Università di Padova Facoltà di Ingegneria

Corso di Laurea in Ingegneria Informatica

Tesi di Laurea

—

UN APPROCCIO COMBINATORIO ALLA DECOMPOSIZIONE DI BENDERS

Laureando: Gianni Codato
(matricola 423878/IF)

Relatore: Prof. Matteo Fischetti

14/04/03

Indice

Sommario

Con questa tesi si é studiata e realizzata una nuova strategia di risoluzione di problemi di ottimizzazione lineare misti-interi (MILP o piú semplicemente MIP).

Il metodo sviluppato parte essenzialmente dall'idea di sfruttare la decomposizione di Benders, generando peró tagli di tipo completamente diverso ("combinatori") ed é poi stato arricchito in vario modo.

I risultati sperimentati sono costituiti da un eccezionale miglioramento nella risoluzione di problemi (fino ad ora ritenuti estremamente difficili) con funzione obiettivo contenente solo variabili intere. Nel caso di funzione obiettivo con anche (o solo) variabili continue il metodo non sembra portare particolari benefici, ma é tuttora allo studio una sua variante che sembra promettente.

Capitolo 1

Introduzione

Nell'ambito delle problematiche riguardanti la Ricerca Operativa, il trattamento dei problemi descrivibili tramite modelli MILP (Mixed Interger Linear Programming) costituisce un settore di ricerca molto attivo. Questa tesi si inserisce proprio in questo filone, proponendo un approccio abbastanza diverso da quelli tradizionali (spesso basati sullo studio di separatori di carattere generale o specifico a seguito di analisi poliedrale su certe classi di problemi) e per questo motivo innovativo.

L'idea di base che si intende realizzare ne riprende una introdotta da J.F. Benders nella quale si cercava di suddividere i problemi misti nelle loro componenti intera e continua, risolvendo direttamente solo la prima (la più importante e difficile) e mantenendo la seconda come sistema di verifica e supporto. Al fine di ripristinare nella parte intera "l'informazione" sottratta dal modello originale escludendo la parte continua, il sistema di Benders prevedeva la generazione di particolari disequazioni da reintegrare nella prima. In questa tesi viene completamente modificata la gestione della parte continua ed il tipo di equazioni che vengono generate. Questo nuovo sistema introdotto risulta particolarmente adatto a problemi di ottimizzazione combinatoria con variabili binarie, specie se queste vengono usate per controllare la disattivazione di alcuni vincoli lineari (tecnica dal big–M). Per questo motivo l'implementazione e la verifica dell'efficacia del metodo ideato verrá fatta solo su questa classe specifica di problemi, sebbene venga anche introdotta la teoria di base nella sua forma più generale (si veda l'appendice A).

Lo scopo di questa tesi risulta quindi quello di descrivere, implementare e verificare il nuovo approccio alla decomposizione di Benders basato su tagli "combinatori".

Per una spiegazione generale relativa ai piú classici sistemi di risoluzione di problemi MILP, oltre che ILP ed LP si può far riferimento a [1].

In [2] è possibile reperire una descrizione abbastanza aggiornata sullo stato dell'arte per quel che rigurada le prestazioni di software per la risoluzione di problemi di programmazione lineare.

Nell'articolo [3] si puó trovare la spiegazione dello stesso Benders della sua idea; altrimenti si possono trovare altre varie fonti che ne riportano il contenuto, tra cui [4] e [5].

Si descrive ora brevemente la struttura di questa tesi riportando in sintesi il contenuto dei vari capitoli.

Tagli Combinatori di Benders Viene esposta, prima in modo intuitivo, poi piú formalmente, la teoria che sta alla base del sistema di risoluzione esposto in questa tesi. In chiusura di capitoli vi é la descrizione dell'algoritmo euristico di ricerca di Sottoinsiemi Minimali Impossibili (MIS) che é utilizzato per generare i tagli combinatori.

Strumenti software utilizzati Sono presenti delle sezioni dedicate all'utilizzo di Ilog Cplex e del General B&C Framework, due librerie di software (la prima commerciale, la seconda sviluppata dall'autore di questa tesi in collaborazione il collega Giuseppe Andreello) usate quale supporto al programma sviluppato.

Implementazione dell'algoritmo In questo capitolo si espongono le caratteristiche salienti del programma che é stato creato e che implementa l'algoritmo esposto nel capitolo 2. Vi sono inoltre descritte alcune importanti variazioni fatte alla teoria di base allo scopo di migliorarne l'efficienza.

Risultati Computazionali Il capitolo comprende una descrizione della collezione di problemi utilizzati per eseguire le prove comparative sull'efficienza del software prodotto, nonché, ovviamente, i risultati ottenuti. Vi é anche la descrizione di una ulteriore importante modifica introdotta per migliorare le prestazioni per alcuni tipi di problemi.

Conclusione Vengono riassunti scopi, obiettivi raggiunti e sviluppi futuri del lavoro portato a termine in questa tesi.

Estensioni al modello Si descrivono due estensioni al modello particolarmente interessanti: la prima permette di trattare alcuni modelli non lineari, la seconda amplia di molto l'insieme di problemi lineari al quale é possibile applicare la teoria sviluppata.

Framework In questa appendice si trova un manuale di riferimento (reference) relativo al software General B&C Framework.

Capitolo 2

Tagli Combinatori di Benders

2.1 Discussione preliminare: nascita dell'idea

L'idea di fondo che sta alla base dell'approccio che si é implementato nasce dall'accostamento di tre elementi: la decomposizione di Benders, la volontá di eliminare problemi sia di instabilitá numerica che di modellizzazione poco efficace di molti problemi MIP, l'esistenza di un ottimo metodo di soluzione per problemi di map labelling.

Prima di spiegare essi abbiano portato al metodo qui implementato é perció opportuno fornirne una breve panoramica.

2.1.1 Decomposizione di Benders

In questa sotto–sezione si esporrá brevemente la teoria di base della decomposizione di Benders. Dato un qualsiasi sistema lineare intero misto (ovvero comprendente sia variabili continue che intere), esso puó essere scritto nella seguente forma:

$$z^\star = \min \quad c^T x + d^T y \tag{2.1}$$

$$\text{s. t.} \quad Qx = q \tag{2.2}$$

$$Ax + By \geq b \tag{2.3}$$

$$x \geq 0, \qquad \text{integer} \tag{2.4}$$

$$y \geq 0, \tag{2.5}$$

nella quale si sono semplicemente messe in evidenza la parte continua e quella intera del modello. Da questa formulazione si puó facilmente passare alla seguente:

$$
\begin{aligned}
z^\star = \min \quad & c^T x + \psi(x) && (2.6)\\
\text{s.t.} \quad & Qx = q && (2.7)\\
& x \geq 0, \qquad \text{integer} && (2.8)
\end{aligned}
$$

dove

$$
\begin{aligned}
\psi(x) = \min \quad & d^T y && (2.9)\\
\text{s.t.} \quad & By \geq b - Ax && (2.10)\\
& y \geq 0, && (2.11)
\end{aligned}
$$

In pratica non si é fatto altro che scorporare il modello originale in due parti. La prima, che chiameremo *master*, é composta dalla "parte intera" del modello originale piú la funzione $\psi(x)$ che incapsula e nasconde tutto quanto manca. La seconda, che chiameremo *slave*, é invece composta dalla "parte continua". Si fa notare che la presenza della funzione $\psi(x)$ rende non-lineare la funzione obiettivo del master e che la parte slave risulta essere un semplice modello LP (cioé senza variabili intere) in quanto le variabili x ne costituiscono solamente un parametro. Grazie alla teoria della dualitá possiamo a questo punto affermare che:

- per la dualitá forte si ha

$$
\begin{aligned}
\psi(x) = \max \quad & u^T (b - Ax) && (2.12)\\
\text{s.t.} \quad & u^T B \leq d^T && (2.13)\\
& u \geq 0; && (2.14)
\end{aligned}
$$

- per la dualitá debole si ha che, per ogni $u \geq 0$ soluzione base di $B^T u \leq d$, vale la disuguaglianza

$$\psi(x) \geq \; u^T (b - Ax).$$

Sulla base di queste considerazioni é possibile riformulare il modello originale come segue:

$$
\begin{aligned}
z^\star = \min \quad & c^T x + \delta && (2.15)\\
\text{s.t.} \quad & Qx = q && (2.16)\\
& \delta \geq u^T b - u^T A x, \quad \forall u \geq 0 \text{ soluzione base di } B^T u \leq d && (2.17)\\
& x \geq 0, \qquad \text{integer} && (2.18)\\
& \delta, \qquad \text{libera.} && (2.19)
\end{aligned}
$$

Come si puó osservare questo modello contiene una sola variabile continua (δ). Tuttavia le disequazioni 2.17 sono in numero generalmente troppo elevato per poter essere gestite esplicitamente inserendole in modo statico in un modello concreto. La soluzione consiste nel gestirle dinamicamente, generando ed aggiungendo al modello solo quelle strettamente indispensabili. In definitiva l'approccio che si puó utilizzare é quello di risolvere il modello 2.15 - 2.19, privato delle disequazioni 2.17, tramite branch & cut aggiungendo di volta in volta, in qualitá di tagli, quelle tra le disequazioni 2.17 che risultano violate.

La procedura di separazione che cerca di individuare quali tra le disequazioni 2.17 risultano violate dovrebbe quindi ricevere in input i valori $x^\star$ e $\delta^\star$, soluzione corrente del problema master, e ricercare (se esiste) un vettore $\hat{u} \geq 0$ tale che:

$$B^T\hat{u} \leq d \;\text{ e }\; \delta^\star < \hat{u}^T b - \hat{u}^T A x^\star.$$

Un tale vettore puó ovviamente essere facilmente calcolato come segue. Dapprima si risolve il seguente problema lineare:

$$z_S = \max \quad u^T b - u^T A x^\star \tag{2.20}$$
$$B^T u \leq d \tag{2.21}$$
$$u \geq 0. \tag{2.22}$$

Si puó facilmente osservare che questo problema non é altro che il problema duale dello slave in cui si sia sostituita la soluzione $x^\star$ del nodo corrente. Detta $(z_S^\star, u^\star)$ la soluzione ottima di quest'ultimo problema [1] si verifica se $\delta^\star < z_S^\star$. Se sí allora il vettore cercato $\hat{u}$ esiste ed é naturalmente pari a $u^\star$: quindi si puó generare il taglio

$$\delta \geq u^\star b - u^\star A x,$$

che risulta ovviamente violato dalla soluzione del nodo corrente del problema master. Altrimenti un tale vettore non esiste e perció non é possibile generare alcun taglio; si lascia quindi che l'algoritmo di branch & cut segua il suo decorso senza aggiungere alcun vincolo.

2.1.2 Problemi numerici e teorici di alcuni modelli MIP

Tra i vari modelli di programmazione mista intera, ne esistono alcuni di grande importanza pratica che risultano spesso molto difficili da risolvere a causa di problemi dovuti sia ad instabilitá numerica che ad una non troppo efficace modellizzazione.

Questi modelli verranno qui chiamati modelli "con coefficiente big–M" (il perché sará presto chiaro). Ció che caratterizza tali problemi di ottimizzazione é il fatto di dover rappresentare delle situazioni in cui alcuni dei vincoli lineari che compongono il modello devono essere "attivi" solo se una certa variabile, anch'essa appartenente al problema, assume un determinato

[1] Supposto limitato per semplicitá di trattazione

valore. In pratica questi sono problemi in cui compaiono vincoli di questo tipo:

$$z_k = 1 \Rightarrow a_i^T x \geq b_i, \tag{2.23}$$

dove z_k é una variabile 0–1 (binaria).

Chiameremo z_k "variabile di controllo" e la disequazione/equazione alla destra del segno di implicazione logica "vincolo ad attivazione condizionale" (o piú semplicemente vincolo condizionale). Chiameremo inoltre, per differenziarli, vincoli statici le restanti normali disequazioni lineari che possono essere presenti.

Come si vede, in questo tipo di problemi, il legame esistente tra le variabili di controllo e le variabili del vincolo ad attivazione condizionale é di tipo non–lineare. Tuttavia, il fatto che la parte ad attivazione condizionale sia una disequazione lineare, fa percepire il modello in questione come "quasi lineare". Per questo quello che si tende a fare é rimodellare il problema tramite una formulazione meno naturale ma lineare: cosí non é necessario inventare tecniche di soluzione ad hoc bensí é possibile ricorrere ai ben noti, collaudati e robusti metodi della Ricerca Operativa.

La tecnica piú comune é quella di riscrivere le equazioni tipo 2.23 in questa forma:

$$a_i^T x \geq b_i - M(1 - z_k), \tag{2.24}$$

dove M é un coefficiente positivo sufficientemente grande (big–M), dal quale prende il nome questo tipo di formulazione. Come é facile osservare quando $z_k = 1$ il vincolo 2.24 si riduce a quello ad attivazione condizionale. Quando invece $z_k = 0$ il vincolo si riduce a $a_i^T x \geq b_i - M$: se M é scelto sufficientemente grande (sulla base di segno e modulo degli a_{ij} e dei lower e upper bounds delle variabili x_j) si puó fare in modo che il vincolo sia sempre soddisfatto, qualsiasi siano i valori assunti dalle variabili x_j, rendendolo di fatto "disattivato".

Questa soluzione sembra, a prima vista, semplice ed elegante; tuttavia nasconde alcune problematiche.

Problemi numerici Il primo tipo di problemi che si riscontrano sono di instabilitá numerica. Spesso il coefficiente M deve essere scelto molto grande (almeno rispetto agli altri coefficienti del problema) per garantire la disattivazione del vincolo. La presenza di numeri con troppa diversitá nell'ordine dei grandezza causa generalmente l'insorgere di problemi negli algoritmi di risoluzione: questi si manifestano in rallentamenti dovuti alla ripetuta applicazione/rimozione di perturbazioni oppure, nei casi peggiori in soluzioni errate o nell'impossibilitá di portare a termine la computazione. C'é comunque da notare che i notevoli progressi compiuti dai software risolutori negli ultimi anni hanno notevolmentente ridotto l'impatto di queste problematiche.

Problemi connessi al sistema di risoluzione Il secondo tipo di problemi che si verificano sono direttamente connessi all'interazione che si instaura tra l'approccio di risoluzione usato ed il tipo di modello utilizzato. Come noto i problemi MIP vengono risolti tramite branch & bound: ció comporta il fatto che si risolvano una sequenza di problemi LP che derivano dal problema originale tramite rimozione del vincolo di interezza sulle variabili intere. In questo modo si permette alle variabili di controllo (che normalmente sono binarie) di assumere un qualsiasi valore frazionario nell'intervallo $[0, 1]$. La maggiore libertá che ne deriva permette solitamente all'algoritmo di risoluzione di assegnare alle variabili di controllo valori prossimi a zero (ovvero sufficientemente piccoli da causare la disattivazione di piú vincoli condizionali possibili) ma non esattamente nulli (in modo che siano comunque tali da soddisfare i vincoli statici che le coinvolgono). Tutto questo comporta il fatto che i lower bound individuati siano solitamente molto distanti dalle soluzioni a valori interi (grazie al fatto che molte disequazioni ad attivazione condizionale sono disattivate, producendo quindi un problema meno vincolato dell'originale) e che il modello rilassato si avvicini all'originale solo a livelli profondi dell'albero di branching (solo a seguito del fissaggio di variabili a valori interi). In definitiva si puó affermare che la presenza dei coefficienti big–M allontana maggiormente il modello dall'originale affidando al solo branching il compito di riavvicinarli, il che comporta tempi di risoluzione lunghi ed alberi di branching dalle dimensioni ingestibili.

2.1.3 Problemi di Map Labelling

In questa sezione si dará una descrizione, per sommi capi, dei problemi di map labelling e di una possibile modalitá di soluzione (per una trattazione piú dettagliata si puó consultare [7] o [6]).

Questo tipo di problemi consiste nel massimizzare il numero di etichette (label) che é possibile apporre su un certo spazio bidimensionale: le etichette sono rappresentate da rettangoli che ne descrivono lunghezza ed altezza (ovvero l'area che queste occupano); ogni etichetta é associata ad un preciso punto dello spazio (point feature) e, se viene collocata, deve essere nei pressi di tale punto (questa condizione si traduce nell'imporre che il punto giacia sul bordo del rettangolo dell'etichetta corrispondente); inoltre le etichette (ovvero i rettangoli) non devono sovrapporsi. Un classico caso concreto modellabile in questo modo é quello di massimizzare il numero di nomi di cittá apponibili su una carta geografica, mantenendo la massima leggibilitá della stessa (cioé senza sovrapposizioni), situazione dal quale prende il nome la classe di problemi.

Il sistema di risoluzione efficiente ha come base l'utilizzo di due grafi di precedenze, uno per ogni dimensione del problema, i quali descrivono i possibili posizionamenti delle etichette nella mappa. Ognuno di questi grafi puó essere pensato come composto da due parti: una sempre presente (statica), serve a descrive la posizione dei point feature e la dimensione delle etichette, l'altra (dinamica — composta di soli archi) serve a descrive i vincoli di non sovrapposizione tra etichette e di prossimitá tra point feature e corrispondenti etichette. La dinamicitá dipende dal fatto che questa parte varia per ogni possibile scelta di etichettatura della mappa, ovvero per ogni insieme di etichette che si desidera posizionare esiste una diversa parte dinamica e quindi un diverso grafo: solo gli archi che sono relativi alle etichette presenti saranno inseriti nei due grafi. Ció che si riesce a dimostrare é che una soluzione é valida per il problema di map labelling (ovvero rispecchia le condizioni di non sovrapposizione tra etichette e di prossimitá tra etichetta e point feature) se e solo se non vi sono cicli positivi nei due grafi che la descrivono.

Sulla base di questa considerazione si puó implementare un algoritmo di risoluzione che funziona come segue. Ad ogni etichetta si associa una variabile binaria il cui valore indica se l'etichetta sará presente o meno nella soluzione; ad ogni arco della parte dinamica si associa una seconda variabile binaria che indica se tale arco é presente o meno nel grafo che descrive una soluzione. Si scrivono un insieme di vincoli che indicano quali archi vanno posti nel grafo quando una certa etichetta o coppia di etichette entrano a far parte di una soluzione: si dimostra facilmente che tali vincoli sono lineari. Aggiungendo ad essi una funzione obiettivo che tende a massimizzare il numero di etichette da apporre, ció che si ottiene é un modello ILP. Tale modello ignora peró proprio le condizioni di non sovrapposizione e di prossimitá che sono proprie del problema di map labelling e quindi, in generale, le violerá. A questo punto entra in gioco la coppia di grafi: sulla base della soluzione fornita dal modello ILP si completano le parti dinamiche dei due grafi, si scoprono tutti i cicli positivi in essi contenuti e si genera una disequazione di questo tipo per ognuno di tali cicli:

$$\sum_{i \in C} x_i \leq |C| - 1, \tag{2.25}$$

dove x_i sono le variabili binarie[2] associate agli archi che formano il ciclo e C é l'insieme degli indici delle variabili alle quali é associato un arco che partecipa al ciclo. Una disequazione di questo genere impedisce che tutte le variabili coinvolte assumano il valore 1 e quindi, una volta aggiunta al modello ILP, "taglia" la soluzione corrente ed evita il riformarsi del ciclo nel grafo corrispondente alla nuova soluzione generata. Alla fine si otterrá una soluzione cui non corrisponde piú alcun ciclo positivo nei grafi ad essa associati: quella sará la soluzione ottima. Il sistema puó essere migliorato se, al posto di aspettare di ottenere una soluzione intera del modello ILP, si opera anche su soluzioni non completamente intere, fornite ad esempio dai

[2]Si é implicitamente assunto che $x_i = 1$ significhi arco presente nel grafo e $x_i = 0$ arco assente

nodi intermedi dell'albero di branching che si genera se si risolve il problema ILP tramite branch & cut. [3]

Un problema di map labelling ammette peró anche una formulazione MIP che riesce a descrivere l'intero problema con vincoli lineari e quindi non necessita della presenza dei grafi di precedenza. Tale modello contiene tutti i vincoli del modello ILP originale piú altri vincoli ed altre variabili continue:

- ogni variabile continua rappresenta un nodo dei grafi di precedenza
- per ogni arco statico (che rappresenta un vincolo di precedenza tra due nodi) si scrive il corrispondente vincolo lineare di precedenza tra le variabili associate ai nodi alle sue estremitá
- per ogni arco dinamico si fa la stessa cosa ma in piú si aggiunge al vincolo che ne risulta un termine costituito dalla variabile binaria che controlla la presenza o l'assenza dell'arco nel grafo moltiplicata per un coefficiente sufficientemente grande: in questo modo si simula l'assenza o la presenza del vincolo nel modello grazie alla tecnica del big–M.

Alla luce di quanto detto fino ad ora si puó perció riassumere nel modo seguente un metodo efficace per risolvere un problema di map labelling che sia fornito tarmite il suo modello MIP:

1. si estrae dal modello MIP la parte intera (ILP) in esso contenuta
2. si ricavano dalla restanti disequazioni le parti statiche e dinamiche dei grafi di precedenza
3. si appronta una procedura di separazione per un sistema branch & cut, la quale, sulla base della parte intera della soluzione che gli viene fornita, attiva gli archi "dinamici" opportuni, verifica la presenza di cicli positivi, genera e restituisce i tagli associati a tali cicli
4. si risolve tramite branch & cut il problema ILP estratto utilizzando il separatore approntato al punto prcedente.

2.2 Idea di base

Ecco come dai tre elementi indicati si puó comprendere l'idea che sta alla base del metodo sviluppato in questa tesi. Se si cerca di risolvere un problema di map labelling tramite la decomposizione di Benders si osserva che essa ha delle analogie con il meccanismo di risoluzione appena esposto: é infatti sufficiente accostare il problema master al modello ILP del map labelling e

[3]Naturalmente in questo caso le disequazioni generate dai grafi assumeranno il valore di tagli atti a separare la soluzione corrente

si nota che in entrambi i casi quel che si fa é estrarre un sottoinsieme di equazioni (le stesse), ampliando cosí l'insieme delle soluzioni ammissibili. A questo punto, si cerca di risolvere questo problema che risulta semplificato, verificando in opportuni momenti, che le soluzioni trovate soddisfino anche le disequazioni che erano state tolte. Se ció avviene allora tali soluzioni sono anche soluzioni del problema originale, altrimenti occorre fornire nuovamente al master quell'informazione che é stata sottratta scorporando da esso le disequazioni finite nello slave o trasformate in grafi di precedenza. Per far questo si sfrutta la caratteristica del branch & cut di poter aggiungere disequazioni run-time. Le differenze tra i due sistemi citati (Benders Decomposition e algoritmo per il map labelling) consistono nelle modalitá di verifica delle soluzioni trovate e nella tipologia di disequazioni (tagli) generate[4]. Nel primo caso infatti affinché la soluzione sia valida occorre che, una volta sostituita nello slave, quest'ultimo sia feasible: se cosí non é allora il taglio generato é una opportuna combinazione lineare delle disequazioni dello slave. Nel secondo caso invece la validitá della soluzione é determinata dall'assenza di cicli positivi nei grafi di precedenza mentre i tagli generati sono costituiti da una particolare somma algebrica a coefficienti interi delle variabili intere.

Dato che la parte master é la stessa risulta chiaro che l'informazione apportata dai tagli aggiunti (per quel che riguarda i valori interi assumibili o meno dalle variabili intere) nei due casi deve essere la stessa (sebbene possano essere diversi i rilassamenti continui che tali tagli inducono). Ed in effetti si puó rendere l'analogia piú evidente se si considerano i seguenti cambiamenti:

- si particolarizza il metodo di Benders alla situazione in cui non ci sono variabili continue nella funzione obiettivo
- nei problemi di map labelling si descrive il grafo di precedenze tramite la matrice di incidenza e le disequazioni che si possono associare in modo naturale ai suoi archi

Grazie a queste due modifiche si osserva che in entrambi i casi i due metodi non fanno altro che osservare la feasibility di un opportuno sistema lineare (slave o LP corrispondente al grafo di precedenze) per decidere se generare o meno dei tagli. La differenza si restringe quindi al solo tipo di disequazioni che vengono aggiunte. Da questo punto di vista l'utilizzo di un garfo di precedenze nei problemi di map labelling si configura come una semplice particolarizzazione della decomposizione di Benders che é possibile grazie alla peculiare struttura del problema: una volta sostituiti i valori interi della soluzione del master, la matrice dei coefficienti dello slave puó essere

[4]Esiste anche una terza differenza dovuta alle diverse "parti di soluzione" che é possibile usare nella verifica (cioé intera o frazionaria); questa differenza é peró di minor conto e risulta ininfluente nell'economia del ragionamento

vista come la trasposta di una matrice di incidenza e quindi la traduzione dello slave in un grafo é estremamante semplice.[5]

A questo punto si puó analizzare la tipologia dei tagli prodotti: qui si riscontrano sia somoglianze che differenze. Per prima cosa si osserva che i coefficienti dei tagli di Benders hanno generalmente dei valori tali per cui l'assegnamemento del valore zero ad una variabile comporta il soddisfacimento del taglio per qualsiasi altra combinazione di valori interi delle altre variabili coinvolte. Questa proprietá é analoga anche per i tagli 2.25 generati nel map labelling. Ció riflette l'uguale necessitá che hanno i tagli di fornire al master la medisima informazione (che é contenuta nel problema originale ma non piú nel master), ovvero che alcune combinazioni di valori per certe variabili binarie non é ammissibile. La differenza sostanziale riguarda quindi solo i coefficienti/termine noto dei tagli: nel caso di Benders questi non sono necessariamente interi ed in piú sono (almeno per questo tipo di problemi) "grandi": risentono infatti della presenza dei termini big-M nella formulazione MIP iniziale del problema. I tagli generati dall'ispezione dei grafi di precedenza non presentano invece questo problema.

Il passo successivo sará quindi quello di cercare se esiste un metodo in grado di permettere l'utilizzo dei tagli pensati per il map labelling all'interno della strategia dovuta a Benders, genralizzandoli cosí ad un qualsiasi altro tipo di problema al fine di trasportare in una piú vasta gamma di casi i benefici (di tempo di risoluzione e di stabilitá numerica) dovuti all'assenza dei coefficienti big–M e sperimentati nel map labelling. Questa generalizzazione é stata inizialmente pensata e sperimentata in questa tesi solo per un sottoinsieme dell'intera classe di problemi di ottimizzazione combinatoria: le caratteristiche di questa classe di problemi e la descrizione dettagliata del metodo di risoluzione generalizzato sará il soggetto della sezione seguente.

2.3 Algoritmo di risoluzione: descrizione teorica

Come annunciato, in questa sezione ci si occuperá della descrizione dei problemi su cui si sono effettuate le prove e della descrizione del funzionamento dell'algoritmo.

2.3.1 Classe di problemi trattati

I problemi di cui ci occuperemo in questa tesi sono problemi che devono avere le seguenti caratteristiche:

- devono essere problemi effettivamente MIP, ovvero problemi che comprendono sia variabili continue che variabili intere (ed ovviamente deve

[5] Quel che si é detto é rigorosamente vero per le soluzioni master completamente intere; per quelle frazionarie l'analogia é meno forte, a causa di una leggera differenza nella gestione della "parte slave" fatta nel map labelling, ma é comunque presente

esserci almeno un vincolo di tipo "misto", ovvero un vincolo che coinvolge almeno una variabile continua ed una intera, altrimenti il problema potrebbe essere partizionato in due sottoproblemi indipendenti: uno LP e l'altro puramente intero)

- i vincoli misti devono avere la parte intera costituita da una sola variabile binaria.

In questa classe di problemi rientrano quasi tutti quelli che fanno uso della tecnica del big–M di cui si é giá discusso, ovvero quelli che fanno uso di una variabile binaria per controllare la "presenza o assenza" di un vincolo nel modello lineare.

Si vuole far notare che non si fa alcuna altra assunzione sui vincoli non misti, il che significa che possono esserci o meno vincoli che coinvolgono solo variabili intere o solo variabili continue; inoltre possono anche essere presenti delle variabili intere non binarie, a patto che esse non compaiano in vincoli misti. Infine si mette in evidenza che nessuna restrizione é stata fatta sulla funzione obiettivo: ció comporterá delle complicazioni e delle conseguenze che verranno chiarite successivamente.

2.3.2 Algoritmo di risoluzione: introduzione

Prima di esporre l'algoritmo di risoluzione in dettaglio nel suo complesso si vuole anticipare il nucleo su cui esso si basa: il tipo di tagli generati. Se si verifica che una certa soluzione master $\hat{x}$ induce uno slave infeasible, per informare il risolutore del problema master che la soluzione appena individuata non é ammissibile, si puó generare un taglio del tipo individuato nei problemi di map labelling, che qui riportiamo in una forma leggermente diversa e generallizzata:

$$\sum_{j:\hat{x}_j=0} x_j + \sum_{j:\hat{x}_j=1} (1 - x_j) \geq 1. \tag{2.26}$$

La generalizzazione consiste nel fatto che qui si é inclusa anche una seconda sommatoria ($\sum_{j:\hat{x}_j=0} x_j$, non presente nella disequazione 2.25) per tener conto delle variabili che contribuiscono all'infeasibility dello slave, avendo assunto un valore pari a zero.

Sebbene in linea di principio questo approccio sia sicuramente corretto, risulta peró completamente impraticabile anche in casi reali di piccole dimensioni: un problema di sole venti variabili potrebbe necessitare di oltre un milione di tagli di questo tipo, una quantitá davvero eccessiva. Tuttavia quando si é alle prese con un sistema slave infeasible, succede raramente che tutte le disequazioni che lo compongono siano la causa della sua infeasibility: ad esempio, nel caso dei problemi di map labelling l'infeasibility é connessa ai soli vincoli che descrivono il ciclo positivo formatosi nel grafo di precedenze associato allo slave. É quindi logico, dato un sistema senza soluzioni, restringere la propria analisi ad uno dei sotto–sistemi minimali impossibili (MIS) [6] che lo compongono. In questo modo si ottiene un sotto–sistema dello slave originale, ancora infeasible ma costituito da un numero generalmente molto minore di vincoli. Per questo sotto–sistema é ancora lecito e possibile scrivere un'equazione tipo 2.26.

Ció che si é cosí fatto puó essere spiegato in termini piú intuitivi in questo modo: l'aver fissato nel master le variabili intere a determinati valori, senza tener conto di tutti i vincoli del problema (che in parte sono stati tolti e posti nello slave) comporta che non necessariamente la soluzione intera trovata dal master stesso sia valida per il sistema originale nel suo complesso, ovvero non necessariamente porti ad uno slave feasible. Per questo si devono ricercare quali sono le variabili che hanno dei valori incongruenti (ovvero che non permettono alle altre variabili intere o continue di assumere alcun valore in grado di soddisfare tutti i vincoli): l'intero insieme di variabili del problema constituisce sicuramente un gruppo con questa caratteristica, ma tra queste ce ne possono essere alcune (ed in pratica saranno molte) che non sono causa di infeasibility, cioé il cui valore non é in "contrasto" con quello delle altre variabili. In altre parole ci sono delle variabili per le quali, anche se si cambiasse il valore loro assegnato dal master, lo slave resterebbe infeasible, ció significando che non é imputabile a tale variabile l'infeasibility stessa: tale variabile puó quindi essere esclusa dal gruppo totale ed il ragionamento puó essere ripetuto per le restanti variabili. Ció andrá avanti finché non si otterrá un gruppo di variabili per il quale vale la proprietá che anche escludendo una sola di esse dall'insieme preso in considerazione, l'infesibility dello slave cesserebbe, ovvero esiste almeno un valore possibile per una qualsisi di queste variabili che tramuta lo slave da problema infeasible in uno feasible. Ció significa che é proprio l'assegnamento di valori che il master ha proposto per questo insieme minimale di variabili a costituire il nocciolo dell'infesibility dello slave (a causarla): infatti se si cambiasse uno solo dei valori di queste variabili si potrebbe rendere lo slave feasible e d'altra parte l'infesibility dello slave non dipende dalle restanti variabili escluse.

Il fatto che si possa parlare indifferentemente di MIS dello slave e di sottoinsieme minimale di variabili intere discende direttamente dalla seconda condizione che si é imposta sulla classe di problemi che ci si propone di risolvere: tale condizione permette infatti di stabilire una corrispondenza biunivoca tra vincoli misti presenti nello slave e variabili intere (binarie) di controllo. In questo modo si puó passare naturalmente dal concetto di minimizzazione di un insieme di disequazioni infeasible a quello di minimizzazione dell'insieme di variabili coinvolte in un taglio e viceversa.

L'importanza dell'operazione di minimizzazione dei tagli é massima: basti pensare che un taglio che coinvolge ad esempio quattro variabili (nel solito

[6] Si veda 2.4 per una discussione piú approfondita sui MIS

problema giocattolo da venti variabili binarie) svolge il compito di un numero di tagli che puó arrivare fino a $2^{20-4} \approx 64000$ (nel caso peggiore sarebbe infatti necessario generare una disequazione per ogni possibile assegnamento delle sedici variabili non coinvolte).

Si puó osservare che nel caso dei problemi di map labelling la minimalitá del numero di variabili coinvolte nei tagli che si generano é implicita nell'algoritmo stesso: ogni taglio rappresenta un ciclo positivo e quindi, escludere una variabile dal taglio equivarebbe a togliere un arco dal grafo, distruggendo quindi il ciclo individuato. In una situazione piú generale, quale quella in cui ci si pone qui, é invece necessario ricercare la minimalitá dei tagli da aggiungere in modo esplicito: questo fatto ha rappresentato un grosso problema in termini di efficienza dell'algoritmo. Tuttavia, grazie all'equivalenza appena evidenziata tra minimalitá di un taglio e minimalitá di un infesible sub–system, si é in gardo di spostare un problema sull'altro indifferentemente: questo é stato di fondamentale importanza dato che, per quel che rigurda la ricerca di MIS si é in possesso di un algoritmo di grande efficienza. Questo algoritmo é spiegato nella sezione 2.4 ed é stato appositamente sviluppato per questa tesi, sebbene sia del tutto generico.

2.3.3 Algoritmo di risoluzione: dettaglio

Ora che si sono esposte in modo informale ed esemplificativo le idee e le motivazioni che hanno portato alla creazione di questo nuovo approccio risolutivo ai problemi MIP é possibile descrivere tutto l'algoritmo in modo piú formale ed esaustivo.

Sia da risolvere un problema di programmazione lineare intera-mista di questo tipo:

$$
\begin{aligned}
z^\star = \min \quad & c^T x + d^T y && (2.27)\\
\text{s.t.} \quad & Qx \ge q && (2.28)\\
& Mx + By \ge b && (2.29)\\
& Ry \ge r && (2.30)\\
& x \ge 0, \quad \text{integer} && (2.31)\\
& y \ge 0. && (2.32)
\end{aligned}
$$

con M [7] matrice con uno ed un solo elemento diverso da zero per ogni riga e con $x_i \in \{0,1\}$ (cioé variabile binaria) se $M_{(\cdot,i)} \neq 0$. Come si nota le uniche imposizioni che si sono fatte sono sulla forma dei vincoli di tipo 2.29 e si possono brevemente riassumere dicendo che in tali vincoli deve comparire una ed una sola variabile binaria. Per semplicitá chiameremo

[7] Si é denotata la matrice in questione con la lettera M perché i suoi elementi non nulli sono generalmente i coefficienti big–M

queste disequazioni *vincoli misti* e indicheremo la j–esima di queste con C_j. A parte la restrizione appena menzionata qualsiasi problema MIP puó essere posto nella forma indicata (eventualmente con l'ininfluente assenza di vincoli di tipo 2.28 o 2.30). Inoltre supporremo che esista almeno un vincolo misto (se cosí non fosse significherebbe che il problema si puó spezzare in due sottoproblemi piú semplici, uno LP ed uno di programmazione intera pura, per cui esistono tecniche ad hoc) e che il rilassamento continuo di questo problema sia feasible (in caso contrario, a meno di situazioni patologiche, anche il problema con i vincoli di interezza sarebbe infeasible o unbounded).

Per semplicitá supporremo, momentaneamente, che $d = 0$, trattando il caso $d \neq 0$ successivamente. Si decomponga quindi il problema nei due seguenti sotto–problemi:

- problema *master*:

$$z^\star = \min \quad c^T x \tag{2.33}$$
$$\text{s.t.} \quad Qx \geq q \tag{2.34}$$
$$x \geq 0, \quad \text{integer} \tag{2.35}$$
$$\tag{2.36}$$

- problema *slave*, con parametro $\tilde{x}$:

$$\min \quad 0^T y \tag{2.37}$$
$$\text{s.t.} \quad By \geq b - M\tilde{x} \tag{2.38}$$
$$Ry \geq r \tag{2.39}$$
$$y \geq 0. \tag{2.40}$$

Si risolva il problema master all'interezza. Se dovesse risultare infeasible, anche il problema originale lo sarebbe, altrimenti sia $x^\star$ la soluzione ottima[8]. Si sostituisca $x^\star$ nel parametro $\tilde{x}$ del problema slave, il quale diventa il seguente semplice sistema LP:

$$\min \quad 0^T y \tag{2.41}$$
$$\text{s.t.} \quad By \geq b^\star \tag{2.42}$$
$$Ry \geq r \tag{2.43}$$
$$y \geq 0, \tag{2.44}$$

con $b^\star = b - Mx^\star$.

Se tale problema risulta feasible, allora sia $y^\star$ la sua soluzione ottima. Si ha quindi che $(x^\star, y^\star)$ é la soluzione ottima del problema originale. Se infatti supponiamo per assurdo che esista un'altra soluzione $(x_1^\star, y_1^\star)$ che fornisca un

[8] Si trascura per semplicitá il raro caso in cui il problema master risulti illimitato

valore della funzione obiettivo migliore, in virtú dell'ipotesi $d = 0$ dobbiamo concludere che $x_1^\star$, che é ovviamente anche soluzione del problema master, fornisca un valore della funzione obiettivo migliore di $x^\star$, il che é impossibile essendo $x^\star$ la soluzione ottima del master.

Se invece il problema slave fosse infeasible (si nota che a causa della scelta della funzione obiettivo questo non puó essere illimitato) allora si individui un Minimal Infeasible Subsistem (MIS[9]) di questo problema. Sia F l'insieme degli indici delle variabili binarie x_i tali che almeno un vincolo misto di cui queste variabili fanno parte sia compreso nel MIS, ovvero

$$F = \{i|\exists j \text{ tale che } M_{j,i} \neq 0 \wedge C_j \in MIS\}.$$

Si aggiunga quindi al problema master il seguente taglio:

$$\sum_{j \in F : x_j^\star = 0} x_j + \sum_{j \in F : x_j^\star = 1} (1 - x_j) \geq 1. \tag{2.45}$$

A questo tipo di taglio si dá il nome di *Tagli Combinatori di Benders*, in inglese *Combinatorial Benders' Cut* (CBC). Come giá spiegato nelle sezioni precedenti questo taglio ha il compito di escludere dalla regione feasible del problema master una serie di soluzioni intere che non sono parte della proiezione della regione feasible del problema originale sul sotto–spazio delle sole variabili intere (tra queste soluzioni escluse vi é in particolare $x^\star$).

Se vi fossero altri MIS oltre al primo individuato é possibile iterare il procedimento indicato, generando ed aggiungendo al master un taglio per ogni nuovo MIS individuato.

A questo punto si risolva nuovamente il problema master arricchito con i tagli combinatori di Benders cuts individuati e si ricominci il procedimento da capo, fino a che o il problema master diventa infeasible (problema originale infeasible) o si giunge ad uno slave feasible (problema originale feasible).

Passiamo ora a trattare il caso in cui $d \neq 0$. In questa situazione non ha senso inserire la funzione obiettivo del problema originale nel master, dato che in nessun vincolo di quest'ultimo compaiono le variabili y. La soluzione piú semplice é allora quella di inserire la funzione obiettivo nello slave, sostituendovi di volta in volta i valori delle variabili intere (come si fa con i vincoli misti). Occorre peró apportare una modifica al procedimento finora descritto: il motivo risiede nel fatto che in questo caso non é piú vero che la giustapposizione di una soluzione ottima del master ($x^\star$) e della corrispondente soluzione slave ($y^\star$) sia la soluzione ottima del problema originale (non é piú possibile ripetere la dimostrazione per assurdo fatta in precedenza). Infatti in questo caso il master risulta privo della funzione obiettivo e quindi una soluzione ottima del problema master é semplicemente la prima

[9]Per maggiori informazioni si veda la sezione 2.4

soluzione ammissibile che si trova e nulla garantisce che tale soluzione attribuisca alle variabili x proprio i valori tali da permettere alle variabili y di far assumere il valore minimo possibile alla funzione obiettivo. Per far fronte a questa situazione, quando ci si trova in presenza di uno slave feasible, occorre operare come segue:

1. si salva la soluzione corrente $(x^\star, y^\star)$ ed il valore ottimo corrispondente dello slave (che é anche il valore della funzione obiettivo del problema originale per questa soluzione)

2. si inserisce nel master il taglio $\sum_{j:x_j^\star=0} x_j + \sum_{j:x_j^\star=1}(1 - x_j) \geq 1$, dove, a differenza del caso precedentemente esposto, non si effettua nessuna minimizzazione del numero di variabili coinvolte (non siamo in presenza di infeasibility) e quindi la somma va fatta su tutte le variabili binarie del problema (infatti diversamente da 2.45 non si richiede piú che l'indice j appartenga all'insieme F).

In questo modo, con la successiva risoluzione, si forza il problema master a cercare un'altra soluzione diversa dalla precedente, visto quella viene esclusa proprio dal taglio appena aggiunto (il quale del resto, vista la sua conformazione, non esclude nessun'altra soluzione intera). Con questo modo di procedere l'algoritmo, nel suo complesso, ha termine solo quando il master diventa infeasible a causa del fatto che tutte le possibili soluzioni sono state escluse dai tagli inseriti al punto 2). Alla fine, tra tutte le soluzioni salvate al punto 1) si sceglie quella che ha associato il valore minimo. Quando ci si trova invece in presenza di uno slave infeasible si continua ad operare come spiegato per il caso $d = 0$.

Occorre far notare che, con questo modo di procedere é necessario enumerare tutte le possibili soluzioni intere del problema originale per provare l'ottimalitá della soluzione trovata. Ció puó risultare improponibile per molti problemi concreti: di questo fatto e di eventuali stratagemmi per farvi fronte verra fatta menzione piú avanti in questa tesi.

2.3.3.1 Strategia Branch & Cut

L'algoritmo fin'ora descritto si configura come una strategia di soluzione di tipo cutting plane: si risolve il problema master all'ottimo, si aggiungono dei tagli generati sulla base dello slave, quindi si riottimizza il master e cosí via. Il dover risolvere una sequenza di problemi master via via sempre piú difficili (perché piú ricchi di vincoli) non é peró un approccio molto promettente: infatti, a meno che il master che si estrae dal problema originale non abbia una particolare struttura per cui si possano applicare algoritmi di soluzione specifici e veloci, l'approccio piú naturale sará quello di risolverlo tramite un algoritmo di branch & cut; ció comporta peró il fatto che ogni volta é necessario ripartire da zero nella soluzione del master visto che l'ottimo trovato viene "tagliato" dai tagli combinatori di Benders aggiunti.

Esiste peró una soluzione semplice e molto efficace per far fronte a questa situazione: risolvere una sola volta il problema master sempre tramite un algoritmo di branch & cut in cui i "cuts" che si aggiungono proprio i Combinatorial Benders' Cuts. In realtá questi non sono veri e propri tagli nel senso tradizionale, infatti non si limitano a escludere zone del rilassamento continuo del problema cercando di avvicinarsi al convex hull, ma anche escludono vertici a coordinate intere: ció non costituisce comunque alcun problema. In una implementazione semplice, seguendo le teoria fin'ora esposta, si potrebbe aspettare che l'algoritmo di branching giungesse a generare un nodo dell'albero decisionale in cui tutte le variabili binarie hanno assunto un valore intero e quindi, sfruttando tali valori, generare i consueti tagli di Benders combinatori. Si puó peró fare anche di piú: con delle lievi modifiche a quanto detto fino ad ora sará infatti possibile tentare la generazione dei tagli ad ogni nodo dell'albero di branching e non solo in quei nodi in cui la soluzione del master presenta caratteristiche di interezza. In questo modo la procedura di generazione dei tagli di Benders combinatori diventa una vera e propria *procedura di separazione*, all'interno di una strategia di B&C per risolvere il problema master. La modifica cui si é appena fatto cenno consiste semplicemente nel non inserire nello slave tutti i vincoli misti C_j, ma soltanto quelli per i quali la variabile di controllo x_i che vi compare sia stata fissata all'interezza dalla soluzione dell'LP del nodo corrente. In questo modo si puó comunque generare il taglio in caso di successo nella ricerca del MIS (si ricorda che i coefficienti $+1$ o -1 del taglio dipendono dal valore intero assunto dalla variabile corrispondente e quindi non é possibile utilizzare valori frazionari). Inoltre il fatto che questo procedimento (ossia la generazione di tagli in presenza di uno slave incompleto infeasible) sia lecito é garantito dalla semplice considerazione che l'unica differenza rispetto a quanto precedentemente descritto é che ad ogni nodo si opera con uno slave i cui vincoli costituiscono un sotto–insieme di quelli dello slave originale e perció ogni MIS di questi slave é anche un MIS dell'originale. Ció fa anche capire il perché i tagli prodotti con questo sistema siano comunque validi "globalmente" ovvero essi valgono per l'intero problema master e non solo per il nodo in cui sono generati e la sua discendenza.

É forse superfluo precisare che in questo caso solo per i nodi in cui tutte le variabili intere assumono un valore intero continua a velere l'affermazione secondo cui ad uno slave feasible corrisponde una soluzione del problema originale (per giustapposizione con la soluzione master); per gli altri nodi infatti, oltre ad avere una soluzione che non renderebbe soddisfati tutti i vincoli di interezza, si ha che l'assenza di alcuni vincoli C_j nello slave (dovuta ai valori frazionari assunti nella soluzione dalle variabili di controllo corrispondenti) ci consente di affermare solamente che un sottoinsieme dei vincoli del problema originale sono soddisfatti, ma nulla ci dice sui vincoli esclusi. Per questo, nel caso di nodi interni dell'albero di branching, se lo slave é feasible semplicemente non si intraprende nessuna particolare azione, lasciando che la procedura di branching continui da sé. É sempre per questo motivo che in precedenza si é detto che si puó *tentare di generare tagli ad ogni nodo*: per quanto detto finora non é infatti detto che vi si riesca ad un certo nodo, anche se quel nodo non dovesse poi generare in futuro alcuna soluzione feasible.

2.3.4 Algoritmo di risoluzione: riassunto

Possiamo ora riassumere sinteticamente l'intero algoritmo di risoluzione su cui si basa l'utilizzo dei tagli di Benders combinatori. Esso consta dei seguenti passi:

1. si estrae il problema master da quello originale, ovvero la "parte intera" di quest'ultimo
2. si risolve il problema master tramite un algoritmo B&C
3. ad ogni nodo dell'albero di branching si attuano i seguenti passi:
   - si costruisce il problema slave corrispondente al nodo corrente, ovvero quel sistema contenente tutti i vincoli continui ed i soli vincoli misti per i quali la corrispondente variabile di controllo ha assunto un valore intero nella soluzione del nodo corrente
   - si sostituisce nel problema slave la parte intera della soluzione trovata
   - se il problema slave cosí ottenuto é infeasible, si cercano uno o piú MIS, si generano i tagli corrispondenti e li si aggiunge al problema master
   - se il problema slave é feasible e tutte le variabili intere soddisfano i vincoli di interezza del master siamo in presenza di una soluzione del problema originale (se qualche variabile non soddisfa tali vincoli non occorre intraprende nessuna azione particolare)

Come é facilmente intuibile le operazioni di generazione dei vari problemi slave e di ricerca di MIS, vista la loro frequenza, possono costituire il collo di bottiglia dell'intero approccio: per questo motivo si é dedicata loro una particolare attenzione ed alcuni accorgimenti approntati in fase di implementazione dell'algoritmo (vi veda il capitolo 4) hanno permesso di ottenere ottime prestazioni.

2.4 Ricerca veloce di MIS

In questa sezione si presenterá il concetto di MIS e si descriverá un algoritmo efficiente per la loro ricerca all'interno di un sistema infeasible.

Minimal Infeasible Subsystem Dato un sistema di disequazioni che non ammetta alcuna soluzione (infeasible), é lecito chiedersi quale sia la causa di questa infeasibility. In altre parole ci si domanda quali siano le disequazioni incompatibili tra loro, ovvero quelle che non permettono di individuare una assegnamento delle variabili tale da soddisfarle tutte contemporaneamente. Inoltre si desidera generalmente che il numero di queste disequazioni sia il piú piccolo possibile.

Da un punto di vista piú formale si puó formulare il problema come segue: dato un sistema di disequazioni che non ammetta soluzione, trovare un sottoinsieme delle disequazioni di tale sistema anch'esso impossibile e di cardinalitá minima. Nonostante il questo problema sembri in apparenza semplice, si puó dimostrare che appartiene alla classe dei problemi NP-hard. Per qusto motivo ci si accontenta di trovare un sottoinsieme che sia non necessariamente di cardinalitá minima ma quantomeno minimale; la minimalitá é da intendersi nel senso che il sottoinsieme in questione deve avere la proprietá di diventare feasible (ammettere soluzioni) non appena si tolga una qualsiasi delle disequazioni che lo compongono. A questi sottoinsiemi di equazioni si dá il nome di MIS.

Questo tipo di problema é di grande interesse pratico, oltre che teorico: infatti l'infeasibility di un sistema é generalmente non voluta e puó dipendere da un errore di modellizzazione della realtá che il sistema vuole descrivere, da imprecisioni numeriche (errori di misura) o altro. L'individuazione e l'analisi dei MIS permette generalmente di individuare e correggere le cause dell'infeasibility.

Si vuole far notare che esiste una sorta di equivalenza tra un sistema di disequazioni infeasible ed un problema di programmazione lineare infeasible: infatti in quest'ultimo l'infeasibility dipende solo dalle disequazioni che determinano la regione ammissibile del problema (che in questo caso é l'insieme vuoto) e non dalla funzione obiettivo. Per questo motivo é possibile ignorare, in questi casi, la funzione obiettivo o viceversa pensare un qualsiasi sistema infeasible come dotato di una arbitraria funzione obiettivo trasformandolo di fatto in un problema LP.

D'ora in poi parleremo perció indifferentemente di problema LP o di sistema di disequazioni, quando questi sono infeasible.

Algoritmo Euristico di Ricerca di MIS Per quanto appena affermato é sempre possibile pensare il sistema di disequazioni di cui si ricerca un MIS come un problema LP con funzione obiettivo a coefficienti tutti nulli. Per ragioni di semplicitá espositiva supporremo inoltre che i vincoli siano tutti in forma di maggiore uguale e che le variabili non siano vincolate da bounds inferiori o superiori (se esistono si possono considerare delle semplici diseguaglianze). In sostanza il problema infeasible di cui si ricerca un MIS sia scritto in questa forma:

$$\min \quad 0^T x \tag{2.46}$$

$$\text{s.t.} \quad Ax \geq b \tag{2.47}$$

$$x \text{ free} \tag{2.48}$$

A questo punto é possibile applicare la teoria della dualitá e passare al problema duale, il quale ha la seguente forma:

$$\max \quad u^T b \tag{2.49}$$

$$\text{s.t.} \quad u^T A = 0^T \tag{2.50}$$

$$u \geq 0 \tag{2.51}$$

É noto che se il problema primale é infeasible, il corrispondente duale puó essere illimitato oppure infeasible. Grazie alla particolare scelta della funzione obiettivo fatta é possibile escludere quest'ultimo caso (ció si puó facilmente verificare osservando che $u = 0$ é una soluzione del problema duale). Il fatto che il problema 2.49–2.51 sia illimitato ci consente di concludere che esiste almeno un vettore u^* con componenti non negative e non tutte nulle tale che $A^T u^* = 0$ e tale che $b^T u^* > 0$; inoltre qualsiasi vettore ku^* con $k \geq 0$ é anche soluzione del problema duale. Sulla base di quanto detto possiamo quindi aggiungere al problema 2.49–2.51 la seguente disequazione, senza che questa ne cambi la natura:

$$u^T b \geq m \tag{2.52}$$

con $m > 0$, costante arbitraria. Il significato dei vincoli del problema duale (insieme al'equazione appena aggiunta) e della soluzione u^* che si ottiene é il seguente: esiste una combinazione lineare, a coefficienti non tutti nulli (u^*), dei vincoli del problema di cui si cerca un MIS tale che il vincolo risultante abbia i coefficienti delle variabili x tutti nulli ($u^{*T} A = 0^T$) ed il termine noto diverso da zero ($u^{*T} b > 0$). Ció ovviamente é conseguenza ed indice del fatto che il problema é infeasible.

A questo punto é possibile fare un'osservazione che sará la chiave di tutto il sistema di ricerca di MIS, osservazione che discende direttamente da quanto si é appena fatto notare: se dall'insieme di disequazioni del sistema primale si escludono quelle che hanno un valore nullo nella corrispondente variabile duale ($u_i^* = 0$) si ottiene un sistema piú piccolo ma ugualmente infeasible.

La situazione puó essere ulteriormente migliorata se si sostituisce la funzione obiettivo 2.49, che in virtú del vincolo 2.52 é diventata ridondante, con la seguente:

$$\min \sum_i u_i = 1^T u. \tag{2.53}$$

Questo cambiamento tende a far scartare, tra tutte le soluzioni possibili, quelle per cui la somma delle variabili duali é alta e quindi tende a far scartare quelle con molti non-zero. Sebbene ció non sia garantito (infatti potrebbero esservi soluzioni con tanti valori non nulli ma piccoli, la cui somma é minore di altre con pochi valori non nulli ma grandi), si é verificata in pratica la grande efficacia di questo approccio: quasi sempre infatti gli insiemi generati da questo sistema sono effettivamente MIS. Naturalmente, anche quando non lo fossero, ne contengono sicuramente uno e quindi il sistema indicato puó comunque essere usato come punto di partenza per sfoltire enormemente in partenza il numero di disequazioni "candidate", per poi usare metodi di ricerca esatti (non euristici) piú lenti ma su una istanza di dimensioni inferiori.

Considerazioni conclusive Oltre ad essersi dimostrato estremamente efficace, si vuole far notare che l'algoritmo fin'ora esposto é suscettibile di una generalizzazione. Infatti, assegnando dei valori diversi dall'unitá ai coefficienti della funzione obiettivo 2.53 che si introduce é possibile "guidare" la ricerca dei MIS spingendo la soluzione ad escludere od includere alcune disequazioni ed ottenendo quindi, tar tutti i MIS contenuti nel sistema, quelli piú vicini alle proprie esigenze. Inoltre, se é noto fin dall'inizio che alcune equazioni non possono far parte di un MIS é possibile direttamente non includere la variabile duale corrispondente nella funzione obiettivo (cioé assegnarle un peso nullo).

Occorre anche far notare che non é necessario verificare a priori che il sistema primale sia infeasible: se cosí non fosse si ha immediatamente che il sistema duale modificato secondo quanto giá descritto risulterebbe infeasible impossibilitando automaticamente la ricerca di un MIS.

Nell'implementazione del programma che accompagna questa tesi, si sono sfruttate entrambe queste due caratteristiche.

Sebbene non siano state condotte prove specifiche direttamente mirate alla verifica dell'efficacia di questo algoritmo, si puó comunque affermare che questo approccio consente comunque di ottenere risultati ottimi in tempi rapidissimi. Infatti inizialmente esso non era stato utilizzato all'interno del programma realizzato per questa tesi e si erano provate, al suo posto, altre soluzioni (in particolare un semplice algoritmo esatto di ricerca e l'algoritmo fornito dal software Ilog CPLEX 8.0 [10]). Queste peró si sono rivelate insufficienti per le esigenze di rapiditá qui necessarie: per questo é stato ideato, implementato ed utilizzato l'algoritmo ora descritto; in seguito a tale utilizzo i tempi spesi dal programma nella ricerca di MIS sono letteralmente crollati, diventando una piccola frazione del tempo totale di calcolo.

[10] Per una descrizione di questo software si veda la sezione 3.2. L'algoritmo cui si fa qui riferimento e quello fornito tramite la funzione `IloCplex::getIIS()`, provato con entrambi i valori del parametro `IloCplex::IISInd`

Capitolo 3

Strumenti software utilizzati

3.1 Panoramica

L'algoritmo fin qui descritto é stato tradotto in un programma completo e funzionale che é parte integrante di questa tesi. A causa della complessitá che la creazione di un tale programma comporta risulta impensabile ed illogico crearlo senza avvalersi dell'ausilio di software preesistente. Infatti, per la sua creazione si é fatto uso di codice contenuto in due pacchetti software. Il primo é un prodotto commerciale: Ilog CPLEX; il secondo é stato scritto dall'autore di questa tesi in collaborazione con il collega Giuseppe Andreello e gli é stato dato il nome di General B&C Framework. Naturalmente lo sviluppo, il debugging e la sucessiva analisi dei dati raccolti durante i test ha coinvolto anche molti altri programmi, ma solo i due menzionati sono indispensabili per la compilazione/linking del programma: per questo il resto del capitolo sará dedicato ad una loro descrizione.

Si fa inoltre presente che lo sviluppo del programma é avvenuto su un PC dotato di un sistema operativo Linux (kernel 2.2) ed il codice é stato scritto nel linguaggio di programmazione C++ e compilato tramite GNU Compiler Collection (GCC/G++) versione 3.0. Per le strutture dati non banali si sono usate le classi messe a disposizione dalla libreria standard del C++ (STL).

3.2 Ilog CPLEX: introduzione e motivazioni

Ilog CPLEX8.0 è un software per la programmazione lineare mista molto usato sia in ambito industriale che di ricerca.

CPLEX deve la sua fama sia alle sue elevate prestazioni che alla stabilità della sua implementazione dell'algoritmo del simplesso. Infatti, fino a cinque anni fa, l'ostacolo principale alla soluzione di istanze difficili era l'instabilità numerica; questa era dovuta sia alle dimensioni delle matrici che ai rapporti fra i coefficienti stessi (si pensi all'uso tipico del "big M" per la modellazione di alcune classi di problemi, di cui si é ampiamente parlato). Capitava spesso, soprattutto in ambito di ricerca, che un branch & bound non potesse essere portato a termine per l'insolubilità di alcuni nodi.

L'uso di CPLEX è quindi inevitabile se non altro come termine di paragone per la misura delle prestazioni di un nuovo software.

In realtà però, per questo lavoro di tesi, CPLEX è stato usato anche per implementare l'algoritmo di branch & cut; infatti, dalla versione 7, CPLEX offre la possibilità di essere esteso con dei moduli esterni chiamati *callbacks*: questi consistono di parti di codice scritte in C, C++ o Java che implementano funzionalità come la scelta della variabile di branching, la separazione di una variabile frazionaria etc. che vengono eseguite al momento opportuno all'interno della normale procedura di branch & cut.

Nel caso di questa tesi, quindi, è stato possibile concentrare gli sforzi sulla sola procedura di separazione (ovvero gestione della parte slave e generazione dei tagli di Benders combinatori) lasciando a Cplex l'onere della risoluzione del problema master. La possibilitá di muoversi all'interno di un già collaudato e apprezzato risolutore ha reso molto più significative le misure effettuate: se fosse stato necessario implementare un solver MIP completo, infatti, tali misure sarebbero state fortemente influenzate anche dalla qualità di tale parte dell'implementazione e non principalmente dall'efficacia del nuovo algoritmo che si é qui studiato.

In generale la possibilità di implementare un separatore all'interno di un solver completo e veloce consente inoltre di stabilire se l'aumento di prestazioni fornito rispetto allo stato dell'arte della tecnologia giustifichi uno sviluppo ulteriore del separatore stesso e la sua implementazione all'interno di software commerciali.

3.2.1 Ilog CPLEX: caratteristiche principali

In questa sezione si analizzano brevemente le caratteristiche delle parti principali di CPLEX8, alla luce di quello che prevedono le tecniche più diffuse di soluzione di problemi MIP.

Per una descrizione dettagliata delle funzionalità si faccia riferimento alla documentazione del software; per avere invece un'idea dell'impatto sulle prestazioni delle singole componenti si vedano [2].

3.2.1.1 Simplesso, preprocessor e prober

Queste sono le parti fondamentali di CPLEX.

Si dispone di tre algoritmi per risolvere il rilassamento continuo: *primale*, *duale* e *barrier*.

Il preprocessor e il prober sono applicati prima della soluzione del rilassamento, per cercare di migliorare la formulazione; il primo cerca di togliere variabili e vincoli inutili, mentre il secondo analizza le implicazioni logiche derivanti dell'assegnamento di valori interi alle variabili intere. Il modello su cui CPLEX applica il solver ha quindi, in genere, meno vincoli, meno variabili e un po' di coefficienti modificati.

Parte di queste due funzionalità vengono disattivate automaticamente se si usa una callback per inserire tagli durante la soluzione di una istanza MIP. In particolare si tratta delle trasformazioni irreversibili; dall'interno di una callback, infatti, si ha disposizione sempre e solo il modello originale; la traduzione viene effettuata in modo automatico dalle API, e naturalmente deve essere sempre possibile.

3.2.1.2 Euristica

Questa è una componente molto utile; come ogni solver MIP, a intervalli casuali CPLEX tenta di individuare soluzioni ammissibili in modo euristico; durante una ottimizzazione quasi tutte le soluzioni ammissibili sono trovate per mezzo dell'euristica, anzichè con la normale esplorazione dell'albero. Purtroppo, a causa di un bug manifestatosi in conseguenza del particolare uso di Cplex (non molto comune) fatto in questa tesi, si é stati costretti a fare un uso ridotto delle funzionalitá euristiche (che sará spiegato nel capitolo 4).

3.2.1.3 Famiglie di tagli

CPLEX fornisce 9 famiglie di tagli di tipo generale (ovvero applicabili a qualsiasi problema, indipendenti dal modello da risolvere): Clique, Cover, Disjunctive, Flow Cover, Flow Paths, Gomory Fractional, Generalized Upper Bound Cover, Implied Bound, Mixed Integer Rounding.

Per un lavoro di ricerca su una nuova famiglia di tagli, il numero di famiglie a disposizione garantisce il fatto di poter capire se la nuova famiglia sia veramente utile oltre a essere efficace in se e per se.

3.2.1.4 Strategie di Branch & Cut

Il Branch & Cut di CPLEX è piuttosto diverso dalla tipica descrizione che si trova nei libri di testo (per esempio [1]). Si dà qui una descrizione molto superficiale delle caratteristiche tipiche di un Branch & Cut, sufficiente però a far capire in cosa si distingue l'implementazione di CPLEX8.

Gli algoritmi della famiglia Branch&Cut, in genere, gestiscono il modello in modo dinamico: i tagli generati dal separatore, di validità globale, sono mantenuti in una struttura dati chiamata *pool*; a ogni nodo viene fatta una scansione del pool: se un taglio del pool è violato dalla soluzione corrente del rilassamento continuo, il taglio viene aggiunto alla formulazione e di conseguenza la soluzione viene aggiornata, tipicamente sfruttando la formulazione duale del simplesso. Se il pool non contiene tagli violati in genere si prova ad applicare il separatore alla soluzione corrente, arricchendo così il pool di nuovi tagli. Ad intervalli regolari si effettua un *purging* del pool per cancellare i tagli inutili perchè dominati da altri tagli più profondi; allo stesso modo anche fra i vincoli della formulazione originale possono essere disattivati quelli che non partecipano alla defizione della regione di politopo che si sta esplorando.

I vantaggi di questo tipo di implementazione sono due:

1. l'applicazione diretta del separatore è tipicamente piuttosto costosa; mantenere i tagli nel pool evita di dover generare più volte lo stesso taglio;

2. i tagli vengono aggiunti alla formulazione solo quando servono, in modo dinamico; cosí facendo la formulazione di ciascun nodo non è appesantita da tagli che non partecipano alla soluzione corrente.

CPLEX funziona in un modo molto più semplice; la sua taratura di default si basa sull'algoritmo 1. Per motivi di chiarezza non è stata esplicitata la gestione delle soluzioni intere trovate, del valore di best-bound, di cut-off etc. L'algoritmo qui descritto non è riportato esplicitamente nella documentazione, ma è stato ricavato osservando l'output delle ottimizzazioni.

Evidentemente, alla base di questo algoritmo ci sono le seguenti assunzioni:

1. se un taglio è utile, deve essere inserito nella formulazione in modo statico;

2. troppi tagli possono essere controproducenti: appesantiscono la formulazione e possono allungare a dismisura il tempo necessario a visitare l'albero, anche se questo è stato "potato" dalla presenza dei tagli stessi;

3. grazie all'efficienza del suo simplesso duale, CPLEX è molto veloce nel visitare nodi in modalità depth-first.

4. CPLEX8 ha un'euristica molto efficace, che di solito riesce a trovare soluzioni intere molto prima di quanto si potrebbe con l'enumerazione; in questo modo il valore di cut-off sale velocemente e l'albero viene ridotto di conseguenza.

Questa configurazione è stata scelta testandola su una vasta collezione di istanze di proprietà di Ilog (si veda [2]). Se necessario, anche senza ricorrere all'uso delle callbacks, il funzionamento di CPLEX8 può essere profondamente modificato in ogni aspetto dell'algoritmo agendo sui parametri che mette a disposizione.

Algorithm 1 Cut & Dive; taratura default di CPLEX8.

```
Require: a: rilassamento continuo del problema
applica il preprocessor e il prober ad a
risolvi a
while non hai aggiunto troppi tagli e puoi generarne ancora do
    separa la soluzione frazionaria di a, aggiungendo i tagli in modo statico
end while
genera e metti in coda due figli facendo branch su una variabile frazionaria
repeat
    a ← un nodo in coda, di costo pari al best-bound
    prova a generare dei tagli, se opportuno
    crea due nodi con un branch su una variabile frazionaria
    a ← uno dei due figli appena creati, l'altro in coda
    dive(a)
until non ci sono più nodi nella coda di attesa

dive:
Require: a: un nodo attivo
loop
    risolvi il rilassamento del nodo a
    if a è feasible, la soluzione è frazionaria e inferiore all'incumbent then
        genera due figli facendo branch su una variabile frazionaria
        a ← uno dei due figli, l'altro in coda
    else
        return
    end if
end loop
```

3.2.2 Ilog CPLEX: modalita d'uso

CPLEX offre due modalità d'uso.

Modalità interattiva: in questa modalità CPLEX si presenta come una qualsiasi altra applicazione con cui si interagisce a riga di comando. Si ha la possibilità di descrivere il modello secondo una sintassi comoda e intuitiva o di importarlo da un file; si possono assegnare valori opportuni ai parametri per configurare l'ottimizzatore, e si può chiedere l'ottimizzazione dell'istanza. L'ottimizzazione può essere interrotta per assegnare un valore diverso a uno dei parametri per poi essere ripresa.

In alternativa, le funzionalità di CPLEX sono a disposizione sotto forma di libreria linkabile in modo dinamico. In questo modo è possibile usare CPLEX come solver a scatola chiusa all'interno di un'applicazione (per esempio un controllore real-time o un software interattivo), in modo che l'utente dell'applicazione non debba interagire direttamente con CPLEX.

Solo in questa modalità si possono sfruttare le funzionalità avanzate che consentono di costruire un Branch&Cut personalizzato usando le callbacks di CPLEX. Ovviamente questa é la modalitá impiegata per sviluppare il software di questa tesi.

3.3 General B&C Framework

Un software di Branch&Cut sviluppato in ambito di ricerca richiede un buon numero di funzionalità dedicate alla parametrizzazione del codice, alla raccolta di dati statistici relativi all'ottimizzazione in corso etc. Basandosi sul fatto che CPLEX è uno strumento di riferimento per questo settore e prevedendo che molti dei progetti di ricerca del gruppo di Ricera Operativa richiederanno l'implementazione di un Branch&Cut basato su CPLEX si é deciso di progettare e implementare dei componenti software che fungessero da struttura di base da poter usare come punto di partenza per le varie implementazioni. In questo modo si favorisce il riutilizzo di codice da un progetto all'altro, evitando inutili perdite di tempo dovute alla reimplementazione di funzionalità basilari o alla fusione di codici sviluppati secondo criteri incompatibili.

Tutto ció ha portato in definitiva alla creazione del General B&C Framework (piú brevemente GBC), ovvero un ulteriore strato di software che si sovrappone alla libreria di funzioni di CPLEX fornendo la struttura (framework) di base per un qualsiasi programma che implementi un algoritmo di tipo Branch&Cut (da cui il nome).

Alla spiegazione della genesi e delle funzionalitá di questo framework sará dedicata la prossima sezione, la quale puó essere vista come una specie di manuale d'uso di GBC. Inoltre si riporta in appendice B la documentazione generata in modo automatico dal tool Doxygen a partire dal codice sorgente di GBC e dai commenti ivi contenuti: tale appendice é pensata come un manuale di riferimento (reference) da consultarsi al momento della scrittura di codice basato su GBC e comprende in particolare le descrizioni delle classi, dei loro metodi/campi ed alcuni diagrammi che aiutano a comprendere le relazioni di ereditarietá, specializzaizone ed uso tra le classi stesse.

3.4 Realizzazione di GBC in Concert Technology

Questo capitolo è stato scritto a quattro mani dall'autore di questa tesi e dal collega Giuseppe Andreello. Durante le rispettive tesi di laurea sono stati sviluppati due diversi algoritmi di Branch&Cut. Avendo notato che si stavano scrivendo programmi che soddisfavano in buona parte le stesse necessità, si é deciso di progettare e implementare una base di codice che potesse fare da punto di partenza per le future implementazioni di Branch&Cut realizzate nell'ambito dei progetti del gruppo di Ricerca Operativa.

Il frutto di quessto lavoro, ovvero GBC, é stato proficuamente utilizzato in entrambe le tesi e si é dimostrato di grande aiuto alla produttivitá.

Seguono una breve analisi dei requisiti e una spiegazione di come i requisiti sono stati trasformati in classi compatibili con l'API C++ di CPLEX (chiamata Concert Technology).

3.4.1 Analisi dei prerequisiti

Sono di seguito presentate le funzionalità che si sono ritenute indispensabili per un Branch&Cut da usare in ambito di ricerca. Il framework sarà costituito da un insieme di classi scritte in un linguaggio orientato agli oggetti; alcune saranno complete, anche se comunque estensibili; altre, come per esempio le classi che implementeranno tagli e pools, saranno delle classi base da cui derivare le implementazioni vere e proprie.

I prerequisiti fondamentali sono quindi proprio la flessibilità e la possibilità di estendere le funzionalità di base.

3.4.2 Gestione dell'ottimizzazione: requisiti

3.4.2.1 Parametri da riga di comando

- scegliere la taratura di un algoritmo per la soluzione di problemi NP richiede di effettuare moltissimi test a tappeto; per questo è indispensabile poter passare argomenti dalla riga di comando; in questo modo si minimizzano le ricompilazioni e si possono automatizzare i test usando un linguaggio di scripting ad alto livello.

3.4.2.2 Log-files, statistiche e profiling

- il framework deve prevedere delle strutture dati in cui raccogliere informazioni su cui poter fare analisi statistiche anche durante l'esecuzione stessa;

- le informazioni raccolte e stampate durante l'esecuzione del programma devono poter essere facilmente suddivise in vari files, in modo da poterle gestire in modo più efficace in fase di analisi dei risulati;

- in particolare è importante poter tenere traccia del tempo di calcolo speso in diverse sezioni del codice.

3.4.2.3 Preprocessing e gestione della formulazione

Spesso è necessario preprocessare la formulazione del problema per renderla più adatta alle tecniche che si stanno sperimentando; esempi di funzionalità implementate nelle nostre applicazioni:

- suddivisione delle variabili in intere e continue;

- manipolazione della funzione obiettivo.

3.4.2.4 Usabilità/raggiungibilità delle informazioni

È indispensabile che le informazioni di partenza (parametri da riga di comando e formulazione) e quelle raccolte durante l'esecuzione (informazioni statistiche) siano facilmente accessibili alle varie parti del codice. Inoltre, secondo noi, è importante evitare di appesantire troppo la sintassi delle chiamate a funzione con il passaggio di un numero inutilmente alto di parametri, che spesso sono informazioni di tipo globale.

Un software scritto in ambito di ricerca, infatti, viene modificato spesso, e questo si deve poter fare in modo agevole.

3.4.2.5 Piani di taglio e pool

Il framework deve prevedere l'implementazione simultanea di più separatori, ciascuno con il prorio pool di tagli, se necessario.

La funzionalità principale del pool è l'estrazione dalle sue strutture dati dei tagli violati da un punto frazionario. È poi molto importante che la ricerca dei tagli violati sia veloce.

3.4.3 Dai prerequisiti alle classi

Per implementare una base di codice di questo tipo è indispensabile l'uso di un linguaggio per la programmazione orientata agli oggetti.

Gli unici due linguaggi a oggetti con cui ci si può interfacciare a CPLEX sono C++ e Java: per questioni di prestazioni abbiamo scelto il primo.

Nel seguito della sezione si dà una descrizione di massima delle classi implementate per poter soddisfare i requisiti compatibilmente con la struttura di classi fornita dalle API di CPLEX8.

3.4.4 CPLEX e Concert Technology

La libreria di classi che si usa per interfacciarsi a CPLEX si chiama Concert Technology; queste classi modellano da una parte il solver e tutte le sue funzionalità (fra cui le callbacks), dall'altra tutto ciò che compone un problema MIP: variabili, vincoli, funzione obiettivo. L'implementazione della libreria si basa sul paradigma *handle-implementation* allo scopo di semplificare la gestione dei riferimenti agli oggetti e per permettere di implementare anche funzionalità di *lazy-copy*. La libreria prevede inoltre un sistema di notifica automatico, per cui una modifica alla formulazione del problema viene automaticamente propagata a tutti gli oggetti a cui interessa.

Apparentemente, molte delle funzionalità necessarie al framework possono essere implementate direttamente sulle classi della Concert Technology; in realtà la Concert Technology modella classi molto generali, pensate sia per la programmazione lineare che per la programmazione vincolata; sono molto comode, per esempio, per la descrizione dell'istanza da risolvere, ma sono troppo lente se usate in modo intenso – ad esempio per l'implementazione dei tagli e dei pool.

Quindi le classi della Concert Technology sono state usate semplicemente per lo scambio di dati con CPLEX, oltre che per il meccanismo delle callbacks; tutte le elaborazioni di espressioni lineari che abbiamo implementato operano invece su strutture dati più snelle scritte da noi.

I nomi delle classi Concert Technology seguono il pattern `Ilo<Nome>`; un `IloRange` è un vincolo lineare, un `IloNumVar` è una variabile numerica, `IloCplex` è il solver.

La classe `IloCplex` ha numerose sottoclassi, fra cui le callbacks, che consentono di prendere il controllo del Branch&Cut: `IloCplex::CutCallback` è la callback dalla quale si possono inserire tagli dopo la soluzione del lineare del nodo corrente; `IloCplex::SolveCallback` è quella da cui si può pilotare la soluzione del lineare; `BranchCallback` è quella in cui vengono creati i figli del nodo appena risolto , e così via. Per implementare una callback si deriva la classe originale e si inserisce il codice che va eseguito nel metodo `main()`. Si tenga presente che ogni classe callback è istanziata una e una sola volta.

3.4.5 Gestione dell'ottimizzazione: le classi di GBC

Parametri da riga di comando Per la gestione dei parametri è stata implementata la struct `Arguments`, della quale viene creata una sola istanza. In pratica si tratta di un semplice insieme di variabili che contengono i valori passati dalla riga di comando; per il parsing vero e proprio è stato usato `argp_parse`, parte di `libc`.

Log-files La gestione dei log-files è attuata per mezzo della classe `OutputLogs`. Di questa classe è definita una sola istanza, che contiene un `ostream` per ogni file di log.

Statistiche Le informazioni raccolte durante l'ottimizzazione sono memorizzate in una istanza della classe `Statistics`; l'unico metodo di questa classe serve a stamparne il contenuto in una forma comoda da leggere.

Profiling La classe `Timers` implementa un insieme di cronometri che possono essere fermati e fatti ripartire, a seconda della sezione di codice da cui si entra o si esce.

Preprocessing e gestione della formulazione Questa parte dei requisiti è soddisfatta dalla classe `Problem`, che contiene le istanze delle classi Concert Technology che modellano il MIP. Qui sono implementate anche le funzioni che preprocessano la fomulazione per l'utilizzo da parte di separatori etc.

Molto importante è l'indicizzazione delle variabili, necessaria a non perdere la corrispondenza fra le strutture Concert Technology e le nostre strutture di tipo C.

Usabilità Per ciascuna delle classi elencate fino a qui è allocata una sola istanza, della quale è disponibile un puntatore globale. In questo modo si minimizzano i parametri da passare ai vari metodi e si evita di duplicare informazioni nelle varie classi.

Anche la variabile `env`, che rappresenta l'ambiente CPLEX in esecuzione e che va passata ad ogni costruttore di oggetti Concert Technology è globale.

3.4.5.1 Piani di taglio e pool

L'implementazione dei piani di taglio e del pool è la parte più complessa del framework, almeno per quanto riguarda la gerarchia delle classi.

La forma più generale di taglio è data dalla classe virtuale `BaseCut`; le funzionalità previste sono quelle di poter interrogare il taglio per sapere se è violato da un punto frazionario e di poterlo convertire in un IloRange.

Figlia di questa è la classe template `SimpleCut`: implementa un vincolo lineare in cui il tipo dei coefficienti è il parametro del template. La struttura dati è costituita da un vettore di indici e uno di coefficienti, in modo da poter calcolare violazioni, slack, prodotti esterni etc nel modo più veloce possibile.

La forma più generale di pool è data dalla classe virtuale `Pool`: i suoi due metodi consentono di aggiungere un `BaseCut` al pool e di estrarre dal pool un `IloRangeArray` di tagli violati da un punto frazionario.

La presenza di queste interfacce standard consente di usare facilmente più separatori all'interno del Branch&Cut. A chi implementa i separatori è lasciata la massima libertà, mentre chi li usa non deve preoccuparsi di come sono stati implementati.

Il codice dei separatori va poi chiamato dal metodo `main` della classe `CutCBI`, che naturalmente discende da `CutCallbackI`.

3.4.5.2 Altre classi a corredo

Per l'effettivo funzionamento del Branch&Cut è stato necessario implementare altre classi.

La classe `DatiNodo` rappresenta un insieme di informazioni che vengono associate a ogni nodo dell'albero di Branch&Cut: la sua profondità, un intero che lo identifica univocamente etc. CPLEX, infatti, prevede di poter associare a ciascun nodo un puntatore a un oggetto di tipo `IloCPlex::NodeData`, in modo da poter associare al nodo tutte le informazioni che possono essere utili alla gestione del Branch&Cut. Le istanze di `DatiNodo` sono assegnate ai nodi al momento della loro creazione da parte dell'istanza di `BrancCBI`, che è una `BranchCallbackI`; possono poi essere raggiunte da tutte le altre callback: per esempio, la `NodeCBI` si occupa di scegliere il prossimo nodo da risolvere.

Capitolo 4

Implementazione dell'algoritmo

4.1 Modifiche nell'implementazione dei tagli combinatori di Benders

La teoria fino ad ora esposta riesce a raggiungere l'obiettivo che ci si era posti: l'eliminazione della influenza negativa causata dalla presenza dei coefficineti big–M. Questo porta sicuramente dei benefici in termini sia di stabilitá e precisione dei calcoli, si in termini di tempo. Ció é peró ottenuto al prezzo di un aumento della complessitá dell'algoritmo di risoluzione e quindi del numero di calcoli necessari, il comporta invece un aumento del tempo necessario alla risoluzione. É quindi necessario capire quali dei due elementi contrastanti sia predominante; per far questo si perció realizzata una implementazione software concreta della teoria sviluppata e si sono fatti dei test raccogliendo dei dati sperimentali su un certo numero di problemi.

Durante la fase di sviluppo di tale software si é dovuti far fronte ad una serie di problemi che hanno permesso di mettere in luce alcune caratteristiche di difficile realizzazione celate nella teoria fin qui esposta: sono state tuttavia trovate delle buone soluzioni e talvolta sono state apportate alcune migliorie alla teoria stessa. Le piú significative tra queste verranno sottolineate nella presente sezione.

Vincoli ad attivazione condizionale Sebbene la teoria esposta non faccia alcuna particolare supposizione di carattere numerico (ovvero sulla "grandezza") circa coefficienti dei vincoli del problema (specie sulla matrice M o sui termini noti) e quindi sia del tutto generale, é noto che buona parte delle istanze di problemi che ci proponiamo di risolvere comprenda dei coefficienti big–M (d'altra parte la loro presenza é stata uno dei motori principali per questa tesi). Questo fatto ha indotto a gestire in modo particolare le situazioni in cui tali coefficienti sono presenti, allo scopo di migliorare l'efficienza, sfruttando le particolaritá specifiche dei vincoli che li comprendono.

Il fatto fondamentale connesso ai vincoli con coefficiente big–M é che ad essi si possono associare due stati, dipendenti dal valore della variabile di controllo: uno attivo, in cui il vincolo concorre alla definizione del dominio del problema, ed uno disattivo, in cui il taglio é superfluo (in quanto "slack"). Questa caratteristica puó essere utilmente sfruttata se si pensa che un tale vincolo, nello stato non attivo, non potrá mai essere indispensabile per la generazione di un MIS: in altre parole se una soluzione del master non dovesse essere valida per il problema originale (ovverro comportasse uno slave infeasible) ci sará sempre un MIS nello slave in grado di escludere tale soluzione generando il taglio combinatorio di Benders appropriato, indipendentemente dalla presenza o assenza del vincolo in questione; questo perché una tale soluzione sarebbe comunque esterna al dominio originale ed alla formazione di tale dominio non concorre nessun vincolo disattivo. Per questo motivo il programma implementato prevede, oltre ad una gestione di tipo generico, una gestione particolare dei vincoli misti: tramite una opzione fornibile dalla linea di comando (`--active_value`) é possibile specificare se il valore delle variabili di controllo al quale corrisponde l'attivazione dei vincoli misti é pari a zero o ad uno. Nel caso questa opzione venga utilizzata, allora il programma evita di inserire nei vari slave che vengono generati ai vari nodi quei vincoli misti che risultano disattivati dalla soluzione master del nodo corrente. In questo modo si riduce la dimensione degli slave, riducendone sia il costo (in termini di tempo) di costruzione che il tempo necessario a cercarne i MIS, dato che risultano assenti tutti i vincoli "superflui". Attualmente questa opzione agisce a livello globale (ovvero tutti i vincoli misti del problema devono attivarsi sullo stesso valore della variabile di controllo): questo non é generalmente un problema dato che molti problemi si presentano in questa forma e comunque é sempre possibile modificare il modello MIP utilizzando le variabili complementari delle variabili di controllo, se il problema presentasse in parte vincoli che si attivano sul valore zero, in parte sull'uno [1].

Naturalmente, se il problema non dovesse avere vincoli ad attivazione condizionale (ovvero la variabile binaria nei vincoli misti non avesse la funzione di variabile di controllo attivante/disattivante) oppure se non si volesse usufruire di questa funzionalitá per qualche motivo sarebbe sufficiente non specificare l'opzione in questione sulla linea di comando ed il programma provvederebbe ad inserire sempre tutti i vincoli possibili negli slave generati.

[1] Il motivo per cui non si é adottato un approccio piú flessibile che potesse specificare singolarmente i valori di attivazione per ogni vincolo é dovuto all'overhead di dati e controlli che ció avrebbe comportato

Tagli locali e tagli globali Uno dei piú grossi problemi cui si é dovuto far fronte é la numerositá dei tagli generati. Mentre in alcuni problemi il numero di tagli necessari a raggiungere la soluzione non superava le poche migliaia, in altri si potevano superare anche i cento mila tagli generati in tempo molto brevi (e senza raggiungere ancora la conclusione dell'ottimizzazione). Questa grande quantitá di tagli che venivano aggiunti al problema master rendeva la sua gestione e la soluzione dei nodi successivi estremamente onerosa ed improponibilmente lenta.

Il principale motivo di questo fatto é da ricercarsi nel sistema di gestione dei tagli adottato da Cplex, il quale non é stato pensato per simili quantitá di tagli: Cplex aggiunge infatti i tagli in modo statico (cioé permanente) al problema che sta risolvendo, senza prevedere alcun meccanismo di purging (ovvero di romozione o disattivazione temporanea) nel caso in cui alcuni dei tagli aggiunti dovessero diventare nel tempo poco o del tutto inutili (come quando diventano slack a causa di altri tagli o della particolare zona dell'albero di branching che si sta visitando). Questo approccio risulta molto differente da quello piú tradizionale secondo il quale i tagli generati dal separatore vengono mantenuti in un pool (un contenitore) e vengono poi aggiunti o tolti dinamicamente dal tableau[2] del problema in corso di risoluzione sulla base delle necessitá del momento: questo spostamento continuo di tagli tra tableau e pool permette di mantenere sempre ridotta al minimo la quantitá di righe del tableau stesso.

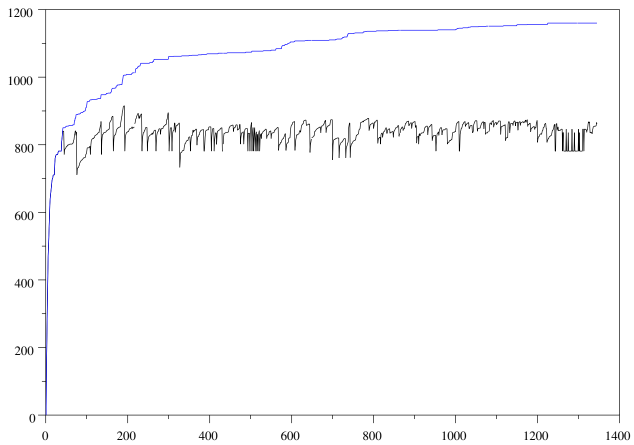
Proprio per risolvere questo inconveniente si é utilizzata una funzionalitá introdotta con la versione 8.0 di Cplex: la possibilitá di inserire tagli locali, cioé tagli la cui validitá é limitata al nodo in cui sono inseriti ed alla sua discendenza. Questa caratteristica comporta il fatto che Cplex debba eliminare questi tagli dalle sue strutture dati interne non appena termina di elaborare un nodo e ne inizia un altro che non faccia parte della discendenza del nodo in cui i tagli locali sono stati inseriti (in generale ció succede quando esegue un backtrack). Ed é proprio questo fatto ad aver permesso di simulare una sorta di gestione tradizionale dei tagli inseriti. In effetti i tagli combinatori di Benders, come giá fatto notare, hanno validitá globale e non solo locale, tuttavia l'utilizzo dei tagli locali risulta essere l'unico modo, in Cplex, di eliminare tagli precedentemente inseriti.

Quello che si é fatto é stato quindi:

- creare una struttura dati che contenesse e gestisse i tagli generati, ovvero si é implementato un pool di vincoli

- inserire i tagli generati come tagli locali in Cplex e contemporaneamente inserirli anche nel pool di modo che nell'istante in cui Cplex

[2]Per semplicitá si usano i termini propri di una risoluzione tramite algoritmo del simplesso, sebbene non sia l'unica possibile



La figura visualizza un tipico andamento della dimensione (numero di vincoli/tagli) dei nodi dell'albero di branching del problema master in funzione del numero di nodi visitati (ascisse). La curva superiore (monotona non–decrescente) rappresenta l'utilizzo di tagli globali, quella inferiore (molto frastagliata), l'utilizzo di quelli locali. Come si puó vedere dopo soli circa 300 nodi la curva superiore ha giá superato i 1000 tagli presenti mentre la curva inferiore si é stabilizzata attorno ad un valor medio di circa 800.

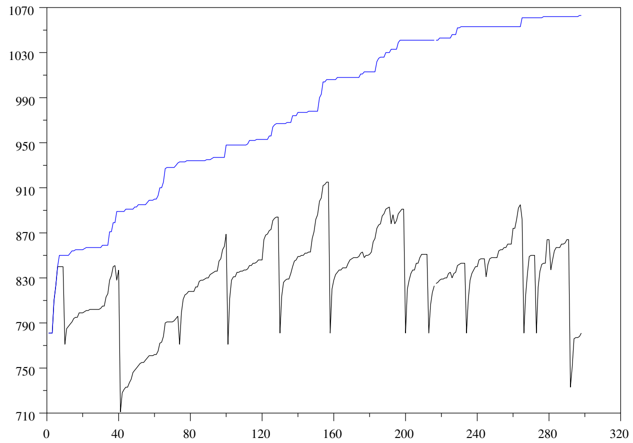
Figura 4.1: Confronto tra tagli locali e globali

elimina i tagli locali non piú validi questi non vengano completamente persi

- oltre a (o, meglio, prima di) verificare la possibilitá che la soluzione del nodo corrente generi uno slave infeasible si verifica che essa non violi i tagli del pool: se ve ne sono di violati questi vengono riaggiunti, come se fossero stati generati dall'ispezione di uno slave; in questo modo si riaggiungono solo i tagli utili, senza doverli nemmeno rigenerare e non vengono mai inseriti i tagli "slack".

Naturalmente questo approccio non é esattamente il migliore immaginabile, visto che la rimozione dei tagli segue una logica diversa (la localitá nell'albero di branching) da quella che servirebbe in questo contesto (l'utilitá a definire il rilassamento continuo al nodo corrente) e quindi tale rimozione puó essere considerata quasi aleatoria. Tuttavia resta l'unica via percorribile con Cplex e comunque porta a risultati decisamente migliori rispetto all'approccio che utilizza i tagli come validi globalmente, non appena il numero di tagli supera le poche migliaia.

Si puó avere un'idea della differenza osservando le figure 4.1 e 4.2. Essa rappresenta la dimensione (in numero di righe) dei problemi risolti ad ogni nodo (in ascisse vi é il numero progressivo che identifica i vari nodi esami-



La figura visualizza un dettaglio di quanto rappresentato nella figura 4.1. Qui sono molto ben visibili i cali nel numero dei vincoli (curva inferiore) dovuti ai backtrack eseguiti durante la soluzione del problema master, i quali causano la rimozione dei vincoli locali non pertinenti alla nuova zona dell'albero di branching visitata.

Figura 4.2: Confronto tra tagli locali e globali (dettaglio)

nati). Si notano bene i bruschi cali della curva che rappresenta l'esecuzione con i tagli di tipo locale in corrispondenza dei backtrack che si sono verificati mentre si nota il continuo aumento subito dalla curva rappresentante l'esecuzione con i tagli globali. Si vede inoltre che a differenza di quest'ultima, la prima tende a stabilizzarsi attorno ad un certo valore medio che risulta assai piú basso della maggior parte dei valori che l'altra curva assume. Ipotizzando una dipendezza approssimativemente lineare tra numero di disequazioni in un nodo e tempo di risoluzione dello stesso si puó facilmente capire quale sia il vantaggio nell'utilizzo del sistema dei tagli locali, vantaggio che compensa ampiamente l'overhead di operazioni introdotto da questo tipo di gestione.

Problemi con variabili reali nella funzione obiettivo Come indicato nella parte di spiegazione teorica, i problemi che hanno una funzione obiettivo contenente delle variabili reali hanno bisogno di una trattazione leggermente diversa rispetto a quelli in cui la funzione obiettivo é composta solo da variabili intere. Da un punto di vista implementativo questo non comporta grossi problemi; tuttavia l'assenza della funzione obiettivo nel problema master ha un impatto notevolissimo sulle prestazioni dell'algoritmo.

Data l'importanza dell'argomento questo verrá trattato a parte, all'interno di una sezione nel capitolo dedicato ai risultati computazionali. In questa sede si anticipa, senza troppe spiegazioni, una modifica che si é introdotta per migliorare la situazione; questo affinché il codice sorgente del software risulti piú comprensibile.

La modifica riguarda i soli problemi in cui la funzione obiettivo non solo coinvolge delle variabili reali ma in piú non ne coinvolge alcuna intera: in sostanza essa é del tipo: min $d^T y$. Per essi, allo slave come definito in 2.37 – 2.40 viene aggiunta la seguente disequazione:

$$d^T y \leq z^\star, \tag{4.1}$$

dove $z^\star$ é il valore della funzione obiettivo corrispondente all'ultima soluzione trovata (incumbent solution). Tale valore é inizialmente impostato a $z^\star = +\infty$. L'effetto della presenza di questa disequazione é quello di forzare un progressivo miglioramento nella sequenza di valori corrispondente alle soluzioni via via trovate.

Tagli 0–1/2 Oltre ai tagli combinatori di Benders, possono essere anche inseriti nel problema master dei tagli di diverso tipo, chiamati tagli 0–1/2. É possibile trovare una trattazione esaustiva su tali tagli in [9]. Questo tipo di tagli é specificatamente pensato per problemi che coinvolgono solo variabili binarie e che hanno coefficienti nelle disequazioni che siano solo interi. Esistono evidenze sperimentali della efficacia dei tagli 0–1/2; per questo motivo, dato che nella maggior parte dei casi il problema master che si estrae dal problema MIP originale soddisfa questo tipo di requisiti, si é deciso di utilizzarli. Inoltre anche se i vincoli del problema master non dovessero soddisfare i requisiti richiesti, certamente li soddisfano tutti i Combinatorial Benders' Cuts, i quali vengono prodotti in gran numero. Ció consente di utilizzare comunque con profitto i tagli in questione semplicemente "nascondendo" al separatore 0–1/2 le disequazioni non conformi alle richieste. Sebbene non siano stati fatti molti test per verificare l'effettiva efficacia di questo tipo di tagli nel contesto della decomposizione introdotta in questa tesi e congiuntamente ai tagli combinatori, qualche prova effettuata durante lo sviluppo del software ha evidenziato risultati molto incoraggianti: per questo, dopo aver individuato una configurazione dei parametri di controllo del separatore 0–1/2 che sembrava dare buoni risultati, si é inserita permanentemente tale procedura di separazione nel programma prodotto.

4.2 Parte specifica del codice sorgente

In questa sezione verrá descritta l'implementazione vera e propria che é stata sviluppata ed utilizzata per eseguire i test comparativi.

L'implementazione é consistita nel personalizzare il framework descritto in 3.4 ed in appendice B, aggiungendo delle opportune parti di codice sorgente in modo che potesse soddisfare alle esigenze specifiche del caso. Sebbene l'approccio fin qui descritto si discosti notevolmente da un tradizionale sistema di risoluzione di tipo B&C, il fatto che possa essere "camuffato" da separatore, ha permesso comunque lo sfruttamento quasi inalterato del framework sviluppato: in effetti l'unico vero cambiamento che si é reso necessario é stata la sostituzione dei riferimenti a Problem::mod e Problem::alg con Problem::master_mod e Problem::master_alg, che come facilmente si intuisce contengono la parte master (parte che in effetti viene risolta in luogo del problema originale).

Non verrá fatta menzione delle modifiche e delle aggiunte necessarie all'integrazione dei citati tagli 0–1/2: a tal proposito si veda [9].

I file sorgenti che hanno avuto bisogno dei principali adattamenti e che contengono il cuore di tutto il sistema di risoluzione sono problem.cpp e cutcb.cpp, con i relativi header. Questi file verranno perció esaminati con maggior dettaglio. I restanti files non hanno addirittura avuto bisogno di aggiunte oppure hanno avuto bisogno solamente di alcuni ovvii cambiamenti di lieve entitá che saranno ora brevemente descritti.

arguments.h e arguments.cpp Si é aggiunto quanto serve alla gestione e manipolazione dei parametri specifici dell'algoritmo per i Combinatorial Benders' Cuts. Questi parametri sono:

input_dir Permette di specificare la directory da cui leggere i dati di input quali i tagli generati in run precedenti o le soluzioni giá note da usare come incumbent solution iniziale (questa funzionalitá non á ancora stata completamente implementata)

active_value Di questo, che é il parametro di maggiore importanza, si é giá discusso in 4.1

preprocess e problem_type Queste opzioni sono presenti a scopo di debugging (preprocess) o per semplificare i test fatti a completamento di questa tesi (problem_type) e vengono qui menzionate solo per completezza

global_cuts Questa opzione permette di inserire i tagli come globali al posto che locali. Visto il vantaggio che i tagli locali comportano, se ne sconsiglia l'uso (anche perché non vi si é dedicata molta attenzione in fase di debugging).

Per la gestione di questi parametri é stato sufficiente inserire le corrispondenti variabili nella struttura Arguments, le relative entry nell'array Arguments::options ed il codice di conversione relativo all'interno delle strutture switch–case nel costruttore di Arguments e nella funzione di stampa, come indicato nella documentazione del framework.

cut.h Non é stata necessaria alcuna aggiunta a questo file dato che le funzionalitá offerte dal template all'implementazione dei tagli utilizzati; essendo inoltre questi ultimi a coeffiecienti interi é stato possibile istanziare il template tramite il tipo int al posto che con double.

solver.h e solver.cpp Le due importanti modifiche introdotte in questo file si sono rese necessarie solamente per porre rimedio a due bugs riscontrati nella versione 8.0 di Cplex. Il primo consisteva un in errato funzionamento del selettore automatico di algoritmo di risoluzione nei problemi LP, il quale faceva abortire bruscamente il programma: si é perció reso indispensabile selezionare manualmente l'algoritmo di risoluzione per il problema slave tramite l'istruzione:

```
prob->dual\_slave\_alg.setRootAlgorithm(IloCplex::Dual);
```

Il secondo consisteva in un errato passaggio dei valori corrispondenti alla soluzione del nodo corrente alla CutCallback. Si é osservato che disabilitando alcune altre funzionalitá di Cplex questo tipo di errori spariva, perció si sono aggiunte le seguenti istruzioni:

```
prob->master_alg.setParam(IloCplex::CutPass, -1);
prob->master_alg.setParam(IloCplex::HeurFreq, -1);
prob->master_alg.setParam(IloCplex::Probe, -1);
```

Come si vede, tra le altre, si é anche disattivata la ricerca euristica delle soluzioni; poiché questo tipo di funzionalitá é estremamente efficace si é deciso di adottare il seguente stratagemma, al fine di non privarsene completamente: si risolve il nodo radice del problema originale lasciando la ricerca euristica attivata e si utilizza la parte intera dell'eventuale soluzione trovata (generalmente in modo euristico) come starting point per l'ottimizzazione del problema master, come si evince dalle seguenti istruzioni:

```
prob->alg.setParam(IloCplex::NodeLim, 1);
prob->alg.solve();
if (prob->alg.getCplexStatus() == IloCplex::NodeLimFeas)
{  IloNumArray soluzione(env);
   prob->alg.getValues(soluzione, prob->int_vars);
   prob->master_alg.setVectors(soluzione, 0, prob->int_vars, 0, 0, 0);
   prob->master_alg.setParam(IloCplex::MIPStart, 1);
}
prob->master_alg.solve();
```

output.h e output.cpp In questi file si é semplicemente aggiunto un ulteriore stream chiamato cb_cuts per la registrazione dei tagli generati.

nodedata.h, nodedata.cpp, callbacks.h, callbacks.cpp, utils.h, utils.cpp, globals.h e globals.cpp Tutti questi files non hanno necessitato particolari modifiche.

pool.h e pool.cpp In linea di principio non sarebbero state necessarie modifiche a questi files: ne sono state fatte solamente per poter avere una gestione unificata dei due pools associati ai tagli 0–1/2 e ai tagli di Benders combinatori. Risulta superfluo entrare nel dettaglio di queste modifiche. Come giá accennato in precedenza, per l'implementazione di tagli e pool non si sono usate le classi messe a disposizione dalla Concert Technology (ovvero IloRange e IloRangeArray), in quanto esse sono risultate inadeguate alle esigenze di rapiditá necessarie: per questo al loro posto il GBF ne mette a disposizione delle nuove, piú rapide (SimpleCut e Pool).

statistics.h e statistics.cpp Sono state aggiunte alla struct Statistics alcune variabili per la raccolta di informazioni utili (quali il numero di soluzioni master intere/frazionarie, il numero di vincoli interi/misti/continui, il numero di variabili di controllo, etc.).

timers.h e timers.cpp Oltre ai cronometri giá presenti sono stati aggiunti altri tre "timers" per monitorare le seguenti tre sezioni di codice: generazione dei problemi slave, loro risoluzione, ricerca di MIS (e relativa generazione dei tagli).

4.2.1 Preprocessing: la classe Problem

I files problem.h e problem.cpp hanno avuto bisogno di sostanziali integrazioni, rispetto alla versione fornita con il framework. Il codice in essi contenuto ha il compito di analizzare il problema fornito e scomporlo nei problemi master e slave; inoltre occorre inizializzare le strutture dati di supporto che sono state previste (e ivi dichiarate) per rendere possibili le computazioni necessarie durante la risoluzione vera e propria del problema.

Prima di descrivere il contenuto del codice si informa che si é scelto di gestire esplicitamente il problema slave sottoforma di problema duale (modificato secondo quanto descritto in 2.4), facendo inoltre completamente a meno della formulazione primale: il motivo di questa scelta risiede nel fatto che questo ha permesso di gestire in modo piú semplice e diretto la ricerca dei MIS. La rinuncia alla formulazione primale ha ancor maggiormante semplificato la situazione senza comportare nessun problema: gli unici due momenti in cui essa veniva utilizzata erano il test di feasibility non appena il master forniva la soluzione del nodo corrente ed il calcolo del valore della funzione obiettivo nel caso di problemi con variabili continue nella funzione obiettivo stessa. Per quel che riguarda la prima situazione, come giá detto in 2.4, il medesimo test puó ugualmente essere fatto sul problema duale (modificato); per quel che riguarda la seconda si é deciso di utilizzare direttamente il problema originale per fare tale calcolo; ció ha comportato anche dei vantaggi: maggior precisione nel risultato dovuto alla presenza di meno frammentarietá nella computazione, disponibilitá di informazioni altrimenti inaccessibili (ad es. analisi di sensitivitá e della qualitá della soluzione) e soprattutto ció ha costituito un ottimo elemento di verifica della validitá della soluzione intera trovata (molto utile in fase di debugging).

Verrá ora presentato e commentato il contenuto di problem.h

```
/* omissis: direttive include, commenti, etc. */

class CondConstr
{
public:
   bool add( IloNumVar var, IloNumVar dual_var, int active_value );

   /* omissis: funzioni di stampa */

   /* Per ogni var attivante master, si mantiene un vettore IloNumVarArray
   di corrispondenti variabili duali "attivate" */
   vector< IloNumVarArray > dual_vars_0;
   vector< IloNumVarArray > dual_vars_1;
};

class ObjFunc
{
public:
   enum objective_t
   {     fo_int,     //solo var intere ==> f.o. e' nel master
      fo_cont,    //f.o. ha solo var continue
      fo_mista    //f.o. ha sia var cont che var int
   };
   objective_t obj_type;

   IloNum obj_value;    //ultimo valore trovato della f.o.

   IloObjective::Sense sense;

   void updateObjValue( IloNum new_obj_value );

   IloNumColumn setObjFunc( IloObjective obj, IloRangeArray dual_cons );

   /* omissis: funzioni di stampa */
};

/* Questa classe descrive il modello di un problema nella forma utile
   al risolutore master/slave con Combinatorial Benders Cuts */
class Problem
{
   /* omissis: parte "ereditata dal framework"

public:
   IloModel master_mod;
   IloCplex master_alg;
   IloNumVarArray int_vars;   //gia' nel framework
   IloRangeArray master_cons; //vincoli del master
```

```
   IloModel dual_slave_mod;
   IloCplex dual_slave_alg;
   IloNumVarArray dual_vars;     //var dello slave duale
   IloRangeArray dual_cons;      //slave dual constraints

   IloNumVarArray master_act_vars;  //var intere che attivano vincoli slave
   IloNumVarArray dual_act_vars; //var slave corrisp. a vincoli ad attiv. cond.

   IloRange dual_ex_obj;         //ex obj func (now constraint)
   IloObjective dual_artif_obj;
   IloNumVar dual_better_obj; //only for obj with only cont vars

   CondConstr * cond_constr;
   ObjFunc * obj_func;
};

/* Questa classe congloba tutti i dati che devono essere associati ad
   una singola variabile */
class VarData
{
   /* omissis: parte "ereditata" dal framework */

public:
   VarData( int _index,  int _int_index, int _cont_index = -1 )
      :     index(_index), int_index(_int_index), cont_index(_cont_index),
         act_int_index(-1), dual_index(-1), dual_act_index(-1), assoc_int_index(-1)
   {}

   int act_int_index;   //index in Problem::act_int_vars

   int dual_index;      //index in Problem::dual_vars
   int dual_act_index;  //index in Problem::dual_act_vars
   int assoc_int_index; //index of master int var that activate this dual
};

/* Inlined Functions for easy access to VarData members
    Viene riportata solo la prima, le altre sono identiche; esse sono:
   getIntVarIndex, getContVarIndex, getActiveIntVarIndex,
   getDualVarIndex, getActiveDualVarIndex, getAssocIntVarIndex */

inline int & getVarIndex( IloNumVar v )
{ return static_cast< VarData * >(v.getObject())->index; }
```

Come si osserva sono state aggiunte due classi: CondConstr e ObjFunc. Quest'ultima serve a gestire la funzione obiettivo e si compone di due funzioni membro (una di inizializzazione, setObjFunc, ed una per l'aggiornamento durante l'algoritmo, updateObjValue) e di altri campi dal nome autoesplicativo che descrivono la funzione obiettivo stessa. La prima serve invece da "collante" tra le variabili binarie di controllo del problema master ed i rispettivi vincoli ad attivazione condizionale del problema slave: visto che il problema slave é dato nella sua formulazione duale ció che ne rappresenta un vincolo é la variabile duale corrispondente. In sostanza la struttura dati contenuta nella classe serve a mantenere una corrispondenza tra ogni variabile di controllo e l'insieme delle variabili duali che rappresentano i vincoli controllati ovvero dual_vars_0[i] contiene l'array che descrive l'insieme delle variabili duali che corrispondono ai vincoli controllati dall'i–esima variabile binaria (ovvero da Problem::master_act_var[i]). Il suffisso zero ed il motivo per cui sembra esserci una duplicazione dell'informazione (nella omologa variabile con il suffisso uno) verrá analizzato tra breve. La funzione membro CondConstr::add é usato in fase di costruzione del problema slave per inizializzare il contenuto delle due variabili membro della classe.

Le due istanze di queste due classi relative al problema in corso di risoluzione vengono gestite tramite i due puntatori contenuti nella classe Problem. Quest'ultima classe si configura come un "contenitore" in cui si immagazzianno tutti i dati relativi al problema corrente e quindi, oltre a tali puntatori ed a quanto ereditato dal framework, essa incapsula delle variabili atte a descrivere i problemi master e slave: modello, algoritmo, array di vincoli e array di variabili; inoltre per il master: array di variabili corrispondente al sottoinsieme delle variabili intere che sono anche variabili di controllo – master_act_vars –; e per lo slave: array di variabili corrispondenti al sottoinsieme di vincoli della formulazione primale che sono vincoli ad attivazione condizionale – dual_act_vars –, vincolo corrispondente alla funzione obiettivo duale – dual_ex_obj – (v. equazione 2.52), funzione obiettivo duale somma delle variabili duali – dual_artif_obj – (v. equazione 2.53), e variabile duale corrispondente al vincolo primale che viene introdotto per gestire problemi con funzione obiettivo costituita da sole variabili reali.

Infine vi é la classe VarData a cui sono state aggiunte quattro variabili intere che fungono da indice ovvero servono ad indicare la posizione della variabile a cui l'istanza di VarData é associata all'interno di determinati array, indicati nei commenti: naturalmente, in base alla natura della variabile (intera, di controllo, slave duale, ecc.) alcuni di tali indici possono non aver senso: in tal caso sono impostati al valore -1. Associata ad ognuno di questi indici vie é una funzione globale inlined, fornita semplicemente per comoditá.

Descritte le strutture dati, é ora possibile passare al contenuto di Problem.cpp dove vi sono le implementazioni dei metodi.

```
Problem::Problem()
{
   // Crea Modello e Algoritmo per il problema
   mod = IloModel(env);
   // A questo punto viene interrogato il license manager e viene stampato
   // un messaggio su cerr (forse perche' e' il 1o IloAlgorithm che si istanzia)
   cerr << "\n\tVerifing license... ";
   alg = IloCplex(env);
   cerr << "\tending license verify... continuing preprocess...";

   // Crea Algoritmo e Modello per il master
```

```
   master_mod = IloModel(env);
   master_alg = IloCplex(env);

   // Crea Algoritmo e Modello per lo slave
   dual_slave_mod = IloModel(env);
   dual_slave_alg = IloCplex(env);

   // Importa il modello per preprocessarlo
   obj = IloObjective(env);
   vars = IloNumVarArray(env);
   cons = IloRangeArray(env);
   alg.importModel(mod, args->input_file.c_str(), obj, vars, cons);

   processVars();

   obj_func = new ObjFunc;
   IloNumColumn better_obj = obj_func->setObjFunc(obj, dual_cons);
   switch (obj_func->obj_type)
   {  case ObjFunc::fo_int:
         better_obj.end();
         master_mod.add(obj);
      break;
      case ObjFunc::fo_cont:
         better_obj += dual_artif_obj(1) + dual_ex_obj(0); //(0) verra' modif.
         dual_better_obj = IloNumVar(better_obj, 0, 0, ILOFLOAT, "dual_better_obj");
         better_obj.end();
         dual_slave_mod.add(dual_better_obj);
      break;
      case ObjFunc::fo_mista:
         better_obj.end();
      break;
   }

   if (!processCons()) {exit(-1);}

   alg.extract(mod);
   master_alg.extract(master_mod);
   dual_slave_alg.extract(dual_slave_mod);

   if (args->preprocess)
   {  /* omissis: salva su disco l'istanza di problem */ }

}  //preprocess

/* Suddivide le var in intere (master) e continue (slave) e per ogni var
cont crea il vincolo duale corrispondente (ed eventualmente le var duali
associate ai vincoli di bounds sulla var primale cont) */
void Problem::processVars()
{  int_vars = IloNumVarArray(env);
   cont_vars = IloNumVarArray(env);
   int int_var_idx = -1;
   int cont_var_idx = -1;
   int num_bin_vars = 0;   //contatore
   int size = vars.getSize();
```

```
for (int i = 0; i < size; ++i)
{
   IloNumVar v = vars[i];
   switch (v.getType())
   {  case ILOBOOL:
         ++num_bin_vars;
         master_mod.add(v);
         int_vars.add(v);
         vars[i].setObject(static_cast< IloAny >(new VarData(i, ++int_var_idx)));
      break;
      case ILOINT:
         if (v.getLB() == 0 && v.getUB() == 1)
         {  master_mod.add(IloConversion(env, v, ILOBOOL));
            ++num_bin_vars;
         }
         else
         {  master_mod.add(v);
         }
         int_vars.add(v);
         vars[i].setObject(static_cast< IloAny >(new VarData(i, ++int_var_idx)));
      break;
      case ILOFLOAT:
         cont_vars.add(v);
         vars[i].setObject(static_cast< IloAny >(new VarData(i, -1, ++cont_var_idx)));
      break;
   }     //switch
   if (i % 100 == 0) cout << ".";
}  //for
stats->num_vars = size;
stats->num_int_vars = int_var_idx + 1;
stats->num_bin_vars = num_bin_vars;
stats->num_cont_vars = cont_var_idx + 1;

// Per ogni var cont, sistema il duale
// (si poteva fare direttam. nel for precedente, ma cosi' e' piu' chiaro)
dual_vars = IloNumVarArray(env);
dual_cons = IloRangeArray(env);
dual_artif_obj = IloMinimize(env, 0, "artif_obj");
dual_slave_mod.add(dual_artif_obj);
dual_ex_obj = IloRange(env, 1, IloInfinity, "ex_obj");
dual_slave_mod.add(dual_ex_obj);
int dual_vars_index = -1;
for (int i = 0; i < stats->num_cont_vars; ++i)
{  // Crea il vincolo duale corrispondente alla var
   IloRange r(env, 0, 0, cont_vars[i].getName());
   dual_cons.add(r);
   dual_slave_mod.add(r);
   // Se la var ha dei bounds (cioe' dei vincoli) crea le var duali corrisp.
   if (cont_vars[i].getUB() < INF)
   {
      IloNumColumn c(env);
      c += dual_artif_obj(1) +
      dual_ex_obj(-cont_vars[i].getUB()) + r(-1);
      string nome= string(cont_vars[i].getName()) + "_ub";
```

```
         IloNumVar u(c, 0, IloInfinity, ILOFLOAT, nome.c_str());
         u.setObject(static_cast< IloAny >(new VarData(-1, -1, -1)));
         getDualVarIndex(u) = ++dual_vars_index; dual_vars.add(u);
         dual_slave_mod.add(u);
      }
      if (cont_vars[i].getLB() > -INF)
      {  IloNumColumn c(env);
         c += dual_artif_obj(1) + dual_ex_obj(cont_vars[i].getLB()) +r(1);
         string nome = string(cont_vars[i].getName()) + "_lb";
         IloNumVar u(c, 0, IloInfinity, ILOFLOAT, nome.c_str());
         u.setObject(static_cast<IloAny >(new VarData(-1, -1, -1)));
         getDualVarIndex(u) = ++dual_vars_index;
         dual_vars.add(u);
         dual_slave_mod.add(u);
      }
      if (i % 100 == 0) cout << ".";
   }
}  //processVars

bool Problem::processCons()
{  // Duplica i vincoli "ranged" (anche i vincoli "lhs == rhs")
   // Pone inoltre tutti i vincoli nella forma "expr >= rhs"
   IloRangeArray tmp_cons = cons;
   cons = IloRangeArray(env);
   int size = tmp_cons.getSize();
   for (int i = 0; i < size; ++i)
   {  IloRange r = tmp_cons[i];
      IloNum lb = r.getLB();
      IloNum ub = r.getUB();
      if (lb > -INF && ub < INF)
      {  IloExpr expr = getExprFromRange(r);
         IloRange new_r1 = (-expr >= -ub);      expr.end();
         expr = getExprFromRange(r);
         IloRange new_r2 = (expr >= lb);  expr.end();

         if (r.getName() != NULL)
         {  string name(r.getName());
            new_r1.setName((name + "_ub").c_str());
            new_r2.setName((name + "_lb").c_str());
         }

         cons.add(new_r1);  //sostituisce il vecchio con uno dei nuovi
         cons.add(new_r2);    //aggiunge il secondo nuovo
      }
      else if (ub < INF)
      {  IloExpr expr = getExprFromRange(r);
         IloRange new_r = (-expr >= -ub);    expr.end();
         new_r.setName(r.getName());
         cons.add(new_r);
      }
      else    {   cons.add(tmp_cons[i]);  }

      if (i % 100 == 0) cout << ".";
   }
```

```
tmp_cons.end();
stats->num_cons = cons.getSize();   //posso aver aggiunto vincoli

dual_act_vars = IloNumVarArray(env);
master_cons = IloRangeArray(env);
master_act_vars = IloNumVarArray(env);
cond_constr = new CondConstr;
int num_vinc_int = 0;
int num_vinc_cont = 0;
int num_vinc_misti = 0;
int dual_vars_index = dual_vars.getSize() - 1;
int dual_act_vars_index = -1;
bool prob_ok = true;   //false se problema non e' risolv con CombBendCuts

for (int i = 0; i < stats->num_cons; ++i)
{
   IloRange r = cons[i];
   IloExpr expr = getExprFromRange(r);
   int num_var_bin = 0;
   int num_var_gen = 0;    //general int: intere ma non binarie
   int num_var_cont = 0;
   IloNumVar act_v;  //var bin "attivante"
   IloNum coef;   //coeff di act_v;
   IloNumColumn col_0(env);   //corrisp a parte cont nel duale (0 active)
   IloNumColumn col_1(env);   //c.s. (1 active)
   for (IloExpr::LinearIterator lit(expr); lit.ok(); ++lit)
   {  IloNumVar v = lit.getVar();
      switch (v.getType())
      {  case ILOFLOAT:
            num_var_cont++;
            col_0 += dual_cons[getContVarIndex(v)](lit.getCoef());
            col_1 += dual_cons[getContVarIndex(v)](lit.getCoef());
         break;
         case ILOINT:
            if (v.getLB() == 0 && v.getUB() == 1)
            {  num_var_bin++;
               act_v = v;
               coef = lit.getCoef();
            }
            else      { num_var_gen++;   }
         break;
         case ILOBOOL:
            num_var_bin++;
            act_v = v;
            coef = lit.getCoef();
         break;
      }
   }
   int num_var_int = num_var_gen + num_var_bin;
   if (num_var_cont != 0 && num_var_int == 0)
   {  num_vinc_cont++;
      col_0 += dual_artif_obj(1) + dual_ex_obj(r.getLB());  //col_0 o 1 e' =
      IloNumVar u(col_0, 0, IloInfinity, ILOFLOAT, r.getName());
      VarData * vdp = new VarData(-1, -1, -1);
```

```
         vdp->dual_index = ++dual_vars_index;
         dual_vars.add(u);
         u.setObject(static_cast< IloAny >(vdp));
         dual_slave_mod.add(u);
      }
      else if (num_var_cont == 0 && num_var_int != 0)
      {  num_vinc_int++;
         master_mod.add(r);
         master_cons.add(r);
      }
      else if (num_var_cont != 0 && num_var_int != 0)
      {  num_vinc_misti++;
         if (num_var_bin != 1 || num_var_gen != 0) {prob_ok = false; }
         else
         {  col_0 += dual_artif_obj(1);   //10
            col_1 += dual_artif_obj(1);   //10
            if (args->active_value != 0)
            {  col_1 += dual_ex_obj(r.getLB() - coef);
               IloNumVar u(col_1, 0, 0, ILOFLOAT, r.getName());
               VarData * vdp = new VarData(-1, -1, -1);
               vdp->dual_index = ++dual_vars_index;
               vdp->dual_act_index = ++dual_act_vars_index;
               dual_act_vars.add(u);
               dual_vars.add(u);
               u.setObject(static_cast< IloAny >(vdp));
               if (cond_constr->add(act_v, u, 1)) { master_act_vars.add(act_v); }
               dual_slave_mod.add(u);
            }
            if (args->active_value != 1)
            {  col_0 += dual_ex_obj(r.getLB());
               IloNumVar u(col_0, 0, 0, ILOFLOAT, r.getName());
               VarData * vdp = new VarData(-1, -1, -1);
               vdp->dual_index = ++dual_vars_index;
               vdp->dual_act_index = ++dual_act_vars_index;
               dual_act_vars.add(u);
               dual_vars.add(u);
               u.setObject(static_cast< IloAny >(vdp));
               if (cond_constr->add(act_v, u, 0)) { master_act_vars.add(act_v); }
               dual_slave_mod.add(u);
            }
         }
      }
      col_0.end();
      col_1.end();
      expr.end();
      if (i % 100 == 0) cout << ".";
   }
   stats->num_int_cons = num_vinc_int;
   stats->num_cont_cons = num_vinc_cont;
   stats->num_mixed_cons = num_vinc_misti;
   return prob_ok;
}   //processCons
```

```
// true se e' una "nuova" var master attivante
bool CondConstr::add( IloNumVar var, IloNumVar dual_var, int active_value )
{
   // Cerca se questa var e' gia' stata aggiunta da precedenti chiamate ad add
   int act_int_index = getActiveIntVarIndex(var);

   bool ret = false;
   if (act_int_index == -1)
   {  // Aggiungiamo una nuova variabile
      IloNumVarArray vec_0(env);
      IloNumVarArray vec_1(env);
      dual_vars_0.push_back(vec_0);
      dual_vars_1.push_back(vec_1);

      getActiveIntVarIndex(var) = act_int_index = dual_vars_0.size() - 1;
      ret = true;
   }

   // Aggiungiamo le nuove informazioni
   if (active_value != 1)
   {  dual_vars_0[act_int_index].add(dual_var); // Attivi sullo zero     }
   if (active_value != 0)
   {  dual_vars_1[act_int_index].add(dual_var); // Attivi sull'uno   }

   getAssocIntVarIndex(dual_var) = act_int_index;
   return ret;
}

/* A causa di un bug, questa funzione esiste nell'implementation ma non
nell'handle di IloObjective (sebbene sia documentata); non chiamare end()
sull'espressione! (e' un handle copia di quello interno dell'IloObjective) */
#define getExpr() getImpl()->getExpr()

IloNumColumn ObjFunc::setObjFunc( IloObjective obj, IloRangeArray dual_cons )
{
   sense = obj.getSense();
   int coef_sign;    //per rendere il taglio che forza miglioramenti come >=
   if (sense == IloObjective::Minimize)
   {  obj_value = IloInfinity;
      coef_sign = -1;
    }
   else
   {     obj_value = -IloInfinity;
      coef_sign = 1;
   }

   int num_var_int = 0;
   int num_var_cont = 0;
   IloNumColumn ret(env);
   IloExpr expr = obj.getExpr(); //non fare expr.end() per come l'ho definita
   IloExpr parte_int(env);

   for (IloExpr::LinearIterator lit(expr); lit.ok(); ++lit)
   {
```

```
      IloNumVar lit_var = lit.getVar();
      IloNum lit_coef = lit.getCoef();
      if (lit_var.getType() == ILOFLOAT)
      {  ++num_var_cont;
         int idx = getContVarIndex(lit_var);
         ret += dual_cons[idx](coef_sign * lit_coef);
      }
      else
      {  ++num_var_int;
         parte_int += lit_var * lit_coef;
      }
   }

   if (num_var_cont == 0 && num_var_int == 0)
   {  cout << "\nWarning: obj func is empty";   }
   else if (num_var_cont == 0 && num_var_int != 0)
   {  obj_type = fo_int;   }
   else if (num_var_cont != 0 && num_var_int == 0)
   {  obj_type = fo_cont;     }
   else if (num_var_cont != 0 && num_var_int != 0)
   {  obj_type = fo_mista;    }
   return ret;
}

void ObjFunc::updateObjValue( IloNum new_obj_value )
{
   obj_value = new_obj_value;
   if (sense == IloObjective::Minimize)
   {  prob->dual_ex_obj.setCoef(prob->dual_better_obj, -new_obj_value + 1e-6);
      prob->dual_better_obj.setUB(IloInfinity);
   }
   else
   {  prob->dual_ex_obj.setCoef(prob->dual_better_obj, new_obj_value + 1e-6);
      prob->dual_better_obj.setUB(IloInfinity);
   }
}
```

Prima di passare alla descrizione delle funzioni appena riportate occorre spiegare una importante scelta di progetto relativa all'implementazione del problema slave: essa riguarda le modalitá di sostituzione dei valori delle variabili binarie entro lo slave e di memorizzazione dei vincoli ad attivazione condizionale. Come é noto, ognuno di tali vincoli assume la forma

$$a_i^T y \geq b_i - m_{i,j} x_j.$$

In base al valore della variabile x_j nella soluzione del nodo corrente del problema master, tale vincolo dovrebbe comparire nel problema slave primale in una delle due seguenti forme:

$$a_i^T y \geq b_i \text{ oppure } a_i^T y \geq b_i - m_{i,j}.$$

Si é detto dovrebbe perché in realtá il vincolo, per effetto del coefficiente $m_{i,j}$, potrebbe corrispondere ad uno stato "disattivato" in una delle due forme e quindi essere presente per un certo valore di x_j ma completamente assente dal problema slave per il valore complementare. Per questo motivo, si é deciso di generare e memorizzare ogni vincolo condizionale in entrambe le sue due forme ed "inserire nel" o "eliminare dal" problema slave la versione appropriata del vincolo, sulla base del valore della variabile binaria di controllo. L'operazione di sostituzione dei valori delle variabili binarie nel problema slave viene quindi a cessare ed é sostituita da opportune operazioni di inserimento e rimozione: tutto ció permette di realizzare uno dei propositi che ci siera posti fin dall'inizio, ovvero l'assenza dei coefficienti "big M" che si manifesta nella presenza nello slave dei soli vincoli "veramente attivi". Si ricorda ora che il problema slave é gestito direttamente in forma duale e non primale. Occorre perció tradurre le operazioni di inserimento di un vincolo ed eliminazione dello stesso in termini duali: queste corrisponderebbero alla inserzione ed eliminazione di una variabile duale. Tuttavia esiste un metodo piú semplice: é sufficiente fissare la variabile duale in questione a zero (agendo sul suo upper bound, dato che il lower bound é giá fissato a zero) per ottenere l'effetto di cancellazione dal problema e ripristinarne i bounds originali per ottenere l'inserzione. Questo approccio ha l'enorme vantaggio, rispetto ad una vera inserzione, di non dover reintrodurre (e quindi anche mantener memorizzati altrove) tutti i coefficienti per cui la variabile duale andava moltiplicata nei vari vincoli: essi restano presenti sempre nella matrice del problema slave ed utilizzati solo quando serve (quando cioé la corrispondente variabile non é fissata a zero). In definitiva perció ogni vincolo ad attivazione condizionale viene sdoppiato nelle sue due versioni le quali vengono entrambe inserite nel problema slave[3]: occorrerá poi di volta in volta sistemare opportunamente gli upper bounds delle corrispondenti variabili duali per garantire la consistenza degli slave via via generati (ovvero per garantire che al massimo una delle due versioni di uno stesso vincolo sia presente in un medesimo slave). É per questo che la classe CondConstr prevede la duplicazione delle sue strutture dati interne cui si é accennato in precedenza (v. variabili dusl_vars_0 e dual_vars_1). Si vuole infine far notare come il fissare una variabile duale a zero abbia una semplicissima interpretazione se rivista alla luce dell'algoritmo di ricerca di MIS: significa impedire che il vincolo associato faccia parte di un qualsiasi MIS (cosa ovvia visto che il vincolo non dovrebbe nemmeno essere presente nel problema).

La funzione ObjFunc::updateObjValue viene invocata solo per quei problemi con la funzione obiettivo senza variabili intere: ha il solo scopo di aggiornare il membro destro della disequazione 4.1; poiché tale disequazione é inizialmente "disattivata" (il membro destro fissato a $+\infty$ nel problema primale), occorre assicurarsi di riattivarla durante l'esecuzione (fissando l'upper bound della variabile duale associata a $+\infty$).

[3]Un problema siffatto sarebbe probabilmente infeasible, se non si apportassero le opportune modifiche grazie ai cambiamenti dei bounds delle variabili duali

La funzione ObjFunc::setObjFunc ha lo scopo di inizializzare l'istanza corrente di ObjFunc sulla base della funzione obiettivo che gli viene passata come parametro. Inoltre costruisce e restituisce l'istanza di IloColumn che costituirá la variabile duale corrispondente al vincolo 4.1: ció é fatto indipendentemente dal tipo di funzione obiettivo del problema ma risulterá utile solo per i problemi non contenenti variabili intere nella funzione obiettivo stessa.

La funzione CondConstr::add serve ad aggiungere una coppia variabile binaria di controllo – variabile duale "controllata" alla struttura dati mantenuta dalla classe CondConstr. Il parametro active_value indica se l'attivazione della variabile duale avviene sul solo valore zero, sul solo valore uno o su entrambi (in tal caso varrá -1). La prima chiamata alla funzione getActiveIntVarIndex serve a verificare se var sia giá presente nella struttura dati a causa di precedenti chiamate ad add: se sí allora tale funzione restituirá il suo indice nei due vettori dual_vars_0 e dual_vars_1, altrimenti essa restutuirá -1 e si provvederá perció a creare un nuovo posto nei due array per questa nuova variabile di controllo. Il valore booleano restituito dalla funzione add serve a distinguere tra questi due casi.

Il costruttore della classe Problem ha il compito di inizializzare opportunamente tutti i campi della classe e contestualmente generare quindi i problemi master e slave: per far ció si avvale essenzialmente delle due funzioni di supporto Problem::processVars e Problem::processCons.

Problem::processVars assolve i seguenti compiti:

- suddivide le variabili in intere/binarie e continue inserendole negli opportuni array
- per ogni variabile continua, genera l'IloRange che costutuirá il vincolo associato a tale variabile nella formulazione duale del problema slave;
- poiché la forma in cui va posto il problema slave primale per la ricerca dei MIS prevede che le variabili siano "free" (ovvero che senza bounds), gli eventuali bounds sulle variabili continue vanno tradotti in disequazioni esplicitamente presenti; in termini di slave duale ció significa generare la variabile corrispondente a questi nuovi vincoli e costruire la colonna di coefficienti moltiplicatori di tale variabile nella matrice del problema duale (IloColumn).

Il primo punto viene realizzato dal codice contenuto nel primo ciclo for, gli altri due nel secondo ciclo for della funzione. Inoltre, per ogni variabile dello slave duale o del problema originale viene creata ed inizializzata una opportuna istanza della classe VarData.

La funzione Problem::processCons presenta invece il seguente funzionamento: il primo ciclo for ha il compito di trasformare i "ranged constraints" (vincoli con ambedue i bounds finiti, il che include anche i vincoli rappresentanti equazioni e non disequazioni) in una coppia di disequazioni entrambe in forma di maggiore–uguale: questa operazione viene fatta solo per semplificare ed uniformare la trattazione successiva dei vincoli del problema. Il secondo ciclo for ha il compito di analizzare i vincoli del problema individuando quali devono finire nel master e quali nello slave, ed in questo caso se si tratta o meno di vincoli ad attivazione condizionale. Esso si compone essenzialmente di due parti: nella prima, grazie ad un IloExpr::LinearIterator si scandisce il vincolo corrente individuandone il tipo; contestualmente si genera una coppia di IloColumn con la parte del vincolo che coinvolge le sole variabili continue, per un successivo uso. La seconda é una cascata di if–then–else che, sulla base delle informazioni raccolte nella prima parte procede nella costruzione di master e slave. In particolare, nel caso di vincoli ad attivazione condizionale si utilizzano le due copie di IloColumn per generare le due variabili duali che rappresentano le due versioni complementari del vincolo e si provvede a chiamare `cond_constr->add` per aggiornare la struttura dati puntata da tale variabile. Tra le altre cose, processCons é anche in grado di capire se un problema sia effettivamente conforme alle restrizioni indicate in 2.3.1 o meno: il valore booleano restituito dalla funzione descrive proprio questa situazione.

4.2.2 Risoluzione: la classe CutCBI

In questa sottosezione si analizzaranno in dettaglio i files cutcb.h e cutcb.cpp, i quali contengono il codice che implementa il separatore che provvede alla generazione dei tagli combinatori di Benders. Ecco il contenuto del file header.

```
/* omissis: includes, commenti, etc. */

typedef IloArray< IloNumArray > IndexArray;

class CutCBI: public virtual IloCplex::CutCallbackI
{
protected:
   /* omissis: dichiarazioni relative ai tagli 0-1/2 */

   IloNumArray sol;  //soluzioni, cosi' non serve inizial. ad ogni chiamata
   int added_cuts;   //num tagli nuovi generati da questa chiamata a callback
   Pool * pool;

   int depth;        //profondita' del nodo corrente
   int last_depth;    //profondita' ultima chiamata a callback
   int this_node;   //nodeId at this call
   int last_node;   //nodeId at last call
   DatiNodo * dnp;

   IloModel d_m;   //conterra' prob->dual_slave_mod (per comodita')
   IloCplex d_a;   //conterra' prob->dual_slave_alg (per comodita')
   IloNumVarArray vars;     //conterra' prob->master_int o act_vars (comodita')
   int num_vars;      //cardinalita' di vars
```

```
   IloNumArray old_sol;    //soluzione precedente
   bool sol_int;     //true se la sol corrente e' intera

   IloNum eps_int;    //tolleranza per test interezza soluzioni
   IloNum eps_feas;  //tolleranza per test feasibility

   bool pool_hit;   //true se tagli violati dal pool


   // Ritorna true se la sol corrente e' intera
   // Arrotonda i valori di sol "quasi interi"
   bool    isIntegerFeasible();

   // "Generano" lo slave corrispond. alla parte intera della sol. corrente
   void generateSlave();
   void setCondConstr( int zero_one, int cc_idx );
   void removeCondConstr( int zero_one, int cc_idx );

   // Cerca tagli violati nel pool e se ci sono li aggiunge al master
   bool poolCuts(Pool::cut_type_t cut_type);

   // Genera un taglio completo (senza minimizzarlo)
   IloRange getFullCut();
   // Genera i tagli
   IloRangeArray makeCuts();
   // Genera un insieme di MIS (di cardinalita' >= 1)
   void getMISSet( IndexArray idx_a );
   //      Cerca di individuare una singola MIS, possibilmente minimale
   IloNumArray dualMISSearch( IloNumArray idx );

   // Alg euristico greedy per ricerca sol int infeasible a partire da sol fraz
   bool greedyHeur();

   // Trova sol slave corrisp. a sol master intera; aggiorna slave e f.o.
   IloNumArray getSlaveSol_And_Update();

public:
   CutCBI();
   IloCplex::CallbackI * duplicateCallback() const {
      return new(env) CutCBI(*this);
   }

   void mainCBC(); // Funzione principale che implementa i CombBendersCuts
   void main012(); // Funzione principale che implementa i tagli 0-1/2
   void main();
};
```

Come si puó facilmente osservare la classe CutCBI, derivante da CutCallbackI, semplicemente definisce una serie di funzioni membro che sono di supporto alla funzione main() nel calcolo dei tagli combinatori di Benders, piú una serie di variabili atte a mantener traccia dello stato del separatore tra diverse chiamate effettuate dal risolutore del problema master. Prima di passare alla descrizione dell'implementazione delle funzioni si vuole rendere noto un accorgimento che si é adottato per rendere maggiormente veloce la generazione e la risoluzione dei vari problemi slave. Il procedimento utilizzato trae origine dalla seguente osservazione: il meccanismo di branching utilizzato per default da Cplex consiste nello scendere il piú possibile in profonditá lungo un ramo dell'albero alla ricerca di soluzioni intere ed effettuare un backtrack solo quando queste vengano trovate o un nodo non generi discendenza. Da questo segue che la sequenza di nodi generata presenta delle differenze sostanziali, in termini di variabili fissate ad un valore intero, solo quando si esegue un backtrack: negli altri casi la maggior parte delle variabili che erano intere al nodo padre lo saranno anche al nodo figlio, con l'aggiunta della variabile sulla quale si é effettuato il branching ed eventualmente di altre, in base alla struttura del problema. Questa osservazione é confermata da dei test che si sono effettuati su alcuni problemi: da questi test si osserva che generalmente meno del $5 \div 10\%$ delle variabili intere variano il loro valore tra un nodo e la propria discendenza, non appena esse abbiano assunto un valore intero. Naturalmente, visto che i problemi slave che vengono generati sono direttamente dipendenti dai valori interi assunti dalle variabili binarie, tutto il ragionamento fatto fin'ora per i nodi dell'albero di branching del problema master puó essere ripetuto per i vari problemi slave. Per questo motivo si é deciso di utilizzare un procedimento che potremmo definire "differenziale", per construire i problemi slave: in pratica si tratta di mantere memorizzata la struttura dell'ultimo slave generato e di effettuare le sole modifiche corrispondenti alla nuova soluzione fornita dal nodo master corrente. Per far ció l'unica informazione che é necessario mantenere memorizzata é la soluzione dell'ultimo nodo per il quale si é costruito lo slave corrispondente (proprio a questo serve la variabile CutCBI::old_sol): confrontando quest'ultima con la soluzione corrente é possibile risalire in modo molto semplice alle modifiche da effettuarsi al problema slave per portarlo alla forma corrispondente ai valori assunti dalle variabili intere del nodo corrente. Tali modifiche consisteranno quindi nel fissare opportunamente a 0 (per disattivare) o a $+\infty$ (per attivare) le variabili duali per le quali la corrispondente variabile di controllo binaria ha cambiato valore rispetto all'ultima soluzione memorizzata. Questo tipo di approccio ha comportato dei vantaggi, dal punto di vista temporale, sia per quel che riguarda la generazione degli slave che per quel che rigurada la loro risoluzione. Infatti tutti gli elementi del problema sono gestiti tramite le strutture offerte dall'interfaccia C++ Concert Technology 1.2: durante la fase di sviluppo si é reso palese il fatto che tale software non era stato pensato per un utilizzo massiccio e continuato all'interno di un algoritmo di risoluzione ma solo quale sistema di modellizzazione (quindi da utilizzarsi solo in fase pre–risolutiva) e perció non garativa prestazioni sufficienti anche per il semplice compito di generazione ripetuta dei problemi slave. Grazie all'approccio differenziale si é minimizzato l'impatto di questo problema.

Per quel che rigurda la risoluzione, questo modo di procedere permette il riutilizzo di gran parte dell'informazione ottenuta dalla risoluzione del problema slave immediatamente precedente dato che la soluzione del problema corrente diventa una semplice riottimizzazione dell'ultimo a seguito delle poche modifiche introdotte. Informazioni statistiche raccolte durante la fase di sviluppo hanno evidenziato gli ottimi risultati ottenuti da questa soluzione, rispetto ad altre che sono state tentate.

Si puó ora passare al contenuto del file cutcb.cpp.

```
CutCBI::CutCBI()
   :  sol(env), added_cuts(0), depth(0), last_depth(0), this_node(0),
      d_m(prob->dual_slave_mod), d_a(prob->dual_slave_alg),
      vars(prob->int_vars), old_sol(env)
{
   eps_int = prob->master_alg.getParam(IloCplex::EpInt);
   eps_feas = prob->master_alg.getParam(IloCplex::EpRHS);
   num_vars = vars.getSize();
   old_sol.add(num_vars, 0.5);   //cosi' costruisce correttam il primo slave
   SimpleCut< int >::setVars(vars);
   SimpleCut< int >::setEpsFeas(eps_feas*100); //problemi di stabilit\'a num. per certi probl.
   pool = new Pool(eps_feas);

   /* omissis: inizializzazione per tagli 0-1/2 */
}

/* Ritorna true se la sol corrente e' intera. Come effetto collaterale
(necessario), arrotonda i valori di sol "quasi interi". I timers sono gestiti
direttamente da CutCBI::main(). Inoltre inizializza anche xstar. */
bool
CutCBI::isIntegerFeasible()
{  int num_fraz = 0;
   for (int i = 0; i < num_vars; ++i)
   {  int rounded = (int)IloRound(sol[i]);
      IloNum tmp = IloAbs(sol[i] - rounded);
      if (tmp > eps_int) { ++num_fraz; }
      else { sol[i] = rounded; }
   }
   return num_fraz == 0;
}

/* Restituisce true se esistono tagli violati nel pool (in tal caso questi
 tagli sono anche aggiunti al master, concordemente a args->global_cuts) */
bool CutCBI::poolCuts(Pool::cut_type_t cut_type)
{
   timers->start(Timers::pool_search);
   IloRangeArray vio_cuts = pool->getViolatedCuts(sol, cut_type);

   int size = vio_cuts.getSize();
   for (int i = 0; i < size; ++i)
   {  if (args->global_cuts)   { add(vio_cuts[i]);     vio_cuts[i].end();     }
      else  {    addLocal(vio_cuts[i]);     }
   }
```

```
   if (size != 0)
   {  stats->pool_access += 1;
      stats->pool_hits += size;
      dnp->node_cons += size;
      output_logs->solver << " pool " << size << "\n";
      output_logs->solver.flush();
   }
   vio_cuts.end();
   timers->stop(Timers::pool_search);
   return size != 0;
}

/* Utility per impostare i bounds di un IloRange dello slave
 Riceve l'indice (relativo a prob->cond_constr) degli IloRange da impostare */
void CutCBI::setCondConstr( int zero_one, int cc_idx )
{  IloNumVarArray va;
   if (zero_one)      { va = prob->cond_constr->dual_vars_1[cc_idx];    }
   else    {   va = prob->cond_constr->dual_vars_0[cc_idx];    }
   for (int j = va.getSize() - 1; j >= 0; --j)
   {  va[j].setUB(IloInfinity);
   }
}

/* Utility per rimuovere un gruppo di conditional constraints dello slave
   Riceve l'indice (relativo a prob->cond_constr) degli IloRange da impostare */
void CutCBI::removeCondConstr( int zero_one, int cc_idx )
{  IloNumVarArray va;
   if (zero_one)   { va = prob->cond_constr->dual_vars_1[cc_idx];    }
   else    {   va = prob->cond_constr->dual_vars_0[cc_idx];    }
   for (int j = va.getSize() - 1; j >= 0; --j)
   {  va[j].setUB(0);
   }
}

/* "Genera" lo slave corrispond. alla parte intera della sol. corrente */
void CutCBI::generateSlave()
{
   timers->start(Timers::slave_generation);
   for (int i = 0; i < num_vars; ++i)
   {  if (sol[i] == old_sol[i])
      {  continue;   //non serve fare niente
      }
      int idx = getActiveIntVarIndex(vars[i]);
      if (idx == -1)
      {  continue;   //non e' una var di controllo
      }

      // Se si arriva qui la var di controllo ha cambiato valore: si rimuove il vecchio
      if (old_sol[i] == 1 && args->active_value != 0)
      {  removeCondConstr(1, idx);
      }
      else if (old_sol[i] == 0 && args->active_value != 1)
      {  removeCondConstr(0, idx);
```

```
      }
      // Aggiunge il nuovo
      if (sol[i] == 1 && args->active_value != 0)
      {  setCondConstr(1, idx);
      }
      else if (sol[i] == 0 && args->active_value != 1)
      {  setCondConstr(0, idx);
      }
   }  //for sulle variabili
   timers->stop(Timers::slave_generation);
}

/* Genera un taglio completo, sulla base della parte intera della sol corrente
Per completo intendo non minimizzato, quindi con una variab contemplata
nel taglio per ogni valore intero nella sol corrente.
Ricordarsi di deallocare il taglio ricevuto, se non piu' usato */
IloRange CutCBI::getFullCut()
{  IloExpr expr(env);
   int count = -1;    //tiene gia' conto del -1 nel rhs
   for (int i = 0; i < num_vars; i++)
   {  if (sol[i] == 1)
      {  expr += vars[i];
         count++;
      }
      else if (sol[i] == 0)
      {  expr -= vars[i];
      }
   }
   IloRange taglio(env, -IloInfinity, expr, count);
   expr.end();
   return taglio;
}

IloRangeArray CutCBI::makeCuts()
{
   timers->start(Timers::MIS_search);
   IndexArray idx_a(env);
   getMISSet(idx_a);
   IloRangeArray ret(env);
   added_cuts = idx_a.getSize();
   if (added_cuts > 0)
   {  for (int i = 0; i < added_cuts; ++i)
      {  ostrstream oss;
         oss << "cut" << stats->added_cuts + i << ends;
         IloNumArray idx = idx_a[i];
         int rhs = -1;
         int size = idx.getSize();
         if (args->global_cuts)
         {  IloExpr expr(env);
            for (int j = 0; j < size; ++j)
            {  if (sol[(int)idx[j]] == 1)
               {  ++rhs;
                  expr += vars[(int)idx[j]];
               }
```

```
               else
               {  expr -= vars[(int)idx[j]];
               }
            }
            IloRange taglio = IloRange(env, -IloInfinity, expr, rhs, oss.str());
            expr.end();
            writeCut(output_logs->cb_cuts, add(taglio));
            taglio.end();
            idx.end();
         }
         else
         {  int * coef = new int[size];
            int * index = new int[size];
            for (int j = 0; j < size; ++j)
            {  if (sol[(int)idx[j]] == 1)
               {  ++rhs;
                  coef[j] = 1;
               }
               else
               {  coef[j] = -1;
               }
               index[j] = (int)idx[j];
            }
            SimpleCut< int > * cut = new SimpleCut< int >(size, index, coef,
                  SimpleCut< int >::lessEq, rhs);
            IloRange taglio = pool->addCut(cut, Pool::combBenders);
            ret.add(taglio);
            writeCut(output_logs->cb_cuts, addLocal(taglio));
         }
      }
      output_logs->cb_cuts << "\n";
      output_logs->solver << " addedCuts:" << added_cuts;
   }
   else
   {  // Tagli anomali aggiunti sempre come globali (sottratti al pool)
      IloRange taglio = getFullCut();
      taglio.setName("\\ Taglio Anomalo (MIS fault): ");
      writeCut(output_logs->warnings, add(taglio));
      taglio.end();
      output_logs->cb_cuts << "\n";
      output_logs->solver << " addedCuts:1(anomalo)";
   }
   idx_a.end();
   timers->stop(Timers::MIS_search);
   return ret;
}

/* Trova la soluzione del problema originale corrispondente ad una
soluzione intera (e feasible) del master; aggiorna, se e' migliore, il valore
della f.o. e genera un taglio per la soluzione trovata, se necessario */
IloNumArray CutCBI::getSlaveSol_And_Update()
{
   prob->alg.extract(prob->mod);
   prob->alg.setVectors(sol, NULL, vars, NULL, NULL, NULL);
```

```
   prob->alg.setParam(IloCplex::IntSolLim, 1);
   prob->alg.setParam(IloCplex::MIPStart, 1);
   prob->alg.setOut(env.getNullStream());
   prob->alg.solve();

   IloNum obj_value = prob->alg.getObjValue();
   IloNumArray sol_cont(env);
   prob->alg.getValues(sol_cont, prob->cont_vars);

   // Controlla se la sol trovata e' migliore dell'ultima
   // (si usa la tolleranza eps_feas: e' come verificare una disuguaglianza)
   if (  (prob->obj_func->sense == IloObjective::Minimize
            && obj_value < prob->obj_func->obj_value - eps_feas)
         || (prob->obj_func->sense == IloObjective::Maximize
            && obj_value > prob->obj_func->obj_value + eps_feas)  )
    {      stats->obj = prob->obj_func->obj_value = obj_value;
      output_logs->solver << "\n\nSoluzione intera (migliore) trovata; f.o. = "
            << obj_value << "\n\n";
      if (prob->obj_func->obj_type == ObjFunc::fo_cont)
      { prob->obj_func->updateObjValue(obj_value);
      }
   }
   return sol_cont;
}

/* Algoritmo euristico (di tipo greedy) per cercare una sol pi\'u intera di
quella frazionaria corrente alla quale corrisponda uno slave infeasible
(in modo da trovare ulteriori tagli da aggiungere al master).
Il taglio generato deve comunque avere la proprieta' di "tagliare" la sol fraz
corrente. Intuitivamente si puo' dire che non fa altro che prendere le var piu'
vicine a zero o ad uno ed arrotondarle finch\'e pu\'o.
Restituisce true se ha modificato lo slave (aggiungendo vincoli) e quindi e'
possibile ricercare nuove MIS, false se invece non serve ricercarle. */
bool CutCBI::greedyHeur()
{  /* omissis */
}

/* Genera un insieme di MIS (di cardinalita' >= 1)
   Restituisce l'insieme di MIS trovate tramite il suo parametro che deve
   essere inizializzato con env ma vuoto */
void CutCBI::getMISSet( IndexArray idx_a )
{
   IloNumArray tolti(env);  //indici in prob->cond_constr dei vincoli tolti
   IloNumArray var_idx(env);  //indici delle var int cui corrisp. tali vincoli

   bool cont = true;
   while (cont && idx_a.getSize() <= 300)
   {
      IloNumArray idx(env);
      IloNumArray MIS;
      MIS = dualMISSearch(idx);
      MIS.end();

      if (idx.getSize() == 0)
```

```
      {  output_logs->warnings << "\nMIS not found";
         idx.end();
         cont = false;
      }
      else
      {
         idx_a.add(idx);
         int k = (int)idx[0];
         int cc_idx = getActiveIntVarIndex(vars[k]);
         if (sol[k] == 1)
         {  removeCondConstr(1, cc_idx);
            tolti.add(cc_idx);
            var_idx.add(k);
         }
         else
         {  removeCondConstr(0, cc_idx);
            tolti.add(cc_idx);
            var_idx.add(k);
            }
         }
         timers->stopStart(Timers::MIS_search, Timers::slave_solve);
         d_a.solve();
         if (d_a.getCplexStatus() != IloCplex::Optimal)  cont = false;

         timers->stopStart(Timers::slave_solve, Timers::MIS_search);
         if (!cont) break;
      }
      int size = tolti.getSize();
      for (int i = 0; i < size; ++i)
      {  int tmp = (int)tolti[i];
         if (sol[(int)var_idx[i]] == 1) { setCondConstr(1, tmp);     }
         else {  setCondConstr(0, tmp);  }
      }
      tolti.end();
      var_idx.end();
}

/* Cerca di individuare un singolo MIS */
IloNumArray CutCBI::dualMISSearch( IloNumArray idx )
{
   IloNumArray MIS(env);   //da ritornare
   MIS.add(num_vars, 0);

   IloNumArray u_values(env);
   d_a.getValues(u_values, prob->dual_act_vars);

   int size = u_values.getSize();
   for (int i = 0; i < size; ++i)
   {  if (u_values[i] > eps_feas)
      {  int indice = getAssocIntVarIndex(prob->dual_act_vars[i]);
         indice = getIntVarIndex(prob->master_act_vars[indice]);
         if (MIS[indice] == 0)
         {  MIS[indice] = 1;
            idx.add(indice);
```

```
            }
         }
      }

      if (idx.getSize() == 0)
      {  output_logs->warnings<<"\n\nUnable to find MIS from DUAL: "<<u_values<<"\n\n";
      }
      u_values.end();
      return MIS;
}

void CutCBI::mainCBC()
{
   /* omissis: operazioni di logging */

   // Verifica presenza di tagli nel pool
   if (poolCuts(Pool::combBenders)) {  pool_hit = true;    return;   }
   pool_hit = false;

   timers->start(Timers::slave_solve);
   d_a.solve();
   IloCplex::Status st = d_a.getCplexStatus();
   if (st == IloCplex::Optimal)
   {
      // Dual Slave feasible, calcola taglio da aggiungere al master
      timers->stop(Timers::slave_solve);
      IloRangeArray added = makeCuts();
      // if (added.getSize() != 0) addRangeArray(added); //0-1/2 cuts
      if (args->global_cuts) added.endElements();
      else    added.end();
      output_logs->solver << "\n";
   }
   else
   {
      if (st == IloCplex::Infeasible || st == IloCplex::InfOrUnbd)
      {  // Dual Slave infeasible
         timers->stop(Timers::slave_solve);
         if (sol_int)
         {  //trovata una possibile soluzione
            stats->int_sol++;
            IloNumArray sol_cont = getSlaveSol_And_Update();
            output_logs->solutions.seekp(0);
            writeSolution(output_logs->solutions, sol, sol_cont);
            output_logs->solver << " sol_int(obj:" << getObjValue() << ")\n";

            if (prob->obj_func->obj_type != ObjFunc::fo_int)
            {  IloRange taglio = getFullCut();
               ostrstream oss;
               oss << "cutSolution" << stats->int_sol << ends;
               taglio.setName(oss.str());
               add(taglio);   //aggiunto sempre come globale
               output_logs->warnings << "\\ Taglio per soluz. intera feasible;";
               writeCut(output_logs->cb_cuts, taglio);
               taglio.end();
```

```
            }
            sol_cont.end();
         }
         else
         {  /* slave feasible e sol fraz:
            qui si pu\'o metter il codice per il "separatore frazionario" */
             // if (greedyHeur())
               { /* ... */ }
         }  //else [ if (sol_int) ]
      }     // else if (d_a.getCplexStatus() == IloCplex::Infeasible) ]
      else
      {  // Problemi numerici
         output_logs->errors << "\n\nSlave duale ne' ottimo, ne' impossibile";
         d_a.exportModel("strange.sav");
         cout << "Error on dual slave: cplex status = " << d_a.getCplexStatus();
         exit(-1);
      }
   }
   stats->added_cuts += added_cuts;
   dnp->node_cons += added_cuts;
   output_logs->flushAll();
}

void CutCBI::main()
{
   timers->stop(Timers::solve);
   stats->cb_calls++;

   if (getNnodes())  {  dnp = static_cast< DatiNodo * >(getNodeData()); }
   else    {   dnp = prob->dnrp; //siamo al nodo radice  }

   added_cuts = 0;
   last_node = this_node;
   this_node = dnp->node_id;
   if (this_node != last_node)
   {  last_depth = depth;
      depth = dnp->depth;
   }

   // Calcola la soluzione master del nodo corrente
   sol = IloNumArray(env);
   getValues(sol, vars);
   sol_int = isIntegerFeasible();
   generateSlave();     // mantiene aggiornato lo slave
   if (sol_int)    { stats->master_int_sol++;   }
   else    {   stats->master_frac_sol++;  }

   if (depth <= 10 || sol_int || depth < last_depth)
   {  mainCBC();
   }
   if ((depth <= 10 && depth < last_depth) && added_cuts == 0 && !sol_int && !pool_hit)
   {      // main012();      //funzione main per i tagli 0-1/2
   }
```

```
    old_sol.end();
    old_sol = sol;
    timers->start(Timers::solve);
}
```

Il metodo piú semplice di analizzare il codice riportato é quello di seguire il flusso di programma che viene eseguito ogni volta che la callback viene invocata dal risolutore del problema master.

Inizialmente viene sempre eseguito il metodo main. La sua struttura é molto semplice: per prima cosa si preoccupa di aggiornare il valore di alcune variabili di stato dell'istanza (added_cuts, last_depth, depth, this_node, last_node che contengono rispettivamente il numero di tagli aggiunti dalla chiamata a callback in corso, la profonditá nell'albero dell'ultimo nodo analizzato e del nodo corrente, l'identificatore del nodo corrente e del nodo su cui é stata fatta la precedente chiamata a callback[4]); quindi, tarmite le funzioni getValue, isIntegerFeasible e generateSlave recupera la soluzione del nodo corrente e genera lo slave corrispondente; poi valuta, tarmite le variabili di stato, se le condizioni necessarie per effettuare la generazione dei tagli combinatori di Benders e/o 0–1/2 sono verificate ed in tal caso utilizza le funzioni mainCBC e main012 per adempiere al compito; infine sistema opportunamente le variabili sol e old_sol affinché contengano il valore corretto per la successiva chiamata.

Le funzioni isIntegerFeasible e generateSlave sono semplici ed auto–esplicative: lo scopo della prima é determinare se la soluzione del nodo master corrente é intera o meno e contestualmente arrotondare quei valori che dovrebbero essere interi ma non lo sono a causa dell'aritmetica a precisione finita del calcolatore. La seconda genera il problema slave sulla base dell'aggiornamento differenziale dei vincoli descritto in precedenza e per far questo si avvale anche dell'ausilio delle due funzioni membro setCondConstr e removeCondConstr.

Prima di descrivere mainCBC che é la funzione che contiene il cuore dell'implementazione dell'algoritmo di generazione dei tagli, é necessario un chiarimento sul perché questa funzione non viene invocata ad ogni nodo dell'albero di branching ma solo se alcune condizioni sono verificate. Il motivo deriva da alcune evidenze sperimentali raccolte durante la fase di sviluppo/debugging e sulla base di alcuni test specifici che si sono eseguiti. Ció che si é notato é che mentre nei nodi molto vicini al nodo radice vengono individuati molti MIS e quindi generati molti tagli, nella maggior parte dei nodi piú profondi non si riesce a generare quasi nessun taglio (a meno di non essere in presenza di soluzioni del tutto intere, o quasi). Per questo eseguendo la funzione mainCBC su questi nodi non si faceva altro che risolvere dei problemi slave senza peró trarne alcuna informazione utile. Per

[4]Si ricorda che in generale vengono eseguite piú chiamate per ogni nodo e quindi last_depth non si riferisce necessariamente all'ultima chiamata fatta e this_node e last_node possono avere lo stesso valore

questo si é deciso di imporre tre condizioni, almeno una delle quali deve essere soddisfatta affinché mainCBC venga invocata; come si vede dal codice, esse sono:

- la profonditá del nodo corrente sia minore o uguale a dieci
- la soluzione sia intera (questo é indispensabile per evitare di avere soluzioni non valide)
- ci sia stato un backtrace (lo spirito secondo cui opera questa condizione é garantire che almeno il nodo da cui parte la generazione di un intero sotto–albero a seguito di un backtrace non contenga variabili con valori incompatibili tra loro)

Si é verificato che questo tipo di approccio comporta un notevole speed–up nella generazione dei nodi: sebbene infatti in questo modo si ritardi la generazione di alcuni tagli che potrebbero giá essere individuati nei nodi interni dell'albero a profonditá maggiore di 10, il numero di nodi che si riesce a generare é molto piú grande e di conseguenza la porzione di albero visitata aumenta grazie al recupero del tempo non piú impiegato per inutili risoluzioni di problemi slave; questo secondo effetto é decisamente prepronderante e compensa il primo. Si fa inoltre notare che questo modo di procedere é certamente lecito finché si garantisce che almeno in presenza di soluzioni intere si producano i tagli combinatori di Benders: ció si deduce immediatamete dal fatto che, come spiegato nella sezione teorica, le chiamate alla procedura di separazione su nodi con soluzione frazionaria non sono strettamente indispensabili alla convergenza dell'algoritmo e quindi é possibile farle o meno a propria discrezione.

Detto questo é ora possibile osservare la struttura di mainCBC: questa funzione per prima cosa controlla che non vi siano tagli violati nel pool dalla soluzione master corrente. Ció é fatto tramite la funzione ausiliaria poolCuts, la quale ha il compito di interrogare il pool relativamente alla presenza o meno di tagli violati secondo un'opportuna interfaccia ed eventualmente di aggiungere tali tagli al master in modo locale o globale. Se vengono trovati tagli violati nel pool mainCBC restituisce direttamente il controllo alla funzione main: in questo modo si forza l'algoritmo di branching a riottimizzare il nodo corrente cercando una soluzione che non violi piú i tagli del pool. Questo evita che il problema slave venga utilizzato per cercare dei MIS e generare dei tagli che sono giá stati generati in precedenza e quindi sono giá presenti nel pool, riducendo il tempo di calcolo impiegato. Se invece non vi sono tagli violati nel pool allora la funzione mainCBC risolve il problema slave duale modificato: se questo risulta feasible allora significa che esiste almeno un MIS e quindi viene invocata la funzione makeCuts che si occupa di individuare quanti piú MIS possibili, generare i tagli relativi ed aggiungerli al problema master; se invece risultasse infeasible significa che non esistono MIS. In tal caso se la soluzione del nodo corrente é intera, allora siamo in presenza di una nuova soluzione per l'intero problema originale, altrimenti non é necessario intraprendere nessuna particolare azione. Tuttavia, come indicato da un commento–segnaposto nel codice qui si potrebbe inserire quello che si puó chiamare un sistema euristico di generazione di tagli, ovvero un sistema che cerchi di individuare una disequazione che "tagli" la soluzione corrente basandosi anche (o solo) sulla parte frazionaria della soluzione corrente (la parte intera é giá tuttora sfruttata). Un tale sistema basato su un algoritmo euristico di arrotondamento é anche stato sviluppato nel contesto di questa tesi (funzione greedyHeur) ma si é dimostrato inefficace (raramente si riuscivano ad individuare nuovi tagli ed il tempo utilizzato per i calcoli risultava sproporzionato rispetto al guadagno): per questo non viene qui riportato. Per quel che rigurda le azioni intraprese quando si é in presenza di una soluzione intera del problema originale, esse consistono nell'utilizzare la funzione getSlaveSol_And_Update (che ha il compito di calcolare la soluzione utilizzando il problema originale ed effettuare gli opportuni aggiornamenti del problema slave, se necessari), salvare la soluzione trovata ed infine generare (tramite la funzione getFullCut) il taglio che permette il procedere dell'algoritmo per i problemi con funzione obiettivo contenete anche o solo variabili reali.

Restano ora da analizzare solamente la funzione makeCuts e le sue due funzioni di supporto getMISSet e dualMISSearch. Eccone il dettaglio:

dualMISSearch Ha semplice compito di individuare quali vincoli fanno parte del MIS e di generare il taglio relativo. Per individuare i vincoli é sufficiente individuare quali variabili duali assumono un valore diverso da zero (in realtá é utilizzata una piccola tolleranza numerica – eps_feas – per assorbire le imprecisioni di calcolo); per generare il taglio si utilizzano le informazioni associate ad ogni variabile nell'istanza corrispondente di VarData ed il taglio stesso, sottoforma di array di indici delle variabili coinvolte é restituito nel parametro idx della funzione. L'unica accortezza che merita una sottolineatura é l'utilizzo di un array di supporto (la variabile MIS) per tener traccia di quali variabili siano giá state inserite nel taglio: ció permette di evitare che se una variabile di controllo attiva molte disequazioni e piú di una di queste entra a far parte di un MIS, la variabile venga conteggiata piú di una volta nel taglio generato (il che sarebbe scorretto).

getMISSet Ha il compito di cercare quanti piú MIS possibili in un certo problema slave (anche se in realtá si é imposto un limite superiore di 300 MIS per evitare che il programma rimanga troppo a lungo su uno stesso nodo). Il metodo che si é implementato per la ricerca di piú MIS diverse all'interno dello stesso sistema é molto semplice ma efficace: una volta individuato il primo MIS non si fa altro che "togliere" (che nel caso in questione significa fissare a zero la variabile duale corrispondente) dal problema slave una delle disequazioni che ne fanno parte. In questo modo, proprio per la sua definizione, i restanti vincoli del MIS individuato non possono piú causare l'infeasibility del sistema e perció, se questo rimane infeasible deve esserci un altro MIS che puó essere scoperto sempre con il solito approccio: a questo punto si itera il procedimento. Agendo cosí si possono individuare quasi tutti i MIS del sistema: restano esclusi solo quelli che condividono una delle disequazioni che vengono eliminate. La funzione getMISSet implementa questo approccio tramite il ciclo while e servendosi di dualMISSearch per la generazione del singolo taglio. All'uscita del ciclo while ripristina tutti i vincoli che ha dovuto togliere per individuare i MIS.

makeCuts Ha il compito di aggiungere al master l'insieme di tagli restituito dalla funzione getMISSet. Per far ció occorre che i tagli siano convertiti in un appropriato formato; in particolare se i tagli vanno aggiunti come globali essi devono essere convertiti in un IloRange per poter essere fornito come parametro alla funzione add; se invece vanno aggiunti come tagli locali allora devono anche essere convertiti nel formato utilizzato dal pool, che come detto é diverso da un IloRange per ragioni di efficienza. Inoltre questa funzione ha anche un compito di "ancora di salvezza": nel caso in cui getMISSet non riesca ad individuare nessun taglio (cosa che non dovrebbe teoricamente accadere) a causa per esempio di problemi di stabilitá numerica allora il primo if della funzione trasferirebbe il controllo ad una sezione del codice che inserirebbe un taglio completo (tramite una chiamata a getFullCut); questo taglio completo (ovvero che coinvolge tutte le variabili senza alcuna forma di minimizzazione e che nel codice sorgente é indicato come taglio anomalo) servirebbe ad evitare situazioni quali la terminazione prematura dell'ottimizzazione o l'individuazione di soluzioni errate che si verificherebbero altrimenti.

Capitolo 5

Risultati Computazionali

Per provare l'efficacia del programma che si é implementato sono stati fatti dei test su una gamma di problemi. In questo capitolo verranno descritti il test–bed utilizzato ed i risultati ottenuti. Verranno inoltre indicate delle possibili varianti (di cui una giá incorporata nel codice [1] ed altre in fase di sperimentazione) atte a migliorare la procedura di risoluzione per certi tipi di problemi.

5.1 Test–bed utilizzato

L'insieme di istanze su cui si sono eseguite le prove appartengono a quattro tipologie diverse di problemi. Eccone una breve descrizione.

Map Labelling Questo tipo di problemi sono giá stati ampiamente descritti in 2.1.3. Sono state utilizzate 18 istanze, alcune delle quali di tipo 4–slider (indicate in seguito con la sigla "4S"), altre di tipo 4–position (indicate con "4P"). Il motivo principale per cui si é scelto di includere questo tipo di problemi é quello di poter avere un termine di paragone costituito dal software specificatamente sviluppato per risolvere questo tipo di problemi (descritto sempre in [6] ed in [7]), di modo da poter individuare eventuali pecche e colli di bottiglia presenti nell'algoritmo dei tagli combinatori di Benders e/o nella implementazione qui fornita.

Two Group Statistical Classification (discriminant analysis) Una spiegazione esauriente di questo tipo di problemi puó essere trovata in [8]; ci riferiremo ad essi per brevitá con il nome di problemi di analisi statistica. Essi vengono infatti generalmente utilizzati in tale settore di ricerca e si possono brevemente descrivere in questo modo: sia data una popolazione di individui (non necessariamente persone) suddivisa in due classi ben distinte

[1] Si tratta di quanto giá anticipato relativamente alla disequazione 4.1

(ad es. coloro che hanno contratto una certa malattia e le persone sane); siano inoltre misurabili una serie di caratteristiche che hanno (o si presume che abbiano) una certa correlazione con la suddivisione in classi appena fatta (ad. es alcuni parametri biofisici in grado di evidenziare la presenza della malattia in questione). Sulla base delle misurazioni di questi parametri fatti su un campione della popolazione di cui sia nota l'appartenenza ad una o all'altra classe si vuole individuare una funzione lineare tale che, inserendovi i parametri relativi ad un nuovo individuo, questa fornisca un valore positivo se tale individuo appartiene ad una classe, negativo se appartiene all'altra. In generale non é detto che questa funzione esista (a causa dell'imposizione che obbliga la funzione ad essere lineare); perció ci si accontenta di individuare la funzione lineare che minimizza il numero di classificazioni errate all'interno del campione noto (per classificazioni errate si intende una situazione in cui la funzione indica che un individuo appartiene ad una classe mentre é nota la sua appartenenza all'altra). Questo tipo di problema puó essere modellato tarmite un sistema lineare MIP che utilizza la tecnica del big–M: in pratica ad ogni individuo del campione viene associata una certa disequazione: questa disequazione va disattivata se per quell'individuo avviene una errata classificazione; lo scopo é quindi minimizzare il numero di disequazioni disattivate; da ció si capisce che nella funzione obiettivo di questo tipo di problemi sono presenti le sole variabili binarie di controllo.[2] Le variabili continue conterranno invece i coefficienti della funzione lineare cercata.

Sono state utilizzate 38 istanze di questo tipo, ottenute da molte diverse situazioni reali (classificazione di persone con malattie, classificazoine di diversi fenomeni fisici, classificazione di comportamenti animali, etc.).

I dati per generare i problemi sono stati tratti da un'archivio liberamente utilizzabile mantenuto presso l'UCI[3]. Questi dati grezzi sono stati poi convertiti nel modello MIP tramite una versione lievemente modificata di un programma fornito da P. A. Rubin[4]. Non tutti i problemi ottenibili in questa maniera partendo dai dati originali sono stati utilizzati: molti infatti erano troppo semplici da risolvere o avevano dimensioni esageratamente grandi per poter essere gestiti. Infatti i dati raccolti in tale database non

[2] In realtá nell'articolo [8] si descrive un modello MIP contenente anche una variabile continua nella funzione obiettivo: qul tipo di modello puó essere trasformato facilmente in quello qui descritto fissando tale variabile al suo lower–bound. Se si desiderasse comunque risolvere il modello descritto nell'articolo si puó ugualmente utilizzare la parte intera della soluzione trovata dal modello semplificato senza variabile continua: infatti all'ottimo tale parte non cambia tra i due modelli perció, una volta individuata, sará sufficiente sostituirla nel modello originale, risolvendo a questo punto un semplice problema LP

[3] P. M. Murphy, D. W. Aha (1994), *UCI Repository of machine learning databases* [http://www.ics.uci.edu/ mlearn/MLRepository.html]. Irvine, CA: University of California, Department of Information and Computer Science.

[4] Department of Management — The Ili Broad Graduate School of Management, Michigan State University

sono specificatamente pensati per un utilizzo simile a quello fatto in questa tesi. Per questo motivo molti dei problemi che sono stati qui utilizzati si sono ottenuti a seguito di modifiche effettuate sui dati contenuti nel database; in particolare: riduzione della dimensione del campione (cancellazione di righe), riduzione del numero di attributi misurati (numero di colonne), accorpamento di diverse classi (per ridurne il numero a due).

Broken Glass Questa classe di problemi rientra nel piú vasto insieme dei problemi di nesting (impaccamento). Dati una serie di oggetti bidimensionali dal contorno poligonale, si vuole posizionare tali oggetti (senza peró avere la possibilitá di ruotarli sul piano) entro uno spazio rettangolare di altezza prefissata e di larghezza minima. Il loro nome deriva dall'idea che questa situazione si verifica nel caso si rompa una lastra di vetro rettangolare di altezza nota (in frammenti poligonali) e si voglia successivamente ricomporne i cocci. Si avevano a disposizione 6 istanze di questo tipo. riferimento a cosa?

Questo tipo di problemi hanno la funzione obiettivo che é composta da un'unica variabile reale che indica la larghezza del rettangolo ed utilizzano la tecnica del big–M per attivare o disattivare alcuni vincoli i quali hanno lo scopo di limitare le possibilitá di posizionamento reciproco delle coppie di poligoni.

riferimenti?

Railway Time–tabling Questa classe di problemi rientra nel dominio dei problemi di scheduling ciclico (noti con la sigla PESP). Il loro scopo é quello di individuare gli orari di partenza e di arrivo dei treni in un contesto in cui l'orario ferroviario si ripete identico ad ogni ora del giorno ed in modo che alcuni vincoli vengano soddisfatti (ad es. le coincidenze tra treni o tempi di attesa minimi/massimi tra due treni correlati).

Si é utilizzata questa classe di problemi in quanto presenta caratteristiche diverse dalle precedenti: in questo caso infatti non si utilizza la tecnica del big–M ma la variabile binaria presente in alcuni vincoli serve a "trasformarli" in due versioni diverse (ovvero con un diverso termine noto che serve proprio a medellare la ciclicitá dell'asse dei tempi utilizzato). Inoltre la funzione obiettivo di questi problemi é mista, ovvero comprende sia variabili binarie che continue.

Si é in possesso di 6 istanze di questo tipo. Tuttavia alcune di esse presentavano delle variabili intere in alcuni vincoli misti che potevano assumere tre diversi valori $(0, 1, 2)$: per questo é stato necessario operare una modifica che ha comportato la sostituzione di tali variabili ternarie con una coppia di variabili binarie affinché tali problemi rientrassero nella classe trattabile con i tagli combinatori di Benders.

5.2 Risultati: prima parte

Per tutti i tipi di problemi sono stati condotti due tipi di test: prima si é tentato di risolverli usando il programma nella modalitá "cplex_only" (si veda la descrizione del framework: in questa modatá il programma non utilizza alcuna forma di separatore ma si limita a risolvere il problema passandolo a Cplex con le impostazioni di default di quest'ultimo); successivamente si é passati alla risoluzione tramite il programma nella sua modalitá naturale (ovvero tramite l'uso dei tagli combinatori di Benders e con le impostazioni di default, a parte l'utilizzo del parametro active_value). Quindi si sono confrontati i risultati ottenuti nei due casi in termini di tempo impiegato e/o di gap tra migliore soluzione intera trovata e best bound. Nella gran parte dei problemi si é imposto un tempo massimo di tre ore, anche se in qualche caso si sono fatte prove di lunghezza diversa. Inoltre si é fatto in modo che il programma terminasse non appena esaurita la memoria RAM fisicamente disponibile cercando di evitare le operazioni di swapping su disco fisso (questo perché nell'istante in cui ció avveniva i tempi di swap erano decisamente preponderanti rispetto al tempo di calcolo e impedivano progressi consistenti nella risoluzione dei problemi): approssimativamente si puó pensare che il limite di occupazione di memoria sia circa pari a 900 MByte. Per eseguire i test si é fatto uso di un PC dalle seguenti caratteristiche: CPU = Athlon 2100+, RAM = 1GByte, OS = Linux (kernel 2.4).

I risultati ottenuti sono suddivisibili in due categorie:

1. eccellenti per i problemi di Map Labelling e di Analisi Statistica

2. non buoni per i problemi di Broken Glass e di Railway Time–tabling.

Il motivo di questa differenza cosí marcata ed alcune possibilitá di ridurla saranno esposte nella prossima sezione. In questa ci occuperemo dei soli problemi su cui il metodo esposto in questa tesi ha dato i risultati piú brillanti. Esporremo quindi i risultati relativi ai 56 problemi appartenenti alle prime due classi $(38 + 18)$. Prima di entrare nel dettaglio si vuole anticipare che i risultati ottenuti sulle istanze di map labelling, seppur lievemente inferiori, sono del tutto confrontabili con il software specificatamente pensato per la loro risoluzione. Per comoditá d'ora in poi ci riferiremo al programma di risoluzione in modalitá cplex_only con il nome di Cplex ed al programma in modalitá "Combinatorial Benders' Cuts" con il nome CBC. Alla fine dei test condotti si possono raggruppare i problemi in tre insiemi:

1. problemi risolti sia da Cplex che da CBC

2. problemi risolti da CBC ma non da CPlex

3. problemi non risolti da entrambi.

Gruppo 1		**Cplex**	**CBC**	**Statistiche**	
Categoria e Nome file	**f.o.**	**Time h : m : s**	**Time h : m : s**	**t.CX/ t.CB**	**t.CB/ t.CX**
Chorales					
Chorales-116	24	1 : 24 : 52	0 : 10 : 18	8.2	12.1%
Balloons		: :	: :		
Balloons76	10	0 : 0 : 10	0 : 0 : 0.14	71.4	1.4%
Breast Cancer (1)		: :	: :		
BCW-367	8	0 : 8 : 33	0 : 0 : 13	39.4	2.5%
BCW-683	10	2 : 2 : 29	0 : 0 : 32	229.7	0.4%
WPBC-194	5	0 : 57 : 17	0 : 3 : 32	16.2	6.2%
Breast Cancer (2)		: :	: :		
Breast-Cancer-400	0.06	0 : 2 : 50	0 : 0 : 0.16	1062	0.1%
Glass		: :	: :		
Glass-163	13	0 : 56 : 17	0 : 0 : 5	675.4	0.1%
Horse colic		: :	: :		
Horse-colic-151	5	0 : 4 : 50	0 : 0 : 23	12.6	7.9%
Iris Plants		: :	: :		
Iris-150	18	0 : 9 : 29	0 : 1 : 10	8.1	12.3%
Credit Screening		: :	: :		
Credit-300	8	0 : 19 : 35	0 : 0 : 2	587.5	0.2%
Lymphography		: :	: :		
Lymphography-142	5	0 : 0 : 11	0 : 0 : 1	11.0	9.1%
Mech. Analysis		: :	: :		
Mech-analysis-107	7	0 : 0 : 5	0 : 0 : 1	5.0	20.0%
Mech-analysis-137	18	0 : 7 : 44	0 : 0 : 27	17.2	5.8%
Monk's Probl.		: :	: :		
Monks-tr-122	13	0 : 2 : 5	0 : 0 : 5	25.0	4.0%
Page blocks		: :	: :		
Pb-gr-txt-198	11	0 : 4 : 21	0 : 0 : 5	52.2	1.9%
Pb-pict-txt-277	7	0 : 2 : 7	0 : 0 : 2	63.5	1.6%
Pb-hl-pict-444	10	0 : 4 : 17	0 : 0 : 27	9.5	10.5%
Post Operative		: :	: :		
Postoperative-88	16	0 : 15 : 16	0 : 0 : 1	916.0	0.1%
Vehicle Silhouttes		: :	: :		
BV-OS-282	6	0 : 5 : 13	0 : 0 : 24	13.0	7.7%
Opel-Saab-80	6	0 : 1 : 3	0 : 0 : 13	4.8	20.6%
Bus-Van-437	6	0 : 9 : 17	0 : 0 : 28	19.9	5.0%
Voting-records		: :	: :		
HouseVotes84-435	6	0 : 4 : 59	0 : 0 : 11	27.2	3.7%
Water Treat.		: :	: :		
Water-treat-206	4	0 : 1 : 10	0 : 0 : 6	11.7	8.6%
Water-treat-213	4	0 : 17 : 0	0 : 0 : 51	20.0	5.0%
Map Lab. (4S)		: :	: :		
CMS 600-1	600	1 : 8 : 41	0 : 4 : 34	15.0	6.6%
TOTALI e MEDIE		**8 : 29 : 51**	**0 : 24 : 11**	**21.1**	**5%**

Tabella 5.1: Gruppo 1. Problemi risolti sia da Cplex che da CBC

É giá significativo il fatto che non siano stati individuati problemi che Cplex riesce a risolvere mentre CBC no.

Per quel che riguarada il primo gruppo di problemi, i principali tra i molti dati raccolti, sono stati inseriti nella tabella 5.1; questa riporta il valore ottimo della funzione obiettivo, il tempo impiegato per trovare la soluzione ottima dai due programmi, il rapporto tra il tempo impiegato da Cplex e quello impiegato da CBC ed il suo inverso. Tutti gli appartenenti a questo gruppo, tranne l'ultimo, sono problemi di Analisi Statistica.

Ció che spicca é il fatto che in tutti i casi CBC é riuscito a trovare la soluzione ottima molto piú rapidamente di Cplex, talvolta in modo clamoroso: infatti tutti i rapporti dei tempi nella penultima colonna della tabella 5.1 sono maggiori dell'unitá (e quindi indicano "quante volte CBC é stato piú veloce di Cplex") e per questo i loro inversi, nell'ultima colonna, sono stati espressi in percentuale (ed indicano quale frazione del tempo usato da Cplex é stata sufficiente a CBC per eseguire lo stesso compito). Nell'ultima riga sono stati riportati le somme dei tempi e le medie aritmetiche dei rapporti.

Osservando i valori di questa tabella si nota che mai CBC ha impiegato piú del 20.6% del tempo impiegato da Cplex e che in tutti i casi tranne due é rimasto sotto al 12.3% (il che significa che in tali casi é stato almeno otto volte piú veloce). In cinque di questi problemi CBC é stato addirittura centinaia o mgliaia di volte piú rapido di Cplex. I dati che rendono meglio l'idea sono quelli riassuntivi: in totale Cplex ha impiegato quasi 8 ore e mezzo per risolvere tutte le 25 istanze di questo primo gruppo mentre CBC solamente poco piú di 24 minuti: ció significa che quest'ultimo programma é risultato essere oltre 21 volte piú veloce del primo (impiegando meno del 5% del tempo del primo).

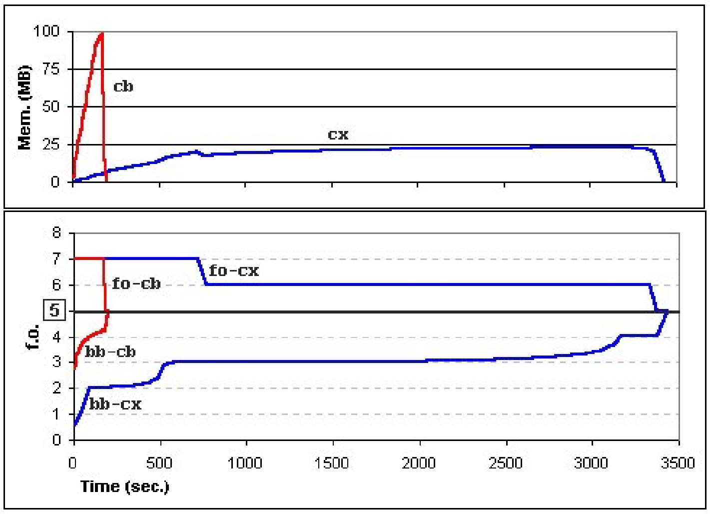


Figura 5.1: Grafici sugli andamenti delle ottimizzazioni, file WPBC-194

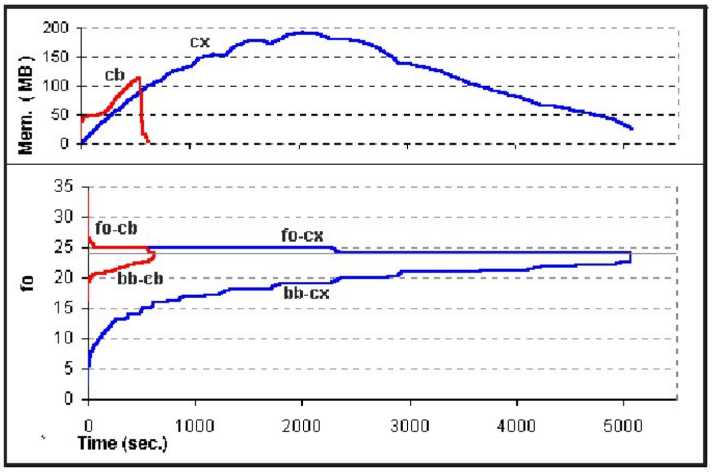


Figura 5.2: Grafici sugli andamenti delle ottimizzazioni, file Chorales-116

Per rendere piú immediato il confronto tra i due algoritmi sono stati prodotti alcuni grafici, riportati nelle figure da 5.1 a 5.3. In ognuna di tali figure sono rappresentati due grafici: il primo di ogni coppia rappresenta la quantitá di memoria utilizzata, il secondo rappresenta l'andamento del best bound ed i valori interi man mano trovati, tutto in funzione del tempo. Su ogni grafico si trovano sia le curve relative a Cplex (etichettate con la sigla "cx"), sia quelle relative a CBC (etichettate invece con "cb"). I quattro problemi che sono stati presi in considerazione sono Chorales-116, BCW-683, WPBC-194, Water-treat-213.

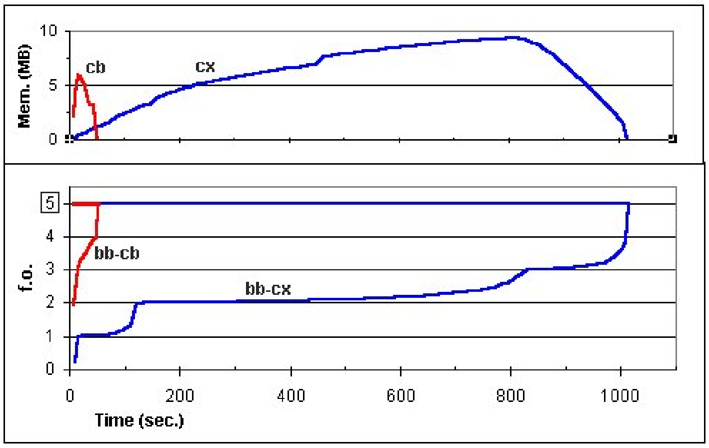


Figura 5.3: Grafici sugli andamenti delle ottimizzazioni, file Water-treat-213

Nel caso del problema BCW-683 si é reso necessario eseguire un ingrandimento della zona in cui si sviluppano le curve relative alla esecuzione di

CBC dato che l'enorme differenza dei tempi di risoluzione rende impossibile una chiara rappresentazione di questi dati sulla scala temporale dei dati di Cplex. Si informa che esistono delle leggere discrepanze tra i tempi della tabella 5.1 e quelli deducibili dai grafici stessi. Il motivo é dovuto al fatto che mentre nella tabella sono riportati i tempi globali impiegati dai programmi, i dati dei grafici si riferiscono solo i tempi di risoluzione veri e propri (enumerazione dei nodi e procedura di separazione): mancano quindi i tempi di setup e preprocessing iniziali (in particolare il tempo impiegato dal sistema euristico sul rilassamento continuo del nodo radice, come spiegato nel capito precedente), che comunque risultano quasi sempre ininfluenti.

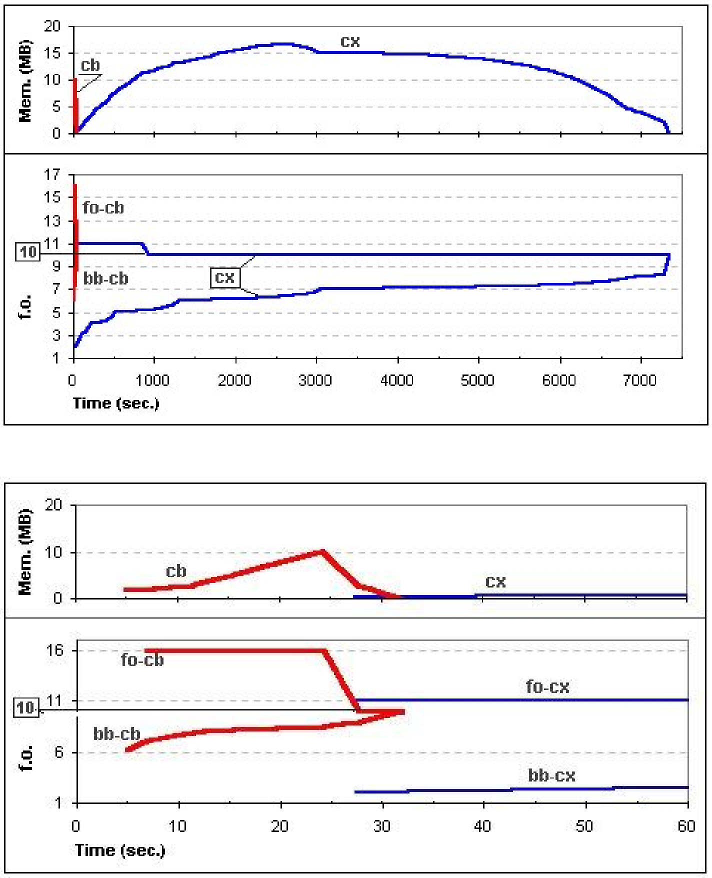


Figura 5.4: Grafici sugli andamenti delle ottimizzazioni, file BCW-683 (sopra). Ingrandimento della zona iniziale (sotto)

I dati relativi al secondo gruppo di problemi sono riassunti nelle due tabelle 5.2 (per quelli di Analisi Statistica) e 5.3 (per i Map Labelling). Per questo gruppo la differenza é per certi versi ancora piú marcata rispetto al primo: non solo infatti Cplex non risolve all'ottimo alcuno di questi problemi entro i limiti di tempo e di spazio occupato in memoria che sono stati imposti, ma in piú osservando i trend di variazione dei bound si deduce che molto probabilmente Cplex non risolverebbe nessuno di questi problemi nemmeno avendo a disposizione giorni/mesi interi di calcolo e maggiori quantitá di memoria.

Gruppo 2	**Cplex**					**CBC**	
Categoria e Nome file	**Time h:m:s**	**f.o.**	**bb**	**Gap %**	**Mem MB**	**Time** : :	**f.o.**
Chorales	: :					: :	
Chorales-134	0:36:23	33	16.2	51%	422	0:36:23	30
	2: 7 :41	30	20.0	33%	918	: :	
Chorales-107	0: 4 :19	28	12.1	57%	61	0: 4 :19	27
	3: 1 :27	27	22.2	18%	711	: :	
Breast Cancer (2)	: :					: :	
Breast-Cancer-600	0: 0 :13	108	1.5	99%	—	0: 0 :13	16
(vedere didascalia)	3: 0 :11	16	13.2	18%	45	: :	
Bridges	: :					: :	
Bridges-132	0: 3 :39	33	5.1	85%	40	0: 3 :39	23
	1:25:38	26	9.0	65%	876	: :	
Mech. Analysis	: :					: :	
Mech-analysis-152	0:34:12	22	12.1	45%	331	0:34:12	21
	2: 9 :26	22	15.1	31%	871	: :	
Monk's Probl.	: :					: :	
Monks-tr-124	0: 1 :55	27	8.1	70%	25	0: 1 :55	24
	3: 0 :35	24	20.0	17%	381	: :	
Monks-tr-115	0: 4 :16	29	9.0	69%	63	0: 4 :16	27
	1:53:40	27	18.0	33%	877	: :	
Solar Flare	: :					: :	
Solar-flare-323	0: 0 :4	51	4.1	92%	—	0: 0 :4	38
	2:30:28	43	16.1	63%	917	: :	
Vehicle Sil.	: :					: :	
BV-OS-376	0: 9 :4	9	3.1	65%	9	0: 9 :4	9
	3: 0 :10	9	6.0	33%	56	: :	
BusVan-445	0:10:31	13	3.0	77%	11	0:10:31	8
	3: 0 :6	9	5.1	43%	56	: :	
TOTALI e	: :	**Gap (t. =)**		**71%**		**1:44:36**	
MEDIE	**25: 9 :22**	**Gap (t.fin.)**		**36%**		: :	

Tabella 5.2: Gruppo 2. Problemi di Analisi Statistica risolti da CBC ma non da Cplex (I valori della f.o. del problema Breast-Cancer-600 sono stati moltiplicati per 1000)

Per rendere piú immediato il confronto nelle tabelle vengono riportati i gap ottenuti da Cplex al termine dell'esecuzione su ogni istanza ed anche allo stesso istante di tempo in cui CBC ha terminato la propria ottimizzazione. Nelle ultime due righe si riportano le medie di questi due valori. Si osserva che il programma CBC risolve tutte le istanze di Analisi Statistica in meno di un'ora e tre quarti mentre, a paritá di tempo, Cplex ottinene un gap

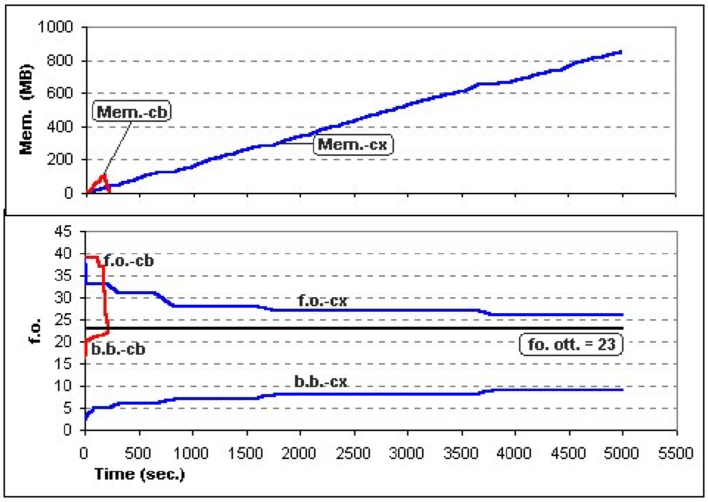


Figura 5.5: Grafici sugli andamenti delle ottimizzazioni, file Bridges-132

medio superiore al 70%. Inoltre dopo quasi un ulteriore intera giornata di calcolo il gap si é solamente dimezzato e metá dei problemi in questione hanno esaurito la memoria disponibile. Un comportamento del tutto simile si é riscontrato nei problemi di Map Labelling (qui addirittura il gap non si é nemmeno dimezzato dopo altre 35 ore di calcolo).

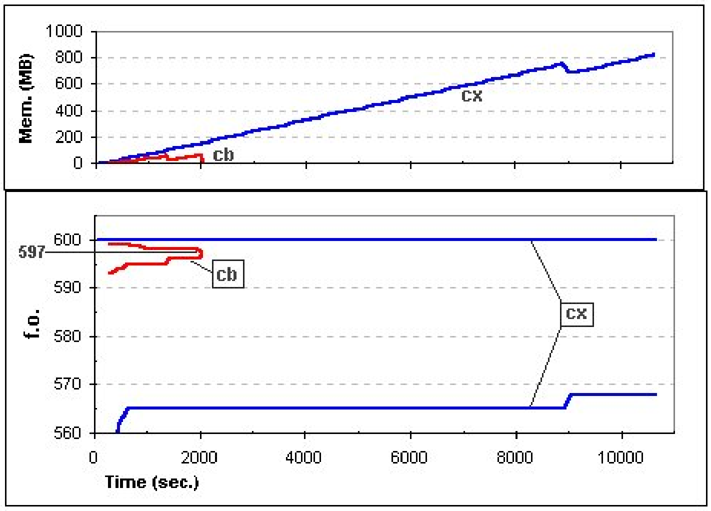


Figura 5.6: Grafici sugli andamenti delle ottimizzazioni, file CMS_600-1

Anche per questo secondo gruppo di problemi sono stati selezionate quattro istanze e sono stati prodotti dei grafici esemplificativi dello stesso tipo di quelli realizzati per il primo gruppo. I problemi presi in considerazione sono

Gruppo 2	**Cplex**					**CB**	
Categoria e Nome file	**Time h:m:s**	**f.o.**	**bb**	**Gap %**	**Mem MB**	**Time h:m:s**	**f.o.**
Map lab (4S)	: :					: :	
CMS 600-0	0:04:27	592	600	1.35%	18	0: 4 :27	600
	3:03:00	594	600	1.01%	770	: :	
CMS 650-0	0:06:26	638	650	1.88%	20	0: 6 :26	649
	3:02:34	646	650	0.62%	480	: :	
CMS 650-1	0:04:50	647	650	0.46%	7	0: 4 :50	649
	3:03:13	648	650	0.31%	904	: :	
CMS 700-1	0:13:06	686	700	2.04%	58	0:13:6	699
	3:03:00	691	700	1.30%	1045	: :	
CMS 750-1	0:07:53	738	750	1.63%	28	0: 7 :53	750
	3:02:19	741	750	1.21%	521	: :	
CMS 750-4	0:07:05	736	750	1.90%	28	0: 7 :5	748
	5:03:57	743	750	0.94%	810	: :	
CMS 800-0	0:19:15	773	800	3.49%	55	0:19:15	798
	3:02:16	773	800	3.49%	533	: :	
CMS 800-1	0:22:24	784	800	2.04%	92	0:22:24	800
	3:02:30	786	800	1.78%	761	: :	
Railway	0:00:31	95	103	8.42%	1	0: 0 :31	100
	4:42:11	100	101	1.00%	19	: :	
Map lab (4P)	: :					: :	
CMS 600-0	0:00:0.1	543	600	10.5%	2	0: 0 :0.04	600
	3:02:57	574	600	4.53%	782	: :	
CMS 600-1	0:39:07	565	600	6.19%	184	0:39:7	597
	3:02:55	568	600	5.63%	831	: :	
TOTALI e	: :	**Gap (t. =)**		**3.6%**		**2: 5 :4.1**	
MEDIE	**37:10:52**	**Gap (t.fin.)**		**2.0%**		: :	

Tabella 5.3: Gruppo 2. Problemi di Map Labelling risolti da CBC ma non da Cplex

tre di analisi statistica (Chorales-134, Bridges-132 e Mech-analysis-152) ed uno di map labelling (CMS_600-1) ed i grafici sono riportati rispettivamente nelle figure dalla 5.5 alla 5.8. Si coglie l'occasione per ricordare che mentre le istanze di analisi statistica sono problemi di minimizzazione, quelle di map labelling sono in forma di massimo. Tutti i grafici fino ad ora presentati mettono in luce un fatto molto importante, relativo al diverso comportamento dei due programmi rispetto all'andamento del best bound. Si vuole premettere che in tutti i problemi del test–bed utilizzato si osserva che non c'é quasi nessuna difficoltá nel trovare buone soluzioni intere (generalmente giá da subito vicine all'ottimo) mentre invece risulta molto piú difficile far muovere il bound, al fine di provare l'ottimalitá delle soluzioni; ed é proprio in questo che CBC si comporta enormemente meglio di Cplex: i grafici dimostrano che in tutti i casi le curve relative al bound di CBC risultano molto

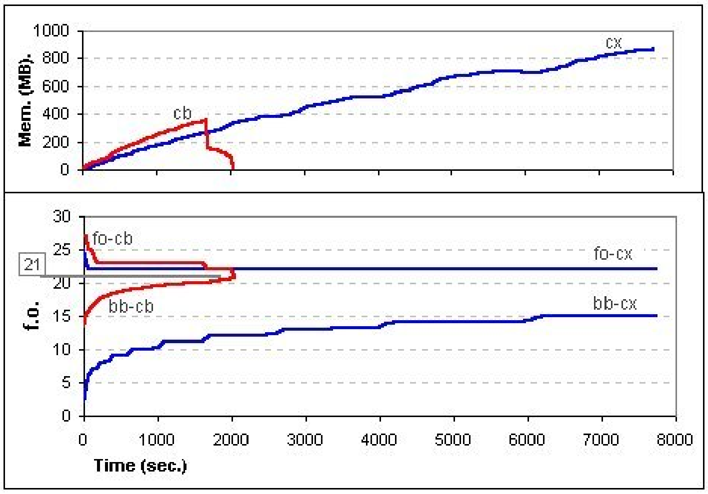


Figura 5.7: Grafici sugli andamenti delle ottimizzazioni, file Mech-analysis-152

meno affette dal fenomeno di "appiattimento" che invece si manifesta, col passare del tempo, nelle curve relative a Cplex. Inoltre é di fondamentale importanza notare che il punto di partenza delle due curve (ovvero il valore della funzione obiettivo al termine delle computazioni eseguite sul nodo radice) é estremamente diverso: quello di CBC fornisce un gap notevolmente minore. In parecchi casi si é osservato che il valore del bound dato da CBC all'inizio é addirittura migliore di quello che Cplex riesce a fornire alla fine della computazione (ovvero dopo ore di calcolo).

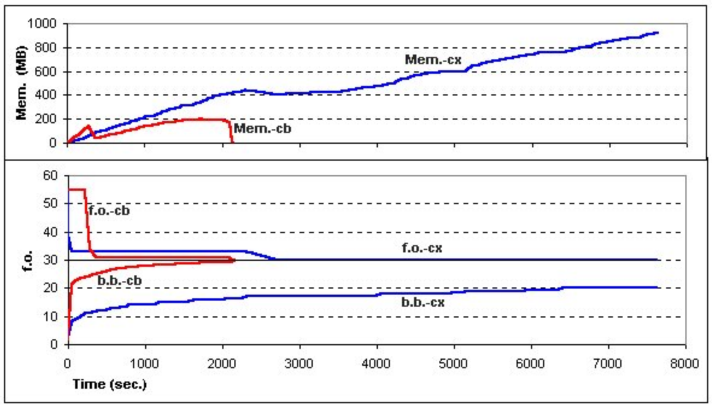


Figura 5.8: Grafici sugli andamenti delle ottimizzazioni, file Chorales-134

Uno dei motivi di questo comportamento é certamente da ricercarsi nell'assenza dei vincoli con coefficienti big–M: come spiegato nella sotto–sezione

2.1.2 la loro presenza induce una maggiore diversitá tra il rilassamento continuo del problema ed il suo convex–hull; quando invece il dominio viene definito tramite l'applicazione dei tagli combinatori di Benders esso risulta decisamente piú "vicino" al convex–hall, con i grandi benefici che ne conseguono e sono palesati dai tempi di risoluzione.

Per quel che rigurada il terzo gruppo i dati raccolti sono riassunti nelle due tabelle 5.5 (per quelli di Analisi Statistica, spezzata i due per ragioni di spazio) e 5.4 (per i Map Labelling, anch'essa spezzata in due parti per ragioni di spazio). Il termine dell'elaborazione é stato determinato dal un superamento di un limite di tempo o di memoria occupata. Come per i problemi del secondo gruppo, anche per i problemi di analisi statistica di questo terzo gruppo sono stati riportati, nella parte della tabella 5.5 relativa a Cplex, i dati presi in due istanti diversi: a paritá di tempo impiegato dal programma CBC ed alla fine dell'elaborazione. Sebbene qui le differenze non siano cosí eclatanti come nei due gruppi precedenti, sono comunque molto marcate, come si puó notare osservando i gap ottenuti. Per comoditá si sono riportate le differenze, a paritá di tempo, di tali gap nella colonna aggiuntiva che compare nella parte dedicata a CBC delle due tabelle 5.5 e 5.4.

Anche per questo terzo gruppo sono stati prodotti dei grafici esemplificativi; i problemi presi ad esempio sono stati Solar-flare-1066 (di analisi statistica) e CMS_900-0 (di map labelling), i cui dati sono rappresentati rispettivamente nelle figure 5.9 e 5.10.

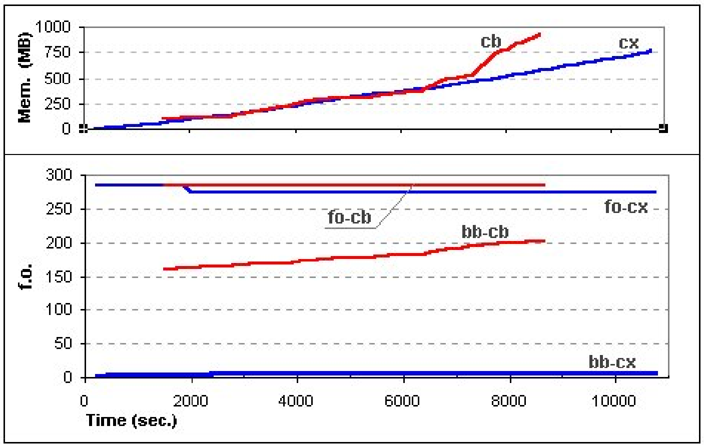


Figura 5.9: Grafici sugli andamenti delle ottimizzazioni, file Solar-flare-1066

In tutti i casi tranne uno, al CBC aveva ottenuto un risultato migliore, non a paritá di tempo ma anche al termine dell'esecuzione di Cplex. Inoltre nell'unico caso in cui Cplex si comporta meglio (uno dei due problemi della sezione denominata Horse–colic) il lower bound individuato da CBC dopo 34 minuti é migliore non solo di quello trovato da Cplex a paritá di tempo,

Gruppo 3	**Cplex**				
Categoria e Nome file	**Time h:m:s**	**fo**	**bb**	**Gap %**	**Mem MB**
Map lab (4S)					
Berlin	2:25:18	38	47.6	25.4%	835
CMS 900-0	3:02:47	881	900	2.2%	676
CMS 1000-0	3:01:46	945	1000	5.8%	566
US-Abbrv	3:01:18	73	104.8	43.6%	740
Map lab (4P)					
CMS 650-0	3:02:47	609	650	6.7%	706
CMS 650-1	3:02:51	604	650	7.6%	798
TOTALI e MEDIE	**19:35:17**		**Gap**	**15.22%**	

Gruppo 3	**CBC**					**Statis.**
Categoria e Nome file	**Time h:m:s**	**fo**	**bb**	**Gap %**	**Mem MB**	**ΔGap (t. =)**
Map lab (4S)						
Berlin	2:25:18	38	45.2	19.1%	474	6.3%
CMS 900-0	3: 2 :47	897	898.5	0.2%	283	2.0%
CMS 1000-0	3: 1 :46	978	998.3	2.1%	509	3.7%
US-Abbrv	3: 1 :18	77	99.7	29.5%	428	14.1%
Map lab (4P)						
CMS 650-0	2:11:23	634	646.8	2.0%	1093	4.7%
CMS 650-1	3: 2 :51	638	648.0	1.6%	706	6.0%
TOTALI e MEDIE	**16:45:23**		**Gap**	**9.07%**		**6%**

Tabella 5.4: Gruppo 3. Problemi di Map Labelling non risolti da entrambi

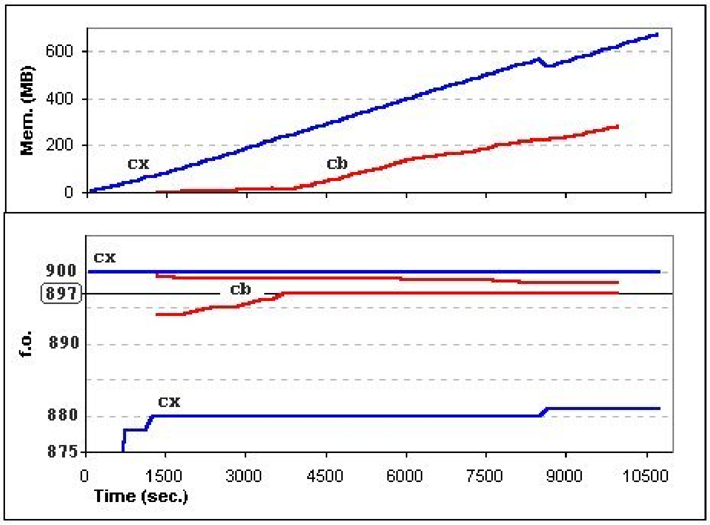


Figura 5.10: Grafici sugli andamenti delle ottimizzazioni, file CMS_900-0

Gruppo 3	**Cplex**				
Categoria e Nome file	**Time h:m:s**	**fo**	**bb**	**Gap %**	**Mem MB**
Flags					
Flags-169	0:31:37	10	4.0	59.9%	—
	3:00:19	10	5.0	49.9%	270
Horse Colic					
Horse-colic-253	0:43:21	13	4.0	69.2%	—
	3:00:15	13	5.0	61.5%	260
Horse-colic-185	0:34:27	11	4.0	63.5%	62
	3:00:16	11	5.0	54.5%	250
Solar Flare					
Solar-flare-1066	2:20:57	273	7.2	97.4%	556
	3: 0 :18	273	7.4	97.3%	754
TOTALI e MEDIE	**4:10:22**	**Gap (t. =)**		**72.51%**	
	12:01:8	**Gap (t.fin)**		**65.78%**	

Gruppo 3	**CBC**					**Statist**
Categoria e Nome file	**Time h:m:s**	**fo**	**bb**	**Gap %**	**Mem MB**	**ΔGap (t.=)**
Flags						
Flags-169	0:31:37	10	5.57	44.3%	1123	15.6%
Horse Colic						
Horse-colic-253	0:43:21	13	7.97	38.7%	1131	30.5%
Horse-colic-185	0:34:27	24	5.47	77.2%	1042	-13.7%
Solar Flare						
Solar-flare-1066	2:20:57	284	201.27	29.1%	968	68.2%
TOTALI e MEDIE	**4:10:22**	**Gap (t. =)**		**47.33%**		**25%**

Tabella 5.5: Gruppo 3. Problemi di Analisi Statistica non risolti da entrambi

ma anche di quello raggiunto dopo tre intere ore di calcolo. Il motivo per cui Cplex si comporta meglio é perció dovuto ad una migliore soluzione intera trovata: tuttavia si ricorda a questo proposito che CBC non puó avvalersi delle funzionalitá di ricerca euristica delle soluzioni intere (a parte al nodo radice) a causa di un giá menzionato bug presente nella libreria di funzioni fornita nel pacchetto di Cplex8.0; quando il programma é eseguito nella modalitá cplex_only utilizza invece permanentemente le funzioni euristiche e la soluzione migliore é stata trovata proprio grazie al loro utilizzo, senza il quale probabilmente non sarebbe stata individuata. Ció spiega (anche se solo parzialmente) il motivo per cui dopo appena poco piú di mezz'ora CBC abbia esaurito la memoria disponibile: non avendo un upper bound molto efficace il meccanismo di cut–off riesce ad eliminare troppo pochi nodi e perció l'albero di branching che si genera é di dimensioni maggiori. Infine occorre sottolineare un fatto che merita attenzione: se si escludono i casi in cui entrambi i problemi hanno raggiunto il time–limit, in tutti gli altri casi CBC ha terminato l'esecuzione prima di Cplex a causa del fatto che ha esaurito prima la memoria disponibile. Non sono state fatte particolari indagini per capire l'effettiva natura di questo maggior consumo di memoria, tuttavia si possono elencare un paio di elementi che vi constribuiscono: uno é la maggiore quantitá di strutture dati ausiliare che devono essere mantenute per implementare questo metodo; il secondo é l'utilizzo dei tagli locali che costringe a mantenere attive altre informazioni che consentano di indicare in quali nodi sono attivi i vari tagli.

5.3 Risultati: seconda parte

In questa sezione si analizzaranno i risultati ottenuti sui problemi appartenenti alle categorie broken glass e railway timetabling. A differenza di quanto fatto per i precedenti test, si é utilizzata la versione 7.0 di Cplex per eseguire le prove sui problemi qui descritti.

Prima di tutto é peró necessario capire il motivo per cui le prestazioni ottenute sono cosí diverse da quelle degli altri tipi di problemi. La causa é probabilmente da ricercarsi nel diverso tipo di funzione obiettivo. Infatti mentre in questo tipo di istanze la funzione obiettivo prevede la presenza di variabili continue, nei problemi di Map Labelling e di Analisi Statistica, vi sono solo variabili binarie. Tale presenza comporta, come giá spiegato, l'impossibilitá di inserire la funzione obiettivo nel problema master e ció ha l'importante conseguenza di impedire il meccanismo del cut–off. Questo fatto é normale tutte le volte che non si usa una funzione obiettivo: tuttavia nei problemi in cui essa é assente normalmente si é interessati ad individuare una soluzione qualsiasi (non si dá un valore/peso diverso alle diverse soluzioni) e perció l'ottimizzazione si interrompe alla prima soluzione trovata. Nel nostro caso invece non si é interessati ad una soluzione qualsiasi: é per questo che, come giá spiegato nella sezione teorica, si é costretti ad individuare tutte le soluzioni possibili e poi valutare la migliore sulla base della funzione obiettivo del problema originale. Si capisce anche il motivo per cui occorre aggiungere un taglio completo ad ogni soluzione trovata (al fine di eliminare la soluzione stessa dalla regione feasible): proprio per evitare che l'ottimizzazione del problema master termini, come normalmente avviene. Inoltre l'assenza della funzione obiettivo fa sí che l'algoritmo di risoluzione (simplesso) ricerchi i vertici costituenti la soluzione ottima dei vari nodi in zone arbitrarie del politopo costituente il dominio del problema, al posto che concentrarsi attorno alla zona ove dovrebbero trovarsi le soluzioni ottime, sulla base della funzione obiettivo. In modo informale ma espressivo si puó dire che senza una funzione obiettivo "il simplesso gira per tutti i vertici del politopo senza una precisa meta" individuando una miriade di soluzioni che si rivelano poi molto distanti dall'ottimo e che non sarebbero mai state generate se vi fosse stato attivo un meccanismo di cut–off. Tradotto in termini di branching questo ragionamento fa capire che l'albero che si viene a generare é decisamente piú grande dato che comprende anche la discendenza di tutti quei nodi che forniscono un valore ottimo maggiore della soluzione incumbent trovata e l'unico caso in cui un nodo non genera figli é che esso risulti infeasible. Si potrebbe pensare che sarebbe sufficiente individuare proprio quei nodi che dovrebbero subire il cut–off se vi fosse una funzione obiettivo ed invece non lo subiscono, eliminandoli artificialmente: ció non é peró corretto infatti proprio l'assenza della funzione obiettivo fa sí che (come detto) si individui in un vertice arbitrario la soluzione di ogni nodo; se si utilizzasse tale soluzione per calcolare la funzione obiettivo si correrebbe il rischio di eliminare un nodo che invece possiede altre soluzioni migliori di quella trovata, magari proprio l'ottima.

Tutto quanto detto fa sí che il miglioramento di prestazioni introdotto dalla risoluzione tramite tagli combinatori di Benders sia completamente sovrastato dal peggioramento indotto dall'assenza della funzione obiettivo. Questa deduzione é supportata dalla seguente prova che si é fatta: si é iniziato a risolvere uno dei problemi di railway timetabling tramite CBC; ad un certo punto si é interrotta l'ottimizzazione, salvando su disco tutti i tagli fino ad allora generati (qualche migliaio); si sono aggiunti tali tagli alla formulazione originale del problema e si é risolta tramite normale branch & bound questa formulazione arricchita. In questo modo si é riusciti a risolvere in circa 20 minuti un problema che prima non veniva risolto entro l'ora di calcolo. Questo testimonia da una parte l'efficacia dei tagli cmmbinatori di Benders (si rammenta che la formulazione arricchita aveva un numero molto maggiore di vincoli e quindi ogni nodo richiedeva piú tempo di calcolo) e dall'altra che l'assenza della funzione obiettivo contribuisce in modo determinante a peggiorare la situazione.

Sulla base di quanto esposto fin'ora e di quanto verrá ora detto si puó affermare che i test effettuati su problemi con variabili continue nella funzione obiettivo non risultano molto significativi: lo sarebbero stati solo se il programma CBC avesse portato a termine l'ottimizzazione di qualche problema in tempi non eccessivi. Ció peró é avvenuto solo in tre dei sei problemi di broken glass, i quali peró sono talmente piccoli da poter essere ritenuti dei "problemi giocattolo" in cui addirittura anche l'enumerazione di tutte le soluzioni possibili potrebbe essere portata a termine in tempi ragionevoli. In tutti i restanti problemi non si é riusciti ad individuare la soluzione ottima ma solo delle soluzioni intere. L'assenza di un lower bound (a causa sempre dell'assenza della funzione obiettivo nel master) non permette di capire peró quanto possa mancare al termine dell'elaborazione e quindi ogni confronto con i risultati di Cplex risulta molto difficile. Il tutto é reso ancor piú complicato dal fatto che anche Cplex, a parte i tre problemi giá menzionati, non sia riuscito a terminare l'elaborazione di nessuno degli altri problemi.

Per cercare di ottenere indicazioni piú precise sull'andamento dell'ottimizzazione e sopperire alla mancanza dell'informazioni normalmente apportata dal lower bound, si é anche cercato di creare dei parametri dai quali poter dedurre un'indicazione circa quanto potesse mancare alla fine della computazione. Alla fine si é riusciti a costrutire in modo molto semplice un indice che rivela la percentuale di nodi dell'albero di branching ancora da esplorare. Per ottenerlo é sufficiente sommare i numeri massimi di figli che ogni nodo attivo potrebbe generare, dividendo poi tale somma per il numero di massimo possibile di "figli dell'intero albero"; in formule:

$$(\sum_{i \in F} 2^{n-d_i}) \cdot \frac{1}{2^n} \cdot 100 = 100 \sum_{i \in F} 2^{-d_i},$$

dove n é il numero di variabili binarie del problema (ovvero l'altezza massima possibile dell'albero), F é l'insieme dei nodi attivi (ovvero ancora da processare in un certo istante) e d_i é la profonditá del nodo attivo i–esimo rispetto al nodo radice. Purtroppo questo indice ha domostrato delle proprietá non sempre utili: in alcuni problemi infatti, dai suoi valori si riescono ad ottenere previsioni di termine dell'ottimizzazione abbastanza veritiere; in altri peró presenta andmenti con zone lungamente stazionarie seguite da bruschi cali improvvisi che lo rendono di difficile interpretazione. Si é comunque fatta menzione di questo indice perché é possibile che esso possa diventare la base di calcolo di altri parametri piú sofisticati e precisi.

Si vuole infine terminare indicando i piú significativi tra i tentativi fatti per compensare il peggioramento che si induce con la rimozione della funzione obiettivo dal problema master, mantenendo peró i benefici dei tagli combinatori di Benders.

Una possibile via é quella di ricorrere all'uso combinato dei tagli di Benders tradizionali e di quelli combinatori: la presenza dei primi consente il mantenimento della funzione obiettivo nel problema master mentre i secondi dovrebbero rafforzare la formulazione a livello del rilassamento continuo del problema, avvicinandolo al convex hull.

Un'altra starda percorribile é quella di inserire, oltre ai Combinatorial Benders' Cuts, anche le disequazioni ad attivazione condizionale come se fossero tagli, ovvero non appena se ne trovasse qualcuna che viene violata dalla soluzione del nodo corrente. In pratica in questo modo si posticiperebbe l'inserzione di tali vincoli nel modello in modo che la loro presenza non influenzi negativamente (come di solito avviene) il calcolo dei valori da far assumere alle variabili di controllo.

Purtroppo entrambi questi metodi hanno il difetto di reintrodurre direttamente o indirettamente i coefficienti big–M. Esiste tuttavia un terzo modo di procedere che merita una certa attenzione. Esso, come é giá stato ampiamente anticipato nei capitoli precedenti, consiste nell'inserire la disequazione 4.1 nel problema slave. Grazie a questa disequazione si riesce a "simulare" una specie di cut–off. Si puó capire questa affermazione se si pensa a quanto segue. Supponiamo che il nodo corrente del problema master generi una soluzione intera che sia ammissibile anche per il problema originale ma dia un valore della funzione obiettivo peggiore rispetto alla soluzione incumbent. Se la disequazione 4.1 non fosse presente, per le ipotesi fatte il problema slave risulterebbe feasible indicando l'avvenuta scoperta di una nuova soluzione valida per il problema originale; la soluzione verrebbe scartata in quanto peggiore e si produrrebbe un taglio completo per far procedere la ricerca di soluzioni nel master. Tuttavia se la disequazione 4.1 é inserita nello slave questo diventa necessariamente infeasible: a questo punto é possibile innescare il meccanismo di minimizzazione ed inserire un taglio combinatorio simile a tutti gli altri; l'effetto di questo taglio sará quello di impossibilitare la generazione di tutte le soluzioni che contengono le variabili contenute nel taglio con valori fissati ai medesimi valori della soluzione corrente: tutte queste soluzioni non vengono piú generate proprio perché produrrebbero un valore della funzione obiettivo peggiore di quello della soluzione incumbnt attuale. In sostanza si riproduce lo stesso effetto del cut–off a livello di soluzioni generate al posto che a livello dei nodi dell'albero decisionale: naturalmente non é detto che ció abbia la medesima efficacia. In termini intuitivi si é spostato il cut–off dal master allo slave (dove va per la sua natura la funzione obiettivo) tramutandolo inoltre in una condizione di infeasibility dato che questa é proprio la situazione naturale secondo cui operano i tagli di Benders combinatori, che sono poi il tramite per riportare l'informazione ottenuta nel problema master. Tutto quanto detto vale naturalmente anche in situazioni in cui il nodo master corrente generi una soluzione che attribuisca valori frazionari ad alcune variabili intere.

A questo punto é anche possibile spiegare il motivo per cui questo tipo di taglio si possa usare solo con problemi il cui la funzione obiettivo non presenti variabili intere ma non con quelli con la funzione obiettivo "mista". In apparenza non ci sono infatti grosse differenze nei due casi; tuttavia se ben si riflette si osserva che in quest'ultimo caso la minimizzazione del taglio che si dovrá fare non é piú lecita, rendendo di fatto completamente inutile l'utilizzo di questo sistema. Il motivo di quanto datto si puó piú facilmente capire con un esempio: supponiamo vi siano n variabili binarie tutte fissate ad 1 dalla soluzione del nodo corrente e che tali volori producano uno slave infeasible proprio a causa della disequazione che si é aggiunta. Supponiamo inoltre che il MIS individuato contenga tutti i vincoli ad attivazione condizionale tranne quelli che coinvolgono la variabile di indice 0 e che questa sia proprio la sola variabile intera presente nella funzione obiettivo. Il taglio che si genererebbe in questo caso sarebbe:

$$\sum_{i=1}^{n-1} x_i \leq n-2.$$

Questo taglio esclude peró anche le soluzione $x_0 = 0, x_1 = ... = x_{n-1} = 1$, senza che di questa soluzione si abbia alcuna informazione (tra l'altro questa soluzione aumenta, grazie alla variabile x_0, di una unitá il termine noto della disequazione 4.1 rendendola piú difficile da violare e potrebbe quindi fornire un valore della funzione obiettivo potenzialmente migliore della soluzione incumbent). É immediato rendersi conto dell'errore se si pensa che anche la disequazione aggiunta in questo particolare esempio risulta avere la forma di un vincolo ad attivazione condizionale e quindi andrebbe contato anch'esso nella trasposizione da MIS a taglio di Benders combinatorio: se lo si facesse si otterrebbe infatti un taglio completo che non escluderebbe la soluzione con $x_0 = 0$. Ció non é tuttavia possibile in uno scenario piú complesso in cui vi fossero piú variabili intere contemplate nella funzione obiettivo. Inoltre si avrebbero delle difficoltá anche quando, pur essendoci una sola variabile intera nella funzione obiettivo, questa assumesse un valore frazionario nella soluzione del nodo master corrente. A prescindere da questo esempio particolare il problema che insorge in casi come questo é la presenza di un vincolo che coinvolge piú di una variabile intera, il che non rende piú valido il meccanismo di traduzione da MIS a taglio. [5]

Il motivo per cui si é scelto di implementare solo questo terzo tipo di soluzione é essenzialmente di carattere pratico: é infatti relativamente contenuto il numero di cambiamenti da effettuarsi sul codice. Inoltre é proprio grazie a questa modifica che si sono riusciti a risolvere i tre problemi di broken glass citati in precedenza. Tuttavia i test effettuati sono in numero troppo limitato ed ad una fase embrionale per poter affermare che i primi due metodi indicati siano di utilitá inferiore a quest'ultimo, senza contare il fatto che altre idee e altre soluzioni potrebbero affiancarsi alle attuali.

Resolution search Si vuole infine concludere con una interessante osservazione di carattere teorico che ha proprio a che fare con il sistema appena descritto: l'approccio dei tagli combinatori di Benders unitamente alla disequazione 4.1 ha, per i problemi con funzione obiettivo con sole variabili continue, delle analogie con un metodo di enumerazione implicita noto con il nome di Resolution Search. Si puó trovare una descrizione di tale metodo, che é applicabile solo a problemi MIP non contenenti variabili intere non binarie, in [10]. Tralasciando molti dettagli si puó cosí riassumere la strategia di Resolution Search; esistono due soggetti principali: il primo é costituito dall'insieme di una famiglia di clausule e di un gruppo di regole che governa tale famiglia; il secondo é il cosiddetto oracolo. Questi due soggetti interagiscono in questo modo: la famiglia di clausule, grazie alle regole ad essa associata, genera delle possibili soluzioni e le fornisce all'oracolo; quest'ultimo le analizza e se non le trova conformi ai suoi vincoli restituisce alla famiglia una nuova clausula che descrive questa situazione. Tutto ció sembrerá di certo molto oscuro ma non lo sará piú se si cerca di tradurre questi nomi con dei concetti equivalenti e con una terminologia piú familiare: si puó infatti dire che le clausule non sono altro che delle particolari

[5] Si badi che la ricerca dei MIS resta comunque lecita

disequazioni lineari le quali hanno esattamente la stessa struttura dei tagli combinatori di Benders e l'oracolo non é altro che il il sistema MIP originale senza vincoli di interezza. Se si prescinde dal fatto marginale che qui le disequazioni che coinvolgono solo variabili binarie ("disequazioni intere") sono poste insieme alle disequazioni "miste" ed a quelle "continue", [6] si riesce facilmente ad individuare una analogia forte tra pool di vincoli e famiglia di clausule e tra problema slave ed oracolo, nonché una anoligia nelle modalitá di interrelazione tra tali entitá. Ció che viene a mancare é il problema master: nel sistema descritto in questa tesi il compito del master é essenzialmente quello di individuare delle soluzioni che non violino nessuno dei tagli fino a quel momento generati e nessuno dei vincoli interi del problema; il fatto che nel sistema resolution search non vi sia un elemento master é dovuto alla presenza dell'insieme di regole che governano la famiglia di clausule: grazie ad esse é sempre possibile individuare un assegnamento per le variabili binarie che soddisfi tutte le clausule. Fino a questo punto l'analogia sussiste anche con problemi in cui non vi sia il taglio 4.1: tuttavia bisogna dire che il sistema resolution search é in grado di generare le sue clausule anche nel caso si individui una soluzione intera feasible ma peggiore di quella incumbent; é proprio in questo comportamento che si evince la somiglianza cui si é accennato in precedenza.

[6] Si é detto marginale perché ognuna delle disequazioni intere si puó trasformare in un insieme di clausule ovvero di Combinatorial Benders' Cuts, i quali possono quindi essere spostati all'interno della famiglia di clausule

Capitolo 6

Conclusione

La realizzazione di questa tesi ha comportato notevoli sforzi ed il superamento di parecchie difficoltà, tuttavia si può sicuramente affermare che gli scopi che ci si erano proposti sono stati in gran parte raggiunti. Il principale obiettivo, consistente nello studio di fattibilità e realizzazione di un nuovo approccio per la soluzione di problemi di ottimizzazione combinatoria, è stato portato a termine: quanto è stato prodotto ha dimostrato prestazioni ottime, sebbene su un sottoinsieme della classe di problemi che ci si era proposti di affrontare. In ogni caso, già in fase di sviluppo si sono osservate varie possibilità di miglioramento e generalizzazione (alcune esposte in questa tesi) ed anche interessanti agganci con altre metodologie e sistemi di risoluzione.

Inoltre, come prodotti collaterali (ma comunque importanti), sono stati conseguiti altri risultati di grandi utilità e sono state aperte le porte a futuri lavori di ricerca.

In definitiva possiamo riassumere quanto ottenuto con i seguenti punti:

- definizione e implementazione di un framework di carattere generale, basato su Concert Technology 1.2 e Cplex 8.0, adatto alla creazione di separatori per nuove strategie di branch & cut; questo prodotto risulterà di certo utile per incrementare la produttività del gruppo di Ricerca Operativa dell'università di Padova

- ideazione e sviluppo di un algoritmo "fast MIS search" flessibile ed efficente

- ideazione, analisi, implementazione e test di un risolutore di problemi MIP, estremamente efficiente per problemi con funzione obiettivo comprendente solo variabili intere

Si vuole infine concludere indicando i principali obiettivi che si hanno per l'immediato futuro: si intende cercare, oltre che di migliorare le prestazioni già ottenute, anche di generalizzare gli ottimi risultati a classi più vaste di problemi, tra cui problemi con funzione obiettivo contenete variabili reali, problemi con più variabili di controllo per vincolo (come spiegato in appendice A) e problemi con variabili di controllo intere non binarie.

Appendice A

Estensioni a modello

In questa appendice si introdurranno due interessanti estensioni alla teoria fino ad ora esposta; queste permettono di ampliare la gamma di problemi che é possibile risolvere con l'approccio dei tagli combinatori di Benders.

A.1 Modelli non lineari

Consideriamo il seguente sistema:

$$\begin{aligned} z = \min \quad & a_0^T y + f_0(x) && \text{(A.1)} \\ \text{s.t.} \quad & Qx \geq q && \text{(A.2)} \\ & a_i^T y + f_i(x) \geq b^i, \quad i = 1, ..., n && \text{(A.3)} \\ & Ry \geq r && \text{(A.4)} \\ & y \geq 0 && \text{(A.5)} \\ & x \geq 0, \qquad \text{integer}, && \text{(A.6)} \end{aligned}$$

dove le funzioni $f_i(x)$, $i = 0, ..., n$ sono generiche funzioni $\mathbb{R}^n \to \mathbb{R}$, possibilmente anche non lineari. Supponiamo per ora, per semplicitá, che le funzioni f_i, $i = 1, ..., n$ dipendano da una sola variabile intera ovvero (con leggero abuso di notazione) $f_i(x) = f_i(x_{j_i})$. Si osserva subito che la classe di problemi cosí individuata include quelli su cui possono operare i Combinatorials Benders' Cuts (non appena si imponga la linearitá sulle funzioni $f_i(x_{j_i})$). Si nota peró altrettanto facilmente che anche se le funzioni f_i fossero di tipo non–lineare, l'algoritmo descritto nella sezione 2.3 si potrebbe applicare totalmente immutato. Infatti, non appena si ottiene una soluzione dal problema master, la sostituzione all'interno del problema slave comporta l'eliminazione di tutti gli elementi non lineari (le funzioni f_i diventano parte del termine noto), riconducendo nuovamente il problema slave nel dominio dei problemi lineari (il che permette di risolverlo con le note tecniche di programmazione lineare e soprattutto di poter ricercare i MIS secondo quanto descritto in 2.4). Il fatto che le f_i dipendano da una sola variabile permette di derivare il taglio combinatorio di Benders dal corrispondente MIS individuato.

Se invece le funzioni f_i dovessero dipendere da piú di una variabile intera, il meccanismo di linearizzazione e di ricerca di MIS funzionerebbe ancora ma occorre prestare attenzione a come si utilizza il principio di passaggio da MIS a taglio combinatorio. Ecco come occorre procedere per tener conto di questa nuova situazione: é sufficiente inserire nell'insieme F tutti gli indici delle variabili intere da cui dipendono quelle funzioni f_i che compaiono nei vincoli coinvolti nella MIS e poi il taglio si puó scrivere come sempre:

$$\sum_{i \in F: x_i = 0} x_i + \sum_{i \in F: x_i = 1} (1 - x_i) \geq 1.$$

In generale si dovranno quindi inserire in F piú indici per ogni vincolo coinvolto nel MIS. Si osserva subito che questa definizione dell'insieme F comprende come caso particolare quella data in 2.3. Tuttavia questo sistema nasconde una insidia: se il numero di variabili da cui dipendono le varie funzioni f_i é molto elevato si otterranno dei tagli che coinvolgono molte variabili binarie: ció comporta una loro efficacia molto limitata e la necessitá di impiegarne un numero troppo grande (che potrebbe diventare ingestibile in termini di occupazione di memoria ed allungare eccessivamente i tempi di computazione); inoltre si renderebbe poco utile la procedura di ricerca di MIS, la quale ridurrebbe il numero di disequazioni incompatibili senza che ció comporti una sostanziale riduzione del numero delle variabili coinvolte nei tagli.

In definitiva, il sistema qui descritto consente di trattare anche problemi che presentano delle componenti non lineari purché essi siano del tipo inizialmente indicato e purché le funzioni f_i dipendano ognuna da poche variabili intere (meglio ancora se da una sola).

Si vuole infine far notare che tutto quello che si é detto relativamente all'estensione della generazione dei tagli alla situazione in cui ci siano piú variabili binarie coinvolte nelle disequazioni "miste" non dipende assolutamente da nessuna particolare caratteristica delle funzioni f_i, quindi queste potrebbero benissimo essere lineari. In tal caso l'implementazione di questo sistema permetterebbe di far cadere uno dei requisiti che si erano imposti sulla classe dei problemi trattabili con il metodo dei tagli combinatoria di Benders (quello di avere una sola variabile binaria di controllo per ogni disequazione): i motivi per cui questo sistema non é stato implementato sono il fatto che sarebbe stato necessario un programma piú complicato che avrebbe influenzato negativamente i tempi di risoluzione ed il fatto che, come detto, l'efficacia dei tagli sarebbe fortemente influenzata (cioé penalizzata) del numero di variabili coinvolte per ogni vincolo misto. C'é poi una considerazione ancora piú importante: se ci limitiamo ai sistemi lineari esiste un approccio molto piú semplice che risolve il problema della dipendenza dal numero di variabili binarie presenti in ogni vincolo: lo si esporrá nella prossima sezione.

A.2 Generici modelli lineari con variabili binarie

Si consideri il seguente problema MIP:

$$z = \min \quad c_b^T x^b + c_i^T x^i + d^T y \tag{A.7}$$
$$\text{s.t.} \quad Q_b x^b + Q_i x^i \geq q \tag{A.8}$$
$$M x^b + B y \geq b \tag{A.9}$$
$$R y \geq r \tag{A.10}$$
$$y \geq 0 \tag{A.11}$$
$$x^i \geq 0, \qquad \text{integer} \tag{A.12}$$
$$x^b \geq 0, \qquad \text{binary,} \tag{A.13}$$

A parte la distinzione di comodo che si é fatta all'interno delle variabili intere x (suddivise in variabili binarie x^b e variabili x^i che possono assumere anche altri valori oltre a 0 ed 1), il problema descritto differisce da quello utilizzato per sviluppare la teoria dei tagli combinatori di Benders solo per il fatto che non sono stati imposti vincoli sulla matrice M. Se si cercasse di risolvere un tale problema come fatto fino ad ora ci si scontrerebbe con la ormai nota difficoltá di non poter associare ad ogni vincolo del problema slave una sola variabile di controllo e quindi di non poter convertire un MIS nel corrispondente taglio.

Tuttavia con una lieve modifica alla struttura del problema slave si puó aggirare l'ostacolo. Infatti, al posto di sostituire i valori delle variabili intere all'interno delle disequazioni che le contengono si puó operare in modo del tutto equivalente aggiungendo allo slave delle equazioni di questo tipo:

$$x_j^b = \hat{x}_j^b, \tag{A.14}$$

dove $\hat{x}_j^b$ é il valore (intero) assunto dalla variabile x_j^b nella soluzione del nodo corrente del problema master. Tali equazioni andranno aggiunte per le sole variabili binarie che hanno assunto un valore intero e non per quelle alle quali viene attribuito un valore frazionario. Bisognerá inoltre avere cura di rimuovere i vincoli di interezza sulle variabili x^b nel problema slave, altrimenti questo verrebbe visto dal risolutore come un problema MIP e non come un problema LP. Con questi due semplici ritocchi si ottinene uno slave sostanzialmente equivalente a quello che si é utilizzato fino ad ora: l'unica lieve differenza consiste nel fatto che in precedenza venivano inseriti nello slave solo i vincoli le cui variabili binarie di controllo avevano assunto un valore intero mentre qui i vincoli misti sono presenti sempre tutti. Come vedremo ció non influenzerá in alcun modo il meccanismo.

L'osservazione chiave consiste nel notare che ora, indipendentemente dal numero di variabili binarie coinvolte nei vincoli misti del problema originale (cioé indipendentemente dai coefficienti di M), si é in presenza di un sistema slave in cui vi sono delle equazioni (A.14) che coinvolgono una sola variabile binaria: ció suggerisce l'idea di poter generare comunque i tagli combinatori di Benders basandosi sul solo sottoinsieme di queste equazioni che andranno a far parte del MIS, ignorando semplicemente le altre disequazioni presenti nel MIS (anche se queste dovessero coinvolgere delle variabili x_j^b). Non é del tutto ovvio peró se questa operazione abbia senso (ovvero generi comunque dei tagli soddisfacenti gli scopi per cui essi sono generati) e se sia lecito restringere la ricerca ai soli MIS che contengano almeno una delle equazioni A.14. La risposta a questi due quesiti é sí; infatti, ammesso che il sistema originale non sia infeasible, le disequazioni A.10 e A.9 da sole non possono produrre infeasibility e perció non potrá esserci alcun MIS che non contenga almeno una delle equazioni A.14. Questa osservazione ci permette anche di sfruttare meglio una delle caratteristiche del sistema di ricerca di MIS: quello di poter orientare la ricerca maggiormente verso alcune disequazioni/equazioni piuttosto che altre variando i coefficienti della funzione obiettvo "artificiale" che si inseisce nel problema duale; in questo caso si potranno far comparire nella funzione obiettivo del problema duale solo le variabili duali corrispondenti ai vincoli A.14 e nessuna delle altre.

In definitiva quello che si é fatto é semplicemente rendere sempre possibile la traduzione di un MIS in un taglio aggiungendo delle equazioni che avessero proprio le caratteristiche necessarie (caratteristiche che venivano a mancare cambiando la matrice M) e facendo in modo che l'infeasibility dipendesse proprio da queste equazioni (e non piú esclusivamente da quelle 2.29).

Si vuole infine far notare come questa nuova forma dello slave permetta di utilizzare il sistema realizzato per i problemi con funzione obiettivo composta di sole variabili continue tramite l'aggiunta dell'equazione 4.1 anche per sistemi con funzione obiettivo "mista": il motivo é che ora non sussiste piú né il problema dovuto alla presenza di piú di una variabile binaria nei vincoli dello slave, né il problema dovuto al'impossibilitá di far variare i valori delle varaibili binarie (problema esposto nell'esempio con cui é stato introdotto l'argomento nel capito dei risultati computazionali) dato che ora le variabili binarie sono a tutti gli effetti parte del problema slave.

Appendice B

General B&C Framework reference

B.1 Arguments Struct Reference

Variables related to command line parameters.

`#include <arguments.h>`

Public Types

- enum **node_sel_t** { **cplex**, **deeper**, **upper** }

  Next-node selection policy.

Public Methods

- ostream & **print** (ostream &os) const

  Prints every parameter's value.

- **Arguments** (int argc, char *argv[ ])

Static Public Methods

- error_t **parse_opt** (int key, char *arg, struct argp_state *state)

  Command line parser from libc.

Public Attributes

- string **input_file**

  Usually a .lp file.

- string **output_dir**
  Directory to put log files.
- bool **cplex_only**
  Enable/disable callbacks.
- **node_sel_t node_sel**
  Next node selection strategy.
- IloCplex::MIPEmphasisType **MIPEmphasis**
  See Cplex docs.
- int **MIPInterval**
  See Cplex docs.
- int **MIPDisplay**
  See Cplex docs.
- bool **cplex_cuts**
  Wether to disable or not Cplex's cutting plane families.
- bool **cplex_heur**
  Whether to disable or not cplex's heuristic.
- int **time_limit**
  A time limit in seconds.
- int **sol_limit**
  Maximum number of int sol to be found.
- bool **null_callback**
  Use an empty cut callback.

Static Public Attributes

- char * **doc**
  General info about the application.
- char * **args_doc** = "INPUT_FILE"
  A description of parameters' syntax.

- argp_option **options** [ ]

  Every argp_option describes a parameter.

- argp **argp**

  Passed to arp_parse.

Detailed Description

Variables related to command line parameters.

The management of command line parameters is based on Argp, an interface for parsing unix-style argument vectors, part of libc. In case of sintax errors a usage message is displayed.

There is one and only (global) instance of this struct during execution.

Please read libc's documentation to understand how to add variables to manage parameters needed by your application.

Watch out: command line parameter strings are not always equal to variables names: use "./exe –usage" to get a list of accepted parameters.

Member Enumeration Documentation

enum Arguments::node_sel_t

Next-node selection policy.

You can force the program to always process the deepest or the least deep among branch tree active nodes. This is implemented by means of class **NodeCBI** (p. 114).

Enumeration values:

cplex Let cplex choose (default).

deeper Always the deepest.

upper Always the least deep.

Member Function Documentation

error_t Arguments::parse_opt (int *key*, char ∗ *arg*, struct argp_state ∗ *state*) [static]

Command line parser from libc.

Fills in the fields, according to descriptions provided in options[].

Member of struct argp argp.

ostream & Arguments::print (ostream & *os*) const

Prints every parameter's value.

Used to print a final report about the optimization

Member Data Documentation

struct argp Arguments::argp [static]

Passed to arp_parse.

Needed to pass doc, args_doc, options[] and parse_opt to argp_parse()

char * Arguments::args_doc = "INPUT_FILE" [static]

A description of parameters' syntax.

Member of struct argp argp

bool Arguments::cplex_cuts

Wether to disable or not Cplex's cutting plane families.

true = cuts are enabled (default);
false = cuts are disabled.

bool Arguments::cplex_heur

Whether to disable or not cplex's heuristic.

true = enabled;
false = disabled.

bool Arguments::cplex_only

Enable/disable callbacks.

true = run with std cplex (no callbacks);
false = otherwise (default).

char * Arguments::doc [static]

Initial value:

```
"Branch&Cut Framework for MIP solvers using Cplex8.0"
              "\v"
              "..."
```

General info about the application.

Member of struct argp argp.

bool Arguments::null_callback

Use an empty cut callback.

Using a cut callback forces cplex to avoid applying irreversible transformations to the model before branch & cut. These transformations can make a huge difference in optimization time; so, if you are testing a cutting plane family set this to true.

Default is false.

struct argp_option Arguments::options[] `[static]`

Every argp_option describes a parameter.

Member of struct argp argp.

int Arguments::sol_limit

Maximum number of int sol to be found.

Typically used for sat problems with a fake objective function

int Arguments::time_limit

A time limit in seconds.

A negative value means no limits (default)

The documentation for this struct was generated from the following files:

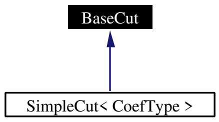
- **arguments.cpp**
- **arguments.h**

B.2 BaseCut Class Reference

Very general, abstract base class for cuts.

`#include <cut.h>`

Inheritance diagram for BaseCut:



Public Methods

- virtual ~**BaseCut** ()

  Destructor.

- virtual BaseCut & **operator=** (BaseCut const &c)

  Assignement operator.

- virtual IloRange **toIloRange** ()=0

  Conversion to IloRange.

- virtual bool **isViolated** (IloNumArray sol)=0

  Is this cut violated by solution sol?

- virtual void **print** (ostream &out)

  Used by the output operator.

Detailed Description

Very general, abstract base class for cuts.

At this level, we assume that a cut:

- is able to convert itself to the IloRange format
- can tell wether it is violated or not by a given solution.

We did not make any assumption on the implementation to let you choose the most efficient one for your application.

Member Function Documentation

virtual bool BaseCut::isViolated (IloNumArray *sol*) `[pure virtual]`

Is this cut violated by solution sol?

Parameters:

sol The solution is in IloNumArray format, which is the format used by CPLEX to return solutions.

Implemented in **SimpleCut< CoefType >** (p. 122).

virtual IloRange BaseCut::toIloRange () `[pure virtual]`

Conversion to IloRange.

We suggest you to implement a cache mechanism; it is easy, since IloRanges are handles.

Implemented in **SimpleCut< CoefType >** (p. 123).

The documentation for this class was generated from the following file:

- **cut.h**

B.3 BranchCBI Class Reference

A BranchCallbackI.

`#include <callbacks.h>`

Public Methods

- void **main** ()
- IloCplex::CallbackI * **duplicateCallback** () const

Detailed Description

A BranchCallbackI.

This callback lets you:

- override Cplex's branching algorithm;
- build a **DatiNodo** (p. 109) object and stick it to its node (more details in class **DatiNodo** (p. 109) documentation).

The documentation for this class was generated from the following files:

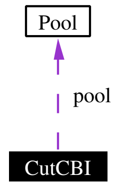
- **callbacks.h**
- **callbacks.cpp**

B.4 CutCBI Class Reference

A CutCallbackI.

`#include <cutcb.h>`

Collaboration diagram for CutCBI:



Public Methods

- IloCplex::CallbackI * **duplicateCallback** () const
- virtual void **main** ()

  The real entry point.

- **CutCBI** ()

Protected Attributes

- IloNumArray **sol**

  Here is put the current solution.

- **Pool** * **pool**

  A pool.

- int **num_int_vars**

  Num of integer vars.

- IloNumVarArray **int_vars**

  A handle pointing to the vector in class **Problem** (p. 118).

- int **depth**

  Current node depth.

- int **last_node**

  nodeId at last call.

- int **added_cuts**

number of new cuts added douring last call.

- int **glob_cuts**

  num of cuts added globally.

- int **total_node_cuts**

  total number of cuts applied to the current node.

- int **this_node**

  nodeId at this call.

Detailed Description

A CutCallbackI.

To make the code reusable – think about implementing a cut callback which adds two different cut families – we chose to put all the code outside method **main**() (p. 107).

So far, we merged different cut callbacks just cutting and pasting source codes. A more mature implementation should be based upon multiple inheritance.

The documentation for this class was generated from the following files:

- **cutcb.h**
- **cutcb.cpp**

B.5 DatiNodo Class Reference

Node infos: size, depth etc.

`#include <nodedata.h>`

Public Methods

- **DatiNodo** ()
- ∼**DatiNodo** ()
- **DatiNodo** (DatiNodo &padre)

  This is not a copy constructor.

Public Attributes

- int **node_cons**

  Number of constraints in the node.

- int **node_id**

  Node identifier.

- int **depth**

  Node's depth in the branch tree.

- IloExpr **lhs**

  IloExpression representing the branch sequence that built the node.

- IloInt **rhs**

  Right hand side of the cut representing the branch sequence.

Detailed Description

Node infos: size, depth etc.

An object of this class is bound to every node in the branch callback; it stores mainly depth and size informations; there is also an IloExpression which can be used to record the branch sequence that leaded to this node.

Constructor & Destructor Documentation

DatiNodo::DatiNodo (DatiNodo & *padre*)

This is not a copy constructor.

This constructor is used to initialize the object related to a node starting from its father's one.

The documentation for this class was generated from the following files:

- **nodedata.h**
- **nodedata.cpp**

B.6 HeurCBI Class Reference

A HeuristicCallbackI.

`#include <callbacks.h>`

Public Methods

- void **main** ()
- IloCplex::CallbackI $*$ **duplicateCallback** () const

Detailed Description

A HeuristicCallbackI.

Here you can implement your heuristic.

The documentation for this class was generated from the following files:

- **callbacks.h**
- **callbacks.cpp**

B.7 IncumCBI Class Reference

An IncumbentCallbackI.

`#include <callbacks.h>`

Public Methods

- void **main** ()
- IloCplex::CallbackI * **duplicateCallback** () const

Detailed Description

An IncumbentCallbackI.

This one is called each time an integer feasible solution is found; you can accept or reject it, according to some problem specific (non-linear) criteria.

The documentation for this class was generated from the following files:

- **callbacks.h**
- **callbacks.cpp**

B.8 MIPCBI Class Reference

A MIPCallbackI.

`#include <callbacks.h>`

Public Methods

- void **main** ()
- IloCplex::CallbackI * **duplicateCallback** () const

Detailed Description

A MIPCallbackI.

We do not really know how to use this one.

The documentation for this class was generated from the following files:

- **callbacks.h**
- **callbacks.cpp**

B.9 NodeCBI Class Reference

A NodeCallbackI.

`#include <callbacks.h>`

Public Methods

- void **main** ()
- IloCplex::CallbackI * **duplicateCallback** () const

Detailed Description

A NodeCallbackI.

This is were the choice for the next node takes place.

In general, this class gives you access to the queue of pending nodes; we use it also to get a measure of how much of the tree has not yet been explored.

The documentation for this class was generated from the following files:

- **callbacks.h**
- **callbacks.cpp**

B.10 OutputLogs Struct Reference

A struct of ofstreams.

`#include <output.h>`

Public Methods

- **OutputLogs** ()

  Constructor.

- ∼**OutputLogs** ()
- void **flushAll** ()

  Flush all the files at once.

Public Attributes

- ofstream **solver**

  For the usual node log.

- ofstream **warnings**

  Info about recoverable problems.

- ofstream **errors**

  Not recoverable problems.

- ofstream **solutions**

  Solutions found douring the optimization.

- ofstream **perNode**

  Per node infos.

- ofstream **perCallback**

  Per callback infos.

Detailed Description

A struct of ofstreams.

These ofstreams are provided to send log infos to several different files

Constructor & Destructor Documentation

ILOSTLBEGIN OutputLogs::OutputLogs ()

Constructor.

Opens the files and writes a header each, in order to separate multiple tests output to the same directory.

The output directory is read from global **Arguments** (p. 99) $*$ args.

perNode and perCallback are initialized accordingly to the syntax of the plotting program used.

Member Function Documentation

void OutputLogs::flushAll ()

Flush all the files at once.

You should flush all the files no more often than once per node; use "
", and not "<< endl".

Member Data Documentation

ofstream OutputLogs::perCallback

Per callback infos.

We output to this stream gnuplot or scilab formatted data

ofstream OutputLogs::perNode

Per node infos.

We output to this stream gnuplot or scilab formatted data

The documentation for this struct was generated from the following files:

- **output.h**
- **output.cpp**

B.11 Pool Class Reference

Base class for pools.

`#include <pool.h>`

Public Methods

- virtual IloRangeArray **getViolatedCuts** (IloNumArray sol)=0

  Returns an IloRangeArray of cuts violated by solution sol.

- virtual void **addCut** (**BaseCut** *cut)=0

  Intserts **BaseCut** (p. 104) *cut into the pool.*

- virtual ~**Pool** ()

  Destructor (empty).

Detailed Description

Base class for pools.

This virtual class provides an interface for the most general pool: there are only two virtual methods, representing the actions of pushing cuts into the pool and getting the cuts violated by a solution. Added cuts' type must be **BaseCut** (p. 104).

Member Function Documentation

virtual void Pool::addCut (BaseCut * *cut*) `[pure virtual]`

Intserts **BaseCut** (p. 104) cut into the pool.

The argument is a pointer for efficiency reasons.

virtual IloRangeArray Pool::getViolatedCuts (IloNumArray *sol*) `[pure virtual]`

Returns an IloRangeArray of cuts violated by solution sol.

Parameters' types come from Concert Technology: the assumption is you obtained the solution via CutCallback::getValue() and are just going to add the cuts calling CutCallbackI::add() or CutCallbackI::addLocal() methods.

The documentation for this class was generated from the following files:

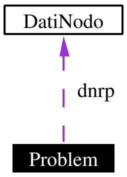
- **pool.h**
- **pool.cpp**

B.12 Problem Class Reference

Encapsulates and initializes objects which describe the model.

`#include <problem.h>`

Collaboration diagram for Problem:



Public Methods

- **Problem** ()

Public Attributes

- IloModel **mod**

  Model to be optimized.

- IloCplex **alg**

  Algorithm to be used.

- IloObjective **obj**

  Objective Function.

- IloNumVarArray **vars**

  Array of variables' handles.

- IloNumVarArray **int_vars**

  Integer variables' handles.

- IloNumVarArray **cont_vars**

  Continuous variables' handles.

- IloRangeArray **cons**

  Model's constraints.

- **DatiNodo** ∗ **dnrp**

  Root node's **VarData** (p. 131) *object It is stored here since there is no branch callback or similar to initialize root node.*

Protected Methods

- void **processVars** ()

  Initializes int_vars and cont_vars and constructs **VarData** (p. 131) *objects.*

- void **processCons** ()

  Clears up cons; checks that all constraints have one and only bound.

Detailed Description

Encapsulates and initializes objects which describe the model.

Loads the model from **Arguments::input_file** (p. 99), processes vars separating integer from continuous ones and provides a method to build a fake obj func for sat problems.

prob (see **globals.h**) is a global pointer to the one and only instance of this class.

The documentation for this class was generated from the following files:

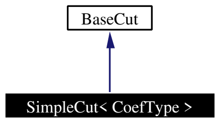
- **problem.h**
- **problem.cpp**

B.13 SimpleCut< CoefType > Class Template Reference

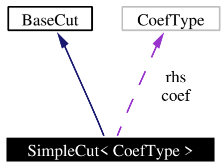
Linear cut template.

`#include <cut.h>`

Inheritance diagram for SimpleCut:



Collaboration diagram for SimpleCut< CoefType >:



Public Types

- enum **Sense_t** { **lessEq**, **equal**, **greaterEq** }

Public Methods

- **SimpleCut** (int _cnt, int ∗_ind, CoefType ∗_coef, Sense_t _sense, CoefType _rhs)
- **SimpleCut** (SimpleCut const &c)

  This class has value semantics.

- virtual ∼**SimpleCut** ()
- virtual SimpleCut & **operator=** (const SimpleCut &a)

  This class has value semantics.

- virtual IloRange **toIloRange** ()

  Conversion to IloRange.

- virtual bool **isViolated** (IloNumArray sol)

  Is this cut violated by solution sol?

- virtual void **print** (ostream &out)

  Cuts are printed in ".lp" format.

Static Public Methods

- void **setEpsFeas** (IloNum _eps_feas)

  To set the _eps_feas tolerance.

- void **setVars** (IloNumVarArray _vars)

  To initialize _vars.

Protected Attributes

- int **cnt**

  Number of non-zero coefficients in the lhs.

- int * **ind**

  Array of variables' indexes.

- CoefType * **coef**

  Array of coefficients' values.

- Sense_t **sense**

  Inequality's sense.

- CoefType **rhs**

  Right hand side.

- int **cut_id**

  Every SimpleCut is identified by this integer index.

- IloRange **cache**

  IloRange representation of the cut.

Static Protected Attributes

- int **num_cuts** = 0

  Total number of SimpleCut objects constructed so far.

- IloNum **eps_feas** = 0.0

Feasibility tolerance.

- IloNumVarArray **vars** = 0

  The array of variables, needed to convert the cut to IloRange.

Detailed Description

template<typename CoefType> class SimpleCut< CoefType >

Linear cut template.

This template models a linear constraint; coefficients' type is a parameter, not to slow down applications based on cuts with integer coefficients. Since the heavily object oriented implementation of IloRanges is too slow to use it in a pool based separator, this one is based on a more traditional array of non-zero coefficients matched by an array of column coefficients, both allocated in dynamic memory.

If you are going to derive from this class, pay attention to memory management.

Constructor & Destructor Documentation

template<typename CoefType> SimpleCut< CoefType >::SimpleCut (SimpleCut< CoefType > const & *c*)

This class has value semantics.

Please, note that IloRange has reference semantics instead.

Member Function Documentation

template<typename CoefType> bool SimpleCut< CoefType >::isViolated (IloNumArray *sol*) `[virtual]`

Is this cut violated by solution sol?

The check is based on a tolerance value stored in member eps_feas.

Implements **BaseCut** (p. 105).

template<typename CoefType> SimpleCut< CoefType > & SimpleCut< CoefType >::operator= (const SimpleCut< CoefType > & *a*) `[virtual]`

This class has value semantics.

Please, note that IloRange has reference semantics, instead.

template<typename CoefType> void SimpleCut< CoefType >::print (ostream & *out*) `[virtual]`

Cuts are printed in ".lp" format.

We always use this format to import models in Cplex.

Reimplemented from **BaseCut** (p. 104).

template<typename CoefType> IloRange SimpleCut< CoefType >::toIloRange () `[virtual]`

Conversion to IloRange.

We suggest you to implement a cache mechanism; it is easy, since IloRanges are handles.

Implements **BaseCut** (p. 105).

Member Data Documentation

template<typename CoefType> IloRange SimpleCut< CoefType >::cache `[protected]`

IloRange representation of the cut.

It is inizialized during the first execution of **toIloRange**() (p. 123), and deleted into the object destruction. In this way we build it if and only if it is usefull, and only once.

The documentation for this class was generated from the following file:

- **cut.h**

B.14 SolveCBI Class Reference

A SolveCallbackI.

`#include <callbacks.h>`

Public Methods

- void **main** ()
- IloCplex::CallbackI * **duplicateCallback** () const

Detailed Description

A SolveCallbackI.

We had lots of problems with this callback.

The documentation for this class was generated from the following files:

- **callbacks.h**
- **callbacks.cpp**

B.15 Statistics Struct Reference

A data structure of infos collected during execution.

`#include <statistics.h>`

Public Methods

- **Statistics** ()

  Assignes Initial values to vars.

- ostream & **print** (ostream &os) const

  Prints a complete report.

Public Attributes

- int **num_vars**

  num of vars.

- int **num_int_vars**

  num of integers vars.

- int **num_cont_vars**

  num of continuous vars.

- int **num_cons**

  num of constraints.

- int **initial_cuts**

  num of cons read from file.

- int **cplex_added_cuts**

  num of cuts added by Cplex MIP solver.

- int **added_cuts**

  num of cuts added by our callbacks.

- int **pool_access**

  num of pool queries.

- int **pool_hits**

  num of successfull pool queries.

- int **int_sol**
  num of integer solutions found.
- int **cb_calls**
  num of callback calls.
- int **num_nodes**
  num of nodes visited.
- int **num_nodes_left**
  num of nodes left.
- double **obj**
  objective function value.
- double **comb**
  fraction of the branch tree left to explore.
- double **best_bound**
  best bound.
- double **best_int_obj**
  best integer solution's cost.

Detailed Description

A data structure of infos collected during execution.

The documentation for this struct was generated from the following files:

- **statistics.h**
- **statistics.cpp**

B.16 Timers Class Reference

This class implements profiling utilities.

`#include <timers.h>`

Public Types

- enum **codeSection** { **pool_search**, **solve**, **cut_cb**, **solve_cb**, **branch_cb**, **node_cb**, **num_sections** }

  Has a member for each code section to monitor.

Public Methods

- **Timers** ()

  Just some asserts on initial conditions.

- void **start** (**codeSection** _start)

  Starts the specified timer.

- void **stop** (**codeSection** _stop)

  Stops the specified timer.

- void **stopStart** (**codeSection** _stop, **codeSection** _start)
- double **getTime** (**Timers::codeSection** clock)

  Gets time spent in specified codeSection.

- double **getBeCTime** ()

  Gets branch & cut time.

- ostream & **print** (ostream &) const

  Prints a profile of the process up to this point.

Static Public Methods

- double **totalTime** ()

  Gets total cpu time of the process.

Static Public Attributes

- char const *const **names** [ ]

  Names needed to print profile informations.

Protected Attributes

- double **acc_times** [num_sections]
- clock_t **last_start** [num_sections]

  Documented in file timer.cpp.

Detailed Description

This class implements profiling utilities.

As the name suggests, an object of this class is a set of clocks, a global one and one for each section of code you want to monitor.

Clocks must be manually started and stopped when the program flows leaves a code section to enter another one.

To add a clock, you have to add its name both to member enum codeSection and to member string array names.

For future developmens: maybe this class' design could be more object oriented.

It is not based on class IloTimer: it seemed broken.

Member Enumeration Documentation

enum Timers::codeSection

Has a member for each code section to monitor.

Add your members before the last one (num_sections), since it is used to count the number of sections.

Enumeration values:

pool_search Time spent looking for violated cuts in the pool.

solve Time spent outside callbacks.

cut_cb Time spent in CutCallbackI's function methods.

solve_cb Time spent solving lp relaxations.

branch_cb Time spent in BranchCallbackI's function methods.

node_cb Time spent in NodeCallbackI's function methods.

num_sections This is not a timer!! It serves only as a counter.

Add your code sections before this one, and remember to add a proper element to string array **Timers::names** (p. 129).

Member Function Documentation

double Timers::getTime (Timers::codeSection *clock*)

Gets time spent in specified codeSection.

You can query active timers without consequences.

ostream & Timers::print (ostream &) const

Prints a profile of the process up to this point.

It can be called at any moment, without worrying: no clock is stopped.

void Timers::start (codeSection *_start*)

Starts the specified timer.

See method **Timers::stop** (p. 129)

void Timers::stop (codeSection *_stop*)

Stops the specified timer.

When a timer is stopped, elapsed time, in seconds, is accumulated in a double.

void Timers::stopStart (codeSection *_stop*, codeSection *_start*) `[inline]`

Parameters:

_start == stop(_stop); start(_start). This is very usefull if you are profiling a program made of a main code section periodically interrupted by other sections

double Timers::totalTime () `[static]`

Gets total cpu time of the process.

Implemented by means of libc's getrusage(). This assures that time spent building the model, preprocessing it,etc is taken into account.

Member Data Documentation

char const *const Timers::names `[static]`

Initial value:

```
{
        "Pool_Search",
        "Solve",
        "CutCB",
        "SolveCB",
        "BranchCB",
        "NodeCB",
}
```

Names needed to print profile informations.

Please note that names and code sections are matched if and only if this array and enum codeSection are ordered the same way.

The documentation for this class was generated from the following files:

- **timers.h**
- **timers.cpp**

B.17 VarData Class Reference

An instance of this data structure is bound to each variable.

`#include <problem.h>`

Public Methods

- **VarData** (int _index, int _int_index, int _cont_index=-1)

Public Attributes

- int **index**

  Variable's index in **Problem::vars** (p. 118)*.*

- int **int_index**

  Variable's index in **Problem::int_vars** (p. 118)*; -1 otherwise.*

- int **cont_index**

  Variable's index in **Problem::cont_vars** (p. 118)*; -1 otherwise.*

Detailed Description

An instance of this data structure is bound to each variable.

There are only index informations so far, to deal with C orinted linear expressions.

The documentation for this class was generated from the following file:

- **problem.h**

Bibliografia

[1] M. Fischetti, "Lezioni di Ricerca Operativa", Edizioni Libreria Progetto Padova, 1995.

[2] R. E. Bixby, M. Fenelon, Z. Gu, E. Rothberg, R. Wunderling, "MIP: Theory and practice – closing the gap" rivista?

[3] J. F. Benders [1962], "Partitioning procedures for solving mixed variables programming problems", Numerische Mathematik, 4 , 238–255 volume, mese?

[4] P. Serafini, Ottimizzazione, Zanichelli editore, 453–456

[5] G. L. Nemhauser, L. A. Wolsey, Integer and Combinatorial Optimization, Wiley Interscience publication, 1988

[6] G. W. Klau, P. Mutzel, Optimal Labelling of Point Features in Rectangular Labelling Models, rivista?, nome Klau?

[7] G. W. Klau, A combinatorial Approach to Orthogonal Placement Problems, Shaker Verlag Aachen, 2002

[8] P. A. Rubin, Solving Mixed Integer Classification Problem by Decomposition, Annals of Operations Research, 1997 mese?

[9] A. Giuseppe, Nuove procedure di separazione per la programmazione lineare intera, 2003 riferimento giusto?

[10] Resolution Search, Chvatal sistemare questo riferimento