explica quins són els plans futurs.
- Conclusions
- Bibliografia

2. Programació matemàtica

Quan em van oferir fer el projecte final de carrera al meu lloc de treball jo no coneixia que és la optimització o programació matemàtica. De la mateixa manera que jo no ho sabia, és possible que el lector d'aquesta memòria tampoc estigui familiaritzat en aquest àmbit. En aquest cas la lectura d'aquesta secció és fonamental per poder entendre que s'ha fet en aquest projecte. En cas contrari aquesta secció és fàcilment prescindible i l'única part interesant podria ser la conversió que he necessitat fer del problema d'entrada a forma estàndard.

Gran part de la informació aquí explicada està basada en les "Linear Programming FAQ" [14] i "Nonlinear Programming FAQ" [19] de la "NEOS Optimization Guide" [1], així com les transparències de l'assignatura de lliure elecció "Modelització en Programació Matemàtica". El que aquí s'explica només són unes pinzellades bàsiques; per a informació més concreta i precisa les referències anteriors poden ser un bon punt de partida.

2.1 Què és la programació matemàtica?

La programació matemàtica tracta de donar resposta a un tipus general de problemes matemàtics en els que es vol elegir el millor entre un conjunt d'elements. Així doncs són problemes on es vol maximitzar o minimitzar una funció objectiu, per exemple maximitzar els beneficis o minimitzar el temps d'execució.

Formalment, i en la forma més simple equival a resoldre una equació d'aquest tipus:

$$(P)\begin{cases} \min\limits_{x\in\mathbb{R}^n} & f(x) & f\colon \mathbb{R}^n \to \mathbb{R} \\ \text{subjecte a} & h(x) = 0 & h\colon \mathbb{R}^n \to \mathbb{R}^m \\ & g(x) \leq 0 & g\colon \mathbb{R}^n \to \mathbb{R}^p \\ & x \in X & \end{cases}$$

On $x = (x_1, \dots, x_n)^T$ és un vector i representa les variables de decisió, $f(x)$ és la funció objectiu i representa la qualitat de les decisions preses, $h(x) = \left(h_1(x), \dots, h_m(x)\right)^T = 0$ són les restriccions d'igualtat, $g(x) = \left(g_1(x), \dots, g_p(x)\right)^T \leq 0$ són les restriccions de

desigualtat i $x \in X$ és el domini de les variables. El conjunt de decisions que satisfan les restriccions del problema és conegut com el conjunt factible $\Omega = \{x \in X \mid h(x) = 0, g(x) \leq 0\}$. Llavors les restriccions ens diuen que no totes les possibles decisions són bones i ens acoten el problema.

D'aquí en endavant només parlarem de minimitzar ja que $\min f(x) \equiv -\max -f(x)$.

2.2 Programació lineal

El tipus de problemes més senzills dintre de l'àmbit de l'optimització són els problemes lineals, que en forma estàndard poden ser representat com el següent:

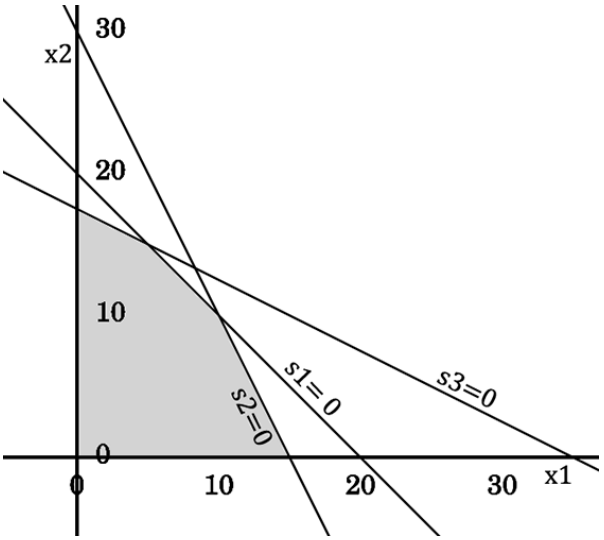
$$\begin{aligned} &\text{minimitzar} && c^T x \\ &\text{subjecte a} && Ax = b \\ & && x \geq 0 \end{aligned}$$

On $c \in \mathbb{R}^n$ i $b \in \mathbb{R}^m$ són vectors de coeficients coneguts i $A \in \mathbb{R}^{m \times n}$ és una matriu amb normalment més columnes que files, és a dir, hi ha més variables que restriccions deixant així moltes possibilitats en l'elecció de x.

Visualment podem representar un problema lineal situant cada variable com un eix de coordenades, on el conjunt factible és l'espai contingut entre els talls marcats per les restriccions. Per exemple, el problema:

$$\begin{aligned} &\text{minimitzar} && -4x_1 - 5x_2 \\ &\text{subjecte a} && x_1 + x_2 + s_1 = 20 \\ & && 2x_1 + x_2 + s_2 = 30 \\ & && x_1 + 2x_2 + s_3 = 35 \\ & && x_1, x_2, s_1, s_2, s_3 \geq 0 \end{aligned}$$

té la següent representació gràfica en $\mathbb{R}^2$:



La regió de solucions factibles és l'àrea ombrejada i la solució òptima es troba a un dels vèrtex del poliedre.

Existeixen dues famílies de tècniques per solucionar aquest tipus de problemes. Les dues visiten progressivament diferents solucions millorant l'actual fins que es troba una que satisfà les condicions per a ser una solució òptima. Els mètodes *Simplex* van recorrent els diferents vèrtex del poliedre fixant variables a un valor límit. En canvi els mètodes de *punt interior* van recorrent els punts per l'interior de la regió factible. BlockIP és un algorisme d'aquest últim tipus.

Tot i que el problema presentat en forma estàndard només té restriccions d'igualtat i variables més grans o iguals que zero, podem convertir qualsevol problema lineal a forma estàndard realitzant una sèrie de transformacions que més endavant veurem. De fet normalment els solvers accepten problemes de la forma:

$$\begin{aligned} &\text{minimitzar} \quad c^T x \\ &\text{subjecte a} \quad b_1 \leq Ax \leq b_2 \\ &\qquad\qquad\quad l \leq x \leq u \end{aligned}$$

On l i u són vectors coneguts com a *lower* i *upper bounds*, els elements dels quals poden prendre valors de $+\infty$ o $-\infty$. De la mateixa manera també es permet qualsevol valor per b_1 i b_2.

Un exemple de problema lineal podria ser planificar la producció volent maximitzar els beneficis tenint en compte els recursos dels que disposem i la demanda que tindrem.

Altre exemple és el fer una dieta el menys costosa possible tenint en compte que ens ha d'aportar tots els nutrients necessaris.

2.2.1 Programació lineal entera

Dintre dels problemes lineals ens trobem uns en particular on les variables han de prendre un valor enter, aquests problemes són més realistes i aquestes variables enteres es poden interpretar com prendre una decisió entre un conjunt finit de decisions. Formalment es poden representar com:

$$\begin{aligned} &\text{minimitzar} \quad c^T x \\ &\text{subjecte a} \quad Ax = b \\ &\qquad\qquad\quad x \geq 0, x \in \mathbb{Z}^n \end{aligned}$$

La dificultat d'aquests tipus de problemes és que són molt més durs per a trobar la solució òptima ja que si en els anteriors la solució òptima es trobava a un vèrtex del poliedre, en aquests no té perquè ser així i el nombre de solucions que opten a ser òptim creix exponencialment.

A més l'ús de variables enteres ens dóna més recursos per modelitzar els problemes, com per exemple la creació de condicionals. Un exemple típic de problema enter és el de la motxilla, en el que és vol omplir una motxilla amb diferents objectes maximitzant la utilitat sense superar el volum de la motxilla.

BlockIP no tracta variables enteres, per tant no són utilitzades ni tractades en cap part del projecte.

2.2.2 Models estructurats

Hi ha ocasions en les que un model té una estructura determinada on el problema es podria dividir en petits subproblemes i només una petita part de les restriccions acoblen aquets subproblemes en un.

Exemples de problemes d'aquests tipus són problemes de transport multiarticle, en els que s'ha de decidir com transportar productes diferents d'uns centres d'origen a altres centres de destí, cada producte té les seves característiques pròpies, però existeixen unes restriccions que acoblen els subproblemes entre ells en el moment que es vol transportar els productes junts al mateix medi de transport.

Més formalment es pot representar com:

$$\begin{aligned} \text{minimitzar} \quad & \sum_{i=1}^{k} {c^i}^T x^i \\ \text{subjecte a} \quad & A^i x^i = b^i \quad \text{i} = 1 \dots \text{k} \\ & \sum_{i=1}^{k} B^i x^i = \text{u} \\ & x^i \geq 0 \end{aligned}$$

BlockIP és un solver específic per solucionar aquest tipus de problemes i més endavant veurem amb més detall aquest tipus de problemes.

2.3 Programació no lineal

Els problemes no lineals tenen funcions no lineals en la definició de la funció objectiu i/o restriccions:

$$\begin{aligned} \min \quad & f(x) \\ \text{subjecte a} \quad & h_i(x) = 0 \quad i = 1, \dots, m \\ & g_j(x) \leq 0 \quad j = 1, \dots, p \end{aligned}$$

L'estudi del problemes no lineals és un àrea difícil, així que històricament la recerca s'ha centrat en casos d'estudi particulars. Un cas ben estudiat, que inclou BlockIP, és el cas on totes les restriccions g i h són lineals. També existeixen els anomenats problemes quadràtics, els quals es caracteritzen perquè la seva funció objectiu només està formada per coeficients lineals o quadràtics i les restriccions són lineals.

En programació no lineal també existeixen problemes sense restriccions. Un dels grans inconvenients dels problemes no lineals és que hi ha molts òptims locals i això dificulta la mobilitat de l'algorisme d'un punt a un altre.

Un exemple de problema no lineal és el de construir un cilindre metàl·lic de volum màxim de manera que la seva superfície sigui una concreta.

2.4 Solvers

Quan parlem de *solver* ens referim a una peça de software matemàtic que resol algun tipus de problema d'optimització matemàtica. Un solver pot existir com una aplicació en si mateixa o bé com una llibreria.

Degut a que els problemes que resolen estan molt presents a la nostra vida hi ha un gran negoci darrere d'aquests solvers, on inclús empreses tan potents com IBM comercialitzen un. Alguns dels solver comercials millors i més utilitzats són Cplex [9] i Xpress [5]. També existeixen bons solvers de lliure ús, però en general el temps que triguen en resoldre un problema és molt més llarg; alguns d'aquests solvers són CLP[13], GLPK [8] o SYMPHONY [21].

2.4.1 Llenguatges de modelització

Entrar dades a un solver sol ser una operació farragosa, i codificar un problema real en les matrius i vectors que utilitzen els solvers com a dades d'entrada encara més. Per aquet motiu es van crear els llenguatges de modelització, els quals faciliten aquesta tasca a l'usuari, que no té perquè saber res de programació.

Per una banda serveixen per prototipar models de forma ràpida i comprovar si el pas del problema real al model ha sigut correcte i per altra creen una capa d'abstracció entre el model i el solver. També ens permeten separar un model de les dades concretes, d'aquesta manera no hem de tornar a refer tot el model només perquè una dada hagi canviat.

Hi ha diversos llenguatges de modelització, alguns solvers tenen el seu propi, però els més utilitzats són AMPL [20] i GAMS [3]. Una gran part d'aquest projecte es basa en una adaptació d'AMPL, així que més endavant veurem tant el format com les modificacions concretes que s'han fet.

2.4.2 MPS

MPS [9] és un format de dades creat per IBM que es va convertir en un format de dades estàndard el qual pràcticament tots els solvers accepten com entrada de dades. Permet codificar en codi ASCII (llegible per l'ésser humà, no binari) problemes lineals i

problemes lineals enters. També hi ha extensions que permeten l'ús de funcions objectius quadràtiques.

L'altra entrada de dades de la plataforma creada (la primera és una adaptació del format AMPL) és una adaptació del format MPS, així que més endavant veurem amb més detall la seva sintaxi i les modificacions realitzades.

2.5 Quins tipus de problema soluciona BlockIP?

El solver al voltant del qual s'ha desenvolupat el projecte és BlockIP, tot i que altres solvers es poden utilitzar des de la plataforma. Així, s'ha mirat de fer una entrada de dades que pugui explotar tot el potencial del solver. Més endavant s'explica com s'ha aconseguit això, però de moment anem a veure quin tipus de problema pot solucionar.

BlockIP resol problemes amb una estructura angular per blocs, tals que:

$$\begin{aligned} &\text{minimitzar} \quad f(x,s) \\ &\text{subjecte a} \quad \begin{bmatrix} N^1 & & \\ & \ddots & \\ & & N^k \end{bmatrix} \begin{bmatrix} x^1 \\ \vdots \\ x^k \end{bmatrix} = \begin{bmatrix} b^1 \\ \vdots \\ b^k \end{bmatrix} \\ &\qquad\qquad L^1x^1 + \ldots + L^kx^k + Is = b^{k+1} \\ &\qquad\qquad 0 \le x \le u, 0 \le s \le u_s \end{aligned}$$

- f és una funció no lineal amb la peculiaritat de que la seva Hessiana ha de ser diagonal i no negativa, és a dir no pot haver producte de dues variables diferents. Evidentment f també poden ser coeficients lineals ($f(x) = c^T x$) o una funció quadràtica ($f(x) = \frac{1}{2} x^T Q x - b^T x$) sempre i quan respecti la norma anterior. BlockIP necessita $f(x)$, gradient $\nabla f(x)$ i Hessiana $\nabla^2 f(x)$
- $x = \left(x^{1^T}, \ldots, x^{k^T}\right)^T$ són les variables del problema que es divideixen per blocs, dos blocs diferents no poden compartir variables. Cada bloc $x^i = \left(x_1^i, \ldots, x_{n_i}^i\right)$ té dimensió n_i. La dimensió total de x és $n = \sum_{i=1}^{k} n_i$.
- $s \in \mathbb{R}^l$ són els *slacks* o folgues associats a cada *linking contraint*, els quals poden prendre cost a la funció objectiu.

- $b = \left(b^{1^T}, \ldots, b^{k^T}, b^{k+1^T}\right)^T$ és el vector de límits a la restriccions. Cada bloc $b^i = \left(b_1^i, \ldots, b_{m_i}^{\ i}\right), i\epsilon(1, \ldots, k)$ té dimensió m_i i $b^{k+1} \in \mathbb{R}^l$. La dimensió total de b és m= $\sum_{i=1}^{k} m_i + l$
- $u \in \mathbb{R}^n$ és el vector de límits superiors a les variables.
- $u_s \in \mathbb{R}^l$ és el vector de límits superiors de les folgues.
- $N^i \in \mathbb{R}^{m_i \times n_i}$ són els blocs de coeficients que formen les restriccions.
- $L^i \in \mathbb{R}^{l \times n_i}$ són els blocs de coeficients que formen les *linking contraints* o restriccions d'acoblament.
- $I \in \mathbb{R}^{l \times l}$ és una matriu identitat.

2.5.1 Conversió a forma estàndard

BlockIP internament només és capaç de solucionar problemes en la forma anteriorment descrita, tot i això per a comoditat de l'usuari es permet donar el problema amb la màxima llibertat. De forma que el problema d'entrada pot tenir la forma:

$$\text{minimitzar} \quad f(x,s)$$

$$\text{subjecte a} \quad \begin{bmatrix} b_1^1 \\ \vdots \\ b_1^k \end{bmatrix} \leq \begin{bmatrix} N^1 & & \\ & \ddots & \\ & & N^k \end{bmatrix} \begin{bmatrix} x^1 \\ \vdots \\ x^k \end{bmatrix} \leq \begin{bmatrix} b_2^1 \\ \vdots \\ b_2^k \end{bmatrix}$$

$$L^1 x^1 + \ldots + L^k x^k + Is = b_2^{k+1}$$

$$l \leq x \leq u, 0 \leq s \leq u_s$$

El canvis respecte a la representació anterior és que ara les restriccions no han de ser d'igualtat sinó que poden ser desigualtats amb límit inferior i superior, de la mateixa manera ara les variables poden prendre també valors negatius.

En el cas de que no es vulgui donar pes als slacks a la funció objectiu no és necessari que apareguin i d'aquesta manera les linking constraints es poden expressar també com a desigualtats:

$$\text{minimitzar} \quad f(x)$$

$$\text{subjecte a} \quad \begin{bmatrix} b_1^1 \\ \vdots \\ b_1^k \end{bmatrix} \leq \begin{bmatrix} N^1 & & \\ & \ddots & \\ & & N^k \end{bmatrix} \begin{bmatrix} x^1 \\ \vdots \\ x^k \end{bmatrix} \leq \begin{bmatrix} b_2^1 \\ \vdots \\ b_2^k \end{bmatrix}$$

$$b_1^{k+1} \leq L^1x^1 + \ldots + L^kx^k \leq b_2^{k+1}$$

$$l \leq x \leq u$$

Una de les tasques a realitzar al projecte ha sigut la conversió del problema general a forma estàndard, les transformacions que s'apliquen son les següents:

- **Restricció de desigualtat amb límit superior**

Quan ens trobem amb una restricció tal que $b_1 \leq a^T x \leq b_2,\ b2\ \neq \infty$, el que es fa és crear una nova variable *slack* que permeti el moviment entre els dos límits i així aconseguir una igualtat, és a dir, passem a tenir una restricció tal que $a^T x + s = b_2,\ 0 \leq s \leq b_2 - b_1$.

Aquesta nova variable creada no té impacte a la funció objectiu i per tant no cal tenir cap més consideració.

- **Restricció de desigualtat sense límit superior**

Quan ens trobem amb una restricció tal que $b_1 \leq a^T x$, el que es fa és crear una nova variable *surplus* que permeti el moviment fins al límit aconseguint una igualtat, és a dir, passem a tenir una restricció tal que $a^T x - s = b_1,\ s \geq 0$

Aquesta nova variable creada no té impacte a la funció objectiu i per tant no cal tenir cap més consideració.

- **Variable amb límit inferior diferent de zero**

Quan ens trobem una variable tal que $l_i \leq x_i \leq u_i,\ l_i \neq -\infty$, el que es fa és un canvi de variable tal que $\tilde{x_i} = x_i - l_i,\ 0 \leq \tilde{x_i} \leq u_i - l_i$, llavors on apareixia x_i, s'ha d'intercanviar per $\tilde{x_i} + l_i$.

Estenent aquesta transformació als vectors als quals pertanyen les components abans anomenades i tenint en compte que $x = (x_1, \ldots, x_i, \ldots, x_n)^T$ i e_i és la i-èssima columna de la matriu identitat, és a dir $e_i = (0, \ldots, 0, 1, 0, \ldots, 0)^T$, amb l'1 a la posició i, tenim que $\tilde{x} = x - l_i e_i = (x_1, \ldots, x_i - l_i, \ldots, x_n)^T$, per tant $x = \tilde{x} + l_i e_i$; i que a tota restric-

ció tal que $a^T x = b_j$ on $a \in \mathbb{R}^n$ i $b = \left(b_1, \dots, b_j, \dots, b_m\right)^T$, ens quedarà $a^T(\tilde{x} + l_i e_i) = b_j$ o el que és el mateix $a^T \tilde{x} = b_j - a^T l_i e_i$, per tant hem de modificar el límit de les restriccions.

La funció objectiu serà $\tilde{f}(\tilde{x}) = f(\tilde{x} + l_i e_i) = f(x)$. I calcularem $\frac{\partial \tilde{f}(\tilde{x})}{\partial \tilde{x_i}} = \frac{\partial f(x)}{\partial \tilde{x_i}} =$

$\frac{\partial f(x)}{\partial x_i} \cdot \frac{\partial x_i}{\partial \tilde{x_i}} = \frac{\partial f(x)}{\partial x_i} \cdot 1 = \frac{\partial f(x)}{\partial x_i}$. De forma similar $\frac{\partial^2 \tilde{f}(\tilde{x})}{\partial \tilde{x_i}} = \frac{\partial^2 f(x)}{\partial x_i}$.

- **Variable sense límit inferior**

Quan ens trobem una variable tal que $x_i \leq u_i,\ u_i \neq -\infty$, el que es fa és un canvi de variable tal que $\tilde{x_i} = -x_i + u_i,\ \tilde{x_i} \geq 0$, llavors on apareixia x_i, s'ha d'intercanviar per $-\tilde{x_i} + u_i$.

Estenent aquesta transformació als vectors als quals pertanyen les components abans anomenades i tenint en compte que T és una matriu identitat de dimensió n amb la posició i canviada de signe, és a dir $T = diag(1, \dots, 1, -1, 1, \dots, 1)$, amb el -1 a la posició i, tenim que $\tilde{x} = Tx + u_i e_i = (x_1, \dots, -x_i + u_i, \dots, x_n)^T$, per tant $x = T\tilde{x} + u_i e_i$; i que a tota restricció tal que $a^T x = b_j$, ens quedarà $a^T(T\tilde{x} + u_i e_i) = b_j$ o el que és el mateix $a^T T\tilde{x} = b_j - a^T u_i e_i$, per tant hem de modificar el límit de les restriccions i de signe el coeficient que multiplica a $\tilde{x_i}$.

La funció objectiu serà $\tilde{f}(\tilde{x}) = f(T\tilde{x} + u_i e_i) = f(x)$. I calcularem $\frac{\partial \tilde{f}(\tilde{x})}{\partial \tilde{x_i}} = \frac{\partial f(x)}{\partial \tilde{x_i}} =$ $\frac{\partial f(x)}{\partial x_i} \cdot \frac{\partial x_i}{\partial \tilde{x_i}} = \frac{\partial f(x)}{\partial x_i} \cdot -1 = \frac{-\partial f(x)}{\partial x_i}$. I també $\frac{\partial^2 \tilde{f}(\tilde{x})}{\partial \tilde{x_i}} = \frac{\partial}{\partial \tilde{x_i}}\left(\frac{\partial \tilde{f}(\tilde{x})}{\partial \tilde{x_i}}\right) = \frac{\partial}{\partial \tilde{x_i}}\left(\frac{-\partial f(x)}{\partial x_i}\right) =$

$\frac{-\partial^2 f(x)}{\partial x_i \partial x_i} \cdot \frac{\partial x_i}{\partial \tilde{x_i}} = \frac{-\partial^2 f(x)}{\partial x_i \partial x_i} \cdot -1 = \frac{\partial^2 f(x)}{\partial x_i}$.

- **Variable lliure**

Quan ens trobem amb una variable lliure, és a dir, que no te límits, $-\infty \leq x_i \leq +\infty$, ens podem trobar dos situacions, si el valor corresponent a aquesta variable a la diagonal de la matriu Hessiana és diferent de 0, llavors el solver pot *marcar-la* com a variable lliure i no cal fer cap transformació.

En el cas de que aquest valor a la Hessiana sigui 0 desdoblem la variable en dos variables tals que $x_i = x_i^+ - x_i^-$, $x_i^+ \geq 0$, $x_i^- \geq 0$, llavors on apareixia x_i, s'ha d'intercanviar per $x_i^+ - x_i^-$.

Estenent aquesta transformació al vector al qual pertany la component abans anomenada tenim que $x = (x_1, \dots, x_i^+ - x_i^-, \dots, x_n)^T$, x $\in \mathbb{R}^n$ i $\tilde{x} = (x_1, \dots, x_i^+, -x_i^-, \dots, x_n)^T$, $\tilde{x} \in \mathbb{R}^{n+1}$; i que tota restricció tal que $a^T x = b_j$, ens quedarà $\sum_{l=1}^{n} a_l x_l = (\sum_{l \neq i} a_l x_l) + a_i x_i = \sum_{l \neq i} a_l x_l + a_i x_i^+ - a_i x_i^- = b_j$, o el que és el mateix $\tilde{a}\tilde{x} = b_j$, on $\tilde{a} = (a_1, \dots, a_i, -a_i, \dots, a_n)$, $\tilde{a} \in \mathbb{R}^{n+1}$, per tant s'ha d'incloure una nova variable amb el mateix coeficient però canviant el signe.

La funció objectiu serà $\tilde{f}(\tilde{x}) = f(x)$ on $x_i = x_i^+ - x_i^-$. I calcularem:

- $\frac{\partial \tilde{f}(\tilde{x})}{\partial x_l \; l \neq i} = \frac{\partial f(x)}{\partial x_l}$.
- $\frac{\partial \tilde{f}(\tilde{x})}{\partial x_i^+} = \frac{\partial f(x)}{\partial x_i^+} = \frac{\partial f(x)}{\partial x_i} \cdot \frac{dx_i}{dx_i^+} = \frac{\partial f(x)}{\partial x_i} \cdot 1 = \frac{\partial f(x)}{\partial x_i}$.
- $\frac{\partial \tilde{f}(\tilde{x})}{\partial x_i^-} = \frac{\partial f(x)}{\partial x_i^-} = \frac{\partial f(x)}{\partial x_i} \cdot \frac{dx_i}{dx_i^-} = \frac{\partial f(x)}{\partial x_i} \cdot -1 = \frac{-\partial f(x)}{\partial x_i}$.

La Hessiana serà 0 ja que era el requisit per a desdoblar la variable.

3. Tecnologies utilitzades

Per a la realització d'aquest projecte he mirat d'utilitzar les eines més adients per a cada tasca, intentant no crear codi ja existent a altres llibreries de lliure ús i intentant que fos el més compatible possible en qualsevol sistema operatiu. A continuació es detalla que s'ha utilitzat i de quina manera.

3.1 C++

El llenguatge de programació utilitzat per a tota la plataforma ha sigut C++, el motiu és que els solvers són programes numèrics que necessiten realitzar càlculs molt intensivament i una gran quantitat de memòria per poder resoldre els problemes, la interacció amb l'usuari és escassa i la major part del temps és consumit de forma interna sense cap entrada i sortida de dades cap a l'exterior.

Degut a això la gran majoria de solvers estan implementats en C o C++ ja que és el llenguatge més eficient per a realitzar aquestes tasques. A més l'ús de C++ permet dues coses molt importants. Per una banda, un control a molt petit detall del que realitza internament el processador, de manera que és pot programar molt eficientment tasques que es realitzaran moltíssimes vegades durant l'execució. I per un altra banda, permet crear una interfície molt abstracta de cara a l'usuari, qui només veu una caixa negra que fa el que li demana de forma molt senzilla.

Per tant, per coherència i facilitat d'ús s'ha optat per implementar la plataforma en C++. Tot i que la plataforma només és un embolcall dels solvers que fa més senzilla l'entrada i sortida de dades; i el temps d'execució que necessita és insignificant comparat amb el que es triga en solucionar un problema.

3.2 Doxygen

Doxygen [7] és un generador de documentació que funciona amb diferents llenguatges de programació. Permet generar el manual de referència, inclús el manual d'usuari, d'una llibreria o aplicació a partir dels comentaris fets al codi.

Per fer-ho els comentaris han d'estar en un format específic utilitzant algunes paraules clau i símbols concrets [7], un cop apressos és molt ràpid afegir-los mentre que fem el codi i posem els comentaris. D'aquesta manera podem donar informació rellevant que tenim present en el moment de fer el codi, però que es podria oblidar si tinguéssim que fer la documentació a posteriori. Un exemple de com es pot documentar una funció amb Doxygen és el següent (els caràcters i paraules en negreta formen part del llenguatge de Doxygen):

```
int suma(int x, int y)
//! Realitza una suma i retorna el resultat
/*!
    @param x Enter a sumar
    @param y Enter a sumar
    @return x+y
*/
```

A més permet que el manual de referència s'actualitzi dinàmicament a la vegada que actualitzem el codi. Per tots aquests motius s'ha utilitzat Doxygen per a la creació del manual de referència de les diferents llibreries i aplicacions.

3.3 BlockIP

BlockIP [11] és un solver creat per Jordi Castro que permet la resolució de problemes de gran escala amb estructura angular per blocs, funcions objectius no lineals i restriccions lineals.

Actualment estic col·laborant en la implementació de BlockIP però malauradament encara no està finalitzada (desembre de 2012) la versió en C++ (només està disponible la versió Matlab) i per tant actualment només s'utilitza la llibreria del solver per realitzar algunes tasques que més endavant veurem. Tot i això, segurament quan estigui finalitzat serà el solver més ràpid per resoldre problemes del tipus anteriorment descrit

3.4 Osi

Osi [4] és una llibreria creada dins del projecte COIN-OR (*COmputational INfrastructure for Operations Research*), que és una comunitat que creen software *open source* per investigació operativa i en el seus inicis va ser promoguda per IBM. La finalitat d'Osi és

donar una classe base abstracta i genèrica que permeti la comunicació amb virtualment qualsevol solver.

Per a cada nou solver afegit es crea una classe derivada de la genèrica que implementa les operacions que es pugin fer amb aquest solver. Osi pot fer d'interfície per a una gran quantitat de solvers diferents, entre ells estan softwares comercials com Cplex i Xpress i també solvers de lliure ús com Glpk, Clp, Symphony, Gurobi, etc.

Llavors es té un accés comú per a les operacions genèriques, però també es té accés directe al propi entorn del solver per tal de poder ajustar paràmetres més específics de cada solver.

La idea en si és molt bona ja que en principi pots utilitzar el mateix codi per cridar a diferents solvers. En la pràctica, i després de trobar-m'ho, hi ha funcions genèriques que es comporten de forma diferent segons el solver, que no funcionen bé o inclús que no estan implementades. Degut a això per a la gran majoria d'accions s'ha d'utilitzar el propi entorn del solver. La meva conclusió és que per evitar maldecaps i errors que no saps d'on provenen el millor és utilitzar Osi tan sols per passar les dades del problema i qualsevol altre paràmetre que s'hagi de donar fer-ho a través del propi solver.

D'aquesta manera, i tenint present que volia que la plataforma fos modular, és a dir, que si una persona mai vol utilitzar altres solvers que no siguin BlockIP no té perquè tenir el codi compilat per aquesta part, vaig crear una llibreria que serveix com una capa d'abstracció a Osi i que en certa manera facilita algunes tasques a l'usuari final. Més endavant veurem amb més detall aquesta llibreria.

3.5 AMPL

AMPL [20], un acrònim de "*A Mathematical Programming Language*", és un llenguatge de modelització algebraic per a problemes d'optimització lineals i no lineals amb variables enteres o continues. Va ser desenvolupat per Robert Fourer, David Gay i Brian Kernighan a Bell Laboratories. Segons les estadístiques dels servidors de NEOS, AMPL és el format més popular per representar problemes d'optimització matemàtica.

AMPL està enllaçat a multitud de solvers i a través d'AMPL es poden resoldre els problemes als solvers i després recuperar informació sobre el resultat de l'optimització.

Una gran avantatge d'AMPL és que la seva sintaxi és molt semblant a la notació matemàtica utilitzada als problemes d'optimització. Això permet una definició molt concisa i llegible dels problemes.

El format d'entrada d'AMPL es basa en el model, les dades i eventualment una seqüencia d'instruccions, tot i que AMPL permet donar aquesta informació barrejada el més habitual i correcte és dividir-ho en tres tipus de fitxers:

- Fitxers amb extensió ".mod" contenen la definició del model del problema. Aquí no es donen dades concretes sinó que es declaren conjunts i paràmetres que seran assignats en algun moment.
- Fitxers amb extensió ".dat" contenen la assignació dels conjunts i paràmetres, declarats en el model, a dades concretes.
- Fitxers amb extensió ".run" contenen una seqüència de comandes per realitzar certes tasques com per exemple carregar el model i les dades, resoldre un problema i obtenir informació. Pot ser utilitzat per resoldre conjunts de problemes mitjançant iteracions per exemple.

Un exemple extret del llibre d'AMPL és el següent, el fitxer ".mod":

```
set P;

param a {j in P};
param b;
param c {j in P};
param u {j in P};

var X {j in P};

maximize Total_Profit: sum {j in P} c[j] * X[j];

subject to Time: sum {j in P} (1/a[j]) * X[j] <= b;

subject to Limit {j in P}: 0 <= X[j] <= u[j];
```

I el fitxer ".dat":

```
data;

set P := bands coils;

param:     a     c     u  :=
  bands   200   25   6000
  coils   140   30   4000 ;

param b := 40;
```

3.5.1 Enllaçant un solver

AMPL permet que qualsevol persona pugui enllaçar AMPL a un solver; aquest procés és anomenat *hooking* en l'ambit d'AMPL i està extensament descrit a la documentació [6].

Bàsicament la comunicació entre AMPL i un solver es realitza mitjançant fitxers. AMPL escriu fitxers amb extensió ".nl" que conten tota la informació relacionada al problema. Aquest fitxer és llegit pel solver, qui resol i problema i després escriu un altre fitxer amb extensió ".sol" que és llegit per AMPL tancant el cicle. A partir d'aquest moment AMPL torna a tenir el control i pot donar informació a l'usuari sobre com ha acabat l'optimització.

Existeixen unes rutines implementades en C que tenen com a propòsit ser utilitzades des del solver per obtenir la informació sobre els problemes que es troba continguda als fitxers .nl; també poden ser utilitzades per escriure el resultat cap a AMPL.

Els fitxers .nl són de vital importància per al projecte ja que són la base a partir de la qual es pot extreure tota la informació sobre el problema quan utilitzem l'entrada de dades basada en AMPL (però modificada com veurem més endavant).

A més el fitxers .nl contenen tota la informació necessària per a poder realitzar càlculs amb funcions objectiu no lineals. En concret permeten calcular la funció objectiu en un punt i obtenir tant el gradient com la matriu Hessiana.

Aquesta funcionalitat ens permet utilitzar funcions objectiu no lineals utilitzant el format AMPL com a entrada de dades del problema, i el que és encara més impressionant des de el punt de vista d'un programador, poder utilitzar diferents funcions objectius no lineals sense haver de tornar a compilar el programa.

Aquests fitxers .nl permeten ser oberts de diferents maneres segons la quantitat d'informació que es vol extreure i de quina manera es vol rebre. Així per exemple s'obren amb unes opcions si només volem les primeres derivades o amb unes altres si volem a més la matriu Hessiana.

Els fitxers .nl s'utilitzen de forma indirecta al projecte a través de SML, però he tingut que tractar amb ells directament per afegir funcionalitats a SML que no existien. En concret, funcionalitats per obtenir informació relacionada amb funcions quadràtiques i no lineals.

També vaig haver de modificar el codi font d'AMPL per tal d'evitar un error d'incompatibilitats amb diferents formes d'obrir el fitxer .nl utilitzant problemes amb funcions objectiu no lineals. Més endavant s'explica quines són les funcions creades i s'entra en detalls d'implementació.

L'ús dels fitxers .nl en aquest projecte no és el convencional ja que normalment l'aplicació principal és AMPL cridant a un solver i aquest retorna informació a AMPL. En aquest projecte s'utilitza AMPL com un conversor d'un problema escrit en format AMPL a dades en C, en el següent apartat es veu amb més detall.

3.6 SML

SML [16] (Structured Modelling Language) és un llenguatge de modelització, creat a la Universitat de Edimburg pel Edinburgh Research Group in Optmization, que permet encapsular l'estructura dels problemes i accedir a les dades a través de C++. Per la part de modelització utilitza la sintaxi d'AMPL i l'estén amb unes paraules clau que tenen com a finalitat definir l'estructura.

D'aquesta manera dintre d'un model es poden crear submodels on cadascun d'aquests submodels representa un o més blocs. Així es facilita molt la tasca de crear un model amb estructura per blocs podent explotar l'estructura; fent-ho directament amb AMPL seria una tasca molt complexa sinó impossible, ja que es tindria que analitzar tota l'entrada de dades per separar els blocs.

L'entrada de dades d'SML és un fitxer ".mod" i altre ".dat" del mateix tipus que hem vist abans però utilitzant l'extensió del llenguatge, un exemple de model amb SML és el següent:

```
set ORIG; # origins
set DEST; # destinations
set PROD; # products

param supply {PROD,ORIG} >= 0; # amounts available at origins
param demand {PROD,DEST} >= 0; # amounts required at destinations

param limit {ORIG,DEST} >= 0;
param cost {PROD,ORIG,DEST} >= 0; # shipment costs per unit

block Prod{p in PROD}:
  var Trans {ORIG,DEST} >= 0; # units to be shipped
  minimize Total_cost:
    sum {i in ORIG, j in DEST}
      cost[p,i,j] * Trans[i,j];

  subject to Supply {i in ORIG}:
    sum {j in DEST} Trans[i,j] = supply[p,i];

  subject to Demand {j in DEST}:
    sum {i in ORIG} Trans[i,j] = demand[p,j];
end block;

subject to Multi {i in ORIG, j in DEST}:
  sum {p in PROD} Prod[p].Trans[i,j] <= limit[i,j];
```

Aquest model és una adaptació del problema multiarticle que té AMPL com exemple, per comparar, l'original és:

```
set ORIG;   # origins
set DEST;   # destinations
set PROD;   # products

param supply {ORIG,PROD} >= 0;  # amounts available at origins
param demand {DEST,PROD} >= 0;  # amounts required at destinations
   check {p in PROD}:
      sum {i in ORIG} supply[i,p] = sum {j in DEST} demand[j,p];

param limit {ORIG,DEST} >= 0;

param cost {ORIG,DEST,PROD} >= 0;  # shipment costs per unit
var Trans {ORIG,DEST,PROD} >= 0;   # units to be shipped

minimize Total_Cost:
   sum {i in ORIG, j in DEST, p in PROD}
      cost[i,j,p] * Trans[i,j,p];

subject to Supply {i in ORIG, p in PROD}:
   sum {j in DEST} Trans[i,j,p] = supply[i,p];

subject to Demand {j in DEST, p in PROD}:
   sum {i in ORIG} Trans[i,j,p] = demand[j,p];
```

Com podem veure la diferència més gran és que s'utilitza la paraula clau *block* i dins del bloc és defineixen les variables, la funció objectiu i les restriccions locals al bloc.

Fora del bloc es defineixen els conjunts, paràmetres i les linking contraints.

A més podem observar com el *check* ha desaparegut amb la versió estructurada, això és perquè SML no permet utilitzar algunes expressions d'AMPL.

El fitxer ".dat" seria el mateix pels dos models si l'ordre de les components dels paràmetres fos el mateix pels dos models, ja que s'ha canviat en el model per blocs per facilitar la lectura del model. És a dir, en aquests dos models concrets el fitxer de dades és el mateix però intercanviant l'ordre de la component *PROD*.

El funcionament d'SML, un cop que té els dos fitxers, és analitzar el model i crear tants fitxers .mod com blocs hi hagi. Aquests fitxers contenen només el model per aquell bloc; també crea un altre fitxer .mod arrel que conté la visió global del problema. Aquests fitxers són escrits amb la sintaxi d'AMPL sense utilitzar les extensions de SML. D'aquesta manera AMPL pot tractar aquests fitxers de forma individual.

Després SML utilitza AMPL per analitzar els fitxers creats i convertir-los a fitxers .nl. Normalment aquests fitxers s'utilitzarien per connectar directament amb un solver cridat per AMPL mitjançant un hook, però en aquest cas qui té el control és SML i un cop que AMPL ha generat els fitxers ja només queda extreure la informació allà continguda i mitjançant la lògica de SML crear els blocs en C++. Llavors qualsevol usuari pot extreure informació de cada bloc cridant a SML directament.

3.6.1 Ampliant SML

Originalment SML només era capaç de tractar amb problemes lineals i no permetia ni problemes quadràtics ni no lineals. Com hem vist abans BlockIP també tracta problemes on la funció objectiu pot ser no lineal, llavors vaig haver de modificar la llibreria de SML afegint funcions que permetessin a un usuari extreure tant la informació sobre les funcions objectius quadràtiques com sobre les funcions objectius no lineals.

La major part d'aquestes funcions es basen en crides a la llibreria d'AMPL per connectar-se als fitxers .nl. Per fer això el major problema que em vaig trobar va ser haver d'aprendre tant el funcionament intern de SML com de la llibreria d'AMPL.

A més per poder extreure la matriu de coeficients quadràtics a la funció objectiu d'un bloc era necessari obrir el fitxer .nl d'una manera determinada que era incompatible amb la existència de funcions objectius no lineals a la resta de blocs. Bàsicament AMPL el que feia era avortar si s'intentava obrir un fitxer .nl que contenia variables no lineals utilitzant un mètode d'apertura determinat. De manera que vaig haver de modificar la llibreria d'AMPL eliminant les comprovacions de la existència de variables no lineals per tal de que no donés un error. I finalment va funcionar.

SML utilitza un model de tres capes per accedir a les dades de l'.nl:

- Una classe interfície que és la que crida l'usuari i que té tots els mètodes definits com a virtuals sense implementar.
- Una classe controlador que deriva de la classe interfície i implementa els mètodes, però sense cridar directament a AMPL.
- Una classe per obtenir les dades, és la classe que crida a AMPL i la que fa realment la feina.

Aquestes classes fan més coses però les descrites només són referides a l'accés de l'.nl.

A continuació especifico les funcions afegides a la interfície SML que s'han escrit seguint l'estructura de tres capes de SML:

```
//! Return the number of local nonlinear objective functions
int getNLocalNonlinearObj();

//! Return the number of local variables appearing nonlinearly in
    objective functions
int getNLocalVarsNonlinearObj();

//! Return the number of local variables appearing nonlinearly in
    constraints
int getNLocalVarsNonlinearCons();

//! Return the number of nonzeros in the Quadratic Coeficients Matrix,
    return 0 if linear, return -1 if nonlinear
int getQuadraticObjNz();
```

```
//! Get the quadratic coeficients matrix for the local variables in
    this model
/*!
  The arrays must have the correct dimension before the calling
  @param[out] quadCoefsColsBeg
              Index to the columns beginning
  @param[out] quadCoefsRows
              Index to the rows
  @param[out] quadCoefsRows
              Quadratic coeficients
  @param[in] freeTemporalMatrix
              If false the matrix will be cached for future calls
  @return Number of nonzeros in the Quadratic Coeficients Matrix,
          0 if linear, return -1 if nonlinear
*/
int getQuadraticObj(int quadCoefsColsBeg[], int quadCoefsRows[],
    double quadCoefsVals[], bool freeTemporalMatrix=false);
//! Evaluate the objective function, gradient and Hessian matrix in a
    point
/*!
  The arrays must have the correct dimension before the calling
  @param[int] x
              Point where evaluate the objective function
  @param[out] fx
              Objective function value in x
  @param[out] Gx
              Gradient in x
  @param[out] Hx
              Hessian matrix in x
  @param[in]  diagHessian
              If true only the diagonal of the Hessian matrix is
              returned, if false the dense upper triangle of the
              Hessian stored by columns is returned
*/
void getNonlinearObj(double x[], double &fx, double Gx[], double Hx[],
     bool diagHessian=true);
```

3.7 MPS

Com hem vist a la secció 2, MPS és un format d'arxiu que permet codificar problemes lineals; i tot i que existeix un estàndard de com és un fitxer MPS s'han anat creant diferents extensions en funció de les necessitats del solver. Per exemple, per Cplex es va crear una extensió per poder representar funcions objectius quadràtiques, ara aquesta extensió és utilitzada per més solvers.

De la mateixa manera que es va fer amb les funcions quadràtiques, per BlockIP és necessari poder expressar blocs, com fins ara no es podia fer, vaig crear una extensió del format MPS per poder-ho fer.

Un dels requisits que volia complir amb aquesta extensió era que el mateix fitxer MPS pogués ser llegit tant per BlockIP com per altres solvers que no entenen de blocs, és a

dir que si es vol es pot aprofitar l'estructura per blocs, però manté retro compatibilitat amb altres solvers. Això permet poder utilitzar el mateix fitxer per diferents solvers.

Donat que BlockIP permet donar un cost als slacks de les linking constraints també volia que això es pogués representar.

I per últim, al realitzar la transformació d'un problema d'entrada a forma estàndard podia passar que la funció objectiu ens quedés modificada amb una constant, això també tenia que ser representable.

La solució per aconseguir tots el propòsits va ser la més senzilla, utilitzar el nom de variables i restriccions per representar-ho. És a dir dintre del nom de les variables i restriccions es té que afegir el nom del bloc, per fer-ho només cal definir un caràcter separador que ens delimiti on acaba el nom del bloc i on comença el nom de la variable.

El caràcter escollit va ser els dos punts ":", la elecció del caràcter tot i ser arbitraria es va comprovar que no causava problemes en lectors de MPS ja existents i a més queda bé estèticament. Per exemple, si tenim una variable amb nom "Var1" que està continguda al "Block2" podem codificar-la dintre del MPS com a "Block2:Var1". Aquesta nomenclatura a més no és ambigua ni confusa.

Així doncs ja tenim identificades les variables i restriccions contingudes a blocs, el que ens queda són les que estan fora dels blocs, que són les linking contraints i els slacks. Les linking contraints són identificades per ser restriccions que no pertanyen a cap bloc, sintàcticament les podem identificar per tenir un nom el qual no conté el caràcter ":".

I el mateix fem per als slacks, han de tenir un nom que no contingui ":". Ara bé, els slacks són opcionals, poden aparèixer o no al MPS, però si ho fan ha d'haver un per a cada linking constraint, i s'identifica que un slack pertany a una linking constraint per que apareix en aquesta, i a cap més, amb coeficient 1.

Per representar la constant abans anomenada es fa una petita trampa, es crea una variable fixada a 1, y que apareix a la funció objectiu amb el valor constant que es vol representar. Per identificar aquesta variable s'utilitza el nom especial "Constant".

Per últim, com BlockIP pot treballar amb funcions quadràtiques, s'utilitza l'extensió creada per Cplex per fer-ho, i com a BlockIP no hi pot haver interacció entre dues variables s'utilitza l'extensió QOBJ, que només escriu la meitat superior de la matriu de coeficients quadràtics.

Un exemple de MPS utilitzant l'extensió creada és el següent (el model representat és el mateix que l'ensenyat anteriorment, també s'han utilitzat les dades del propi exemple d'AMPL, però per qüestions d'espai s'han reduït, així que no es garanteix la factibilitat d'aquest problema):

```
NAME multi
ROWS
 N obj
 E Prod_bands:Supply['bands','GARY']
 E Prod_bands:Supply['bands','CLEV']
 E Prod_bands:Demand['bands','FRA']
 E Prod_coils:Supply['coils','GARY']
 E Prod_coils:Supply['coils','CLEV']
 E Prod_coils:Demand['coils','FRA']
 E Multi['GARY','FRA']
 E Multi['CLEV','FRA']
COLUMNS
 Prod_bands:Trans['bands','GARY','FRA'] obj 30
      Prod_bands:Supply['bands','GARY'] 1
 Prod_bands:Trans['bands','GARY','FRA']
      Prod_bands:Demand['bands','FRA'] 1 Multi['GARY','FRA'] 1
 Prod_bands:Trans['bands','CLEV','FRA'] obj 22
      Prod_bands:Supply['bands','CLEV'] 1
 Prod_bands:Trans['bands','CLEV','FRA']
      Prod_bands:Demand['bands','FRA'] 1 Multi['CLEV','FRA'] 1
 Prod_coils:Trans['coils','GARY','FRA'] obj 39
      Prod_coils:Supply['coils','GARY'] 1
 Prod_coils:Trans['coils','GARY','FRA']
      Prod_coils:Demand['coils','FRA'] 1 Multi['GARY','FRA'] 1
 Prod_coils:Trans['coils','CLEV','FRA'] obj 27
      Prod_coils:Supply['coils','CLEV'] 1
 Prod_coils:Trans['coils','CLEV','FRA']
      Prod_coils:Demand['coils','FRA'] 1 Multi['CLEV','FRA'] 1
 Slacks[1] Multi['GARY','FRA'] 1
 Slacks[2] Multi['CLEV','FRA'] 1
RHS
 rhs Prod_bands:Supply['bands','GARY'] 300
      Prod_bands:Supply['bands','CLEV'] -0
 rhs Prod_bands:Demand['bands','FRA'] 300
      Prod_coils:Supply['coils','GARY'] 800
 rhs Prod_coils:Supply['coils','CLEV'] 450
      Prod_coils:Demand['coils','FRA'] 1250
 rhs Multi['GARY','FRA'] 625 Multi['CLEV','FRA'] 625
BOUNDS
 PL BOUND Prod_bands:Trans['bands','GARY','FRA']
 PL BOUND Prod_bands:Trans['bands','CLEV','FRA']
 PL BOUND Prod_coils:Trans['coils','GARY','FRA']
 PL BOUND Prod_coils:Trans['coils','CLEV','FRA']
 UP BOUND Slacks[1] 625
```

```
 UP BOUND Slacks[2] 625
ENDATA
```

Tant la creació de fitxers MPS amb aquestes extensions com la seva lectura han sigut implementats dintre de la llibreria del propi solver BlockIP.

3.8 wxWidgets

wxWidgets [15] és un conjunt de biblioteques multiplataforma i lliures, per al desenvolupament d'interfícies gràfiques. Va ser creat per Julian Smart a la Universitat d'Edimburg i actualment és manté per la comunitat.

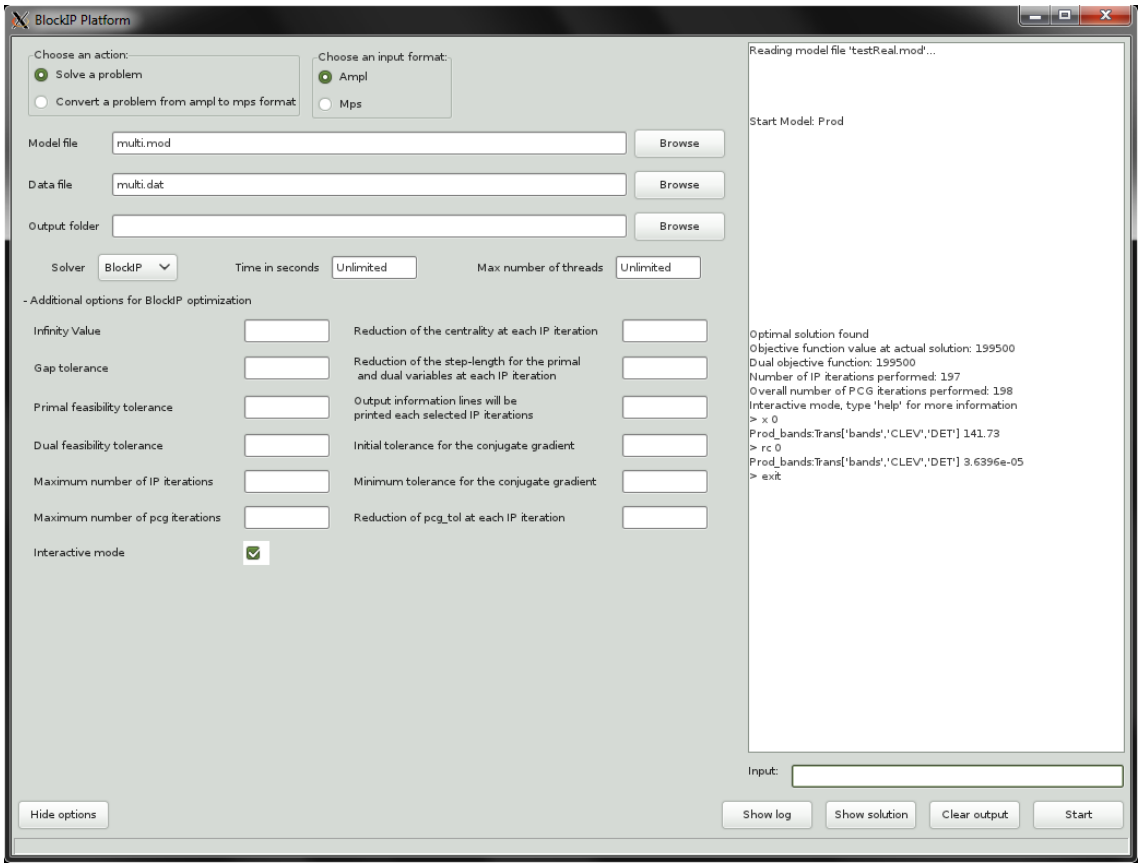
Inicialment només es van crear llibreries per al llenguatge de programació C++ i plataformes Unix i Windows, però actualment pot ser utilitzat amb altres llenguatges de programació com Java, JavaScript, Phython o Perl entre altres. També s'han creat llibreries per al sistema operatiu Mac OS.

L'avantatge de wxWidgets és que proporciona una interfície gràfica basada en les biblioteques natives del sistema operatiu, de manera que l'aspecte de l'aplicació no desentona amb la resta del sistema i a més permet una gran portabilitat entre diferents sistemes operatius. A més, a l'utilitzar biblioteques natives del sistema té millor rendiment que altres biblioteques gràfiques.

wxWidgets proporciona una capa d'abstracció sobre els controls natius d'una plataforma i fa senzilles algunes tasques que d'altra manera requeririen molt de treball. A més es pot compilar des de pràcticament qualsevol compilador de C++ sense haver de passar per una fase prèvia de compilació, tal i com passa amb altres biblioteques gràfiques.

Per últim, hi ha una gran quantitat de documentació, una comunitat darrera i es manté en continu desenvolupament, el que fa més fàcil aprendre a utilitzar-ho i que els errors estiguin depurats.

L'aplicació gràfica de la plataforma ha sigut creada amb wxWidgets. Un cop après el funcionament, el desenvolupament ha sigut sense cap inconvenient i el resultat satisfactori, a continuació es mostra una captura de pantalla de l'aplicació creada:



3.9 wxFormBuilder

wxFormBuilder [2] és una eina RAD (desenvolupament ràpid d'aplicacions) per wxWidgets multiplataforma i de codi obert.

El seu objectiu és ser una aplicació que permet el desenvolupament visual de l'aplicació a la vegada que pot generar codi i permet la inclusió d'elements no gràfics, com la connexió de widgets. On els widgets són cadascú dels elements que formen la interfície gràfica com per exemple botons, text, check box, etc.

D'aquesta manera es pot crear tota la part visual d'una interfície gràfica sense haver de programar res, sinó de forma molt ràpida mitjançant clics. A més permet assignar *events* a cada widget.

Un cop que s'ha finalitzat el disseny, wxFormBuilder genera dos classes, una classe que mai ha de ser modificada i que conté tot el codi que crea l'aspecte visual de l'aplicació i la connexió d'events. I una altra classe que hereta de l'anterior i que ha de ser omplerta pel programador per definir la lògica de l'aplicació i realitzar qualsevol canvi.

L'aplicació mostrada anteriorment amb wxWidgets s'ha realitzat amb l'ajuda de wxFormBuilder i m'ha ajudat molt a poder utilitzar wxWidgets ja que realitza una part important de la feina.

3.10 Matlab

Matlab [22] és un software matemàtic multiplataforma que ofereix un entorn de desenvolupament integrat amb un llenguatge de programació propi. Permet manipular matrius molt còmodament i implementar algorismes matemàtics ràpidament.

El major inconvenient de Matlab és que és lent. És a dir, per a execucions petites va bé, però per solucionar problemes d'optimització de certa grandària, el mateix algorisme és molt més ràpid en C++ que en Matlab.

L'ús de Matlab al projecte ha vingut donat per que la implementació original de BlockIP, prblock_ip [10], és va realitzar en Matlab. Donat que BlockIP encara està en procés d'implementació, el poder utilitzar la mateixa entrada de dades tant per l'un com per l'altre permet fer un control exhaustiu sobre la implementació actual comprovant que els resultats actuals són coherents amb l'algorisme original. A més quan BlockIP estigui acabat és podran fer comparacions de rendiment entre una implementació i altra molt fàcilment.

El que és volia era connectar una entrada de dades que s'ha transformat a través d'un codi C++ i enviar-la a Matlab de forma transparent per l'usuari. Per fer-ho he utilitzat fitxers i crides a sistema. Així el que es fa és transformar les dades del problema, que es troben carregades al programa, a format Matlab i escriure-ho en un fitxer. Aquest fitxer a més conté un script que crida a prblock_ip i el resultat és escrit en fitxers que llegirà el programa C++.

Tot i que el codi escrit en C++ s'ha fet compatible tant per Unix com per Windows canviant les crides a sistema, el codi prblock_ip conté rutines compilades en Fortran i actualment només es pot utilitzar a sistemes Unix.

4. BlockIP Platform

BlockIP Platform és el nom de la plataforma creada que té com objectiu la resolució de problemes de gran escala amb estructura angular per blocs a partir d'una entrada de dades còmoda per l'usuari.

Consta de diferents llibreries, que poden ser utilitzades com un tot o per separat. S'ha fet aquesta separació en diverses llibreries per tal de que sigui fàcil aprofitar codi per altres aplicacions ja que cada llibreria té una funció molt concreta.

L'estructura de directoris de la plataforma és la següent:

- amplsolvers: Conté el codi d'AMPL per accedir als fitxers .nl amb les modificacions explicades anteriorment.
- bin: Conté el codi font del programa principal per línia de comandes així com els executables.
- BlockIP: Conté el codi font del solver BlockIP.
- BlockIPInterface: Conté el codi font de la llibreria principal de la plataforma.
- Doxygen: Conté la documentació generada a partir dels comentaris del codi.
- GUI: Conté el codi font de la interfície gràfica.
- OsiInterface: Conté el codi font de la llibreria per comunicar-se amb Osi.
- prblock_ip: Conté el codi font Matlab de la versió original de BlockIP.
- sml-0.7.2: Conté el codi font de la llibreria SML amb les modificacions explicades anteriorment.
- SMLBlockIP: Conté el codi font de la llibreria per comunicar un problema amb format SML amb BlockIP.

4.1 Especificació

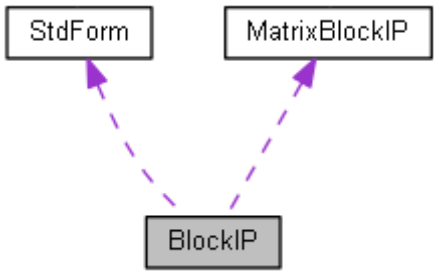
A continuació s'especifiquen els mètodes creats més rellevants de les classes utilitzades. Una documentació més extensa i complerta es pot trobar al directori Doxygen dintre del codi adjuntat.

Als diagrames que apareixen, una línia porpra puntejada significa que la classe apuntada està continguda o utilitzada des de classe on surt la fletxa. Una línia continua significa que la classe apuntada és la classe base de la que surt la fletxa. Un requadre puntejat indica que les classes contingudes al requadre són externes a la llibreria.

4.1.1 Llibreria BlockIP

Tot i que no és propòsit d'aquest projecte implementar un solver, i malauradament la versió C++ de BlockIP encara es troba en fase de desenvolupament i no es pot utilitzar com a solver; he afegit directament a la llibreria de BlockIP funcions que poden ser rellevants per al solver i que són necessàries per al funcionament de la plataforma.

L'estructura de classes que té la llibreria de BlockIP és:



Les funcions que té actualment aquesta llibreria són: convertir un problema a forma estàndard, llegir i escriure MPS amb l'extensió per blocs i operacions matricials, com per exemple convertir tots els blocs del problema en un únic bloc.

Els mètodes més rellevants de la llibreria són:

- **Classe BlockIP**

```
Load a general problem: lb <= variables <= ub and lhs <= constraints
<= rhs
void load_problem(double cost[], double qcost[], void (*fobj) (int n,
     double x[], double &fx, double Gx[], double Hx[], void *params),
     void *params, double lb[], double ub[], double lhs[], double
     rhs[], int numBlocks, bool sameN, MatrixBlockIP N[], bool sameL,
     MatrixBlockIP L[], bool copy_vectors, bool convertToStd)
param cost Linear cost of variables including slacks
```

```
param qcost Quadratic cost of variables including slacks
param fobj User function to calculate the objective function in a
      point. If it is is not NULL, cost and qcost must be NULL
param params User parameters to perform objective function
      calculations. Only can be used when fobj is not NULL
param lb Lower bounds, including slack bounds
param ub Upper bounds, including slack bounds
param lhs Lower constraint limits, including linking constraints
      limits
param rhs Upper constraint limits, including linking constraints
      limits
param numBlocks Number of diagonal blocks
param sameN Define if the same matrix is used for each N block.
      If true array N must have dimension 1
param N Diagonal blocks
param sameL Define if the same matrix is used for each L block.
      If true array L must have dimension 1
param L Linking constraints blocks
param copy_vectors If is true the user must free the memory of
      arguments (arrays and classes). If false the user must allocate
      the memory with new, after this call the user will not have
      control of the arguments (arrays and classes) anymore and
      MatrixBlockIP will delete the arguments
param convertToStd The problem must be converted to standard in order
      to be solved, you can convert the problem now or later calling
      convert_to_standard function.
note If sameN is used each constraint must be the same type in each
     block, e.g., if constraint i is an equality in block j must be an
     equality in all other blocks.
note If sameN is used each free variable with zero value in Hessian
     must be the same type in each block, e.g., if variable i is free
     variable with zero value in Hessian of block j, it must be a free
     variable with zero value in Hessians of all other blocks.

Convert any problem to standard form, restrictions= rhs, variables >=
0, <= ub
void convert_to_standard()

Write a mps file
void write_mps(const char *filename)
param filename Filename for output MPS

Create a problem from a mps file
void read_mps(const char *filename, bool convertToStd)
param filename File name with extension
param convertToStd The problem must be converted to standard in
      order to be solved, you can convert the problem now or later
      calling convert_to_standard function.
note Token : separate the block name and the variable or constraint
     Name. When a constraint or variable does not have a block name
     it is considered a linking constraint or a slack
note If some variable have as a name "Constant" with no block, this
     Variable is considered a constant to add in the objective
     Function.
note Slacks are optional, but if one is defined, all slacks must be
     defined, to relate an slack to one linking constraint, this
     slack must have a 1 coeficient in the related linking
     constraint.
note In QUADOBJ only cannot be relation between two different
     variables.
```

- **Classe StdForm**

```
Function to calculate the objective function in an extended variables
point
void fobj_stdform(int n, double x[], double &fx, double Gx[], double
     Hx[], void *params)
Converts a point from extended variables to original variables space
and call the user function to calculate the objective function in
that a point, after that inverts the transformations to return the
information in the extended variables space
param n Number of variables in extended variables space
param x Point in extended variables space
param fx Objective function value in x
param Gx Gradient in x
param Hx Hessian in x
param params User parameters to perform objective function
      calculations

Converts a point from extended variables to original variables space
void extended_to_original_variables(double* &xout, double*
     xin)
param xout Point in original variables space, is allocated by the
      function with new, must be freed with delete[]
param xin Point in exteded variables space

Converts a point from original variables to extended variables space
void original_to_extended_variables(double* &xout, double*
     xin)
param xout Point in exteded variables space, is allocated by the
      function with new, must be freed with delete[]
param xin Point in original variables space

Transforms objective function from original variables to extended
variables space
void transform_linear_and_quadratic_cost(double* &cost, double*
     &qcost, double &constant)
param cost Linear cost of variables including slacks, input in
      original variables space, output in extended variables space
param qcost Quadratic cost of variables including slacks, input in
      original variables space, output in extended variables space
param constant Constant to add in the objective function
```

- **Classe MatrixBlockIP**

```
Builds a packed rowwise format structured matrix based on diagonal
blocks and linking constraints blocks
void load_full_matrix(int numBlocks, bool sameN, MatrixBlockIP N[],
     bool sameL, MatrixBlockIP L[])
param numBlocks Number of diagonal blocks
param sameN Define if the same matrix is used for each N block
param N Diagonal blocks
param sameL Define if the same matrix is used for each L block
param L Linking constraints blocks

Add new columns into the matrix
void add_new_columns(int num_columns, int size[], int *irows[], double
     *values[])
param num_columns Number of columns to add
param size number of non-zero elements of the new column for each
      column
param irows Row position for each element and column, have to be
```

```
      ordered
param values Value of non-zero elements for each column
```

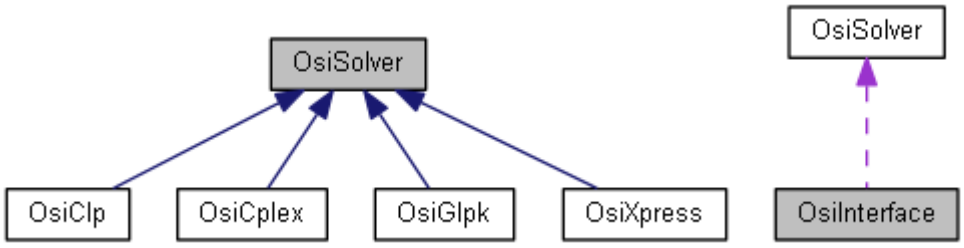
4.1.2 Llibreria OsiInterface

La llibreria OsiInterface té com a propòsit comunicar-se amb altres solvers diferents de BlockIP utilitzant per a tots els mateixos mètodes i la mateixa sintaxi. Per fer-ho es crea una capa d'abstracció sobre els solvers. Aquesta capa d'abstracció la proporciona Osi.

Però com s'ha explicat anteriorment Osi té unes carències que fan inviable l'ús del mateix codi per a tots els solvers. Aquesta llibreria el que fa és arreglar això. La manera d'arreglar-ho és utilitzar Osi quan funciona, i utilitzar directament la classe o llibreria del solver per a les tasques que no estan bé implementades o que no existeixen a Osi.

Idealment aquesta llibreria no hauria d'existir i és possible que en algun moment el codi d'Osi estigui més avançat i no sigui necessària.

L'estructura de classes que té la llibreria de BlockIP és:



OsiSolver defineix totes les funcions de la llibreria que després poden ser especialitzades a cada subclasse i OsiInterface proporciona una capa més d'abstracció en quant a l'elecció del solver i és aquesta classe la que haurà d'utilitzar l'usuari. Així doncs els mètodes més rellevants per OsiInterface són, (per la resta de classes són els mateixos):

```
Load a problem
void loadProblem(int numRows, int numColumns, int nz,
     double obj[], double lb[], double ub[], double lhs[], double
     rhs[], int begRows[], int indCols[], double values[], bool
     copy_vectors)
param numRows Number of rows
param numColumns Number of columns
param nz Number of non-zero elements in the sparse matrix
param obj Objective function values, if NULL, all variables have 0
      objective coefficient
param lb Lower bound of variables
param ub Upper bound of variables, if NULL, all columns have upper
      bound infinity.
param lhs Lower bound of restrictions, if NULL, all columns have lower
```

```
      bound 0
param rhs Upper bound of restrictions, if NULL, all rows have upper
      bound infinity
param begRows Index to the begining of each row, if NULL, all rows
      have lower bound -infinity
param indCols Index of each column
param values Value of each non-zero element
param copy_vectors if false the vectors will be used freed with
      delete[] after optimization

Get solver interface
OsiSolverInterface* getSolverInterface()
return Solver interface

Solve the problem loaded
OPT_STATUS solve()
return Optimization exit status

Get the solution
const double* getSolution()
return Solution
```

4.1.3 Llibreria SMLBlockIP

La llibreria SMLBlockIP té com a propòsit carregar un problema d'entrada en format SML a BlockIP.

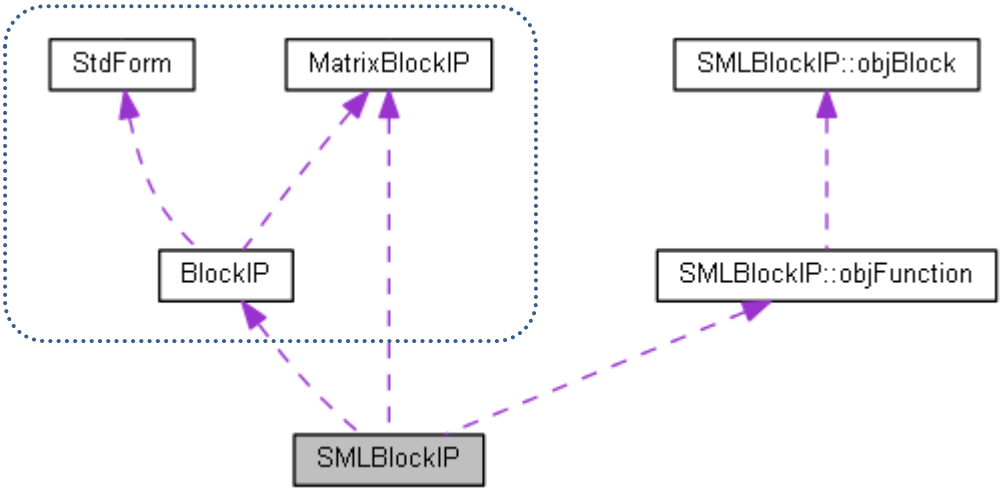
Recordem que el format SML és un el format d'AMPL però estès per permetre definir estructures, com per exemple blocs. De la mateixa manera que vaig estendre el format MPS per permetre la definició del block Slacks, també he estès el format de SML per permetre aquesta definició. L'extensió és molt senzilla, i es basa en que si es vol donar cost als slacks de les linking contraints a la funció objectiu, s'ha de definir un block amb nom "Slacks" que contingui la definició de les variables slacks, un per a cada linking contraint, i la seva funció objectiu. És clar que aquest bloc no pot contenir restriccions. I per relacionar cada slack amb cada linking contraint han d'aparèixer a la restricció sumant. Amb un exemple és veu molt més clar:

```
                                    ...
block Slacks:
    var slacks{(i,j) in ARCS} >= 0, <= mut_cap[i,j];
    minimize Total_Cost: sum {(i,j) in ARCS}(slacks[i,j]);
end block;

subject to MutCap {(i,j) in ARCS} :
    sum {k in COMM} Net[k].flow[i,j] + Slacks.slacks[i,j] =
    mut_cap[i,j];
```

La llibreria SMLBlockIP a més de carregar un problema a BlockIP també pot convertir un problema des del format SML a MPS. I en cas que la funció objectiu sigui no lineal,

en tots o algun dels blocs, crea una funció per a que BlockIP pugui avaluar qualsevol punt en aquesta funció.

L'estructura de classes que té la llibreria SMLBlockIP és:



Les rutines més importants de SMLBlockIP són:

```
Load a problem from ampl files
void readAmpl(const char *modelFilename, const char *dataFilename,
bool convertToStd)
param convertToStd If true the problem will be converted to BlockIP
      standard form after loading

Function to get objective function information in a point when the
objective function is non-linear
void fobjnonlin(int n, double x[], double &fx, double Gx[], double
Hx[], void *params)
param n Number of variables
param x Point
param fx Objective function value in x
param Gx Gradient in x
param Hx Hessian in x
param params User parameters to perform objective function
      calculations

Converts a problem in ampl format to mps format
void amplToMps(const char *modelFilename, const char *dataFilename,
const char *mpsFilename, bool convertToStd)
param convertToStd If true the problem will be written in BlockIP
      standard form
note The problem contained by BlockIP will be not modified
```

4.1.4 Llibreria BlockIPInterface

Aquesta llibreria és la principal de la plataforma. La que qualsevol usuari que vulgui fer servir totes les funcionalitats còmodament ha d'usar. La seva funció és crear una capa

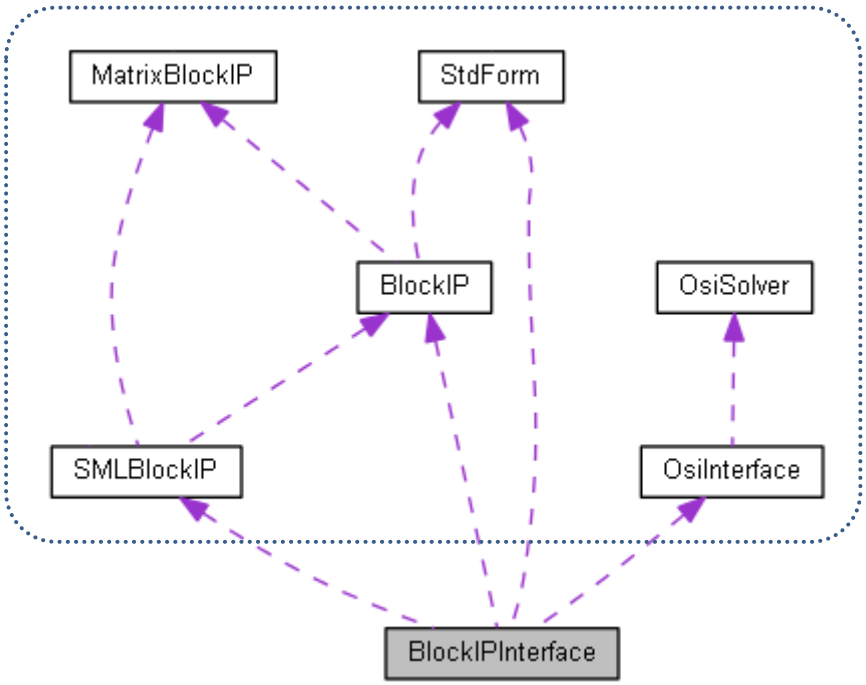
d'abstracció en la qual l'usuari només hagi d'indicar l'acció a fer, el solver a utilitzar i el fitxer d'entrada.

Té dos usos principalment, resoldre un problema i convertir un problema en format SML a format MPS.

Per resoldre el problema es pot utilitzar BlockIP (quan estigui acabat), prblock_ip (la versió BlockIP en Matlab), i els solvers d'Osi.

El convertir un problema en format SML a format MPS serveix sobretot per poder utilitzar l'MPS externament a aquesta llibreria, és a dir, qualsevol solver (teòricament) podrà llegir l'MPS generat, tot i que estigui en format per blocs.

L'estructura de classes que té la llibreria és:



I les rutines més rellevants són:

```
Set the solver to be used
void setSolver(SOLVER solver)
param solver Solver to be used
```

```
Load a problem from a mps file
void readMps(const char *filename)
param filename Filename of mps file

Load a problem from ampl files
void readAmpl(const char *modelFilename, const char *dataFilename)
param modelFilename Filename of model file
param dataFilename Filename of data file

Solve the problem loaded
OPT_STATUS solve()

Converts a problem in ampl format to mps format
void amplToMps(const char *modelFilename, const char *dataFilename,
const char *mpsFilename, bool convertToStd)
param modelFilename Filename of model file
param dataFilename Filename of data file
param filename Filename of mps file
param convertToStd If true the problem will be written in BlockIP
      standard form

Create the files to load and solve a problem in BlockIP Matlab
void createMatlabProblem()
```

4.1.5 mainBlockIPPlatform

És el programa principal per línia de comandes. Bàsicament utilitza totes les funcions de la llibreria BlockIPInterface i en el cas d'utilitzar BlockIP com a solver pot entrar en un mode interactiu per recuperar informació sobre com ha acabat l'execució.

Per tant, el programa té dos usos diferents, resoldre un problema o convertir un problema en format AMPL estès per SML i afegint el bloc slacks a format MPS estès a blocs.

El problema d'entrada pot ser tant en format SML com en MPS. L'estructura ha de ser angular i no poden haver blocs continguts a altres blocs. Els termes quadràtics o no lineals no poden ser productes de variables diferents. És a dir, la matriu Hessiana de la funció objectiu ha de ser diagonal.

Com hem vist abans l'únic solver que pot explotar l'estructura per blocs és BlockIP.

La sintaxi d'ús del programa i les seves opcions es poden veure a l'Annex.

Actualment quan es vulgui utilitzar BlockIP com a solver, automàticament s'utilitzarà la versió Matlab fins que la versió C++ estigui acabada. En aquesta implementació només els paràmetres "-inf" i "-iter" poden ser utilitzats amb la versió Matlab de BlockIP, la resta de paràmetres estan pensats per la versió en C++.

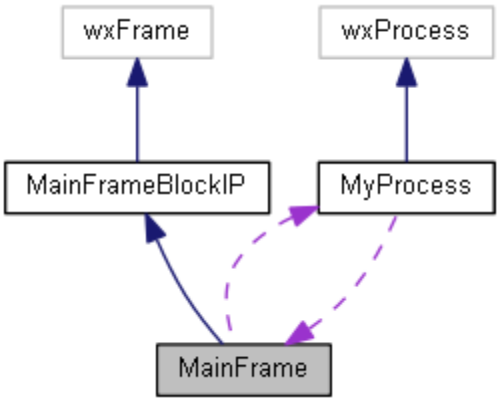
4.1.6 GUI

Dins del directori GUI ens trobem el codi font de la interfície gràfica, la qual com s'ha explicat anteriorment s'ha creat amb wxWidgets i dissenyat amb wxFormBuilder.

Les funcionalitats d'aquesta interfície gràfica són les mateixes que té l'aplicació per línia de comandes. De fet la interfície gràfica serveix com un enllaç entre l'usuari i l'aplicació per línia de comandes.

Això és per no haver de mantenir dos codis diferents i molt similars, i en canvi poder aprofitar codi existent. El que s'ha fet és que la interfície gràfica agafi totes les dades d'entrada i cridi a la aplicació per línia de comandes. D'aquesta manera la interfície el que fa és crear un procés, rebre dades d'aquest procés i mostrar-les; i rebre dades de l'usuari i enviar-les al procés.

L'estructura de classes de la interfície gràfica és:



I els seus mètodes més importants són:

- **Classe MainFrame**

```
Creates a process with the input data and starts the execution
void startClick( wxCommandEvent& event )

Send input data to the process and write output from the process
void OnIdle(wxIdleEvent& event)

The frame returns to its initial state
void OnProcessCompletion()
```

- **Classe MyProcess**

```
When the execution ends or is killed OnTerminate is called and notify
the frame
void OnTerminate(int pid, int status)
```

4.2 Implementació

A continuació es descriuen els detalls d'implementació més rellevants de la plataforma. Per evitar redundàncies em referiré a les rutines abans especificades pel seu nom, els seus paràmetres i les seves especificacions es poden trobar a la subsecció anterior.

4.2.1 Llibreria BlockIP

- **BlockIP::load_problem**

Un paràmetre que ens trobem a diverses funcions de la plataforma i que té una gran importància sobretot a la rutina load_problem és el copy_vectors. Per veure la importància hem de recordar que els problemes a resoldre són molt grans, poden ocupar Gigues de memòria, el que pot implicar que un problema gran no el puguem resoldre si hem de copiar el problema d'entrada.

Es va prendre la decisió d'utilitzar aquest paràmetre com un avís. A C++ la memòria dinàmica es pot reservar de dues maneres diferents, utilitzant la comanda new o qualsevol de les variants d'alloc. La primera forma és la estàndard de C++, les altres són heretades de C; i cadascuna també té un mètode d'alliberació diferent i no compatible.

A tota la plataforma es reserva la memòria amb new, i per màxima eficiència no s'han de copiar els vector, però llavors l'usuari ha de ser conscient que els paràmetres passats han d'haver sigut reservats amb new i que un cop passats deixa de tindre el control sobre aquests paràmetres.

Altra característica important per evitar un gran consum de memòria, tot i que per la naturalesa de SML i dels MPS no s'utilitza en aquest projecte, és que si tots el blocs tenen la mateixa estructura i els mateixos coeficients es pot indicar i només passar un únic bloc. Això té un impacte important a la conversió a estàndard i altres operacions que transformen matrius ja que s'han de tenir en compte molts cassos diferents.

- **BlockIP::convert_to_standard**

Aquesta rutina és una de les més rellevants projecte, la que dóna llibertat a l'usuari de poder passar el problema en el format que vulgui i una de les que més hores raonant té darrere. Aquí és on es realitzen totes les transformacions explicades a "Conversió a forma estàndard".

Hem de tindre en compte que no només s'ha de convertir el problema a forma estàndard, sinó que a més hem de poder tornar a desfer el canvi en quant a les variables ja que el nou espai de variables no és el que espera l'usuari. Per fer aquests canvis d'un espai de variables a un altre es va crear la classe StdForm, que després veurem amb més deteniment.

A més, degut a les possibles limitacions de memòria s'ha procurat reservar la mínima memòria possible per a cada tram del codi i alliberar-la en quant fos possible.

Tots els canvis a realitzar a les dades del problema en sí bàsicament es limiten a afegir columnes a matrius, canviar coeficients de columnes i files i multiplicar vectors per matrius. El problema era en quin moment realitzar els canvis, en quin ordre i seguint una casuística complexa.

- **BlockIP::write_mps**

Aquesta rutina tot i ser molt llarga no té més complexitat que el seguir estrictament el format MPS, tot i que en C el tractar amb fitxers era poc còmode, en C++ s'ha millorat molt. Gràcies al *Output file stream*, ha sigut senzill de programar.

- **BlockIP::read_mps**

Contràriament a la rutina anterior, la lectura d'un fitxer mps és molt més complexa ja que s'han de realitzar múltiples comprovacions de que el fitxer d'entrada sigui correcte, tant sintàcticament com semànticament. Degut a que inicialment no es coneix la dimensió del problema d'entrada l'ús de la classe *Vector*, ha sigut de gran ajuda. Així com l'ús de *string stream* junt a *getline* per la lectura del fitxer ha ajudat molt a detectar errors de sintaxi.

- **StdForm**

Per guardar la relació entre l'espai de variables original i l'espai de variables esteses s'han utilitzat 3 vectors de dimensió número de transformacions:

- transforms: Guarda el tipus de transformació per a cada transformació.
- origVars: Té els índexs de les variables que han rebut alguna transformació en l'espai original de variables. En el cas de que la transformació sigui una variable nova, llavors l'índex és -1.

- extVars: Té els índexs de les variables que han rebut alguna transformació en el nou espai de variables.

Guardant la informació d'aquesta manera el canvi d'un espai a un altre és molt ràpid i a més la quantitat de memòria utilitzada és petita.

- **StdForm::fobj_stdform**

Aquesta rutina és la que crida BlockIP cada vegada que vol avaluar un punt i la funció objectiu és no lineal. Per fer-ho primer es realitza un canvi de variables del nou espai a l'original, tenint en compte les transformacions abans explicades.

Un cop tenim les variables en l'espai original ja podem utilitzar la rutina que ens ha donat l'usuari per calcular la funció objectiu. I el que ens és retornat s'ha de tornar a convertir al nou espai de variables, també tenint en compte les transformacions de la segona secció.

La resta de rutines abans especificades són trossos o especialitzacions d'aquesta.

- **MatrixBlockIP**

Aquesta classe conté una matriu, aquesta matriu pot estar guardada de diferents maneres segons la seva estructura, però per el relacionat al projecte només ens interessen dos maneres. Així doncs la matriu es guarda de forma esparsa, i pot estar per columnes o per files.

Els mètodes més interesants d'aquesta classe són els que transformen la matriu d'un format a un altre, per exemple una matriu de xarxa a una matriu esparsa per files. Però com això no s'utilitza a la resta del projecte anem a veure només una funció que si té s'utilitza.

- **MatrixBlockIP::load_full_matrix**

Tot problema per blocs el podem veure com un únic bloc que conté tant els blocs diagonals com les linking contraints, el que fa aquesta funció és crear aquesta matriu global. Partint de matrius esparses per files és una operació fàcil de realitzar, tant sols cal concatenar les matrius desplaçant els índexs de columnes tenint en compte l'estructura de les matrius.

4.2.2 Llibreria OsiInterface

La característica més curiosa d'implementació d'aquesta classe és que l'usuari de la llibreria no es té que preocupar, en el moment de crear el codi, de quins solvers estan enllaçats a l'aplicació. Per entendre això anem a recordar com funciona l'herència en C++ i la compilació.

En el nostre cas particular tenim que OsiSolver és la classe base i deriven d'ella una classe per a cada solver. Llavors un usuari de la llibreria per fer una instanciació amb el solver Glpk, necessitaria crear una variable de tipus OsiGlpk. OsiGlpk a la seva definició inclou l'entorn de Glpk i la interfície de Glpk amb Osi, per tant, en el moment de la compilació serà necessari que existeixen aquests fitxers.

Si un usuari de la llibreria vol crear una aplicació que permeti l'ús de diferents solvers (el que és una cosa normal si s'utilitza Osi), llavors aquest usuari ha de tenir en compte que alguns solvers poden estar o no en el sistema. Per fer això a C s'utilitzen definicions de precompilació, és a dir, uns trossos de codi existeixen o no (de cara al compilador) depenent dels flags de precompilació utilitzats a la compilació.

Ara bé, aquesta plataforma va dirigida a gent no experta, així que amb la creació de la classe OsiInterface vaig crear una capa d'abstracció que conté tots els flags de precompilació necessaris per a que l'usuari no es tingui que preocupar, només haurà de dir quins solvers té en el moment de compilar. En codi, la creació d'OsiInterface es tradueix a:

```
OsiInterface(SOLVER solver) {
#ifndef CPLEX_
  if (solver == CPLEX)
    throw new ExceptionOsiSolver(SOLVER_NOT_AVAILABLE);
#endif
                                    ...
#ifdef CPLEX_
  if (solver == CPLEX)
    os = new OsiCplex();
#endif
                                    ...
}
```

I el flag de precompilació que ha de definir l'usuari al compilar si té Cplex és, CPLEX_. Per a tot solver el flag s'ha definit com el nom del solver en majúscules seguit d'una barra baixa.

De la mateixa manera s'han posat aquests flags al principi i final dels fitxers que defineixen cadascuna de les classes derivades, per exemple (i simplificant molt):

```
#ifdef CPLEX_
#include "OsiSolver.h"
#include <ilcplex/cplex.h>
class EXPDLL OsiCplex : public OsiSolver {
                                        ...
};
#endif //CPLEX_
```

4.2.3 Llibreria SMLBlockIP

- **SMLBlockIP::readAmpl**

Aquesta rutina és l'altra rutina principal del projecte. És la que a partir de fitxers AMPL/SML carrega el problema a BlockIP. És la que es connecta amb la llibreria de SML i fa de parser del bloc Slacks.

Aquí es van creant les matrius i vectors que es passaran a BlockIP, però hi ha diverses coses a tindre en consideració:

- AMPL desordena les variables, més ben dit, les ordena col·locant primer les que apareixen de forma no lineal a la funció objectiu i després la resta, però aquest no és l'ordre que espera l'usuari. Una de les feines a fer és guardar l'ordre original per a que quan tinguem la solució puguem ordenar-la abans de passar-la a l'usuari.
- El tipus de funció objectiu es defineix per bloc, llavors un bloc pot tindre la funció objectiu lineal, quadràtica o no lineal. En el cas de que tots els blocs tinguin la funció objectiu lineal o quadràtica s'han de passar els vectors dels coeficients directament a BlockIP. Però en el cas de que algun dels blocs tingui la funció objectiu no lineal es crea una funció objectiu no lineal que serà la que cridarà BlockIP cada vegada que vulgui calcular el valor de la funció objectiu en un punt.
- S'ha de detectar si existeix un bloc d'slacks i en el cas de que existeixi s'ha d'ordenar internament i comprovar que l'usuari l'ha passat correctament.

- Hi ha moltes comprovacions per fer. SML permet definir problemes amb moltes característiques que no són compatibles amb BlockIP, per tant s'ha de filtrar tot el problema d'entrada comprovant la seva correcció per BlockIP.

A més es creen els noms dels blocs, variables i restriccions. S'ajunten tant els límits de les variables com de les restriccions en vectors no individuals per blocs. I es carreguen les matrius de coeficients que venen donades en forma esparsa per columnes.

- **SMLBlockIP::amplToMps**

Aquesta rutina aprofita el codi ja fet en una funció molt útil. Per fer-ho primer es carrega en BlockIP el problema des de el format AMPL com hem vist en la rutina anterior; després s'utilitza la funció d'escriure MPS ja implementada a BlockIP.

- **SMLBlockIP::fobjnonlin**

Aquesta rutina és un atribut de SMLBlockIP i és la que es passa a BlockIP com a paràmetre per a que calculi la funció objectiu. Llavors aquesta rutina ha de ser una rutina genèrica però que pugui calcular qualsevol funció objectiu.

Per fer-ho hi ha un paràmetre params, que també és passat a BlockIP, i que BlockIP el passa cada cop que crida a aquesta funció. Dintre d'aquest paràmetre el que tenim és una estructura que guarda la quantitat de blocs, els coeficients linears i quadràtics per a cada bloc (si correspon) i la connexió amb la funció no lineal d'AMPL per a cada block (si correspon). A més, si hi ha slacks també es guarda el seu ordre.

D'aquesta manera és fàcil calcular la funció objectiu, gradient i Hessiana per a cada bloc i tornar-ho tot junt.

4.2.4 BlockIPInterface

Aquesta llibreria fa servir els recursos ja explicats en altres llibreries però combinant-los i decidint quin utilitzar en funció del solver. També conté el codi de connexió amb Matlab. Per a usuaris més avançats permet obtenir directament l'entorn del solver a utilitzar per tal de poder ajustar els paràmetres més minuciosament.

- **BlockIPInterface::createMatlabProblem**

De la mateixa manera que per escriure fitxers MPS, s'utilitza l'Output file stream de C++, però ara utilitzant la sintaxi de Matlab creem directament els vectors i matrius que demana prblock_ip com a paràmetres, també escrivim la crida a prblock_ip i al acabar instruccions per a que Matlab escrigui en un fitxer tots els paràmetres de sortida que te prblock_ip. També es crea un fitxer que conté un script que crida a Matlab en mode consola passant com a fitxer d'entrada el fitxer anteriorment creat.

- **BlockIPInterface::readMps**

Si el solver a utilitzar és BlockIP utilitza la rutina pròpia de BlockIP per llegir fitxers MPS, si el solver a utilitzar és algun dels que estan enllaçats a través d'Osi utilitza el mètode natiu d'Osi per llegir-lo. En canvi si el solver a utilitzar és prblock_ip (BlockIP a Matlab) llavors s'ha de fer més feina, primer llegim el problema amb la rutina de BlockIP, ens guardem informació sobre els noms del problema i la seva dimensió, convertim el problema a estàndard, cridem a createMatlabProblem, recuperem la instancia de la classe StdForm associada al problema i esborrem la instancia de BlockIP per tal de deixar la màxima memòria lliure per quan cridem a Matlab.

- **BlockIPInterface::readAmpl**

Llegim el problema amb SMLBlockIP i ens guardem l'ordre de les variables (recordem que AMPL canvia l'ordre), si el solver a usar és BlockIP ja hem acabat, però si és un solver a través d'Osi hem de convertir tots els blocs en un únic bloc amb load_full_matrix i ja podem carregar el problema a Osi amb loadProblem i podem lliberar la memòria de SMLBlockIP i de BlockIP.

Si el solver a usar és BlockIP Matlab llavors ens guardem informació sobre els noms del problema i la seva dimensió, convertim el problema a estàndard, cridem a createMatlabProblem, recuperem la instancia de la classe StdForm associada al problema i alliberem la memòria de SMLBlockIP i de BlockIP.

Actualment si el solver a utilitzar és algun d'Osi només es poden utilitzar funcions objectiu lineals, si és BlockIP Matlab lineals o quadràtiques, i les no lineals queden reservades per a BlockIP.

- **BlockIPInterface::solve**

Aquesta rutina és la que crida a resoldre el problema, la part més interessant és la transformació inversa de les variables quan s'utilitza Matlab (del nou espai de variables a l'original), i la reordenació d'aquestes si s'ha utilitzat SML. A part en el cas de Matlab recull informació sobre les variables duals.

4.2.5 mainBlockIPPlatform

La implementació de l'aplicació per línia de comandes és molt simple. Es basa en llegir els arguments d'entrada de argv i en funció del que vol l'usuari cridar a unes rutines de BlockIPInterface o unes altres. De fet s'ha intentat que la lògica estigui el més encapsulada possible dintre de la llibreria, així és més fàcil crear diverses aplicacions diferents. Inclús la part interactiva de l'aplicació només fa d'enllaç amb BlockIPInterface.

4.2.6 GUI

- **MainFrame::startClick**

Aquest mètode llegeix totes les opcions escollides per l'usuari a la GUI, crea un vector de paràmetres que serà els que llegirà el main per argv i crea el procés passant aquests paràmetres. En aquest moment entra en marxa un timer que cada cert temps cridarà a una funció.

També, en el cas de que ja hi hagués un procés en execució el mata ja que el mateix botó Start es converteix en un Stop quan el procés es comença a executar.

- **MainFrame::OnIdle**

Cada cop que el timer envia un event aquesta funció és cridada. Aquesta funció agafa la sortida estàndard i d'error del procés i la mostra per pantalla. També, en el cas de que l'usuari hagi donat informació d'entrada, envia aquesta informació cap a l'entrada estàndard del procés. Per fer-ho s'utilitzen crides pròpies de la llibreria wxWidgets

- **MainFrame::OnProcessCompletion**

Quan un procés finalitza aquesta funció és cridada i l'únic que fa és tornar a deixar la GUI en un estat inicial per tal de poder començar noves execucions.

- **MainFrame::OnTerminate**

Quan un procés mor, ja sigui per que ha rebut un kill o perquè ha acabat de forma natural aquesta funció és cridada automàticament. Com és possible que el procés hagi escrit informació per la sortida estàndard o d'error i que encara no s'hagi mostrat per pantalla (el temps que passa entre un event del timer i un altre), mostra aquesta informació per pantalla i després crida a OnProcessCompletion per notificar que ja s'ha acabat l'execució.

5. Planificació i costos

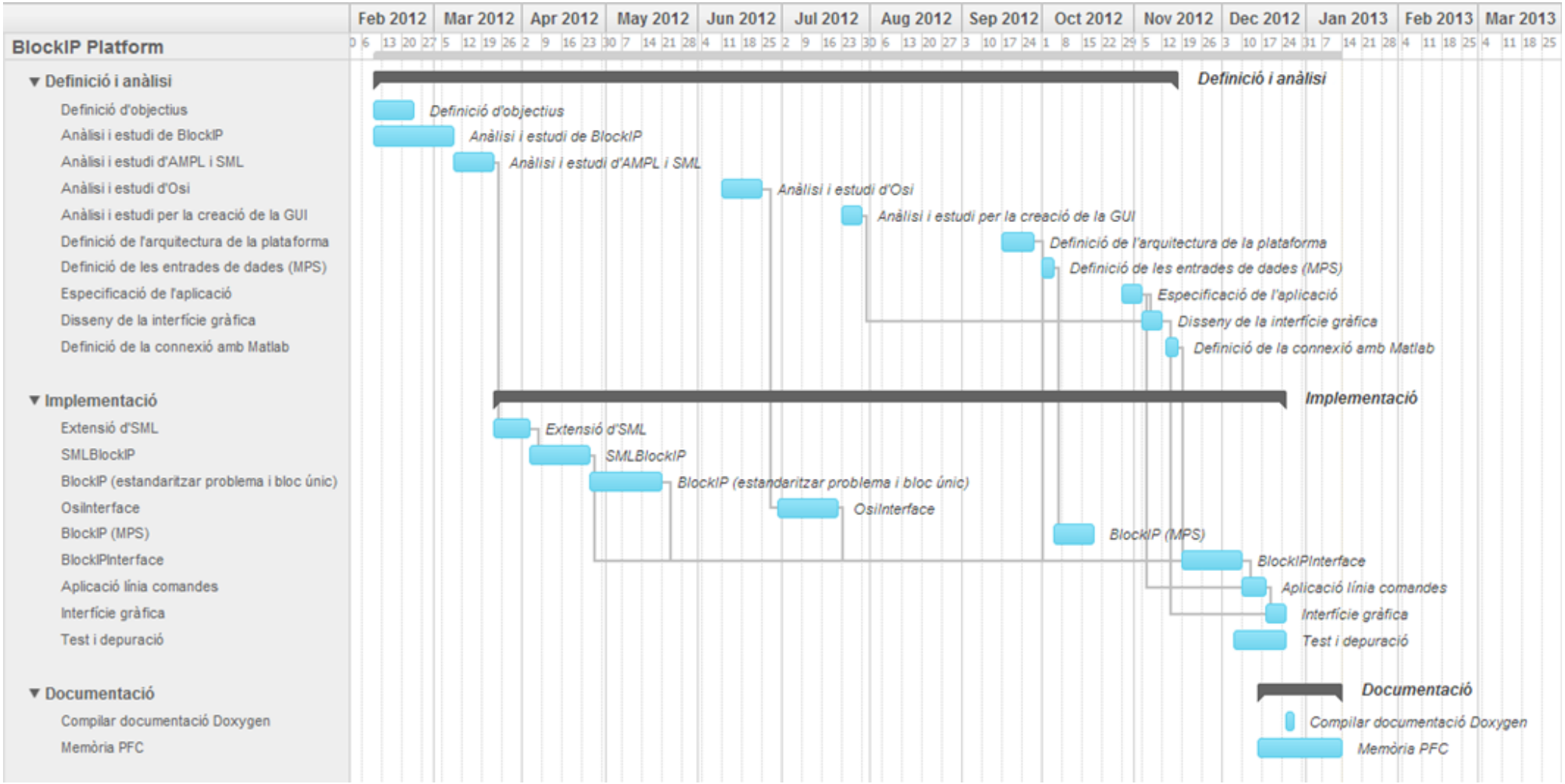
L'idea d'aquest projecte es va començar a crear a la segona meitat del 2011, però no va ser fins a principis de 2012 quan es va concretar, un cop que ja estava familiaritzat amb l'optimització matemàtica, després de fer l'assignatura de lliure elecció "Modelització en programació matemàtica".

Donat que aquest ha sigut un projecte d'empresa, s'ha realitzat en èpoques que no hi havia altre treball més prioritari. Per tant el desenvolupament del projecte no ha sigut de forma continuada sinó a trams.

També part de l'extensió del projecte durant el temps ha vingut donada per que era preferible que estigués finalitzat el solver BlockIP, de manera que el projecte tingués tot el sentit. Malauradament no ha sigut possible degut a limitacions de temps; i finalment es va optar per afegir l'enllaç al codi Matlab que inicialment no estava previst.

Per totes aquestes raons el recompte d'hores de dedicació al projecte durant la primera part d'aquest és una estimació basada en notes i dates dels fitxers.

A la següent pàgina apareix un diagrama de Gantt detallant la planificació, tot i això no totes les tasquen han sigut lineals, sinó que s'han realitzat iteracions sobre la mateixa tasca per arreglar errors o perfeccionar el codi.


Feb 2012
Mar 2012
Apr 2012
May 2012
Jun 2012
Jul 2012
Aug 2012
Sep 2012
Oct 2012
Nov 2012
Dec 2012
Jan 2013
Feb 2013
Mar 2013
BlockIP Platform
▼ Definició i anàlisi
Definició d'objectius
Anàlisi i estudi de BlockIP
Anàlisi i estudi d'AMPL i SML
Anàlisi i estudi d'Osi
Anàlisi i estudi per la creació de la GUI
Definició de l'arquitectura de la plataforma
Definició de les entrades de dades (MPS)
Especificació de l'aplicació
Disseny de la interfície gràfica
Definició de la connexió amb Matlab
▼ Implementació
Extensió d'SML
SMLBlockIP
BlockIP (estandaritzar problema i bloc únic)
OsiInterface
BlockIP (MPS)
BlockIPInterface
Aplicació línia comandes
Interfície gràfica
Test i depuració
▼ Documentació
Compilar documentació Doxygen
Memòria PFC
Definició i anàlisi
Definició d'objectius
Anàlisi i estudi de BlockIP
Anàlisi i estudi d'AMPL i SML
Anàlisi i estudi d'Osi
Anàlisi i estudi per la creació de la GUI
Definició de l'arquitectura de la plataforma
Definició de les entrades de dades (MPS)
Especificació de l'aplicació
Disseny de la interfície gràfica
Definició de la connexió amb Matlab
Implementació
Extensió d'SML
SMLBlockIP
BlockIP (estandaritzar problema i bloc únic)
OsiInterface
BlockIP (MPS)
BlockIPInterface
Aplicació línia comandes
Interfície gràfica
Test i depuració
Documentació
Compilar documentació Doxygen
Memòria PFC

Seguint el diagrama anterior, tenint en compte un treball diari per tasca de 4 hores i sumant entre totes les tasques 207 dies, per realitzar aquest projecte s'han empleat 828 hores.

Tot i que aquesta quantitat d'hores és molt superior a l'establerta per la quantitat de crèdits del projecte final de carrera, hem de considerar que algunes de les tasques d'anàlisi i estudi d'algunes tecnologies, com per exemple Osi i wxWidgets han sigut aprofitades al realitzar altres projectes paral·lels.

A continuació es descriu una aproximació del cost d'aquest projecte suposant el següents perfils: un analista que s'encarrega d'identificar les necessitats d'un client i definir les funcionalitats requerides, un arquitecte que s'encarrega del disseny i estructura de l'aplicació, així com les tecnologies a utilitzar. Un programador que implementa segons les indicacions de l'arquitecte. Un director de projecte que planifica i gestiona el projecte durant totes les fases i realitza un seguiment del projecte per tal de que es compleixin els terminis de lliurament. I finalment un dissenyador que dissenya la interfície gràfica.

Dividint les hores per perfil i assignant un hipotètic preu per hora de cost total per l'empresa tenim:

Perfil	Hores	Euros/hora	Cost
Analista	220	40,00€	8800,00€
Arquitecte	80	40,00€	3200,00€
Programador	424	25,00€	10600,00€
Director del projecte	84	60,00€	5040,00€
Dissenyador	20	30,00€	600,00€
Total	828	-	**28.240,00€**

El cost total del projecte en una empresa que utilitzi aquesta jerarquia de perfils és de 28.240€. A la realitat però, tot i que existeixin diferents perfils, aquest tipus de projecte realitzat no és divisible com a tal, i les empreses que és dediquen a això no utilitzen aquest tipus de jerarquia. Exemples d'aquestes empreses les trobem a les tecnologies utilitzades al projecte, com per exemple AMPL.

6. Demostració

Per il·lustrar com utilitzaria les aplicacions un usuari anem a veure alguns exemples.

Si té un problema amb format AMPL amb l'extensió per blocs, com per exemple:

```
set ORIG; # origins
set DEST; # destinations
set PROD; # products

param supply {PROD,ORIG} >= 0; # amounts available at origins
param demand {PROD,DEST} >= 0; # amounts required at destinations

param limit {ORIG,DEST} >= 0;
param cost {PROD,ORIG,DEST} >= 0; # shipment costs per unit

block Prod{p in PROD}:
  var Trans {ORIG,DEST} >= 0; # units to be shipped
  minimize Total_cost:
    sum {i in ORIG, j in DEST}
      cost[p,i,j] * Trans[i,j];

  subject to Supply {i in ORIG}:
    sum {j in DEST} Trans[i,j] = supply[p,i];

  subject to Demand {j in DEST}:
    sum {i in ORIG} Trans[i,j] = demand[p,j];
end block;

subject to Multi {i in ORIG, j in DEST}:
  sum {p in PROD} Prod[p].Trans[i,j] <= limit[i,j];
```

Suposant que aquest problema té com a nom multi.mod, que també té un fitxer de dades anomenat multi.dat i que vol solucionar el problema amb BlockIP amb les opcions per defecte; només hauria d'escriure:

```
> BlockIPPlatform -ampl multi.mod multi.dat
```

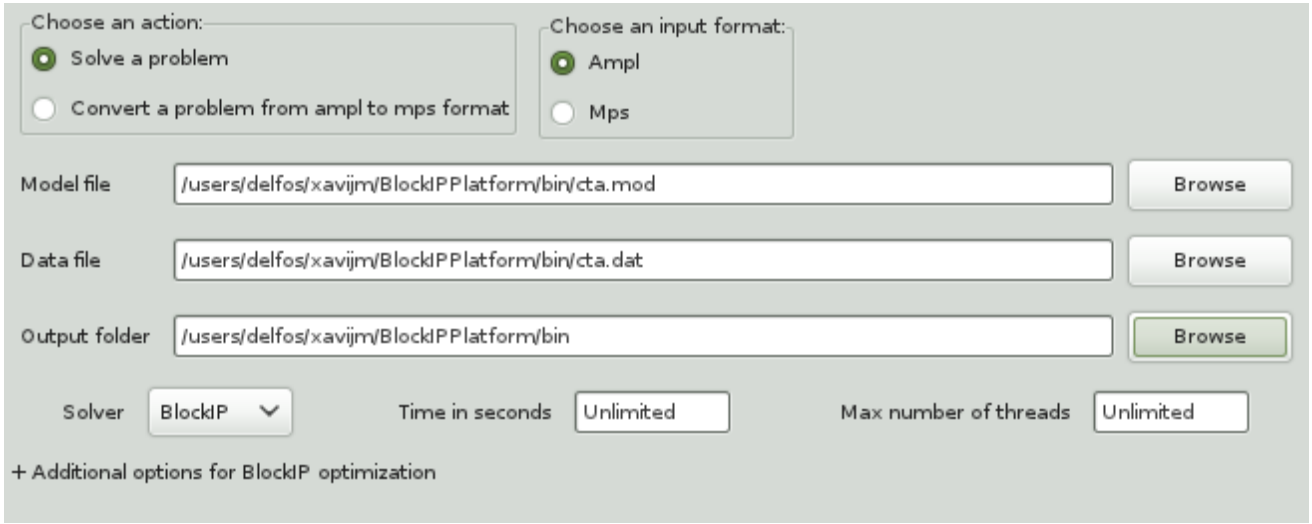
I si tot ha anat bé el problema ja estaria solucionat, amb un fitxer multi.sol que conté la solució del problema i un fitxer multi.log que conté la sortida d'informació del solver, la sortida que veuríem per pantalla és:

```
Reading model file 'multi.mod'...
Start Model: Prod
Optimal solution found
Objective function value at actual solution: 199500
Dual objective function: 199500
Number of IP iterations performed: 197
Overall number of PCG iterations performed: 198
```

Un altre exemple, aquest cop amb una funció objectiu quadràtica, podria ser el següent model:

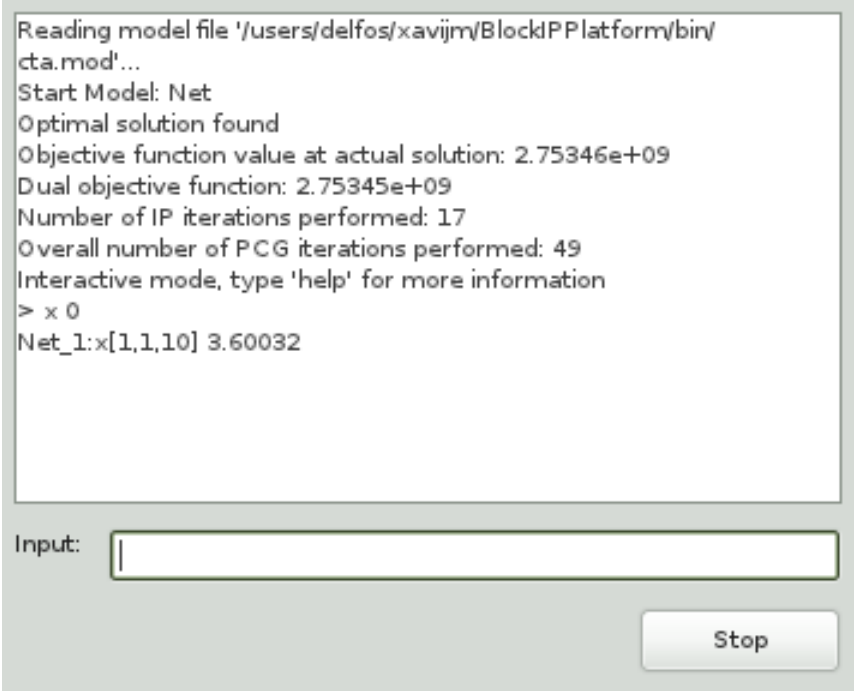
```
param m;
param n;
param l;
param p;

set M := 1..m;
set N := 1..n;
set L := 1..l;
set P := 1..p;

param p_r {P};
param p_c {P};
param p_l {P};
param p_lpl {P};
param p_upl {P};

set M2 := 1..m+1;
set N2 := 1..n+1;
set L2 := 1..l+1;

param a {M2, N2, L2};
param lb{M, N, L};
param ub{M, N, L};

block Net{k in L}:
  var x {(i,j) in {M, N}} >= lb[i,j,k], <= ub[i,j,k];
  minimize Qdistance:
    sum {(i,j) in {M, N}} (a[i,j,k] - x[i,j])^2;
  subject to Row {i in M}:
    sum {j in N} x[i,j] = a[i,n+1,k];
  subject to Column {j in N}:
   sum {i in M} x[i,j] = a[m+1,j,k];
end block;

subject to Linkconst {i in M, j in N}:
    sum {k in L} Net[k].x[i,j] = a[i,j,l+1];
```

Utilitzant la interfície gràfica per resoldre'l, seleccionaríem el model i les dades:



Podem seleccionar diverses opcions, en aquest exemple seleccionem l'opció d'entrar en un entorn interactiu, per preguntar el resultat de variables concretes.

Ja només queda clicar Start per començar l'optimització, un cop finalitzada podem fer consultes al mode interactiu:



Podem convertir aquest mateix model a un fitxer MPS que pot ser llegit per qualsevol lector de MPS; també existeix l'opció d'estandarditzar el problema abans d'escriure'l, així es resoldrà just el mateix problema que resol BlockIP. La manera de fer-ho és:

```
> BlockIPPlatform -convert cta.mod cta.dat cta -standarize
Reading model file 'cta.mod'...
Start Model: Net
Conversion performed successfully
```

Un cop tenim el fitxer MPS podem utilitzar-ho en aplicacions externes com per exemple la de Cplex:

```
> cplex
CPLEX> read cta.mps
Selected objective sense:  MINIMIZE
Selected objective  name:  obj
Selected RHS        name:  rhs
Selected bound      name:  BOUND
Problem 'cta.mps' read.
CPLEX> opt
                  .
                  .
                  .
Total time on 24 threads =    0.84 sec.
Barrier - Optimal:  Objective =  2.7534519825e+09
Solution time =    0.85 sec.  Iterations = 14
Deterministic time = 295.44 ticks  (346.52 ticks/sec)
```

7. Tasques futures

Aquest projecte ha finalitzat com a projecte final de carrera, però encara no està a la seva fase final. El motiu no és un altre que falta la peça al voltant la qual gira aquest projecte, el solver BlockIP.

Així doncs, quan el solver estigui acabat segurament es tingui que adaptar aquesta plataforma als probables canvis que tingui el solver, així com afegir possibles noves funcionalitats que apareguin.

Altra possible extensió és la d'explotar l'estructura interna de cada bloc. Com s'ha comentat referent a la classe MatrixBlockIP, aquesta matriu es pot definir com a diferents tipus de matrius, com per exemple, diagonal, de xarxa, identitat, etc. Però actualment no s'exploten aquestes estructures a través de la plataforma. Així doncs una possible extensió és analitzar l'entrada de dades per extreure l'estructura.

També, tot i que tot el codi escrit s'ha realitzat nativament en Unix, s'ha fet pensant en la compatibilitat amb Windows, i de fet s'han fet proves per verificar la seva correcte compilació. Però no tots els codis que utilitza la plataforma són directament compilables amb Visual Studio, així doncs, altra possible millora seria portar els codis de terceres parts a Windows.

Més possibles extensions són la reoptimització, recordem que amb aquest software és poden resoldre problemes molt grans que poden trigar hores en arribar a un punt òptim. Així que pot ser interesant que donat un límit de temps o d'iteracions, en el moment que s'arribi a aquest límit, en comptes de parar l'execució, mostrar el resultat actual i demanar a l'usuari si vol continuar amb l'optimització.

8. Conclusions

Un cop finalitzat el projecte i mirant enrere em dono compte que he aprés molt. Quan em van proposar de fer aquest projecte ni tan sols sabia que és el que faria realment, l'únic que sabia de tot el que he utilitzat era C++.

Així doncs estic content ja que durant aquest últim any he aprés coses que ni sabia de la seva existència i l'aprendre sobre optimització m'ha obert la visió a molts camps on podria, i de fet és, ser aplicada.

També agraeixo la llibertat que he tingut per fer aquest projecte, tot i ser un projecte d'empresa. Jo mateix he pogut investigar quines eines utilitzar i de quina manera plantejar l'estructura de la plataforma. Al final el resultat ha sigut satisfactori.

Segurament externament aquest projecte no cridi massa l'atenció al ser molt poc visual. Tot i això crec que s'ha fet una plataforma que pot fer més còmode la vida a l'usuari. I al final la informàtica és per això, per fer més còmode la vida als usuaris. A més, com l'àmbit de l'optimització està més centrat en la ciència que en l'usuari, no està de més que existeixen més plataformes pensades per aquest.

Tot i que el meu futur me l'imagino lluny d'aquests temes, la meva passió és la fotografia i vull unir informàtica amb fotografia, crec que l'optimització podria tindre diversos usos en el que vull fer. Així que en definitiva aquest treball m'ha aportat molt.

9. Bibliografia

[1] «NEOS Optimization Guide», http://neos-guide.org/Optimization-Guide, Últim accés: 7 Gener 2013.

[2] «wxFormBuilder HomePage», 14 Agost 2011. http://sourceforge.net/apps/mediawiki/wxformbuilder/index.php, Últim accés: 7 Gener 2013.

[3] Bruce A. McCarl, Alex Meeraus, Paul van der Eijk, Michael Bussieck, Steven Dirkse, Pete Steacy, Franz Nelissen, «McCarl GAMS User Guide», GAMS Development Corporation, 2013.

[4] COIN-OR, «Osi home page», https://projects.coin-or.org/Osi, Últim accés: 7 Gener 2013.

[5] Dash Optimization, «XPRESS Optimizer Reference Manual», DASH, 2011.

[6] David M. Gay, «Hooking Your Solver to AMPL», Technical report, Bell Laboratories, Murray Hill, NJ, 1993; revised 1994, 1997.

[7] Dimitri van Heesch, «doxygen Manual for version 1.8.3», 1997-2012.

[8] GNU Linear Programming Kit, «Reference Manual for GLPK Version 4.45», DRAFT, 2010.

[9] ILOG CPLEX, «ILOG CPLEX 11.0 Reference Manual», ILOG, 2007.

[10] Jordi Castro, «An interior-point approach for primal block-angular problems», Computational Optimization and Applications, nº 36, pp. 195-219, 2007.

[11] Jordi Castro, Jordi Cuesta, «Quadratic regularizations in an interior-point method for primal block-angular problems», Mathematical Programming, nº 130, pp. 415-445, 2011.

[12] Juan Soulié, «C++ Language Tutorial», 2007.

[13] John Forrest, David de la Nuez, Robin Lougee-Heimer, «CLP User Guide», IBM Coportation, 2004.

[14] John W. Gregory, Robert Fourer, Optimization Technology Cente, «Linear Programming FAQ», 1 September 2005. http://neos-guide.org/lp-faq, Últim accés: 7 Gener 2013.

[15] Julian Smart, Robert Roebling, Vadim Zeitlin, Robin Dunn, et al., «wxWidgets 2.8.12», 2011.

[16] Marco Colombo, Andreas Grothey, Jonathan Hogg, Kristian Woodsend, Jacek Gondzio, «SML User's Guide», School of Mathematics and Maxwell Institute, University of Edinburgh, UK, 2011.

[17] Microsoft, «Visual Studio Overview», http://www.microsoft.com/visualstudio/eng/products/visual-studio-overview, Últim accés: 7 Gener 2013.

[18] Novell, Inc. and contributors, «openSUSE Reference», 2011.

[19] Optimization Technology Center, «Nonlinear Programming FAQ», 22 Gener 2007. http://neos-guide.org/non-lp-faq, Últim accés: 7 Gener 2013.

[20] Robert Fourer, David M. Gay, Brian W. Kernighan, «AMPL: A Modeling Language for Mathematical Programming», 2003.

[21] Ted Ralphs, Menal Güzelsoy, Ashutosh Mahajan, «SYMPHONY 5.4.0 Users Manual», 2011.

[22] The MathWorks, Inc., «Matlab R2012b», 2012.

10. Annex

```
usage: BlockIPPlatform mode [options] [additional options]
where mode is:
  If you want to solve a problem must choose an input format:
  -ampl {model filename} {data filename}
    Input problem in sml extended ampl format.
  -mps {mps filename}
    Input problem in mps extended to blocks format.

  If you want to convert a problem from sml extended ampl to mps
  extended to blocks format:
  -convert {model filename} {data filename} {mps filename}
  [-standarize] [-outDir {path}]
    Converts a problem from sml extended ampl to mps extended to
    blocks format.
  -standarize
    Standarize the problem before write the mps converting all
    restrictions in equalities and all variables with 0 lower bound.

options
  Optionally you can set some options:
  -solver {solver}
    Set the solver to use to solve the problem. Depends of your
    installation but are allowed blockip, cplex, xpress, glpk and clp,
    by default blockip is used.
  -outDir {path}
    Set the directory where the output files will we written, must
    exist, by default it is the actual directory.
  -logFile {filename}
    Set the name of the log file, by default the same that input file
    with extension .log.
  -solFile {filename}
    Set the name of the solution file, by default the same that input
    file with extension .sol.
  -time {number of seconds}
    Set the maximum number of seconds that the solver can use to solve
    the problem.
  -threads {number of threads to use}
    Set the maximum number of threads that the solver can use to solve
    the problem, by default unlimited.
  -pathMatlab {path}
    If you want to use the Matlab implementation of BlockIP you need
    prblock_ip.m file. The default directory where is expected to find
    it is "../prblock_ip".

additional options
  If the solver to use is BlockIP, you can set addicional options, for
  the default values see the constant terms in BlockIP.h:
  -interactive
    After solving the problem starts a interactive mode to retrieve
    information
  -inf {value}
    Infinity value.
  -sigma {value}
    Reduction of the centrality parameter at each IP iteration.
  -rho {value}
    Reduction of the step-length for the primal and dual variables at
    each IP iteration.
```

```
  -gap {value}
    Optimality gap tolerance.
  -pfeas {value}
    Primal feasibility tolerance.
  -dfeas {value}
    Dual feasibility tolerance.
  -outFreq {value}
    Output information lines will be printed each selected IP -
    iterations.
  -iter {value}
    Maximum number of IP iterations.
  -initPcgtol {value}
    Initial tolerance for the conjugate gradient (by default
    PCG_TOL_LIN if linear problem, PCG_TOL_QUAD if quadratic or -
    convex- nonlinear).
  -minPcgtol {value}
    Minimum tolerance for the conjugate gradient.
  -iterPcg {value}
    Maximum number of pcg iterations (if 0, then the value will be
    computed by the code).
  -redPcgtol {value}
    Reduction of pcg_tol at each IP iteration.

Interactive mode
  help    Show short help
  help -v    Show extended help
  pobj    Show primal objective function
  dobj    Show dual objective function
  it    Show the number of IP iterations performed
  cgit    Show the overall number of PCG iterations performed
  exit    Ends the interactive mode

  Indices start in 0.
  Display the primal variables value of blocks and slacks
  x
    Show all the primal variables of blocks and slacks
  x {i}    Example: x 5
    Show the value of the i-th primal variable
  x {name}    Example: x Block1:Var5
    Show the value of the named primal variable
  x {k} {i}    Example: x 1 5
    Show the value of the i-th primal variable inside the block k
  x {blockName} {i}    Example: x Block1 5
    Show the value of the i-th primal variable inside the named block
  x block {k}    Example: x block 1
    Show the value of all primal variables inside the block k
  x block {blockName}    Example: x block Block1
    Show the value of all primal variables inside the named block
  x slack    Example: x slack
    Show the value of all primal variables inside the slacks
  x slack {i}    Example: x slack 3
    Show the value of the i-th primal variable inside the slacks

  Display the reduced cost of primal variables of blocks and slacks
  rc
    Show all the reduced cost of primal variables of blocks and slacks
  rc {i}    Example: rc 5
    Show the reduced cost of the i-th primal variable
  rc {name}    Example: rc Block1:Var5
    Show the reduced cost of the named primal variable
  rc {k} {i}    Example: rc 1 5
```

```
    Show the reduced cost of the i-th primal variable inside the block
    k
  rc {blockName} {i}    Example: rc Block1 5
    Show the reduced cost of the i-th primal variable inside the named
    block
  rc block {k}    Example: rc block 1
    Show the reduced cost of all primal variables inside the block k
  rc block {blockName}    Example: rc block Block1
    Show the reduced cost of all primal variables inside the named
block
  rc slack    Example: rc slack
    Show the reduced cost of all primal variables inside the slacks
  rc slack {i}    Example: rc slack 3
    Show the reduced cost of the i-th primal variable inside the
    slacks

  Display the dual variables value of blocks and linking constraints
  cons
    Show all the dual variables of blocks and linking constraints
  cons {i}    Example: cons 5
    Show the dual variable of the i-th constraint
  cons {name}    Example: cons Block1:Cons5
    Show the dual variable of the named constraint
  cons {k} {i}    Example: cons 1 5
    Show the dual variable of the i-th constraint inside the block k
  cons {blockName} {i}    Example: cons Block1 5
    Show the dual variable of the i-th constraint inside the named
block
  cons block {k}    Example: cons block 1
    Show all the dual variables inside the block k
  cons block {blockName}    Example: cons block Block1
    Show all the dual varialbes inside the named block
  cons linking    Example: cons linking
    Show all the dual varialbes inside the linking constraints
  cons linking {i}    Example: cons linking 3
    Show the dual variable of the i-th constraint inside the linking
    constraints
```