UNIVERSITÀ DEGLI STUDI DI MILANO

Facoltà di Scienze Matematiche, Fisiche e Naturali

Corso di Laurea in Informatica

Algoritmi Branch & Bound e Branch & Price per il problema delle p-mediane con capacità

Relatore: Prof. Giovanni RIGHINI

Tesi di Laurea di:
Alberto CESELLI
Matr. 566538

Anno Accademico 2000-2001

Indice

Capitolo 1

Il problema delle p-mediane con capacità

1.1 Introduzione

La logistica della distribuzione di punti di servizio su larga scala ha acquistato, nel corso degli anni, interesse crescente. Data la natura combinatoria di molti problemi reali, la loro risoluzione richiede tempi di calcolo proibitivi. Ad esempio, problemi di *network design* particolarmente complessi prevedono diverse fasi di sviluppo: il *partizionamento* dell'insieme degli utenti (ad esempio clients su rete di telecomunicazione) in clusters, la *localizzazione* delle risorse (i servers) all'interno dei clusters così descritti e la progettazione del sistema di interconnessioni tra utenti e risorse.

La formulazione del problema di partizionare un insieme di entità in gruppi *omogenei*, basandosi sulle *differenze* fra le entità associate ad ogni sottoinsieme è stato a lungo studiato; le sue applicazioni coinvolgono, oltre che ai problemi di network design descritti, situazioni eterogenee come la caratterizzazione di aree geografiche e pattern classification.

Il primo modello per problemi di localizzazione è stato proposto da Alfred Weber nel 1909; a lui si deve soprattutto l'introduzione del paradigma di localizzazione basata sulla minimizzazione dei costi di trasporto.

Nei problemi multirisorse, tuttavia, problemi di localizzazione e cluste-

ring interagiscono tra loro e devono essere, pertanto, affrontati *contemporaneamente*.

Tra i problemi di localizzazione multirisorse, il problema delle p-mediane, oltre ad essere interessante dal punto di vista teorico, trova applicazioni negli scenari di network design appena descritti. Metodi basati su programmazione mista intera hanno dimostrato di essere efficaci in tale ambito; il recente sviluppo tecnologico dei MIP solvers ha favorito, inoltre, l'applicazione di tecniche di programmazione matematica basate sui rilassamenti lineari, come la *column generation*.

Vengono proposti, in questo lavoro, algoritmi per la soluzione *esatta* di un problema di localizzazione multirisorse su grafo: il problema delle p-mediane con capacità (CPMP). In particolare, è presentato un confronto tra un algoritmo di Branch & Bound basato su un rilassamento Lagrangeano ed un algoritmo di Branch & Price.

Viene presentata di seguito una breve panoramica dei problemi di localizzazione multirisorse, per individuare CPMP in tale contesto.

Capacitated Facility Location Problem

Nella famiglia dei problemi di localizzazione multirisorsa hanno grande importanza i problemi di *capacitated facility location* (CFLP), che modellano, ad esempio problemi di distribuzione di servizi di emergenza sul territorio. Su un grafo avente N nodi vengono individuati M potenziali siti candidati all'apertura di facilities (i punti di servizio). Ogni nodo del grafo ha una domanda che deve essere soddisfatta ed ogni facility ha una capacità; l'obiettivo è soddisfare la domanda di ogni nodo, aprendo delle facilities in alcuni degli M siti candidati ed assegnando i nodi alle varie facilities, senza eccederne la capacità. La domanda di ogni nodo può essere soddisfatta utilizzando più facilities. Si vuole, inoltre, minimizzare il costo dell'operazione, dato dai costi di trasporto (proporzionali alla distanza del nodo dalle facilities a cui è

assegnato) e dai costi fissi per l'apertura di ogni facility.
La letteratura inerente i problemi di CFLP è molto ricca; per la soluzione esatta sono stati proposti diversi algoritmi in cui il bound duale è valutato tramite rilassamento lineare [Erle82] e rilassamento Lagrangeano [VRoy86] [Chri83], o anche tramite scomposizione di Benders [Geof74] [VRoy86]. Possono essere risolti all'ottimo, in tal modo, problemi che coinvolgono fino a 50 utenti e 50 siti candidati ad ospitare facilities. Per la risoluzione approssimata si dimostrano molto efficaci algoritmi euristici *greedy* combinati con *scambi miglioranti* [Kueh63] [Lehr66], oppure algoritmi basati su euristiche Lagrangeane [Corn91], che sono in grado di risolvere problemi su grafi di centinaia di nodi.

Per un'analisi completa dei metodi utilizzati per la risoluzione euristica o esatta del problema si rimanda all'articolo di Sridharan [Srid95] ed al testo [FL].

Single source - capacitated facility location problem

Per l'applicazione a problemi di network design è diffuso l'utilizzo di una variante di CFLP: il problema di *capacitated facility location* con *single source constraints* (CFLP-SS), conosciuto anche come problema di *capacitated concentrators location* (CCLP), ovvero un problema di CFLP in cui ogni utente deve essere servito da una sola facility (come nel caso di CPMP).

Neebe e Rao [Neeb83] studiano il problema di CFLP-SS. Viene presentato un algoritmo esatto che ricorre ad una riformulazione di CFLP-SS come problema di *set partitioning* con *side constraints* per sfruttare tecniche di column generation, risolvendo problemi su grafi di 35 nodi aventi fino a 25 possibili facilities, in alcuni secondi. Klincewicz e Luss [Klin86] realizzano un algoritmo esatto con rilassamento Lagrangeano dei vincoli di capacità per lo stesso problema, risolvendo in pochi minuti istanze su grafi di 50 nodi.

Pirkul [Pirk87] propone un argoritmo basato su un rilassamento Lagrangeano per la soluzione esatta del problema, risolvendo all'ottimo problemi su grafi di 100 nodi in pochi minuti. Pirkul analizza anche la possibilità di ot-

tenere soluzioni approssimate con un algoritmo di branch & bound troncato.

p-median problem

Il problema delle p-mediane (PMP) consiste nel partizionare un insieme $\mathcal{N} = \{1 \dots N\}$ di nodi di un grafo in p clusters disgiunti, minimizzando le *differenze* all'interno di ciascun cluster. Il partizionamento è equivalente alla scelta di p nodi (le *mediane*) da un insieme di candidati $\mathcal{M} = \{1 \dots M\}$, $\mathcal{M} \subseteq \mathcal{N}$, che identifichino i clusters. Le differenze tra i nodi all'interno dei clusters sono calcolate come la somma pesata delle distanze (intese come cammino minimo) tra ogni nodo e la mediana del cluster in cui è inserito.

Questo problema, definito per la prima volta da Hakimi (1963), è stato ampiamente studiato all'interno della famiglia dei problemi localizzazione multirisorsa; l'articolo [Hans97] offre una panoramica sui problemi di clustering, nel volume [DLT] (capitolo 2) vengono analizzati nel dettaglio il problema delle p-mediane classico e le sue generalizzazioni. Algoritmi basati su rilassamento Lagrangeano sono stati proposti in [Naru76], [Corn77], [Chri81], [Beas85]. Sono presentati risultati per la soluzione di problemi aventi grafi fino a 200 nodi, con 5 mediane e 150 nodi ed un numero arbitrario di mediane, in alcuni minuti. Utilizzando macchine per il calcolo vettoriale sono stati risolti problemi aventi fino a 900 nodi e 90 mediane o 600 nodi e 120 mediane in alcune decine di minuti. In [Galv79] e [Hanj85] sono presentati approcci basati sulla formulazione duale del problema che risolvono problemi su grafi di 75 nodi ed un numero arbitrario di mediane, in poche decine di secondi.

I problemi PMP in cui il grafo è un albero possono essere risolti in tempo polinomiale, come proposto da Goldman [Gold71] (per $p = 1$) o tramite l'algoritmo di Kariv e Hakimi [Kari79].

capacitated p-median problem

PMP può essere generalizzato associando ad ogni mediana una capacità e ad ogni nodo un peso ed imponendo che la somma dei pesi dei nodi in

ogni cluster non possa eccedere la capacità della mediana relativa. Questa generalizzazione è conosciuta come problema delle p-mediane con capacità (CPMP).

Il problema delle p-mediane con capacità è conosciuto anche come *sum of stars problem* [Hans97]. Garey e Johnson [Gare79] ne hanno dimostrato l'appartenenza alla classe di problemi $\mathcal{NP}$-hard. CPMP si differenzia da CFLP per due aspetti: in CFLP può essere attivato un numero qualsiasi di facilities, l'apertura di una nuova facility comporta, come descritto, dei costi fissi; in CPMP il numero di facilities è assegnato (p) e l'apertura di una facility in un nodo non comporta penalizzazione con costi fissi. CFLP prevede, inoltre, che un utente possa essere servito da più facilities, CPMP impone che ogni utente si riferisca ad una sola facility (come nel caso di CCLP).

Per CPMP non sono stati presentati molti metodi: algoritmi euristici in grado di raggiungere soluzioni per problemi con 100 nodi e 20 mediane in poche decine di secondi sono stati presentati da Mulvey e Beck [Mulv84].

Maniezzo, Mingozzi e Baldacci [Mani98] propongono l'utilizzo di tecniche di programmazione evolutiva, Golden e Skiscim [Gold86] ricorrono a Simulated Annealing, Osman e Christofides [Osma94] sviluppano un algoritmo ibrido tra Simulated Annealing e Tabu Search. Questi algoritmi forniscono soluzioni sperimentalmente "vicine" all'ottimo su problemi aventi N = 100 p = 20 in meno di un'ora.

Baldacci, Hadjiconstantinou, Maniezzo e Mingozzi [Bald02] descrivono un algoritmo per la soluzione euristica di CPMP che permette la valutazione della massima distanza dall'ottimalità della soluzione trovata, presentando risultati in cui problemi su grafi aventi 100 nodi con 10 mediane vengono risolti in meno di un'ora. Analizzano anche il caso di problemi con vincoli aggiuntivi di *incompatibilità* tra nodi su problemi dalle dimensioni analoghe, la cui soluzione è raggiunta entro 2 ore.

Struttura dell'esposizione

Nei capitoli successivi vengono presentati e confrontati tre differenti approcci per la soluzione ottima di CPMP:

- la riformulazione di CPMP come problema di *assegnamento generalizzato* come proposto in [Ross77] e la sua soluzione con l'algoritmo branch & price di Savelsbergh [Save95]
- un algoritmo Branch & Bound per CPMP basato su rilassamento Lagrangeano, in grado di risolvere all'ottimo in meno di un'ora problemi su grafi aventi fino a 100 nodi e 10 mediane
- un algoritmo per CPMP Branch & Price, in grado di risolvere all'ottimo problemi sugli stessi grafi per valori di p *qualsiasi* in meno di un'ora.

Nel capitolo 2 è descritto il problema della ricerca di una soluzione ammissibile per CPMP: dopo aver dimostrato che il problema di decidere l'*esistenza* di una soluzione per CPMP è $\mathcal{NP}$-Completo viene presentato un algoritmo per la soluzione euristica. Il capitolo 3 affronta il problema della riformulazione di CPMP come assegnamento generalizzato. Nel capitolo 4 viene presentato un algoritmo esatto basato su rilassamento Lagrangeano e nel capitolo 5 viene descritto lo sviluppo di un algoritmo Branch & Price per la risoluzione ottima, con i relativi dettagli implementativi. Nel capitolo 6 sono raccolte delle brevi conclusioni.

1.2 Formulazione del problema

Dati:

1. Sia $\mathcal{N} = \{1 \dots N\}$ l'insieme dei nodi di un grafo $G = (\mathcal{N}, E)$ e $\mathcal{M} \subseteq \mathcal{N}, \mathcal{M} = \{1 \dots M\}$ l'insieme dei nodi candidati ad essere scelti come mediane.
2. Ad ogni nodo $i \in \mathcal{N}$ è associato un coefficiente $w_i \in \boldsymbol{R}_+$ che rappresenta la sua domanda.

3. Per i nodi candidati ad essere mediane è definito anche il coefficiente $Q_j \in \boldsymbol{R}_+$, che rappresenta la sua capacità.

4. Ogni coefficiente $d_{ij} \in \boldsymbol{R}_+$ $i, j \in \mathcal{N}$ esprime la *differenza* tra due nodi i e j del grafo ed è proporzionale alla lunghezza del cammino minimo tra di essi (t_{ij}) ed alla domanda del nodo i: $d_{ij} = t_{ij} w_i$.

5. Si assume che il grafo non contenga coefficienti di differenza negativi

   $$d_{ij} \geq 0 \ \forall i, j \in \mathcal{N}$$

6. L'insieme $\mathcal{N}$ dev'essere partizionato in p clusters, dove p è dato.

Variabili:

1. I clusters sono quindi p insiemi $S_j \subseteq \mathcal{N}$, con il vincolo che

   $$\bigcup_{j \in \mathcal{M}} S_j = \mathcal{N}, \ \ S_{j'} \cap S_{j''} = \emptyset \ \forall j', \ \ j'' \in \mathcal{M}.$$

2. Ogni cluster è individuato tramite il suo nodo *mediana* m_j, $j \in \mathcal{M}$.

3. La formulazione classica del problema contempla l'uso di $N*M$ variabili binarie x_{ij}.

4. In particolare $x_{ij} = 1$ se il nodo i viene assegnato alla mediana j e $x_{ij} = 0$ altrimenti ($x_{ij} = 1$ se e solo se $i \in S_j$).

5. M variabili binarie y_j, $j = 1 \ldots M$ indicano, per ogni nodo j, se j è selezionato come mediana o no.

6. Per attinenza a situazioni reali, alcuni autori [Bald02] suppongono che $d_{jj} = 0 \ \forall j \in \mathcal{M}$ e che ogni nodo mediana debba essere inserito nel cluster da esso descritto ($x_{jj} = 1$ se e solo se il nodo j è mediana). Questa assunzione può, tuttavia, portare a soluzioni ottime diverse o anche impedire l'esistenza di soluzioni ammissibili. Per questo il modello adottato negli algoritmi presentati nei capitoli seguenti si riferisce alla versione più generale in cui nodi mediana possono essere assegnati ad altri clusters ed i coefficienti d_{jj} possono essere non nulli.

Vincoli:

1. La cardinalità di ciascun cluster non è vincolata, ma la somma delle domande dei suoi nodi non può superare la capacità della mediana.
2. Ogni nodo deve essere servito da una sola mediana.
3. Le mediane devono essere p.

Funzione obiettivo:

- L'obiettivo è minimizzare la somma delle differenze tra ogni nodo e la mediana del cluster a cui è assegnato, nel rispetto dei vincoli descritti.

Modello:

Una formulazione del problema è quindi la seguente
CPMP)

$$min\ v = \sum_{i \in \mathcal{N}} \sum_{j \in \mathcal{M}} d_{ij} x_{ij} \tag{1.1}$$

s.t.

$$\sum_{j \in \mathcal{M}} x_{ij} = 1\ \forall i \in \mathcal{N} \tag{1.2}$$

$$\sum_{i \in \mathcal{N}} w_i x_{ij} \leq Q_j y_j\ \forall j \in \mathcal{M} \tag{1.3}$$

$$\sum_{j \in \mathcal{M}} y_j = p \tag{1.4}$$

$$x_{ij} \in \{0,1\}\ \forall i \in \mathcal{N},\ \ \forall j \in \mathcal{M} \tag{1.5}$$

$$y_j \in \{0,1\}\ \forall j \in \mathcal{M} \tag{1.6}$$

L'obiettivo (1.1) è minimizzare la somma delle distanze tra ciascun nodo e la mediana associata.

I vincoli di set partitioning (1.2) impongono che ogni nodo sia assegnato ad una mediana. I vincoli (1.3) hanno un doppio effetto: imporre che la

capacità di ciascuna mediana non possa esser inferiore alla somma dei pesi dei nodi ad essa assegnati ed impedire che dei nodi vengano associati a mediane non attive (per le quali $y_j = 0$).

Il vincolo (1.4) impone che le mediane attive siano esattamente p, mentre i vincoli (1.5 e 1.6) impongono l'integralità della soluzione.

Rilassamento continuo:

Nell'algoritmo presentato nel capitolo 5 si ricorre alla soluzione del rilassamento lineare di CPMP, riformulato come problema di set partitioning con side constraints per valutare bounds duali. Per ottenere la soluzione del rilassamento è opportuno disporre di una soluzione di base *ammissibile* con cui inizializzare il metodo del simplesso. Per poter individuare più facilmente soluzioni ammissibili per il rilassamento di CPMP è possibile

- rilassare i vincoli (1.2), portandoli in forma di $\geq$: all'ottimo, infatti, nessun nodo è assegnato a più mediane, poiché tale nodo potrebbe essere rimosso dal cluster più svantaggioso, generando una soluzione migliore;

- portare il vincolo (1.4) in forma di $\leq$: infatti non è mai vantaggioso utilizzare un minor numero di clusters, poiché ogni soluzione con meno di p clusters equivale ad una soluzione con p clusters, alcuni dei quali sono vuoti.

Inoltre, come descritto nel capitolo 2, la ricerca di una soluzione ammissibile per CPMP è un problema "difficile", ed ottenere rapidamente una soluzione *intera* ammissibile è utile per migliorare l'accuratezza del metodo del sottogradiente nell'algoritmo basato su rilassamento Lagrangeano (descritto nel capitolo 4) durante le prime iterazioni (infatti il *passo* nel metodo del sottogradiente è funzione dello scarto tra il bound primale ed il bound duale). La scelta di rilassare i vincoli di uguaglianza consente di mantenere la corrispondenza tra i modelli utilizzati per i due algoritmi.

1.3 Strumenti e dati utilizzati

Per valutare le prestazioni degli algoritmi proposti nei capitoli seguenti sono state considerate 60 diverse istanze di CPMP. Le prime 20, indicate come istanze di *classe A*, corrispondono ai problemi CCPX1 ... CCPX20, utilizzati in [Osma94] [Bald02]. I primi 10 problemi sono definiti su grafi di 50 nodi, di cui 5 debbano essere designati mediane; nei rimanenti 10 problemi i grafi hanno 100 nodi con 10 mediane disponibili. In entrambi i casi $p_A = \frac{N}{10}$.

Sono state generate, inoltre, 40 nuove istanze dalle 20 di partenza, ponendo $p_B = \lfloor \frac{N}{4} \rfloor$ (problemi della *classe B*), e $p_C = \frac{2N}{5}$ (problemi della *classe C*), e modificando in corrispondenza la capacità delle mediane:

$$Q_j^B = \frac{2}{5} Q_j^A$$

$$Q_j^C = \frac{1}{4} Q_j^A$$

Alcuni autori [Chri81] rilevano che il problema delle p-mediane è più arduo se $p \simeq \frac{N}{3}$; quindi è stato creato un nuovo insieme di istanze (classe D), ponendo

$$p_D = \lfloor \frac{N}{3} \rfloor$$

$$Q_j^D = \lceil \frac{3}{10} Q_j^A \rceil$$

Entrambi gli algoritmi esatti sono stati sviluppati secondo il paradigma della programmazione orientata agli oggetti. I test sono stati condotti su PC con processore Pentium II a 350MHz, dotato di 128MB di RAM, sistema operativo Linux (distribuzione RedHat 6.1, kernel 2.2). Il codice è stato scritto in C++ e compilato con gcc/g++ versione 2.91, con impostazioni di default.

Utilizzo di un MIP-solver

Nella tabella 1.1 sono riportati i risultati ottenuti affidando la soluzione del problema ad un *MIP Solver* (ILOG CPLEX versione 6.6). E' indicato il tempo di cpu (in secondi) impiegato per la soluzione ottima. E' stato posto, come per tutti gli esperimenti successivi, un limite di 3600 secondi di utilizzo della cpu per terminare il calcolo.

CPLEX è in grado di risolvere in alcuni minuti istanze definite su grafi

grafo di cardinalità 50

Problema	Classe A	Classe B	Classe C
CCPX1	18.8	16.8	4.0
CCPX2	1.9	10.1	40.0
CCPX3	33.4	53.6	20.5
CCPX4	6.4	2248.9	59.7
CCPX5	10.5	226.8	85.0
CCPX6	34.9	357.4	46.8
CCPX7	77.3	1194.7	65.6
CCPX8	798.8	143.4	332.9
CCPX9	63.4	723.0	-
CCPX10	366.8	-	-

grafo di cardinalità 100

Problema	Classe A	Classe B	Classe C
CCPX11	-	-	1394.1
CCPX12	1780.1	310.0	-
CCPX13	514.3	-	365.9
CCPX14	-	1266.2	-
CCPX15	-	1208.5	-
CCPX16	-	-	2218.4
CCPX17	-	710.2	2873.4
CCPX18	-	445.0	-
CCPX19	-	-	2654.2
CCPX20	-	-	-

Tabella 1.1: Prestazioni del MIP solver

di cardinalità 50, mentre fallisce nel 60% dei problemi su grafi di dimensione maggiore. La scelta di p influenza il tempo necessario per ottenere la soluzione.

1.4 Algoritmi Branch & Bound

Come affermato nel capitolo 2, il problema delle p-mediane con capacità è $\mathcal{NP}$-Hard. Per ottenere la soluzione ottima è necessario ricorre all'enumerazione di tutte le possibili soluzioni. Questa enumerazione viene effettuata in maniera implicita ricorrendo ad algoritmi di *branching*, ovvero esplorando un albero in cui ogni nodo corrisponde ad un sottoproblema. La radice è il problema iniziale, in cui tutte le variabili sono libere. L'insieme di soluzioni corrispondenti (S_0) può essere partizionato in F sottoinsiemi S_k, che rappresentano le soluzioni di F *sottoproblemi* ($S_1 \dots S_F$, $\bigcup_{k=1\dots F} S_k = S_0$), fissando opportunamente il valore di una o più variabili libere. Ogni sottoproblema è radice di un nuovo albero; ai suoi figli corrispondono F sottoinsiemi

dell'insieme di soluzioni S_k. Le foglie dell'albero di branching rappresentano tutte le possibili soluzioni del problema.

Ricorrendo ad algoritmi euristici è possibile sperare di ottenere soluzioni ammissibili partendo dalle soluzioni parziali definite in ciascun sottoproblema. Il valore delle soluzioni ammissibili eventualmente trovate è un *bound primale*.

Per ogni sottoproblema viene valutato il valore di una soluzione ammissibile duale, sicuramente non peggiore di qualsiasi sua soluzione ammissibile primale. Questo valore viene detto *bound duale*.

Se in un sottoproblema il bound duale non risulta migliore del bound primale, la valutazione del nodo dell'albero di branching corrispondente e di tutti i suoi nodi-figlio può essere tralasciata.

Nei capitoli 4 e 5 vengono descritti due algoritmi che utilizzano la tecnica descritta per risolvere all'ottimo CPMP: nel primo il bound duale viene calcolato tramite rilassamento Lagrangeano, nel secondo viene utilizzato il rilassamento lineare di CPMP riformulato come problema di set partitioning; in entrambi soluzioni ammissibili sono raggiunte utilizzando una versione modificata dell'euristica di Martello e Toth per il problema dell'assegnamento generalizzato [MT89], descritta nel capitolo 2.

Capitolo 2

Euristica primale

2.1 Il problema di esistenza di una soluzione

Anche la soluzione euristica di CPMP è un problema "difficile", infatti se $\mathcal{P} \neq \mathcal{NP}$ non esistono per CPMP algoritmi che, in tempo polinomiale, garantiscano di trovare una soluzione ammissibile. La dimostrazione è per riduzione da PARTITION.

Sia $\mathcal{N} = \{1 \dots n\}$. Dato un vettore di n interi positivi $\boldsymbol{w^p} = (w_1^p \dots w_n^p)$, il problema PARTITION, ovvero il problema di determinare se

$$\exists S \subseteq \mathcal{N} \mid \sum_{i \in S} w_i^p = \sum_{i \in \mathcal{N} \setminus S} w_i^p$$

è $\mathcal{NP}$-Completo.
La dimostrazione si deve a Karp (1972).

Il problema di determinare l'esistenza di una soluzione per un'istanza di CPMP (ECPMP) si può formulare come segue:

sia $\boldsymbol{w^c} = (w_1^c \dots w_n^c), \boldsymbol{Q} = (Q_1 \dots Q_m)$

un'istanza di CPMP su un grafo $G = (\mathcal{N}, E)$, $|\mathcal{N}| = n$, il cui insieme di possibili mediane è $\mathcal{M}$, $|\mathcal{M}| = m$ (i coefficienti $[d_{ij}]$ non sono rilevanti ai fini

della riduzione). ECPMP è il problema di determinare se esiste una scelta delle mediane $\mathcal{M}^* = \{j_1^* \dots j_p^*\} \subseteq \mathcal{M}$ tale per cui

$$\exists S_{j_1^*} \dots S_{j_p^*} \mid \bigcup_{j_k^* \in \mathcal{M}^*} S_{j_k^*} = \mathcal{N}, \sum_{i \in S_j} w_i^c \leq Q_j \ \forall j \in \mathcal{M}^*$$

Sia 2-ECMP una *specializzazione* di ECPMP in cui $p = 2$ e $\mathcal{M}^* = \{1,\ 2\}$: chiaramente 2-ECMP $\preceq_m^p$ ECPMP.

Proposizione: 2-ECMP è $\mathcal{NP}$-Completo.
Dimostrazione: Sia $\boldsymbol{w^p}$ istanza di PARTITION e sia $\boldsymbol{w^{2c}}$, $\boldsymbol{Q} = (Q_1,\ Q_2)$ un'istanza di 2-ECMP. Ponendo

$$\boldsymbol{w^{2c}} = \boldsymbol{w^p},\ Q_1 = Q_2 = \frac{1}{2} \sum_{i=1}^{n} w_i^{2c}$$

l'esito di 2-ECMP è positivo se e solo se

$$\exists S_1,\ S_2 \mid S_1 \cup S_2 = \mathcal{N},\ \sum_{i \in S_1} w_i^{2c} \leq Q_1 \text{ e } \sum_{i \in S_2} w_i^{2c} \leq Q_2$$

Poiché $S_1 \cup S_2 = \mathcal{N}$, e quindi $S_2 = \mathcal{N} \setminus S_1$, ciò è equivalente a determinare se

$$\exists S_1 \mid \sum_{i \in S_1} w_i = \sum_{i \in \mathcal{N} \setminus S_1} w_i^{2c} = \frac{1}{2} \sum_{i=1}^{n} w_i^{2c} = \frac{1}{2} \sum_{i=1}^{n} w_i^p$$

da cui si deduce che PARTITION $\preceq_m^p$ 2-ECMP.

Poiché PARTITION $\preceq_m^p$ 2-ECMP $\preceq_m^p$ ECPMP allora ECPMP è $\mathcal{NP}$-Completo. □

La versione di ricerca di un problema di esistenza $\mathcal{NP}$-Completo è almeno difficile quanto quest'ultimo [ICC] [Gare79].

2.2 L'euristica di Martello e Toth modificata

Come affermato nella sezione precedente, se $\mathcal{P} \neq \mathcal{NP}$ non è possibile avere la garanzia di trovare una soluzione ammissibile per CPMP in tempo polinomiale. Per ottenere rapidamente soluzioni primali sperabilmente ammissibili è possibile ricorrere ad una versione modificata dell'euristica MTH di Martello e Toth (1981) [MT89] per il problema dell'assegnamento generalizzato già utilizzata da diversi autori, come ad esempio Savelbergh [Save95] (il problema dell'assegnamento generalizzato è descritto nel capitolo 3).
Le eventuali soluzioni individuate possono essere utilizzate come bounds primali da algoritmi di enumerazione implicita per raggiungere più rapidamente l'ottimo del problema.

Definiti i coefficienti f_{ij} come misura della desiderabilità dell'assegnamento del compito i alla macchina j, MTH prevede due fasi.

- Viene effettuato l'assegnamento di compiti alle macchine secondo i coefficienti di desiderabilità descritti, nel rispetto dei vincoli di capacità. In particolare si assegnano prima compiti che abbiano maggior differenza tra il coefficiente $f_{ij'}$ riferito alla macchina più desiderabile, per la quale non vengano violati i vincoli di capacità ed il coefficiente $f_{ij''}$, riferito alla seconda macchina più desiderabile nel rispetto dei vincoli di capacità.

- Se è possibile trovare una soluzione ammissibile in tal modo, questa può essere migliorata attraverso ricerca locale, in caso contrario l'algoritmo non è in grado di individuare una soluzione.

Per una descrizione formale dell'algoritmo si rimanda a [MT89]. La scelta dei coefficienti f_{ij} determina le prestazioni dell'euristica. E' possibile adattare MTH al problema delle p-mediane con capacità, procedendo per fasi successive.

Fase 1: Scelta delle mediane

Se esiste una soluzione ammissibile per un'istanza di CPMP in cui le capacità dei nodi non sono uniformi, esiste sicuramente una soluzione ammissibile in cui siano mediane i p nodi più capaci. E' possibile, in tal caso, scegliere le mediane in base alla loro capacità. Nelle istanze proposte in letteratura, tuttavia, ogni potenziale mediana ha la stessa capacità. Si può quindi optare per due alternative:

- scegliere le mediane casualmente (mediante estrazione da distribuzione uniforme);
- definire

  $$\psi_j = \sum_{i=1}^{N} f_{ij}$$

  ricavando una distribuzione di probabilità operando opportunamente su questi coefficienti ed estraendo casualmente rispetto ad essa (o semplicemente scegliendo i p nodi più promettenti). In tal modo è possibile iterare l'applicazione dell'euristica per diverse scelte delle mediane.

Fase 2: Allocazione (costruzione di una soluzione parziale)

Nella fase di allocazione dell'euristica proposta

- $\mathcal{N}_W$ rappresenta l'insieme dei nodi non inseriti in nessun cluster
- $\mathcal{M}^i$ è l'insieme dei clusters in cui può essere inserito il nodo i senza che vengano violati i vincoli di capacità
- i coefficienti q_j^r rappresentano le capacità residue di ogni mediana
- i coefficienti D_i rappresentano le *penalità* per non assegnare il nodo i alla mediana più desiderabile
- C^j è l'insieme dei nodi i assegnati alla mediana j (il cluster riferito alla mediana j)

Scelte le mediane, il problema si riduce ad un assegnamento generalizzato: è possibile costruire una soluzione come nell'euristica MTH, proposta da Martello e Toth (Figura 2.1).
La fase di allocazione può essere effettuata utilizzando, come descritto in

```
forall j ∈ M do q_j^r := Q_j

N_W := N
C^1 := C^2 := ... := C^M := ∅

while N_W ≠ ∅ do
      forall i ∈ N_W do
              M^i = {j | q_j^r ≥ w_i}
              if M^i = ∅ then FAIL (endif)
              j'_i := argmax_{j∈M^i}{f_ij}
              if M^i = {j'_i} then D_i := +∞
              else
                    j''_i := argmax_{j∈M^i, j≠j'_i}{f_ij}
                    D_i := f_ij' − f_ij''
              (endif)
      (done)

      i* := argmax_{i∈N_W}{D_i}
      j* := j'_{i*}
      C^{j*} = C^{j*} ∪ {i*}
      N_W := N_W \ {i*}
      q^r_{j*} := q^r_{j*} − w_{i*}
(done)
```

Figura 2.1: Allocazione

[MT89], strutture dati opportune, in tempo $O(N^2 + NM \log M)$
Per alcune istanze di CPMP è difficile raggiungere soluzioni ammissibili in questo modo.

Fase 3: Minimizzazione dell'inammissibilità attraverso ricerca locale

Se la soluzione ottenuta al passo precedente è inammissibile, l'euristica di Martello Toth si arresta. Nella versione modificata si è scelto, invece, di ricorrere a *1-scambi* come descritto di seguito.

In questa fase di minimizzazione dell'inammissibilità

- $l(i, j)$ è la minima capacità residua, maggiore del peso dell'elemento i, ottenibile dal cluster j rimuovendo un solo elemento ad esso associato; potrebbe non essere possibile determinare tale valore
- $k(i, j)$ è l'elemento (nel caso in cui esista) che deve essere rimosso dal cluster j per ottenere la capacità $l(i, j)$
- per ogni elemento i, L_i è l'insieme delle mediane per cui esiste l'elemento $k(i, j)$.

lo pseudocodice relativo a tale fase è riportato in figura 2.2

Un simile algoritmo di ricerca locale non fornisce alcuna garanzia: si compiono un numero finito Γ di iterazioni, nella speranza di ottenere una soluzione ammissibile. Ogni iterazione termina, nel caso peggiore, in tempo $O(N^2M)$.

```
for Γ iterations do
    forall i ∈ N_W do

            L_i = {j | ∃k ∈ C^j | q^r_j + w_k ≥ w_i,  w_k < w_i,  j ∈ M}
            forall j ∈ L_i do
                    l(i,j) := min_{k∈C^j}{(q^r_j + w_k) | q^r_j + w_k ≥ w_i,  w_k < w_i}
                    k(i,j) := argmin{l(i,j)}
            (done)

    (done)

    if ⋃_{i∈N_W} L_i ≠ ∅ then

            (i*,  j*) := argmin_{i∈N_W, j∈L_i}{l(i,j) − w_i}
            C^{j*} := C^{j*} \ {k(i*,j*)}; C^{j*} := C^{j*} ∪ {i*}
            q^r_{j*} := l(i*,j*) − w_{i*}
            N_W := N_W \ {i*}

            if ∃j ∈ M | q^r_j ≥ w_{k(i*,j*)} then
                    C^j := C^j ∪ {k(i*,j*)}
                    q^r_j := q^r_j − w_{k(i*,j*)}
            else N_W := N_W ∪ {k(i*,j*)}
            (endif)

    else FAIL (endif)
(done)
```

Figura 2.2: Minimizzazione dell'inammissibilità

Fase 4: Miglioramento della soluzione attraverso ricerca locale

Se la soluzione prodotta ai passi precedenti è ammissibile, si possono effettuare, come nell'euristica MTH, scambi miglioranti (figura 2.3). In questa fase la soluzione è definita dalle variabili s_i: $s_i = j$ se il nodo i è stato assegnato alla mediana j nelle fasi precedenti. ($s_i = j$ se e solo se $i \in C^j$).

```
if N_W = ∅ then
 forall i ∈ N do

        L_i := {j ∈ M | q^r_j ≥ w_i}
        j* := argmin_{j∈L_i, j≠s_i} {d_ij}

        if d_ij* ≤ d_is_i then

            q^r_{s_i} := q^r_{s_i} + w_i
            q^r_{j*} := q^r_{j*} − w_i
            C^{s_i} := C^{s_i} \ {i}; C^{j*} := C^{j*} ∪ {i}
            s_i := j*

        (endif)
 (done)
(endif)
```

Figura 2.3: Scambi miglioranti

La complessità della fase di miglioramento è $O(NM)$. La soluzione ottenuta può essere migliorata ulteriormente, ricalcolando la mediana di ciascun cluster della soluzione parziale (valutabile in maniera esatta in tempo polinomiale). L'implementazione dell'euristica realizzata non prevede la fase di riottimizzazione.

Capitolo 3

Riformulare come problema di assegnamento generalizzato

Il problema delle p-mediane con capacità è riducibile in tempo polinomiale al problema dell'*assegnamento generalizzato* (GAP). GAP è molto studiato sia per la struttura semplice e per l'importanza teorica, sia perché costituisce un buon modello per problemi di allocazione di processi a macchine per il calcolo.

Ogni istanza del problema dell'assegnamento generalizzato è definita su di un grafo $G = ((\mathcal{N}^G,\ \mathcal{M}^G),\ E)$, in cui $\mathcal{N}^G$ è un insieme di compiti da svolgere e $\mathcal{M}^G$ l'insieme delle macchine in grado di portare a termine i compiti. Ogni compito $i \in \mathcal{N}^G$ deve essere assegnato ad una macchina $j \in \mathcal{M}^G$; il tempo macchina necessario per portare a termine ogni compito $i \in \mathcal{N}^G$ è descritto da un vettore $\boldsymbol{w_i^G}$, il costo necessario per svolgere il compito i è descritto da vettore $\boldsymbol{d_i^G}$. Tempi e costi dipendono dalla macchina $j \in \mathcal{M}^G$ a cui viene assegnato ogni compito: $w_{ij}^G \in \boldsymbol{R}_+$ e $d_{ij}^G \in \boldsymbol{R}_+$ sono rispettivamente il tempo ed il costo necessario alla macchina j per terminare il compito i. Ogni macchina $j \in \mathcal{M}^G$ ha una capacità $Q_j^G \in \boldsymbol{R}_+$, che rappresenta il tempo disponibile per la gestione dei compiti. La somma dei tempi dei compiti associati ad una macchina non ne può eccedere la capacità. Le macchine, in generale, sono diverse tra loro.

GAP può essere formulato come
GAP)

$$min\ v_{gap} = \sum_{i \in \mathcal{N}^G} \sum_{j \in \mathcal{M}^G} d_{ij}^G x_{ij}^G \tag{3.1}$$

$s.t.$

$$\sum_{j \in \mathcal{M}^G} x_{ij}^G = 1\ \forall i \in \mathcal{N}^G \tag{3.2}$$

$$\sum_{i \in \mathcal{N}^G} w_{ij}^G x_{ij}^G \leq Q_j^G\ \forall j \in \mathcal{M}^G \tag{3.3}$$

$$x_{ij}^G \in \{0,1\}\ \forall i \in \mathcal{N}^G,\ \forall j \in \mathcal{M}^G \tag{3.4}$$

Ogni variabile $x_{ij}^G = 1$ se e solo se il compito i è assegnato alla macchina j. L'obiettivo è minimizzare il tempo macchina totale utilizzato (3.1), assegnando ciascun compito ad una macchina (3.2), nel rispetto dei vincoli di capacità (3.4).
La similitudine con CPMP è chiara se si considera l'assegnamento di compiti alle macchine come il partizionamento dell'insieme $\mathcal{N}^G$ in $|\mathcal{M}^G|$ sottoinsiemi disgiunti (clusters), ognuno dei quali deve essere riferito ad un nodo macchina. La somma dei pesi (i tempi di calcolo) dei nodi di ogni cluster non può superare la capacità della macchina associata; l'obiettivo è minimizzare il costo dell'inserimento di ogni nodo in un cluster.
Ross e Soland [Ross77] propongono un algoritmo per la trasformazione di istanze di CPMP in istanze di GAP che richiede tempo lineare nella dimensione dell'istanza.
Data un'istanza $\boldsymbol{w} = (w_1 \dots w_N),\ [d_{ij}],\ \boldsymbol{Q} = (Q_1 \dots Q_M),\ p$ di CPMP su di un grafo $T = (\mathcal{N},\ E),\ |\mathcal{N}| = N$, per il quale sia $\mathcal{M},\ \mathcal{M} \subseteq \mathcal{N},\ |\mathcal{M}| = M$ l'insieme delle possibili mediane, la riduzione si ottiene costruendo un grafo $T^G = ((\mathcal{N}^G,\ \mathcal{M}^G),\ E)$ avente

$$|\mathcal{N}^G| = 2 * |\mathcal{N}|,\ \mathcal{N}^G = \{1 \dots 2N\}$$

$$|\mathcal{M}^G| = |\mathcal{M}| + 1,\ \mathcal{M}^G = \{1 \dots M+1\}$$

L'istanza di GAP costruita ha un numero di compiti pari al doppio della cardinalità del grafo di CPMP, ed un numero di macchine superiore di una unità alla cardinalità del grafo di CPMP. Inoltre, per completare la costruzione di un'istanza di GAP è necessario indicare il valore dei termini $[w_{ij}^G]$, $[d_{ij}^G]$, $\boldsymbol{Q^G} = (Q_1^G \dots Q_{M+1}^G)$.

- Per $i = 1 \dots N,\ j = 1 \dots M$ le variabili x_{ij}^G mantengono il loro significato: $x_{ij}^G = 1$ se e solo se il nodo i è assegnato alla mediana j. Si pone quindi

$$\begin{array}{lll} d_{ij}^G & = d_{ij} & \forall i = 1 \dots N,\ \forall j = 1 \dots M \\ w_{ij}^G & = w_i & \forall i = 1 \dots N,\ \forall j = 1 \dots M \\ Q_j^G & = Q_j & \forall j = 1 \dots M \end{array}$$

  I vincoli di set partitioning (3.2) sono analoghi ai vincoli (1.2), per i quali ogni nodo è assegnato ad una mediana.

- Fra le N^2 nuove variabili $x_{N+i,j}^G,\ i = 1 \dots N,\ j = 1 \dots M$, gli N termini $x_{N+i,i}^G$ rappresentano le variabili y_i della formulazione di CPMP *complementate*: $x_{N+i,i}^G = 0$ se e solo se il nodo i è scelto come mediana. Per imporre questa corrispondenza si sfruttano i vincoli di capacità: se

$$w_{N+i,i}^G = Q_i \ \forall i = 1 \dots N$$

  la condizione $x_{N+i^m,i^m}^G = 1$ implica che $x_{i,i^m}^G = 0\ \forall i \in \mathcal{N}^G$ perché l'i^m-esimo vincolo di capacità non venga violato (nessun nodo può essere assegnato a i^m).

- Le rimanenti variabili $x_{N+i,j}^G,\ i = 1 \dots N,\ j = 1 \dots M,\ i \neq j$ non hanno corrispondenza nell'istanza di CPMP e devono essere poste a 0, attraverso un'opportuna scelta dei pesi (o dei coefficienti):

$$\begin{array}{lll} w_{N+i,j}^G & = +\infty & \forall i = 1 \dots M,\ \forall j = 1 \dots M,\ i \neq j \\ d_{N+i,i}^G & = 0 & \forall i = 1 \dots N \end{array}$$

  considerare qualsiasi $x_{N+i,j}^G = 1$ fra quelle indicate viola i vincoli di capacità

- Ponendo $x^G_{N+i^m,i^m} = 0$, ovvero scegliendo il nodo i^m come mediana, per il vincolo di set partitioning (3.2)

  $$\sum_{j \in \mathcal{M}^G} x^G_{i^m,j} = 1$$

  e per come sono stati scelti i coefficienti $w^G_{N+i^m,j}$ $(i^m \neq j)$ deve necessariamente essere $x^G_{i^m,M+1} = 1$. Per ogni nodo scelto come mediana deve essere presente un "1" nella colonna $(M+1)$-esima. Si sfrutta l'ultimo vincolo di capacità

  $$\sum_{i \in \mathcal{N}^G} w^G_{i,M+1} x^G_{i,M+1} \leq Q^G_{M+1}$$

  per assicurare che le mediane siano al più p definendo

  $$\begin{array}{ll} w^G_{N+i,M+1} & = 1 \quad \forall i = 1 \ldots N \\ Q^G_{M+1} & = p \end{array}$$

  (la designazione di un nodo come mediana comporta l'utilizzo di un'unità delle p disponibili sulla macchina $M+1$) e

  $$w^G_{i,M+1} = +\infty \ \forall i = 1 \ldots N$$

  (nessun nodo è assegnato alla macchina $M+1$, la cui capacità rappresenta il vincolo sul numero di mediane).

- Per completare la definizione dell'istanza di GAP si scelgono arbitrariamente i rimanenti coefficienti, per esempio

  $$\begin{array}{lll} d^G_{i,M+1} & = +\infty & \forall i = 1 \ldots N \\ d^G_{N+i,M+1} & = 0 & \forall i = 1 \ldots N \\ d^G_{N+i,j} & = +\infty & \forall i = 1 \ldots M, \ \forall j = 1 \ldots M, \ i \neq j \end{array}$$

Dalla soluzione ottima $[x^{G*}_{ij}]$ per una simile istanza di GAP, si ottiene la soluzione ottima $[x^{c*}_{ij}], \boldsymbol{y^c}$ per CPMP ponendo

$$x^c_{ij} = x^G_{ij} \ \forall i = 1 \ldots N, \ \forall j = 1 \ldots M$$

$$y^c_j = (1 - x^G_{N+j,j}) \ \forall j = 1 \ldots M$$

3.1 Algoritmo di Savelsbergh

Anche per la soluzione esatta di GAP sono stati sviluppati diversi algoritmi (Ross e Soland [Ross75], Martello e Toth [MT89], Fisher [Fish86], Jörnsten e Näsberg [Jorn86], Guignard e Rosenwein [Guig89]. In particolare, Savelsbergh [Save95] ha proposto un algoritmo molto efficiente basato su *branch & price*, un metodo di cui è proposta una descrizione più approfondita nei capitoli 4 e 5.

Viene costruito un albero di branching per l'enumerazione implicita delle soluzioni. GAP può essere riformulato come problema di *set partitioning* con *side constraints*: per ogni sottoproblema, il rilassamento lineare di questa nuova formulazione costituisce un bound duale valido. Tecniche di *column generation* vengono utilizzate per ottenere il valore ottimo del rilassamento. L'utilizzo iterato dell'euristica MTH di Martello e Toth [MT89], descritta nel capitolo 2, e un'opportuna scelta dei coefficienti f_{ij} dell'euristica, permettono di individuare soluzioni ammissibili durante l'esplorazione dell'albero di branching. Savelsbergh propone due strategie di branching: branching dicotomico e per inserimento di vincoli di *subset sum*; per ogni strategia analizza la possibilità di ricercare soluzioni con visita depth first o best first dell'albero di branching. E' stata realizzata un'implementazione dell'algoritmo di Savelbergh adottando la strategia di branching per inserimento di vincoli di *subset sum* e ricerca best first.

3.2 Risultati sperimentali

La riduzione da CPMP a GAP può essere effettuata come descritto con costo computazionale irrilevante. Sono stati condotti dei test risolvendo alcuni problemi di CPMP riformulati come GAP utilizzando l'algoritmo di Savelsbergh. Nella tabella 3.1 sono riportati, per ogni test, il problema ed il valore di p utilizzati. Nel caso in cui la computazione sia terminata entro 60 minuti, è riportato il tempo di CPU impiegato (in secondi). Per i test in cui sia stato superato il limite di tempo imposto (indicati con -), sono riportati il valore della miglior soluzione ammissibile trovata (Z^*) e del bound duale

(ω_{CG}) dell'ultimo sottoproblema analizzato (la visita *best first* dell'albero di branching garantisce che i bound duali dei successivi sottoproblemi non siano migliori).

Problema		classe A		classe B		classe C	
		$Z^*(\omega_{CG})$	tempo (s)	$Z^*(\omega_{CG})$	tempo (s)	$Z^*(\omega_{CG})$	tempo (s)
N = 50	CCPX1	713	162	383	3348.1	(266;262.495)	-
	CCPX2	740	27.6	412	179.68	(307;294.87)	-
	CCPX3	751	39	(405;403.745)	-	311	2440
	CCPX4	651	36	(386;373.352)	-	277	215.8
	CCPX5	664	35.5	(429;424.662)	-	(357;354.495)	-
	CCPX6	778	31.5	(490;474.937)	-	370	24.9
	CCPX7	787	199.4	(453;431.795)	-	358	54.2
	CCPX8	(823;797.863)	-	(412;400.298)	-	(325;303.328)	-
	CCPX9	715	103.8	(436;429.995)	-	(414;406.423)	-
	CCPX10	829	537.3	(472;450.287)	-	(474;444.995)	-

Tabella 3.1: Algoritmo di Savelsbergh su istanze di CPMP ridotte a GAP

Sebbene l'algoritmo di Savelbergh si dimostri molto efficace per problemi di GAP, sulle istanze trasformate di CPMP ha prestazioni peggiori del MIP solver. Infatti CPLEX termina il calcolo, su problemi con N = 50, nel 90% dei casi, mentre il metodo descritto consente di risolvere solo il 50% delle istanze. Anche paragonando i tempi di calcolo medi quando entrambi terminano, CPLEX è migliore (55.532 s contro i 495.65 s necessari per la soluzione delle istanze trasformate in GAP). E' da notare come il divario di prestazioni con algoritmi specifici per CPMP sia destinato ad aumentare al crescere della dimensione dei problemi: il grafo per GAP costruito ha numero di nodi doppio rispetto al corrispondente grafo di CPMP, ed il numero di possibili soluzioni aumenta in maniera *esponenziale* nella cardinalità del grafo.

Capitolo 4

Rilassamento Lagrangeano

Il rilassamento Lagrangeano si è dimostrato efficace nell'applicazione a molti problemi di ottimizzazione [Beas92], pertanto si presta bene come termine di confronto con altri metodi.
Diversi autori hanno proposto algoritmi basati su rilassamenti di tipo Lagrangeano anche per la soluzione esatta o approssimata del problema delle p-mediane senza capacità (PMP) [Chri81] [Beas85] [Naru76] o per la soluzione di problemi di localizzazione multirisorse con capacità (CFLP) [Srid95]. Nei capitoli successivi è presentato un confronto tra le prestazioni di questo approccio ed altri algoritmi.
Anche Baldacci, Hadjiconstantinou, Maniezzo e Mingozzi [Bald02] comparano le prestazioni del proprio metodo per CPMP con un algoritmo Lagrangeano derivato da quello di Pirkul [Pirk87] per il problema di *localizzazione dei concentratori con capacità* (CCLP), descritto nella sezione 2.

Il modello di Christofides e Beasley

Christofides e Beasley [Chri81] affrontano il problema delle p-mediane classico:
PMP)

$$\min\ v = \sum_{i\in\mathcal{N}} \sum_{j\in\mathcal{M}} d_{ij}x_{ij} \tag{4.1}$$

s.t.

$$\sum_{j\in\mathcal{M}} x_{ij} = 1\ \forall i \in \mathcal{N} \tag{4.2}$$

$$\sum_{i\in\mathcal{N}} x_{ij} \leq N y_j\ \forall j \in \mathcal{M} \tag{4.3}$$

$$\sum_{j\in\mathcal{M}} y_j = p \tag{4.4}$$

$$x_{ij} \in \{0,1\}\ \forall i \in \mathcal{N},\ \ \forall j \in \mathcal{M} \tag{4.5}$$

$$y_j \in \{0,1\}\ \forall j \in \mathcal{M} \tag{4.6}$$

Il PMP è un caso particolare del problema con capacità in cui $w_i = 1\ \forall i \in \mathcal{N}$ e $Q_j = N\ \forall j \in \mathcal{M}$. I vincoli (1.6), del modello di CPMP, sono aggregati nei vincoli (4.3) che, in questo caso, impediscono che vengano inseriti elementi in clusters la cui mediana non è attiva.
Gli autori propongono un confronto fra algoritmi basati sul rilassamento della famiglia di vincoli (4.2) (LR1) ed algoritmi in cui vengono rilassati i vincoli (4.3) (LR2).

L'esperienza computazionale mostra che gli algoritmi basati sul rilassamento *LR1* hanno prestazioni migliori rispetto a quelli basati sul rilassamento *LR2*.

Il modello di Pirkul

Il *problema di localizzazione dei concentratori con capacità* (CCLP) è un problema di localizzazione multirisorse con capacità su grafo, simile a CPMP, in cui non c'è il vincolo (1.4) sul numero di clusters nei quali il grafo deve essere partizionato, ed in cui l'installazione di un nuovo concentratore è penalizzata con dei costi fissi FC_j.
CCLP)

$$\min\ v = \sum_{i\in\mathcal{N}}\sum_{j\in\mathcal{M}} d_{ij}x_{ij} + \sum_{j\in\mathcal{M}} FC_j y_j \tag{4.7}$$

s.t.

$$\sum_{j\in\mathcal{M}} x_{ij} = 1\ \forall i \in \mathcal{N} \tag{4.8}$$

$$\sum_{i\in\mathcal{N}} w_i x_{ij} \leq Q_j y_j\ \forall j \in \mathcal{M} \tag{4.9}$$

$$x_{ij} \in \{0,1\}\ \forall i \in \mathcal{N},\ \forall j \in \mathcal{M} \tag{4.10}$$

Anche Pirkul propone il rilassamento dei vincoli (4.8), come nel rilassamento *LR1* dell'algoritmo di Christofides e Beasley.
Sarebbe possibile risolvere CPMP fissando arbitrariamente uno stesso valore per i costi fissi FC_j ed utilizzando l'algoritmo di Pirkul. In generale, la soluzione ottenuta in tal modo può violare il vincolo sul numero di mediane. E' necessario, in tal caso, modificare il valore dei costi fissi (riducendolo se il numero di concentratori nella soluzione ottima risulta minore di p, incrementandolo altrimenti) e risolvere nuovamente il problema, procedendo *per tentativi* fino ad individuare un valore per il quale i concentratori, nella soluzione ottima, siano esattamente p.

4.1 Un rilassamento per CPMP

Si è scelto di adottare un rilassamento Lagrangeano analogo a quello suggerito da Christofides e Beasley per PMP e da Pirkul per CCLP, in cui viene

rilassato anche il vincolo sul numero di mediane. Il problema rilassato che si ottiene è il seguente:
LR)

$$\min\ \omega_{LR} = \sum_{i\in\mathcal{N}}\sum_{j\in\mathcal{M}} d_{ij}x_{ij} + \sum_{i\in\mathcal{N}} \lambda_i(1 - \sum_{j\in\mathcal{M}} x_{ij}) + \nu(\sum_{j\in\mathcal{M}} y_j - p) \quad (4.11)$$

s.t.

$$\sum_{i\in\mathcal{N}} w_i x_{ij} \leq Q_j y_j\ \forall j \in \mathcal{M} \quad (4.12)$$

$$x_{ij} \in \{0,1\}\ \forall i \in \mathcal{N},\ \ \forall j \in \mathcal{M} \quad (4.13)$$

$$y_j \in \{0,1\}\ \forall j \in \mathcal{M} \quad (4.14)$$

CPMP può essere rilassato, portando i vincoli

$$\sum_{j\in\mathcal{M}} x_{ij} = 1\ \forall i \in \mathcal{N}$$

in forma di $\geq$ ed il vincolo

$$\sum_{j\in\mathcal{M}} y_j = p$$

in forma di $\leq$, come descritto nel capitolo 1, senza influenzare la struttura della soluzione ottima. In tal modo è più semplice trovare rapidamente una soluzione ammissibile ed è quindi possibile migliorare l'accuratezza del metodo del sottogradiente (descritto in seguito) durante le prime iterazioni. Quindi, vincolando in segno i moltiplicatori

$$\lambda_i \geq 0\ \forall i \in \mathcal{N},\ \ \nu \geq 0 \quad (4.15)$$

la violazione dei vincoli viene penalizzata aggiungendo termini positivi nella funzione obiettivo.

4.2 Bound duale

Rilassando il problema come descritto, se ne ottiene la disaggregazione in M sottoproblemi indipendenti

$$min\ \omega_j = \sum_{i \in \mathcal{N}} (d_{ij} - \lambda_i) x_{ij} + \nu y_j$$

s.t.

$$\sum_{i \in \mathcal{N}} w_i x_{ij} \leq Q_j y_j$$

$$x_{ij} \in \{0,1\}\ \forall i \in \mathcal{N},\ \ \forall j \in \mathcal{M}$$

$$y_j \in \{0,1\}\ \forall j \in \mathcal{M}$$

Per ogni sottoproblema si considerano due casi:

- se $y_j = 0$ il problema ha come unica soluzione $x_{ij} = 0\ \forall i \in \mathcal{N}$ e quindi $\omega_j = 0$;
- se invece $y_j = 1$, il sottoproblema diventa un problema di knapsack: KP_j)

  $$\max\ \tau_j = \sum_{i \in \mathcal{N}} (\lambda_i - d_{ij}) x_{ij}$$

  s.t.

  $$\sum_{i \in \mathcal{N}} w_i x_{ij} \leq Q_j$$

  $$x_{ij} \in \{0,1\}\ \forall i \in \mathcal{N}$$

  ed, in tal caso, $\omega_j = -\tau_j^* + \nu$.

Dato che il vincolo di capacità è in forma di $\leq$,

$$\min\ \tau_j = \sum_{i \in \mathcal{N}} (d_{ij} - \lambda_i) x_{ij}$$

è stato posto in forma di massimizzazione, cambiando il segno dei coefficienti.

4.2.1 Il problema Lagrangeano duale

Ottenuti i valori τ_j^*, risolvendo gli M problemi KP_j all'ottimo, è possibile valutare l'effetto di designare il nodo j come mediana, calcolando le *penalties*

$$\pi_j = -\tau_j^* + \nu$$

Il bound duale può essere calcolato come:

$$\omega_{LR} = \sum_{j \in \mathcal{M}} min\{\pi_j,\ 0\} + \sum_{i \in \mathcal{N}} \lambda_i - \nu p$$

Il problema di determinare $\omega_{LD} = max\{\omega_{LR}(\boldsymbol{\lambda},\ \nu)\}$ viene detto *Lagrangeano duale* (LD). Essendo LR una versione rilassata del problema originale, ogni soluzione ammissibile per LD dà un bound duale valido per CPMP.

Per migliorare il bound duale ad ogni nodo si può ricorrere ad una soluzione euristica di LD: il vettore $(\boldsymbol{\lambda},\ \nu) = (\lambda_1,\ \lambda_2,\ \ldots,\ \lambda_N,\ \nu)$ dei moltiplicatori può essere aggiornato con il metodo iterativo del sottogradiente, ampiamente collaudato per i problemi di ottimizzazione basati su rilassamento Lagrangeano [Beas92]. Si definiscano:

$$G_i = 1 - \sum_{j \in \mathcal{M}} x_{ij}\ \forall i \in \mathcal{N}$$

$$G_0 = \sum_{j \in \mathcal{M}} y_j - p$$

$$T_t = \zeta_t \frac{(Z^* - \omega_{LR}(\boldsymbol{\lambda^t}, \nu^t))}{\sum_{i \in \mathcal{N}} (G_i)^2 + {G_0}^2}$$

$$\lambda_i^{t+1} = max\{\lambda_i^t + T_t G_i, 0\}\ \forall i \in \mathcal{N}$$

$$\nu^{t+1} = max\{\nu^t + T_t G_0, 0\}$$

dove Z^* è il valore della miglior soluzione ammissibile corrente per CPMP, $\omega_{LR}(\boldsymbol{\lambda^t}, \nu^t)$ è il valore della soluzione di LR per i valori dei moltiplicatori $\boldsymbol{\lambda^t}$ e ν^t al passo t, ζ_t è un parametro reale positivo, che regola l'ampiezza del

passo T_t. Se un vincolo dell'insieme (1.2) è violato nella soluzione di LR, G_i risulta positivo e λ_i viene incrementato; al contrario, se il sottogradiente G_i corrispondente ad un vincolo risulta negativo, il valore del corrispondente moltiplicatore viene decrementato.

Sia $(\boldsymbol{\lambda^*}, \nu^*)$ il *vettore dei moltiplicatori ottimo* ($\omega^*_{LD} = \omega_{LD}(\boldsymbol{\lambda^*}, \nu^*) = \max_{\boldsymbol{\lambda},\nu}\{\omega_{LD}(\boldsymbol{\lambda}, \nu)\}$), il metodo del sottogradiente è tale che $\max_t\{\omega_{LD}(\boldsymbol{\lambda^t}, \nu^t)\}$ converge a ω^*_{LD} se

$$\lim_{t\to\infty} T_t = 0$$

e

$$\lim_{\Gamma\to\infty} \sum_{t=0}^{\Gamma} T_t = \infty$$

Il metodo del sottogradiente è analizzato da Crowder, Wolfe ed Held in [Held74] e considerazioni più specifiche sono riportate in [Wols98] (pagg. 173-176)
Essendo interessati ad una soluzione non necessariamente ottima per LD, è possibile compiere un numero finito Γ di iterazioni. Inoltre è pratica ricorrente porre il parametro ζ_t ad un valore iniziale arbitrario (nell'implementazione realizzata 1), dimezzarne il valore ogni h_ζ iterazioni non miglioranti (ad esempio 15) e terminare l'algoritmo quando il valore di ζ_t scende sotto una soglia stabilita (nel caso in esame, vengono compiute Γ iterazioni, indipendentemente dal valore di ζ_t).

4.2.2 Tecniche di bounding alternative

Come descritto nella sezione 4, la soluzione ottima per CPMP viene individuata tramite enumerazione implicita di tutte le soluzioni, realizzata mediante la scomposizione ricorsiva del problema in sottoproblemi (figli) dalle dimensioni inferiori. I bounds descritti devono essere valutati per tutti questi sottoproblemi. Per questo motivo e poiché il metodo del sottogradiente è una tecnica computazionalmente dispendiosa, anche se molto efficace, Pirkul

propone di valutare altri tipi di bounds duali, prima di dare inizio all'aggiornamento iterativo dei moltiplicatori (che comporta, ad ogni passaggio, il calcolo della soluzione ottima per M problemi di knapsack). E' possibile

1. valutare un bound duale per ogni sottoproblema figlio utilizzando i valori delle penalties del padre, che dovranno, dunque essere memorizzate. In particolare, se in un figlio è stato imposto che sia mediana un nodo con penalty positiva, il bound duale calcolato in questo modo sul figlio risulta sicuramente più stretto che nel padre. Analogamente, il bound diventa più stringente se in un figlio si impedisce che un nodo con penalty negativa (nel padre) possa essere mediana.

2. risolvere il rilassamento lineare dei problemi di knapsack con il miglior set di moltiplicatori $(\boldsymbol{\lambda^*}, \nu^*)$ incontrato durante la valuatazione del sottoproblema precedente (come descritto in seguito).
   KP_j^L)

   $$\max\ \tau_j^L = \sum_{i \in \mathcal{N}} (\lambda_i - d_{ij}) x_{ij}$$

   s.t.

   $$\sum_{i \in \mathcal{N}} w_i x_{ij} \leq Q_j$$

   $$0 \leq x_{ij} \leq 1$$

   I valori di penalty trovati in questo modo saranno sicuramente non peggiori di quelli ottenibili tramite la soluzione esatta dei problemi di knapsack 0-1, quindi il bound duale è valido. Il rilassamento continuo può essere risolto all'ottimo calcolando l'efficienza di ogni elemento i nel problema j ($e_i^j = \frac{\lambda_i - d_{ij}}{w_i}$), ordinando gli elementi per efficienza decrescente e selezionando l'insieme K dei primi k elementi della successione ordinata tali che $\sum_{i \in K} w_i \leq Q_j$. Sia quindi i^* il primo elemento della successione che, procedendo come descritto, viola il vincolo di capacità

(*break item*). Il valore ottimo del rilassamento lineare per il j-esimo problema di knapsack è allora:

$$\tau_j^{L^*} = \sum_{i \in K} (\lambda_i - d_{ij}) + e_{i^*}^j (Q_j - \sum_{i \in K} w_i)$$

Le penalties sono

$$\pi_j^L = -\tau_j^{L^*} + \nu \quad \forall j \in \mathcal{M}$$

ed il bound duale

$$\omega_{LR} = \sum_{j \in \mathcal{M}} \min\{\pi_j^L, \ 0\} + \sum_{i \in \mathcal{N}} \lambda_i - \nu p$$

In tabella 4.1 è presentato un confronto tra le prestazioni dell'algoritmo di Branch & Bound utilizzando per ogni sottoproblema solo queste tecniche di bounding (ma utilizzando il metodo del sottogradiente per 300 iterazioni sul nodo radice), ricorrendo solo a rilassamento Lagrangeano con metodo del sottogradiente per l'aggiornamento dei moltiplicatori, oppure utilizzando entrambe le tecniche, iniziando le iterazioni del sottogradiente solo nei casi in cui il bound duale ottenuto tramite le tecniche alternative non sia superiore al bound primale.

Come è possibile desumere dai risultati riportati, applicate da sole le tecniche alternative non forniscono bounds sufficientemente stretti. Tuttavia, dato il ridotto carico computazionale richiesto per la loro valutazione, possono contribuire a migliorare l'efficienza dell'algoritmo se abbinati al metodo del sottogradiente.

4.3 Bound primale

Come affermato nel capitolo 2, il problema di esistenza di una soluzione per la generica istanza di CPMP è $\mathcal{NP}$-Completo.
L'euristica proposta nello stesso capitolo può sfruttare l'informazione derivante dalla soluzione del rilassamento Lagrangeano per cercare soluzioni ammissibili: invece di operare una scelta casuale delle mediane si può osservare

classe A

Problema		Solo bounds deboli		Solo Sottogradiente		Entrambi	
		$Z^*(\omega_{CG})$	tempo (s)	$Z^*(\omega_{CG})$	tempo (s)	$Z^*(\omega_{CG})$	tempo (s)
N = 50	CCPX1	713	28.4	713	1.5	713	0.2
	CCPX10	(925;813.381)	-	829	28.7	829	32.1
N = 100	CCPX16	(1046;944.717)	-	954	75.7	954	87.4
	CCPX19	(1250;1018.03)	-	1031	979.8	1031	205.4

classe B

Problema		Solo bounds deboli		Solo Sottogradiente		Entrambi	
		$Z^*(\omega_{CG})$	tempo (s)	$Z^*(\omega_{CG})$	tempo (s)	$Z^*(\omega_{CG})$	tempo (s)
N = 50	CCPX1	(459;370.941)	-	383	167.5	383	245.0
	CCPX10	(584;412.662)	-	(472;412.662)	-	(471;412.662)	-
N = 100	CCPX16	(715;517.611)	-	(572;517.611)	-	(542;517.611)	-
	CCPX19	(695;527.533)	-	(573;527.533)	-	(568;527.533)	-

classe C

Problema		Solo bounds deboli		Solo Sottogradiente		Entrambi	
		$Z^*(\omega_{CG})$	tempo (s)	$Z^*(\omega_{CG})$	tempo (s)	$Z^*(\omega_{CG})$	tempo (s)
N = 50	CCPX1	(416;254.053)	-	(266;254.053)	-	266	1662.0
	CCPX10	(810;365.004)	-	(475;365.004)	-	(483;365.004)	-
N = 100	CCPX16	(720;419.138)	-	(457;419.138)	-	(444;419.138)	-
	CCPX19	(885;431.375)	-	(486;431.375)	-	(543;431.375)	-

Tabella 4.1: Bounds deboli e sottogradiente

la struttura della soluzione del problema rilassato.

Viene risolto LR, ottenendo M vettori $\boldsymbol{x_j^{LR}} = (x_{1j} \dots x_{Nj})$ corrispondenti alla soluzione corrente di LR

Fase 1: Scelta delle mediane

In ogni soluzione ammissibile per CPMP, devono essere selezionate esattamente p mediane. Sia $\mathcal{M}_{LR}$ l'insieme delle mediane nella soluzione corrente di LR e $p_{LR} = |\mathcal{M}_{LR}|$

- Se $p_{LR} < p$, la soluzione è completata considerando come mediane i $p - p_{LR}$ nodi più "promettenti", ovvero inserendo in $\mathcal{M}_{LR}$ i nodi non designati come mediane aventi i minimi valori di $\sum_{i \in \mathcal{N}} d_{ij}$.
- Se $p_{LR} > p$, le $p_{LR} - p$ mediane aventi i cluster di minima cardinalità vengono rimosse da $\mathcal{M}_{LR}$
- Altrimenti $\mathcal{M}_{LR}$ rimane invariato

Per ogni potenziale mediana j i valori $\sum_{i \in \mathcal{N}} d_{ij}$ possono essere calcolati in tempo lineare una sola volta durante l'inizializzazione dell'algoritmo. Anche la cardinalità del cluster associato ad ogni mediana in $\mathcal{M}_{LR}$ è valutabile in tempo lineare. L'inserimento o rimozione di nodi da $\mathcal{M}_{LR}$ affinché $|\mathcal{M}_{LR}| = p$ può essere effettuato in tempo $O(MN)$.

Fase 2: Allocazione

Ora $|\mathcal{M}_{LR}| = p$. Nella soluzione corrente di LR, avendo rilassato i vincoli di set partitioning, alcuni nodi potrebbero essere assegnati a più mediane ed altri a nessuna mediana. Sia $\mathcal{N}_F$ l'insieme dei nodi inseriti in un solo cluster nella soluzione del rilassamento e $\mathcal{N}_W = \mathcal{N} \setminus \mathcal{N}_F$. In tal modo, sono stati considerati "liberi" anche i nodi assegnati a più clusters nella soluzione di LR. Si ricorre all'euristica descritta nel capitolo 2 per assegnare i nodi in $\mathcal{N}_W$ ai vari clusters, ponendo, come scelta dei coefficienti

$$f_{ij} = -d_{ij} \ \forall i \in \mathcal{N}_W, \forall j \in \mathcal{M}_{LR}$$

Fase 3: Minimizzazione dell'inammissibilità

Se il metodo precedentemente descritto non produce una soluzione ammissibile è possibile eseguire una ricerca locale attraverso la rimozione di elementi da clusters ed il reinserimento in altri in funzione del loro peso. Anche in questo caso si termina se $\mathcal{N}_W = \emptyset$, ed in tal caso la soluzione è ammissibile, oppure dopo un numero finito Γ di iterazioni: la ricerca locale di soluzioni ammissibili non offre alcuna garanzia di trovarle, ma all'interno di algoritmi di enumerazione implicita è possibile ricorrere all'euristica per raggiungere prima soluzioni ammissibili, migliorando le prestazioni del metodo.

Fase 4: Miglioramento della soluzione

Se la soluzione ottenuta è ammissibile, si può anche procedere alla ricerca di soluzioni migliori, attraverso scambi miglioranti.

Data l'intenzione di eseguire l'euristica descritta ad ogni iterazione del metodo del sottogradiente, è preferibile, però, rinunciare alle operazioni di ricerca locale, computazionalmente troppo dispendiose. Simili tecniche si rivelano utili in altre circostanze, come nel caso della versione adattata per l'utilizzo in algoritmi branch & price, descritti nel Capitolo 5.

4.4 Strategia di branching

Per risolvere all'ottimo il problema, è possibile ricorrere ad una strategia di enumerazione implicita delle soluzioni. In particolare Pirkul propone di effettuare branching a due livelli.

4.4.1 Albero di primo livello

Nell'albero di branching di primo livello, p nodi vengono designati come mediane, individuando un insieme $\mathcal{M}^B \subseteq \mathcal{M}$ di cardinalità p. Tale scelta viene effettuata con partizionamento binario dell'insieme delle soluzioni.

Scelto un nodo j^b, l'insieme di soluzioni ammissibili del sottoproblema corrente S_0 viene bipartito, creando due sottoproblemi disgiunti:

$$S_l = S_0 \cap \{(x_{ij},\ \ y_j \mid y_{j^b} = 0\}$$

e

$$S_r = S_0 \cap \{x_{ij},\ \ y_j \mid y_{j^b} = 1\}$$

In un sottoproblema-figlio, il nodo j^b viene designato come mediana ($y_{j^b} = 1$); nell'altro sottoproblema-figlio viene impedito che il nodo j^b possa essere designato come mediana ($y_{j^b} = 0$).

La condizione $y_j = 0$ può essere applicata al più a $(N-p)$ nodi: i rimanenti devono essere considerati mediane. Viceversa, se p nodi sono stati considerati mediane, i rimanenti devono avere $y_j = 0$.

Valutato il bound duale ω_{LR} ed il corrispondente insieme di penalties per un sottoproblema, si può scegliere di effettuare branch sulla variabile j^b avente il massimo valore π_j.

4.4.2 Albero di secondo livello

Quando il numero di nodi designati come mediane è p, CPMP si riduce ad un problema di assegnamento generalizzato (GAP), descritto nel capitolo 2. Come per CPMP, il problema di esistenza di una soluzione per GAP è $\mathcal{NP}$-Completo [MT89]. L'obiettivo della ricerca sull'albero di secondo livello è di inserire in modo ottimo ogni nodo in uno dei p clusters fissati.

Branching dicotomico

Scelto un nodo i^b ed una mediana $j^b \in \mathcal{M}^B$ (dove $\mathcal{M}^B$ è il sottoinsieme di $\mathcal{M}$ dei p nodi selezionati come mediane, definito al primo livello di branching), l'insieme di soluzioni del sottoproblema corrente S_0 può essere partizionato come

$$S_l = S_0 \cap \{x_{ij},\ \ y_j \mid x_{i^b j^b} = 0\}$$

e

$$S_r = S_0 \cap \{x_{ij},\ \ y_j \mid x_{i^b j^b} = 1\}$$

La seconda condizione è più forte, poiché implica $x_{i^b k} = 0\ \forall k \neq j^b$, e quindi, in questo caso, l'albero di branching risulta poco bilanciato.
Viene scelto come variabile di branch il termine x_{ij} corrispondente al nodo i^b ed alla mediana j^b per cui il valore $d_{i^b j^b}$ è minimo.

Branching sulle possibili mediane

Ogni sottoproblema S_j può essere ottenuto assegnando un nodo i^b ad ogni possibile mediana $j \in \mathcal{M}$, ovvero ponendo

$$S_j = S_0 \cap \{x_{ij} \mid x_{i^b j} = 1,\ \ x_{i^b k} = 0\ \forall k \neq j\}\ \forall j \in \mathcal{M}^B$$

E' conveniente scegliere il nodo i^b avente maggior peso.

Vari test dimostrano come, generalmente, non solo non sia necessario ricorrere alla soluzione del problema di assegnamento generalizzato (secondo livello di branching), ma spesso non vengano neppure raggiunte le foglie dell'albero di branching di primo livello (tabella 4.2).

Problema		*Sottoproblemi Valutati*	*Accessi al 2º Livello*	*Rapporto*
N = 50	CCPX1	37	1	2,700%
	CCPX7	116	3	2,586%
	CCPX10	409	9	2,200%
N = 100	CCPX11	658	0	0.000%
	CCPX16	739	1	0,135%
	CCPX19	1529	2	0,131%

Tabella 4.2: Livelli di Branching

In tabella 4.3 sono riportati i tempi di calcolo per le due strategie. Generalmente è poco influente scegliere la prima o la seconda politica di branch per il secondo livello. Per alcune istanze (CCPX1-B, CCPX11-B, CCPX7-C, CCPX10-C) la strategia di branching sulle possibili mediane risulta migliore, quindi è stata adottata per i test successivi.

classe A

Problema		Branch Dicotomico		Branch sulle mediane	
		$Z^*(\omega_{CG})$	tempo (s)	$Z^*(\omega_{CG})$	tempo (s)
N = 50	CCPX1	713	2.0	713	1.8
	CCPX7	787	12.8	787	11.0
	CCPX10	829	35.6	829	31.8
N = 100	CCPX11	1006	91.2	1006	90.4
	CCPX16	954	99.2	954	86.8
	CCPX19	1031	221.0	1031	202.5

classe B

Problema		Branch Dicotomico		Branch sulle mediane	
		$Z^*(\omega_{CG})$	tempo (s)	$Z^*(\omega_{CG})$	tempo (s)
N = 50	CCPX1	383	412.9	383	245.5
	CCPX7	(5.5%)	-	(5.5%)	-
	CCPX10	(15.4%)	-	(14.1%)	-
N = 100	CCPX11	(7.2%)	-	(4.5%)	-
	CCPX16	(4.9%)	-	(4.7%)	-
	CCPX19	(7.9%)	-	(7.7%)	-

classe C

Problema		Branch Dicotomico		Branch sulle mediane	
		$Z^*(\omega_{CG})$	tempo (s)	$Z^*(\omega_{CG})$	tempo (s)
N = 50	CCPX1	266	1763.8	266	1650.9
	CCPX7	358	1905.9	358	904.9
	CCPX10	(44.6%)	-	(32.3%)	-
N = 100	CCPX11	(7.7%)	-	(7.7%)	-
	CCPX16	(5.7%)	-	(5.9%)	-
	CCPX19	(25.9%)	-	(25.9%)	-

Tabella 4.3: Branch dicotomico e Branch sulle mediane

4.5 Analisi di sensitività e preprocessing

Sia Z^* un bound primale e ω_{LR} un bound duale calcolato sulla base di M penalties $(\pi_1 \dots \pi_M)$.

- Se $\exists j \mid \pi_j > 0,\ \omega_{LR} + \pi_j \geq Z^*$, imponendo $y_j = 1$ si otterrebbe un bound duale superiore al bound primale. Quindi, per poter ottenere soluzioni migliori dell'ottimo locale incontrato, deve necessariamente essere $y_j = 0$ (e quindi $x_{ij} = 0\ \forall i \in \mathcal{N}$).
- Se $\exists j \mid \pi_j < 0,\ \omega_{LR} - \pi_j \geq Z^*$, imponendo $y_j = 0$ si otterrebbe un bound duale superiore al bound primale. Quindi necessariamente $y_j = 1$.

Una simile analisi può essere effettuata, una volta valutate le penalties, in tempo $O(M)$. Questi semplici test possono portare, come osservato da diversi autori [Beas92], alla rimozione di un numero rilevante di variabili dal problema (per alcuni problemi anche il 75 per cento), incidendo in maniera determinante sulle prestazioni dell'algoritmo. Ponendo una $y_j = 0$, infatti, N variabili vengono rimosse dal problema: alla formulazione standard possono essere aggiunti i vincoli

$$x_{ij} \leq y_j\ \forall i \in \mathcal{N},\ \ \forall j \in \mathcal{M} \tag{4.16}$$

Questi vincoli sono ridondanti, poiché implicati dai vincoli (1.3) per garantire che i nodi siano associati solo a mediane attive, ma possono essere utilizzati per migliorare l'efficienza degli algoritmi proposti con tecniche di preprocessing: considerare $y_j = 0$ implica che debba essere $x_{ij} = 0\ \forall i \in \mathcal{N}$.

Nel rispetto del vincolo (1.4), per il quale le mediane debbano essere esattamente p, anche fissare una $y_j = 1$ riduce il numero di sottoproblemi da analizzare.

4.6 Risultati sperimentali

Di seguito (Tabelle 4.4) sono riportati i risultati relativi all'esecuzione dell'algoritmo descritto per le 60 istanze di CPMP in esame. Sono indicati i

tempi di calcolo ed il valore della miglior soluzione ammissibile trovata (Z^*); per i casi nei quali la computazione non è terminata entro 3600 secondi sono indicati il valore della miglior soluzione primale trovata ed il bound duale riferito all'intero problema. I moltiplicatori sono stati posti inizialmente al valore 10.0. Il bound duale è calcolato utilizzando inizialmente i valori delle penalties del problema padre. Se necessario, in un secondo momento i moltiplicatori sono inizializzati ai valori $(\boldsymbol{\lambda}^*, \nu^*)$ per i quali, nell'ultimo sottoproblema analizzato, è stato individuato il miglior bound duale. Questa scelta non offre alcuna garanzia sulla qualità del bound ottenuto risolvendo il rilassamento. Con questo set di moltiplicatori viene valutato il rilassamento lineare dei problemi di knapsack; se anche questa tecnica non porta alla determinazione di un bound duale superiore al bound primale, si ricorre al metodo del sottogradiente. In quest'ultimo caso si è posto il parametro ζ, che determina l'ampiezza del passo, inizialmente a 1.0, dimezzandolo ogni 15 iterazioni in cui il bound duale non venisse migliorato ed eseguendo complessivamente 300 iterazioni del metodo per il problema radice, 50 iterazioni per ogni sottoproblema di primo livello e 10 per quelli di secondo livello. Si è scelto di effettuare, al secondo livello, branching sulle possibili mediane ed esplorare l'albero di branching mediante strategia depth first ad entrambi i livelli.

I tempi di calcolo per i problemi della classe A sono, in generale, di un ordine di grandezza inferiori rispetto all'utilizzo di CPLEX e due ordini di grandezza inferiori se paragonati ai risultati dell'algoritmo di Savelsbergh applicato ad istanze riformulate come GAP. Ad eccezione di CCPX20, tutti i problemi sono stati risolti in meno di tredici minuti.

Al decrescere del rapporto $\frac{N}{p}$, tuttavia, le prestazioni decadono in maniera molto evidente: i tempi di calcolo diventano superiori a quelli ottenibili utilizzando CPLEX. La strategia di branching adottata, infatti, è molto efficace se $p \ll N$, in quanto fissare un nodo come mediana o impedire che lo sia è più vincolante nella definizione della soluzione.

Problema		classe A		classe B		classe C	
		$Z^*(\omega_{CG})$	tempo (s)	$Z^*(\omega_{CG})$	tempo (s)	$Z^*(\omega_{CG})$	tempo (s)
N = 50	CCPX1	713	2.0	383	245.0	266	1662.0
	CCPX2	740	0.6	412	24.1	(298;273.909)	-
	CCPX3	751	1.6	405	153.0	311	511.8
	CCPX4	651	1.3	(385;364.073)	-	277	212.3
	CCPX5	664	0.8	429	491.1	(362;313.208)	-
	CCPX6	778	1.4	(482;457.975)	-	370	908.7
	CCPX7	787	11.0	(445;421.711)	-	358	903.3
	CCPX8	820	194.6	403	1967.9	(333;272.466)	-
	CCPX9	715	11.1	436	1191.6	(412;401.815)	-
	CCPX10	829	32.1	(471;412.662)	-	(483;365.004)	-
N = 100	CCPX11	1006	92.2	(550;526.089)	-	(427;396.315)	-
	CCPX12	966	772.7	(512;494.934)	-	(427;357.819)	-
	CCPX13	1026	50.1	(567;531.784)	-	(433;400.964)	-
	CCPX14	982	674.7	(555;527.888)	-	(434;397.772)	-
	CCPX15	1091	309.5	(590;569.79)	-	(506;446.195)	-
	CCPX16	954	87.4	(542;517.611)	-	(444;419.138)	-
	CCPX17	1034	498.6	(549;532.382)	-	(779;404.555)	-
	CCPX18	1043	558.2	(508;499.935)	-	(648;402.02)	-
	CCPX19	1031	205.4	(568;527.533)	-	(543;431.375)	-
	CCPX20	(1005;928.29)	-	(622;511.982)	-	(1406;428.018)	-

Tabella 4.4: Algoritmo con Rilassamento Lagrangeano

Capitolo 5

Algoritmo Branch & Price

Nelle sezioni seguenti viene proposto un algoritmo per la soluzione esatta di CPMP: il bound duale, per ogni sottoproblema, viene valutato risolvendo il rilassamento lineare della riformulazione di CPMP come problema di *set partitioning* con *side constraints*. Questa nuova formulazione contempla un numero di variabili esponenziale nella dimensione del problema. E' possibile, tuttavia, considerare solo un sottoinsieme delle variabili del problema riformulato, e le relative colonne nella matrice dei vincoli, ottenendo un problema *ridotto*: partendo da una soluzione di base ammissibile si possono inserire nel problema ridotto solo le colonne corrispondenti a variabili candidate ad entrare in base. Il problema di determinare quali siano le colonne aventi costo ridotto -*price*- opportuno (negativo, per problemi di minimizzazione) è detto *pricing problem*; l'individuazione ed inserimento iterativo di opportune colonne nel problema ridotto è chiamato *column generation*. L'applicazione di column generation a problemi di ottimizzazione (mista) intera, per i quali è richiesta l'enumerazione implicita delle soluzioni, conduce ad algoritmi di *branch & price*.

5.1 Riformulazione secondo Dantzig - Wolfe

La regione di ammissibilità di CPMP è l'intersezione di M insiemi indipendenti Ω^j che soddisfano vincoli di convessità (1.3 e 1.4). Solo i vincoli (1.2) pongono in relazione i diversi insiemi di variabili; i vincoli di questa famiglia vengono detti *di unione*, per la loro proprietà di determinare l'aggregazione degli M sottoproblemi descritti.

Dantzig e Wolfe [Dant60] propongono un metodo generale per riformulare problemi di programmazione (mista) intera in modo da ottenerne la disaggregazione in sottoproblemi distinti. sia $\boldsymbol{x^j} = (x_{1,j} \dots x_{N,j})$ un vettore della matrice delle variabili, $\boldsymbol{d^j} = (d_{1,j} \dots d_{N,j})$ un vettore della matrice dei coefficienti e $\boldsymbol{w} = (w_1 \dots w_N)$ il vettore dei pesi dei nodi. Siano

$$\Omega^j = \{(\boldsymbol{x^j}, y_j) \in \boldsymbol{B}^{N+1} \mid \boldsymbol{w^T x^j} \leq Q_j y_j\} \; \forall j \in \mathcal{M}$$

ogni Ω^j contiene un numero finito L_j di punti, quindi

$$\Omega^j = \{(\boldsymbol{\alpha_1^j}, \beta_1^j) \dots (\boldsymbol{\alpha_{L_j}^j}, \beta_{L_j}^j)\}$$

dove $(\boldsymbol{\alpha_l^j}, \beta_l^j) = (\alpha_{1,l}^j \dots \alpha_{N,l}^j, \beta_l^j) \; \forall l = 1 \dots L_j$. Dunque

$$\Omega^j = \{(\boldsymbol{x^j}, y_j) \in \boldsymbol{R}^{N+1} \mid (\boldsymbol{x^j}, y_j) = \sum_{l=1}^{L_j} \varphi_l^j (\boldsymbol{\alpha_l^j}, \beta_l^j), \sum_{l=1}^{L_j} \varphi_l^j = 1, \boldsymbol{\varphi^j} \in \boldsymbol{B}^{\boldsymbol{L_j}}\} \quad (5.1)$$

dove $\boldsymbol{\varphi^j} = (\varphi_1^j \dots \varphi_{L_j}^j)$.

L'obiettivo è scegliere p di questi punti, appartenenti a Ω^j diversi (e quindi clusters riferiti a mediane diverse), in modo che ogni nodo venga associato ad una mediana minimizzando la somma delle distanze tra i nodi e la mediana in ogni cluster.

Dalla formulazione iniziale

CPMP)

$$min \; v = \sum_{j \in \mathcal{M}} \boldsymbol{d^j}^T \boldsymbol{x^j}$$

s.t.

$$\sum_{j \in \mathcal{M}} \boldsymbol{x^j} = \boldsymbol{e}$$

$$(\boldsymbol{x^j}, y_j) \in \Omega^j = \{(\boldsymbol{x^j}, y_j) \in \boldsymbol{B^{N+1}} \mid \boldsymbol{w^T x^j} \leq Q_j y_j\}$$

$$\sum_{j \in \mathcal{M}} y_j = p$$

in cui $\boldsymbol{e} = (\underbrace{1 \ldots 1}_{N})$ è il vettore dei termini noti associati ai vincoli (1.2), è possibile ottenere una formulazione equivalente *estendendo* il vettore $\boldsymbol{e}$ al vettore $\widetilde{\boldsymbol{e}} = (\boldsymbol{e}, p)$ ed il vettore $\boldsymbol{d^j}$ al vettore $\widetilde{\boldsymbol{d^j}} = (\boldsymbol{d^j}, 0)$:

$$min\ v = \sum_{j \in \mathcal{M}} \widetilde{\boldsymbol{d^j}}^T (\boldsymbol{x^j}, y_j)$$

s.t.

$$\sum_{j \in \mathcal{M}} (\boldsymbol{x^j}, y_j) = \widetilde{\boldsymbol{e}}$$

$$(\boldsymbol{x^j}, y_j) \in \Omega^j = \{(\boldsymbol{x^j}, y_j) \in \boldsymbol{B^{N+1}} \mid \boldsymbol{w^T x^j} \leq Q_j y_j\}$$

Sostituendo $(\boldsymbol{x^j}, y_j)$ con (5.1) si ottiene

$$min\ v = \sum_{j \in \mathcal{M}} [\widetilde{\boldsymbol{d^j}}^T \sum_{l \in L_j} \varphi_l^j (\boldsymbol{\alpha_l^j}, \beta_l^j)]$$

s.t.

$$\sum_{j \in \mathcal{M}} \sum_{l \in L_j} \varphi_l^j (\boldsymbol{\alpha_l^j}, \beta_l^j) = \widetilde{\boldsymbol{e}}$$

$$\sum_{l \in L_j} \varphi_l^j = 1 \ \forall j \in \mathcal{M}$$

ovvero, indicando componente per componente,

$$min\ v = \sum_{j \in \mathcal{M}} \sum_{l \in L_j} \sum_{i \in \mathcal{N}} d_{i,j} \varphi_l^j \alpha_{i,l}^j$$

s.t.

$$\sum_{j \in \mathcal{M}} \sum_{l \in L_j} \varphi_l^j \alpha_{i,l}^j = 1 \ \forall i \in \mathcal{N} \tag{5.2}$$

$$\sum_{j\in\mathcal{M}}\sum_{l\in L_j}\varphi_l^j\beta_l^j = p \tag{5.3}$$

$$\sum_{l\in L_j}\varphi_l^j = 1\ \forall j\in\mathcal{M} \tag{5.4}$$

Sia $Z = \bigcup_{j\in\mathcal{M}}\Omega^j$ l'insieme di tutti i possibili clusters *ammissibili*, $K = |Z|$; si definiscano le variabili γ_j^k come:

$$\gamma_j^k = \begin{cases} 1 & \text{se } \sum_{l=1}^{j-1} L_l < k \leq \sum_{l=1}^{j} L_l \\ \\ 0 & \text{altrimenti} \end{cases}$$

cioè $\gamma_j^k = 1$ se il cluster k è riferito alla mediana j. Ricordando che $\mathcal{M}$ è l'insieme dei nodi candidati ad essere scelti come mediana e $M = |\mathcal{M}|$, le variabili $\boldsymbol{\varphi^j}$ possono essere identificate tramite M indici $1\dots M$. Siano inoltre

$$z^k = \begin{cases} \varphi_k^1 & \text{se } 1 \leq k \leq L_1 \\ \varphi_{k-L_1}^2 & \text{se } L_1 < k \leq (L_1+L_2) \\ \varphi_{k-L_1-L_2}^3 & \text{se } (L_1+L_2) < k \leq (L_1+L_2+L_3) \\ \vdots & \\ \varphi^m_{k-\sum_{l=1}^{m-1} L_l} & \text{se } \sum_{l=1}^{m-1} L_l < k < \sum_{l=1}^{m} L_l \\ \vdots & \\ \varphi^M_{k-\sum_{l=1}^{M-1} L_l} & \text{se } \sum_{l=1}^{M-1} L_l < k < \sum_{l=1}^{M} L_l \end{cases}$$

$$\boldsymbol{a^k} = \begin{cases} \boldsymbol{\alpha_k^1} & \text{se } 1 \leq k \leq L_1 \\ \boldsymbol{\alpha^2_{k-L_1}} & \text{se } L_1 < k \leq (L_1+L_2) \\ \boldsymbol{\alpha^3_{k-L_1-L_2}} & \text{se } (L_1+L_2) < k \leq (L_1+L_2+L_3) \\ \vdots & \\ \boldsymbol{\alpha^m_{k-\sum_{l=1}^{m-1} L_l}} & \text{se } \sum_{l=1}^{m-1} L_l < k < \sum_{l=1}^{m} L_l \\ \vdots & \\ \boldsymbol{\alpha^M_{k-\sum_{l=1}^{M-1} L_l}} & \text{se } \sum_{l=1}^{M-1} L_l < k < \sum_{l=1}^{M} L_l \end{cases}$$

e

$$b^k = \begin{cases} \beta^1_k & \text{se } 1 \leq k \leq L_1 \\ \beta^2_{k-L_1} & \text{se } L_1 < k \leq (L_1 + L_2) \\ \beta^3_{k-L_1-L_2} & \text{se } (L_1 + L_2) < k \leq (L_1 + L_2 + L_3) \\ \vdots & \\ \beta^m_{k-\sum_{l=1}^{m-1} L_l} & \text{se } \sum_{l=1}^{m-1} L_l < k < \sum_{l=1}^{m} L_l \\ \vdots & \\ \beta^M_{k-\sum_{l=1}^{M-1} L_l} & \text{se } \sum_{l=1}^{M-1} L_l < k < \sum_{l=1}^{M} L_l \end{cases}$$

quindi $a^k_i = 1$ se il nodo i è presente nel cluster k, $\boldsymbol{a^k} = (a^k_1 \dots a^k_N)$. L'utilizzo delle variabili γ^k_j consente di estendere le sommatorie sui possibili cluster di ogni mediana all'intero insieme di cluster ammissibili, in modo da ottenere una nuova formulazione equivalente:

$$min\ v = \sum_{j \in \mathcal{M}} \sum_{k \in Z} \gamma^k_j \sum_{i \in \mathcal{N}} d_{i,j} a^k_i z^k$$

s.t.

$$\sum_{j \in \mathcal{M}} \sum_{k \in Z} \gamma^k_j a^k_i z^k = 1 \ \forall i \in \mathcal{N}$$

$$\sum_{j \in \mathcal{M}} \sum_{k \in Z} \gamma^k_j b^k z^k = p$$

$$\sum_{k \in Z} \gamma^k_j z^k = 1 \ \forall j \in \mathcal{M}$$

Inoltre

- Ogni cluster può essere riferito ad una sola mediana: $\sum_{j \in \mathcal{M}} \gamma^k_j = 1$
- Fissato k

$$\sum_{j \in \mathcal{M}} \gamma^k_j a^k_i \ = \ a^k_i \sum_{j \in \mathcal{M}} \gamma^k_j \ = \ a^k_i$$

e

$$\sum_{j \in \mathcal{M}} \gamma^k_j b^k \ = \ b^k \sum_{j \in \mathcal{M}} \gamma^k_j \ = \ b^k$$

Indicando con $c^k = \sum_{j \in \mathcal{M}} \sum_{i \in \mathcal{N}} \gamma_j^k a_i^k d_{i,j}$ il costo di ogni cluster, si ottiene la riformulazione secondo Dantzig-Wolfe per CPMP, che prende il nome di *master problem* (MP):

MP)

$$min\ v = \sum_{k=1}^{K} c^k z^k$$

s.t.

$$\sum_{k \in Z} a_i^k z^k = 1\ \forall i \in \mathcal{N} \tag{5.5}$$

$$\sum_{k \in Z} b^k z^k = p \tag{5.6}$$

$$\sum_{k \in Z} \gamma_j^k z^k = 1\ \forall j \in \mathcal{M} \tag{5.7}$$

$$z^k \in \{0,1\}\ \forall k \in Z \tag{5.8}$$

Ogni *colonna* della matrice dei vincoli rappresenta, quindi, un cluster ammissibile. CPMP viene perciò riformulato come problema di *set partitioning* con dei *side constraints* (i vincoli di capacità).
Se $\beta_l^j = 0$ ($b^k = 0$) allora $\alpha_{i,l}^j = 0\ \forall i \in \mathcal{N}$, il che corrisponde ad un cluster vuoto. Essendo interessati solo ai clusters *potenzialmente* non vuoti, si suppone che ogni punto di Ω^j descriva un cluster ammissibile con $\beta_l^j = 1$ per la mediana j ($b^k = 1\ \forall k \in Z$). Sotto questa ipotesi è necessario porre i vincoli (5.7) in forma di $\leq$: devono essere associati clusters solo a nodi scelti come mediane. Si evitano, in tal modo, potenziali problemi di degenerazione nella risoluzione del rilassamento continuo. Il modello si riduce quindi a

MP)

$$min\ v = \sum_{k=1}^{K} c^k z^k \tag{5.9}$$

s.t.

$$\sum_{k \in Z} a_i^k z^k = 1\ \forall i \in \mathcal{N} \tag{5.10}$$

$$\sum_{k \in Z} z^k = p \tag{5.11}$$

$$\sum_{k \in Z} \gamma_j^k z^k \leq 1 \ \forall j \in \mathcal{M} \tag{5.12}$$

$$z^k \in \{0, 1\} \ \forall k \in Z \tag{5.13}$$

I vincoli assicurano che ogni nodo sia assegnato ad una mediana (5.10) e che il numero totale di clusters (e quindi di mediane) sia p (5.11). E' bene notare come siano stati rimossi da Z i clusters per ogni mediana j con valore $\beta_l^j = 0$, e quindi *sicuramente* vuoti; è tuttavia possibile avere clusters con $\beta_l^j = 1$ e $\alpha_{i,l}^j = 0 \ \forall i \in \mathcal{N}$.

5.2 Bound duale

Rilassamento di MP

I vincoli (5.5) possono essere posti in forma di $\geq$, poiché nella soluzione ottima nessun nodo è assegnato a due clusters; come affermato nel capitolo 3, esisterebbe in tal caso una soluzione ammissibile migliore, ottenibile rimuovendo il nodo dal cluster più svantaggioso. Il vincolo (5.6) può essere posto in forma di $\leq$: in caso di disuguaglianza stretta, è possibile completare la soluzione con dei clusters di costo nullo. Per mantenere la notazione uniforme, tutte le disuguaglianze in forma di $\leq$ sono portate in forma di $\geq$.
L-MP)

$$min\ \omega_{CG} = \sum_{k=1}^{K} c_k z^k \tag{5.14}$$

s.t.

$$\sum_{k \in Z} a_i^k z^k \geq 1\ \forall i \in \mathcal{N} \tag{5.15}$$

$$-\sum_{k \in Z} z^k \geq -p \tag{5.16}$$

$$-\sum_{k \in Z} \gamma_j^k z^k \geq -1\ \forall j \in \mathcal{M} \tag{5.17}$$

$$0 \leq z^k (\leq 1)\ \forall k \in Z \tag{5.18}$$

I vincoli (5.17) implicano i vincoli $z^k \leq 1$, che pertanto possono essere rimossi.

E' possibile dimostrare [Wols98] che questa formulazione fornisce risultati equivalenti a quelli ottenibili risolvendo all'ottimo il *problema duale lagrangiano* o aggiungendo *piani di taglio* al rilassamento lineare della formulazione originale. Se il problema (come nel caso di CPMP) non possiede l'*integrality property*, il bound valutato in tal modo può essere più stretto di quello ottenibile risolvendo il rilassamento lineare della formulazione originale [Wols98]. Purtroppo la cardinalità di Z (il numero di variabili in MP) cresce esponenzialmente nel numero di nodi del grafo ($O(M * 2^N)$). All'aumentare delle dimensioni dell'istanza, risolvere il rilassamento lineare di MP (L-MP) diventa improponibile.

Master Problem Ridotto

Siano $\boldsymbol{\lambda}$, $\boldsymbol{\delta}$ i vettori rispettivamente delle N variabili duali associate ai vincoli (5.15) e delle M variabili duali associate ai vincoli (5.17) sia ν la variabile duale associata al vincolo (5.16); il costo ridotto di una colonna k che rappresenti un assegnamento per la mediana j è

$$\pi_k = \sum_{i \in \mathcal{N}} (d_{i,j} - \lambda_i) a_i^k + \delta_j + \nu$$

dove $\lambda_i \geq 0 \ \forall i \in \mathcal{N}$, $\delta_j \geq 0 \ \forall j \in \mathcal{M}$, $\nu \geq 0$, essendo associati a vincoli di $\geq$.

Utilizzare il metodo del simplesso per risolvere L-MP renderebbe necessario valutare il costo ridotto di $|Z|$ colonne ad ogni iterazione. E' possibile, tuttavia, risolvere il rilassamento lineare con le tecniche di *column generation* precedentemente descritte: rimuovendo dall'insieme Z un opportuno insieme di colonne si ottiene un *Master Problem Ridotto* (RMP). RMP deve contenere almeno una soluzione di base ammissibile. Risolto il rilassamento lineare di RMP (L-RMP), l'informazione derivante dal valore delle variabili duali può essere sfruttata per determinare quali colonne di Z, non inserite in RMP, avendo costo ridotto negativo sono candidate ad entrare in base. Per calcolare il costo ridotto delle colonne si risolve il *pricing problem*, che consiste in M problemi di ottimizzazione volti ad individuare potenziali nuove colonne vantaggiose: $(\forall j \in \mathcal{M})$
PP)

$$min \ \pi_j = \sum_{i \in \mathcal{N}} (d_{i,j} - \lambda_i) a_i^j + \delta_j + \nu$$

$s.t.$

$$\boldsymbol{a^j} \in \Omega^j$$

ovvero M problemi di *knapsack* $(\forall j \in \mathcal{M})$
KP^j

$$min \ \pi_j = \sum_{i \in \mathcal{N}} (d_{i,j} - \lambda_i) a_i^j + \delta_j + \nu$$

s.t.

$$\sum_{i \in \mathcal{N}} w_i a_i^j \leq Q_j$$

$$a_i^j \in \{0,1\} \; \forall i \in \mathcal{N}$$

Risolto il pricing problem, può essere inserito in RMP l'intero insieme delle colonne individuate aventi costo ridotto negativo, o parte di esso, per valutare nuovamente il valore ottimo del rilassamento lineare ed ottenere un nuovo vettore di valori delle variabili duali.

Si procede iterativamente ricercando, come descritto, nuove colonne candidate ad entrare in base. E' importante notare come la successione dei valori del rilassamento lineare L-RMP ad ogni iterazione t di column generation ($\{\omega_{CG}\}_t$) sia monotona decrescente e converga al valore del rilassamento lineare di MP. Se la soluzione esatta dei problemi di knapsack non permette di individuare nessuna nuova colonna potenzialmente vantaggiosa significa che la soluzione di base corrente è ottima. In tal caso il valore ottenuto è un bound duale valido per MP (e quindi anche per CPMP).

5.3 Column generation e rilassamento Lagrangeano

I valori delle variabili duali $(\boldsymbol{\lambda}, \boldsymbol{\delta}, \nu)$ all'ottimo corrispondono ai valori ottimi dei moltiplicatori $(\boldsymbol{\lambda}, \boldsymbol{\eta}, \nu)$ del rilassamento Lagrangeano nella formulazione equivalente
LR)

$$min \sum_{i \in \mathcal{N}} \sum_{j \in \mathcal{M}} d_{ij} x_{ij} \tag{5.19}$$

s.t.

$$\sum_{j \in \mathcal{M}} x_{ij} \geq 1 \ \forall i \in \mathcal{N} \tag{5.20}$$

$$\sum_{i \in \mathcal{N}} w_i x_{ij} \leq Q_j y_j \ \forall j \in \mathcal{M} \tag{5.21}$$

$$y_j \leq 1 \ \forall j \in \mathcal{M} \tag{5.22}$$

$$\sum_{j \in \mathcal{M}} y_j \leq p \tag{5.23}$$

$$x_{ij} \in \{0, 1\} \ \forall i \in \mathcal{N}, \forall j \in \mathcal{M} \tag{5.24}$$

$$y_j \in \{0, 1\} \ \forall j \in \mathcal{M} \tag{5.25}$$

$\boldsymbol{\lambda}$ e ν corrispondono ai vincoli (5.20) e (5.23), come visto nel capitolo 2, $\boldsymbol{\eta}$ è il vettore dei moltiplicatori associato ai vincoli (5.22), portati nella forma

$$-y_j \geq -1 \ \forall j \in \mathcal{M}$$

ed i valori delle variabili duali al termine delle iterazioni di column generation corrispondono ai valori ottimi dei moltiplicatori per il rilassamento Lagrangeano.In questo senso, le iterazioni di column generation sostituiscono il metodo del sottogradiente per l'aggiornamento dei moltiplicatori nel rilassamento Lagrangeano.
Anche se, sfruttando l'informazione duale come valore per i moltiplicatori, la successione dei valori del rilassamento lineare di RMP ($\{\omega_{CG}\}_t =$

$\omega^1_{CG},\ \omega^2_{CG},\ \ldots$) e la successione dei valori del bound Lagrangeano ($\{\omega_{LR}\}_t = \omega^1_{LR},\ \omega^2_{LR},\ \ldots$) convergono allo stesso risultato, la monotonia della prima non implica quella della seconda. Tuttavia, per ogni scelta dei moltiplicatori, il bound Lagrangeano è non superiore al valore del rilassamento lineare di RMP.

5.3.1 Bound duale

Come esposto nella sezione (5.2) è un bound duale valido solo il valore del rilassamento di RMP al termine del processo di column generation (quando non è possibile individuare nessuna nuova colonna di costo ridotto negativo). Per alcuni problemi il numero di passi necessari potrebbe essere molto elevato; inoltre i valori ω^t_{CG} decrescono rapidamente durante le prime iterazioni per variare via via più lentamente.
E' possibile sfruttare la proprietà di equivalenza per i moltiplicatori per ottenere bounds duali validi *durante* il processo di column generation: ad ogni iterazione t è un bound duale valido per CPMP

$$\omega^t_{LD} = \max\{\textstyle\sum_{j\in\mathcal{M}} \min\{\pi_j, 0\} + \sum_{i\in\mathcal{N}} \lambda_i - \sum_{j\in\mathcal{M}} \delta_j - p\nu,\ \omega^{t-1}_{LD}\} =$$

$$= \max\{\omega^1_{LR} \ldots \omega^t_{LR}\}$$

dove $\pi_j = \sum_{i\in\mathcal{N}}(d_{i,j} - \lambda_i)a^k_i + \delta_j + \nu$ è il costo ridotto della colonna generata risolvendo all'ottimo KP^j. Infatti i valori di π_j sono ottenuti in modo del tutto analogo ai valori π_j del rilassamento Lagrangeano nel capitolo 4. Disponendo del valore Z^* di una soluzione ammissibile, il processo di column generation può essere terminato quando $Z^* \leq \omega^t_{LD}$.
Sia ω^t_{CG} il valore del rilassamento lineare di RMP all'iterazione t-esima. Essendo disposti ad accettare bounds duali meno stretti, per la monotonia di $\{\omega^t_{CG}\}$, la condizione di terminazione potrebbe essere posta come $\omega^t_{CG} - \omega^t_{LD} < \Delta$ ($\Delta \geq 0$).

5.3.2 Analisi di sensitività e preprocessing

Come per il rilassamento Lagrangeano, anche applicando tecniche di column generation è possibile effettuare un'analisi di sensitività sulla soluzione par-

ziale corrispondente ad ogni sottoproblema. Ottenuto un bound duale ω_{LR} ed una soluzione ammissibile Z^*,

- Se $\exists j \mid \pi_j > 0,\ \omega_{LR} + \pi_j \geq Z^*$, imponendo $y_j = 1$ si otterrebbe un bound duale superiore al bound primale. Quindi, per poter ottenere soluzioni migliori del bound primale, deve necessariamente essere $y_j = 0$ (e quindi $x_{ij} = 0\ \forall i \in \mathcal{N}$).

- Se $\exists j \mid \pi_j < 0,\ \omega_{LR} - \pi_j \geq Z^*$, imponendo $y_j = 0$ si otterrebbe, come prima, un bound duale superiore al bound primale. Quindi necessariamente $y_j = 1$.

L'analisi richiede tempo $O(M)$, dato che i valori π_j sono già stati determinati durante il processo di column generation. Come nel caso del rilassamento Lagrangeano, alla formulazione standard sono aggiunti i vincoli

$$x_{ij} \leq y_j\ \forall i \in \mathcal{N},\ \ \forall j \in \mathcal{M} \tag{5.26}$$

e considerare $y_j = 0$ implica che debba essere $x_{ij} = 0\ \forall i \in \mathcal{N}$.

Nella tabelle 5.1 è riportato un confronto fra i tempi di calcolo su diverse istanze, nei casi in cui

1. non venga effettuata analisi di sensitività
2. venga effettuata una sola volta utilizzando i moltiplicatori ottimi (al termine delle iterazioni di column generation)
3. venga effettuata ad ogni iterazione t in cui vi sia un miglioramento per ω_{LD}
4. venga effettuata ad ogni iterazione di column generation

Il branching avviene per introduzione di vincoli di *subset sum* (politica "forbid", descritta nella sezione 7) e ricerca best first. Come nei test precedenti, è stato posto un limite di 3600 secondi di tempo di CPU per terminare la computazione. I casi in cui questo limite è stato superato sono indicati

Istanze di Classe A

Problema	Nessuna		Al termine di CG		Iteraz. miglioranti		Ogni iterazione	
	$Z^*(\omega_{CG})$	tempo (s)	$Z^*(\omega_{CG})$	tempo (s)	$Z^*(\omega_{CG})$	tempo (s)		
CCPX1	713	17.5	713	17.8	713	17.7	713	17.8
CCPX3	751	12.9	751	12.4	751	12.5	751	12.4
CCPX10	829	53.0	829	48.1	829	45.9	829	46.2
CCPX13	1026	85.8	1026	78.0	1026	78.7	1026	77.9
CCPX16	954	83.8	954	67.8	954	66.5	954	67.3

Istanze di Classe B

Problema	Nessuna		Al termine di CG		Iteraz. miglioranti		Ogni iterazione	
	$Z^*(\omega_{CG})$	tempo (s)	$Z^*(\omega_{CG})$	tempo (s)	$Z^*(\omega_{CG})$	tempo (s)		
CCPX1	383	17.0	383	16.1	383	16.6	383	17.1
CCPX3	405	28.3	405	28.3	405	27.4	405	27.2
CCPX10	461	1482.4	461	1191.7	461	1199.6	461	1221.0
CCPX13	(562;542.333)	-	(560;542.3)	-	(561;542.267)	-	(561;542.3)	-
CCPX16	(537;527.4)	-	(538;527.374)	-	(537;527.35)	-	(537;527.383)	-

Istanze di Classe C

Problema	Nessuna		Al termine di CG		Iteraz. miglioranti		Ogni iterazione	
	$Z^*(\omega_{CG})$	tempo (s)	$Z^*(\omega_{CG})$	tempo (s)	$Z^*(\omega_{CG})$	tempo (s)		
CCPX1	266	9.7	266	9.1	266	9.8	266	9.7
CCPX3	311	6.8	311	7.4	311	6.7	311	6.6
CCPX10	(458;456.667)	-	(458;456.917)	-	(458;456.893)	-	(458;457)	-
CCPX13	(412;410.5)	-	(412;410.5)	-	(412;410.5)	-	(412;410.5)	-
CCPX16	428	259.1	428	211.8	428	265.2	428	258.5

Tabella 5.1: Analisi di sensititità

con il simbolo"-"; in queste situazioni sono riportati anche il bound primale ed il bound duale (fra parentesi).

L'introduzione dell'analisi descritta comporta, nei casi dei problemi CCPX13-A, CCPX16-A e CCPX10-B, un significativo miglioramento delle prestazioni dell'algoritmo. In generale non ci sono differenze sensibili nei tempi di calcolo effettuando l'analisi solo al termine delle iterazioni di column generation, ad ogni iterazione o ad ogni miglioramento del bound ω_{LD}.

Si può notare come per l'istanza CCPX16-C si registri un *incremento* dei tempi di calcolo effettuando analisi di sensitività ad ogni iterazione: contro l'intuizione, fissare alcune variabili y_j può non essere vantaggioso, poiché equivale a trasformare il problema in esame in uno dei suoi figli in un ipotetico schema di branching binario, orientato alla definizione delle mediane (mentre la politica adottata prevede l'introduzione di un diverso tipo di vincoli). La determinazione del valore di una variabile comporta la necessità di prolungare

la valutazione del rilassamento lineare; il calcolo del bound può risultare quindi più oneroso della valutazione di entrambi i sottoproblemi nello schema di branching "forbid".

5.4 Problema di pricing

Il problema di determinare il vettore $\boldsymbol{a^j}$ degli elementi inseriti nel cluster j in modo che il costo ridotto della colonna sia negativo è analogo al problema di separazione negli algoritmi cutting planes. Il pricing problem è soggetto ai vincoli di capacità. Come descritto, nel caso di CPMP determinare le soluzioni del pricing problem corrisponde a risolvere problemi di knapsack, e quindi problemi $\mathcal{NP}$-hard (in senso debole).Un problema gode dell'integrality property se il corrispondente problema di separazione, o in maniera duale il pricing problem, può essere risolto in tempo polinomiale nella dimensione dell'istanza (così come il valore del rilassamento Lagrangeano, per ogni vettore di moltiplicatori). CPMP, quindi, non ricade in questa categoria.

5.4.1 Preprocessing

Se nel problema sono presenti elementi con coefficiente negativo (nel caso del problema di knapsack in forma di massimizzazione), essi possono essere trascurati, poiché non faranno mai parte della soluzione ottima. La rimozione dal problema degli elementi con coefficiente negativo può essere compiuta in tempo lineare tramite ispezione dei loro valori $(\lambda_i - d_{ij})$.

5.4.2 Soluzione euristica

Per trovare colonne di costo ridotto negativo non è sempre necessario risolvere all'ottimo il pricing problem, ma anche algoritmi euristici possono permettere di individuare colonne vantaggiose. La letteratura inerente la soluzione euristica o approssimata di problemi di knapsack è molto ampia. Si è scelto di privilegiare euristiche semplici e veloci, data l'intenzione di utilizzarle ad ogni iterazione del processo di generazione di nuove colonne.

Euristiche forward e backward

Non essendo gli elementi ordinati, si possono ottenere soluzioni ammissibili in tempo $O(N)$ inserendo nel knapsack di volta in volta qualsiasi elemento il cui peso non ecceda la capacità rimanente. Il valore di ogni elemento non

viene preso in considerazione per la scelta. I pesi degli elementi possono essere esaminati nell'ordine naturale ($w_1 \dots w_N$) (euristica forward) oppure nell'ordine inverso ($w_N \dots w_1$) (euristica backward). E' possibile eseguire entrambe le scansioni e considerare la migliore delle due soluzioni ottenute.

Euristica con ordinamento

Basandosi sull'*efficienza* $e_i^j = \frac{\lambda_i - d_{ij}}{w_i}$ di ogni elemento a_i^j, si può eseguire un'altra euristica:

- si ordinano gli elementi per efficienza decrescente, ottenendo una successione $\{s_1 \dots s_N\}$
- si valuta ogni elemento secondo l'ordine stabilito: se non viene violato il vincolo di capacità, si inserisce l'elemento nel knapsack.

L'ordinamento degli elementi ha complessità $O(N \log N)$ e la ricerca descritta, nel caso siano stati ordinati, può essere effettuata in tempo lineare.

5.4.3 Soluzione esatta

Se le euristiche descritte non generano colonne di costo ridotto negativo, è necessario ricorrere alla soluzione esatta del problema di knapsack. Tecniche molto efficienti sono state sviluppate, tra gli altri, da Martello e Toth [MT89] e Pisinger [Pisi95]. Si è scelto di utilizzare una versione adattata dell'algoritmo *MINKNAP* di Pisinger, che combina programmazione dinamica a tecniche di bounding e riduzione, sfruttando particolari condizioni sul *core* del problema, ovvero un sottoinsieme di variabili il cui valore, all'ottimo, differisce dal valore del rilassamento lineare.

Nell'algoritmo di Pisinger il *core* viene determinato *durante* il calcolo; le variabili vengono partizionate in tre insiemi: S, l'insieme delle variabili fissate a 1, T, l'insieme delle variabili fissate a 0 e Θ, il core del knapsack. Inizialmente $\Theta = \{\xi\}$, dove ξ è il *break item*, ovvero l'unica variabile a valore frazionario nel rilassamento lineare; S e T contengono rispettivamente le variabili a 1 ed a 0 nella soluzione del rilassamento lineare. Variabili che

potrebbero condurre a miglioramenti del bound duale sono iterativamente inserite nel core e rimosse da S e T. Ad ogni iterazione si valuta un bound duale *debole* u_i^w per ogni variabile non inserita in Θ, che indica il valore massimo della soluzione ottenibile inserendo la variabile i nel core (ovvero complementandone il valore). Nel caso di problemi a coefficienti interi, se u_i^w non supera di almeno un'unità il valore z^* della miglior soluzione primale incontrata fino a quel momento, la variabile i può essere fissata al valore corrente (0 se $i \in T$, 1 se $i \in S$). Il problema si riduce a determinare una soluzione ottima per il sottoproblema di knapsack riferito alle variabili presenti in Θ. Quest'operazione viene effettuata tramite programmazione dinamica; il numero di stati da valutare cresce esponenzialmente rispetto alla cardinalità del core. Per questo motivo, per ogni variabile per cui $u_i^w \geq z^* + 1$ viene valutato un bound duale *forte* u_i^s, più stringente, anche se più oneroso computazionalmente (è richiesto l'ordinamento delle variabili sulle quali deve essere valutato). Anche in questo caso, se $u_i^s < z^* + 1$ la variabile i viene fissata al valore corrente. Soluzioni primali z^* sono raggiunte utilizzando l'euristica con ordinamento.

5.4.4 Approssimazione o scalatura

L'algoritmo MINKNAP è stato concepito per problemi a coefficienti e pesi interi. Nel nostro caso, però, nel pricing problem devono essere trattati valori frazionari. Anche per istanze di CPMP a coefficienti interi, infatti, il valore delle variabili duali nel rilassamento lineare è frazionario.

Scalatura dei coefficienti

In un problema di programmazione lineare, la soluzione di base corrente $\mathbf{z_{vb}}$ è tale che

$$B\mathbf{z_{vb}} = \mathbf{b}$$

dove $\mathbf{b}$ e B sono ottenuti rispettivamente dal vettore dei termini noti e dalla matrice dei vincoli, eliminando le colonne delle variabili non in base e le righe dei vincoli non attivi.

Il valore delle variabili per la soluzione in base sono ottenuti come

$$\mathbf{z_{vb}} = B^{-1}\mathbf{b}$$

Quindi valori frazionari possono derivare solo dalla divisione dei coefficienti per il determinante di B, nel calcolo della matrice inversa.

Per ottenere coefficienti interi è dunque necessario calcolare il determinante di B e moltiplicare tutti i coefficienti per il valore ottenuto. Il numero di variabili in base è pari al numero di vincoli in RMP, quindi potenzialmente molto alto. Nei test condotti, inoltre, la densità di B è rimasta intorno al 20%: minore è la densità della matrice, minore è il carico computazionale necessario per calcolarne il determinante. Per il calcolo di quest'ultimo è stata utilizzata la libreria C++ "NewMath" (http://webnz.com/robert/cpp_lib.htm).

Approssimazione della soluzione

E' possibile utilizzare MINKNAP per risolvere problemi di knapsack a coefficienti frazionari, sostituendo i test

$$u_i^w \geq z^* + 1$$

$$u_i^s \geq z^* + 1$$

basati sugli upper bounds debole e forte per l'inserimento della variabile nel core con

$$u_i^w > z^*$$

$$u_i^s > z^*$$

In alternativa, si possono ottenere soluzioni approssimate sostituendo con

$$u_i^w > z^* + \varepsilon$$

$$u_i^s > z^* + \varepsilon$$

con i quali una variabile che non possa produrre un miglioramento del bound superiore a ε viene esclusa dal core. Nel caso peggiore, tutte le variabili potrebbero migliorare il bound, ma vengono tralasciate perché l'entità del miglioramento sarebbe inferiore ad ε. In tal caso, la soluzione differisce dall'ottimo al più $N_{KP} * \varepsilon$, dove N_{KP} è il numero di variabili del problema di knapsack. L'algoritmo che utilizza quest'ultima alternativa viene indicato, in seguito, come ε-approssimato.

Confronto tra approssimazione e scalatura

Ricorrere ad un algoritmo ε-approssimato per la soluzione dei knapsack risulta meno oneroso in termini di tempo di calcolo, ma porta ad un peggioramento del bound duale ad ogni nodo, perché il valore della soluzione ottenuta con algoritmo approssimato deve essere *peggiorato*, come descritto, di $N_{KP} * \varepsilon$.

E' possibile confrontare le prestazioni nei due casi, per alcuni valori di ε: in tabella 5.2 sono riportati il numero di sottoproblemi analizzati ed il tempo impiegato per esaurire il calcolo su alcune istanze di classe A.

Nel confronto tra approssimazione e scalatura proposto, RMP è inizialmente vuoto, l'enumerazione implicita avviene tramite branching *forbid* con politica di esplorazione *best first*, migliorata effettuando analisi di sensitività e preprocessing, ed inserendo in RMP tutte le colonne di costo ridotto negativo individuate.

Problema		*Scalatura*		$\varepsilon = 10^{-6}$		$\varepsilon = 10^{-9}$	
		# s.p.	tempo (s)	# s.p.	tempo (s)	# s.p.	tempo (s)
N = 50	CCPX1	33	407,494	39	21,926	39	21,595
	CCPX2	1	27,345	1	5,276	1	4,836
	CCPX3	3	42,329	3	12,858	3	13,33
	CCPX10	195	4384,822	203	79,765	203	79,983

Tabella 5.2: Confronto tra approssimazione e scalatura senza utilizzo di euristiche

Anche se la scalatura dei coefficienti porta alla valutazione di un numero di nodi minore, il tempo necessario alla valutazione dei determinanti influisce in modo molto negativo sulle prestazioni globali dell'algoritmo. La variazione di tre ordini di grandezza del coefficiente di approssimazione ε non determina

miglioramenti apprezzabili nel bound (il numero di nodi valutati rimane lo stesso). Contro l'intuizione, tuttavia, un'approssimazione più stringente (ε minore) conduce, in alcuni casi, a tempi di calcolo inferiori, anche se il numero di nodi rimane lo stesso: migliore è l'approssimazione, migliori sono le colonne generate e migliori i valori duali ottenuti dal rilassamento lineare.

Utilizzo di algoritmi euristici

Le iterazioni di column generation con algoritmo esatto possono essere precedute da iterazioni in cui si utilizzano solo algoritmi euristici per individuare colonne di costo ridotto negativo. E' possibile ridurre ulteriormente il numero di chiamate all'algoritmo di soluzione esatta invocando l'euristica ad ogni ciclo di generazione delle colonne e ricorrendo alla soluzione esatta solo in caso non si riesca ad individuare una colonna opportuna. In tal modo viene complessivamente ridotto anche il numero di valutazioni del determinante necessarie. Minimizzando il numero di chiamate dell'algoritmo esatto si offre un vantaggio alla politica di scalatura dei coefficienti (in tabella 5.3 sono riportati risultati per istanze di classe A).

Problema		*Scalatura*		$\varepsilon = 10^{-6}$		$\varepsilon = 10^{-9}$		$\varepsilon = 10^{-12}$	
		# s.p.	tempo (s)	# s.p.	tempo (s)	# s.p.	tempo (s)	# s.p.	tempo(s)
N = 50	CCPX1	31	57,385	41	24,174	41	23,993	41	23,186
	CCPX2	1	6,405	1	5,385	1	5,503	1	5,413
	CCPX3	3	15,826	3	14,366	3	14,356	3	14,522
	CCPX10	195	1058,295	173	73,963	173	74,431	173	73,158
N = 100	CCPX11	83	1050,219	81	182,288	81	179,257	81	178,365
	CCPX12	137	4758,156	217	394,717	217	385,996	217	392,476
	CCPX13	19	309,256	11	96,384	11	95,444	11	93,964

Tabella 5.3: Approssimazione e scalatura, con euristiche, su grafi di 50 e 100 nodi

La differenza di prestazioni fra le due politiche si riduce, ma i tempi di calcolo per la soluzione tramite algoritmo ε-approssimato risultano di nuovo migliori. Anche in tal caso per valori di ε diversi si ottengono variazioni minime. Il divario tra le due politiche è destinato ad accentuarsi all'aumentare delle dimensioni del problema (e quindi alle dimensioni delle matrici su cui valutare il determinante).

E' da notare, infine, come per problemi di dimensioni ridotte la soluzione dell'algoritmo ε-approssimato sia sperimentalmente *competitiva* con la soluzione euristica: non si hanno miglioramenti importanti in termini di tempi di calcolo utilizzando o meno algoritmi euristici per la soluzione dei problemi di knapsack del pricing problem, prima dell'esecuzione dell'ε-approssimato.

5.5 Bound primale

Come affermato in precedenza, individuare soluzioni ammissibili durante il processo di enumerazione implicita contribuisce a migliorare le prestazioni del metodo.

Per ottenere bounds primali è stata utilizzata l'euristica descritta nel capitolo 2. Sono state esplorate cinque diverse politiche di definizione dei coefficienti f_{ij} e di scelta delle mediane, effettuando anche le fasi di ricerca locale (3 e 4):

5.5.1 Scelta dei coefficienti

Di seguito (Tabella 5.4) è proposto un confronto fra i risultati ottenuti eseguendo l'euristica per le diverse scelte dei coefficienti e delle mediane:

1. ponendo $f_{ij} = -d_{ij}$, $\psi_j = \sum_{i \in \mathcal{N}, i \neq j} \frac{1}{d_{ij}}$ ed estraendo le mediane casualmente con probabilità proporzionale a ψ_j

2. ponendo $f_{ij} = \frac{1}{d_{ij}}$, $\psi_j = \sum_{i \in \mathcal{N}, i \neq j} \frac{1}{d_{ij}}$ ed estraendo le mediane casualmente con probabilità proporzionale a ψ_j

3. come proposto da Savelsbergh in [Save95] si può sfruttare l'informazione ottenibile durante le iterazioni di column generation, definendo

   $$f_{ij} = \sum_{k \in Z} \gamma_j^k a_i^k z^k$$

   $$\psi_j = \sum_{i \in \mathcal{N}} f_{ij}$$

   e designando come mediane i p nodi aventi migliori coefficienti ψ_j, eseguendo l'algoritmo euristico una sola volta al termine del processo di column generation

4. ponendo, come sopra, $f_{ij} = \sum_{k \in Z} \gamma_i^k y_j^k z^k$, $\psi_j = \sum_{i \in \mathcal{N}} f_{ij}$ e designando come mediane i p nodi aventi migliori coefficienti ψ_j, eseguendo l'algoritmo euristico ad ogni iterazione del processo di column generation

5. generando un set iniziale di colonne con le soluzioni ottenute dall'euristica con coefficienti (1) e (2) ed eseguendo l'euristica con i coefficienti come (4) ad ogni iterazione di column generation

Le prestazioni dell'euristica non hanno variazioni di rilievo utilizzando la prima o la seconda politica di assegnamento dei coefficienti f_{ij}, anche se ponendo $f_{ij} = -d_{ij}$ si hanno risultati in media migliori. Sfruttare l'informazione derivante dalle iterazioni di column generation offre vantaggi rispetto ad altre scelte dei coefficienti. In quattro degli otto casi analizzati la soluzione ottima viene trovata nel nodo radice. Un altro vantaggio è la possibilità di eseguire l'euristica ad ogni iterazione con coefficienti aggiornati. La politica (4) consente un notevole miglioramento del bound primale. Ricorrendo alla strategia (5) si ottiene, in generale, un ulteriore incremento della performance (solitamente di modesta entità). E' da notare come nel caso di CCPX10 il bound ottenuto in tal modo sia peggiore rispetto a quanto offerto da (4): un simile risultato non è incongruente con le considerazioni presentate, poiché inizializzare "male" il pool di colonne in RMP può contribuire ad ottenere rilassamenti lineari dalle soluzioni più frazionarie, dalle quali è più difficile risalire a soluzioni ammissibili vicine all'ottimo. Soluzioni molto frazionarie possono influire in modo negativo anche nella scelta delle variabili di branch. Si può ovviare al problema rimuovendo da RMP le colonne in base meno determinanti nella definizione della soluzione del rilassamento lineare. Possono essere scelte le colonne aventi valore di z^k inferiore ad una soglia, oppure aventi i minimi valori di costo ridotto (in modulo) o ancora le colonne generate meno recentemente, supponendo che gli ultimi clusters generati siano più desiderabili. Rimuovendo colonne di costo ridotto negativo, tuttavia, si rallenta la convergenza all'ottimo del rilassamento lineare.

Questa possibilità non è stata esplorata a fondo, poiché la valutazione del bound duale risulterebbe troppo onerosa e dato che in molti casi una soluzione vicina all'ottimo viene individuata durante la valutazione del nodo radice.

Problema	*Scelta di coefficienti e mediane*	*Bound Primale Trovato*		
		Migliore	*Peggiore*	*Medio (100 iter.)*
CCPX1 (N = 50)	(1)	820	2135	1149.97
	(2)	845	1994	1158.62
	(3)		713	
	(4)		713	
	(5)		713	
	Soluzione ottima: 713			
CCPX2 (N = 50)	(1)	858	1482	1092.51
	(2)	858	1419	1109.22
	(3)		740	
	(4)		740	
	(5)		740	
	Soluzione ottima: 740			
CCPX3 (N = 50)	(1)	844	1638	1188.82
	(2)	839	1737	1210.67
	(3)		1289	
	(4)		788	
	(5)		785	
	Soluzione ottima: 751			
CCPX4 (N = 50)	(1)	719	1657	1101.73
	(2)	700	1712	1116.45
	(3)		651	
	(4)		651	
	(5)		651	
	Soluzione ottima: 651			
CCPX5 (N = 50)	(1)	733	1841	1172.21
	(2)	763	1881	1188.69
	(3)		664	
	(4)		664	
	(5)		664	
	Soluzione ottima: 664			
CCPX10 (N = 50)	(1)	996	1841	1354.6
	(2)	972	1925	1387.4
	(3)		897	
	(4)		892	
	(5)		895	
	Soluzione ottima: 829			
CCPX11 (N = 100)	(1)	1309	2713	1720.36
	(2)	1312	2837	1754.51
	(3)		1331	
	(4)		1009	
	(5)		1007	
	Soluzione ottima: 1006			
CCPX12 (N = 100)	(1)	1287	2658	1794.02
	(2)	1294	2682	1831.92
	(3)		1038	
	(4)		1026	
	(5)		994	
	Soluzione ottima: 966			

Tabella 5.4: Prestazioni dell'euristica primale

5.6 Enumerazione implicita

Descritti i metodi per ottenere bounds primali e duali e per fissare il valore di alcune variabili tramite analisi di sensitività, bisogna ora descrivere la strategia di enumerazione implicita.

Ricordando che l'enumerazione implicita equivale alla visita di un albero di branching in cui ogni nodo ha F figli, possono essere adottati diversi criteri per la generazione dei sottoproblemi-figli $S_1 \dots S_F$ dal sottoproblema-padre S_0, e per la strategia di ricerca nell'albero.

5.6.1 Strategia di ricerca

Depth First

E' possibile valutare per primi gli ultimi sottoproblemi generati, esplorando l'albero di branching con politica *depth first*. In tal modo si possono raggiungere presto soluzioni ammissibili (le foglie dell'albero di branching) che costituiscono un bound primale: ciò è importante poiché se i vincoli di capacità del problema sono molto stretti potrebbe essere molto difficile per l'euristica primale descritta nel capitolo 2 terminare con successo nei nodi intermedi. Inoltre, come descritto nel capitolo seguente, è utile limitare il numero di clusters in RMP, cancellando colonne non vantaggiose. Clusters "buoni" generati per il sottoproblema padre sono probabilmente vantaggiosi anche per il problema figlio: valutando quest'ultimo subito dopo il padre, si rende improbabile la perdita di tali clusters per cancellazione. Considerazioni analoghe possono essere effettuate per le soluzioni di base: in ogni sottoproblema figlio è possibile valutare il rilassamento lineare partendo dalla soluzione di base ottima del padre, che è, in generale, vicina all'ottimo del figlio. Anche se, durante il *backtracking*, quest'informazione viene persa, la valutazione può partire dalla base ottima per l'ultimo nodo visitato prima del backtracking, che ha con molta probabilità comunque una struttura simile al sottoproblema in esame.

Best First

Ricorrendo ad una politica di esplorazione *best first* dell'albero di branching, si dà precedenza alla valutazione di sottoproblemi dal bound duale più promettente. Si affida all'euristica primale il compito di raggiungere soluzioni ammissibili partendo da soluzioni parziali *probabilmente* vicine all'ottimo, descritte dai nodi più promettenti. Adottando questa strategia è possibile rimuovere da RMP clusters aventi costo ridotto superiore alla differenza tra bound primale e bound duale, poiché sicuramente svantaggiosi. Tuttavia non è possibile risolvere il rilassamento lineare di un sottoproblema partendo dalla base del padre senza memorizzare esplicitamente le colonne in base all'ottimo ed, in generale, alla valutazione di un sottoproblema segue quella di un sottoproblema in cui sono state fissate variabili differenti (e quindi con struttura differente).

5.6.2 Branching

Branching sulle possibili mediane

Come proposto nel capitolo 3, l'enumerazione implicita può avvenire generando un albero di branching in cui ogni sottoproblema è ottenuto dal padre imponendo che un nodo i^b sia assegnato rispettivamente ad ogni mediana $j \in \mathcal{M}$, ovvero ponendo per ogni $j \in \mathcal{M}$:

$$S_j = S_0 \cap \{(\boldsymbol{x^j}, y_j) \mid x_{i^b j} = 1,\ \ x_{i^b k} = 0\ \forall k \neq j\}$$

Per il problema radice $F = |\mathcal{M}|$, successivamente il numero di candidati mediane descresce. Imporre che un nodo i^b sia assegnato alla mediana j implica che il nodo j debba essere designato come mediana ($y_j = 1$).

Essendo F indipendente dalla variabile su cui si sceglie di effettuare il branching, è conveniente scegliere come nodo i^b il nodo assegnato con valore frazionario al maggior numero di diverse mediane nel rilassamento lineare. Poiché il numero di potenziali mediane potrebbe essere molto elevato (in generale N), dovranno essere valutati molti sottoproblemi-figlio. Tuttavia, in questo modo, CPMP diventa in breve un problema di assegnamen-

to generalizzato (si veda capitolo 2), che può essere risolto all'ottimo più rapidamente.

Alcuni risultati sono riportati nella tabella 5.5: questa tecnica non conduce a risultati migliori delle strategie di branching analizzate di seguito.

problema (N = 50)	p = 5		p = 12		p = 20	
CCPX1	713	216.6	383	331.3	266	371.5
CCPX3	751	43.8	405	759.9	311	1446.3
CCPX10	829	314.6	(505;460.5)	-	(479;454.4)	-

Tabella 5.5: Branching sulle possibili mediane - Best First

Branching guidato dal rilassamento lineare

Per ogni nodo i, sia M_i l'insieme delle mediane a cui il nodo i è associato nella soluzione di L-RMP. La cardinalità di M_i indica quanto è frazionario l'assegnamento del nodo di indice i alle mediane.
Scelto i^b, si possono generare $|M_{i^b}|$ sottoproblemi in cui i^b è assegnato ad ogni mediana $m_k \in M_{i^b}$ ed un ulteriore sottoproblema in cui è impedito che i^b sia assegnato ad una mediana in M_{i^b}:

$$S_j = S_0 \cap \{(\boldsymbol{x^j}, y_j) \mid x_{i^b j} = 1,\ x_{i^b k} = 0\ \forall k \neq j\} \forall j \in M_{i^b}$$

e

$$S_{|M_{i^b}|+1} = S_0 \cap \{(\boldsymbol{x^j}, y_j) \mid x_{i^b k} = 0\ \forall k \in M_{i^b}\}$$

Scegliendo il nodo $i^b = argmin_{i \in \mathcal{N}}\{|M_i|\}$ si minimizza il numero di figli per ogni sottoproblema. In generale un branching di questo tipo dovrebbe essere più efficace del precedente: poiché diverse soluzioni probabilmente scadenti sono valutate contemporaneamente nell'ultimo sottoproblema ed il numero di valutazioni necessarie può essere notevolmente ridotto.

Tuttavia anche le prestazioni di questa strategia sono inferiori a quelle delle strategie seguenti. I risultati sono riportati nella tabella 5.6.

problema (N = 50)	p = 5		p = 12		p = 20	
CCPX1	713	33.9	383	175.6	(268;265.5)	-
CCPX3	751	19.0	405	340.5	311	2683.6
CCPX10	829	171.1	(498;458.9)	-	(581;446.1)	-

Tabella 5.6: Branching guidato dal rilassamento lineare - Best First

Branching dicotomico

Applicando tecniche che consentano di trovare bounds duali stringenti, ricorrere ad alberi di branching binari rende *potenzialmente* necessario un minor numero di valutazioni di sottoproblemi per terminare la computazione. Le potenzialità di tecniche basate sulla bipartizione dell'insieme delle soluzioni sono già state sottolineate da alcuni autori ([Beas92]).

Come nel secondo livello di branching dell'algoritmo basato su rilassamento Lagrangeano, una strategia per ridurre il numero di figli per ogni sottoproblema può essere, scelto un nodo i^b ed una mediana j^b, ricorrere alla dicotomia

$$S_l = S_0 \cap \{(\boldsymbol{x^j}, y_j) \mid x_{i^b j^b} = 0\}$$

e

$$S_r = S_0 \cap \{(\boldsymbol{x^j}, y_j) \mid x_{i^b j^b} = 1\}$$

La seconda condizione è più forte (poiché implica $y_{j^b} = 1$) e l'albero di branching risulta poco bilanciato. Assume molta importanza la scelta di i^b e j^b. Si è scelto di selezionare la variabile $x_{i^b j^b}$ che ha il valore più vicino a $\frac{1}{2}$ nella soluzione del rilassamento lineare. Se più variabili sono candidate, viene scelta quella avente coefficiente $d_{i^b j^b}$ minore (l'assegnamento "più desiderabile"): è anche possibile selezionare simmetricamente la variabile con coefficiente $d_{i^b j^b}$ maggore, ma imporre che un nodo sia assegnato ad una mediana desiderabile ha maggior impatto nella definizione della soluzione rispetto ad imporre che non sia assegnato ad una mediana poco desiderabile. Di seguito (tabelle 5.7 e 5.8) sono presentati i risultati sulle istanze di classe A, B e C sia per la strategia di ricerca depth first che per quella best first.

Problema		classe A		classe B		classe C	
		$Z^*(\omega_{CG})$	tempo (s)	$Z^*(\omega_{CG})$	tempo (s)	$Z^*(\omega_{CG})$	tempo (s)
N = 50	CCPX1	713	11.6	383	94.9	266	102.8
	CCPX2	740	4.7	412	38.9	298	223.0
	CCPX3	751	11.7	405	81.6	311	282.9
	CCPX4	651	10.2	384	2118.4	277	9.7
	CCPX5	664	5.7	429	105.3	356	5.6
	CCPX6	778	5.7	482	291.7	370	5.4
	CCPX7	787	37.7	445	1999.9	358	154.9
	CCPX8	(821;817.433)	-	403	85.0	312	1137.5
	CCPX9	715	32.6	436	130.6	412	560.7
	CCPX10	829	43.2	461	1946.7	(483;455.389)	-
N = 100	CCPX11	1006	739.7	(599;540.333)	-	(421;410.333)	-
	CCPX12	966	2108.8	(515;504)	-	(408;372.431)	-
	CCPX13	1026	181.7	(584;547.533)	-	(422;411)	-
	CCPX14	982	2433.2	(579;538.733)	-	(442;421)	-
	CCPX15	1091	3515.7	583	715.4	(540;496)	-
	CCPX16	954	127.2	(623;529.333)	-	(439;428)	-
	CCPX17	(1042;1034)	-	542	1689.9	(455;437)	-
	CCPX18	1043	2985.2	508	1723.1	(473;449)	-
	CCPX19	1031	300.9	(591;542.167)	-	(493;447.733)	-
	CCPX20	(1102;983.738)	-	(696;537.024)	-	(506;485.417)	-

Tabella 5.7: Branch dicotomico - Best First

Problema		classe A		classe B		classe C	
		$Z^*(\omega_{CG})$	tempo (s)	$Z^*(\omega_{CG})$	tempo (s)	$Z^*(\omega_{CG})$	tempo (s)
N = 50	CCPX1	713	20.0	383	486.3	266	33.3
	CCPX2	740	4.8	412	33.3	298	508.3
	CCPX3	751	11.9	405	95.9	311	297.2
	CCPX4	651	10.4	384	1767.2	277	7.8
	CCPX5	664	5.8	429	221.7	356	5.7
	CCPX6	778	5.5	482	866.6	370	5.5
	CCPX7	787	30.9	445	2632.1	358	72.1
	CCPX8	(844;777.63)	-	403	2204.5	312	1578.3
	CCPX9	715	57.5	436	690.1	412	612.5
	CCPX10	829	262.6	461	3103.0	(479;441.167)	-
N = 100	CCPX11	1006	1186.5	(595;528.917)	-	(416;405.833)	-
	CCPX12	(982;958.488)	-	(517;496)	-	(400;364.25)	-
	CCPX13	1026	179.2	(584;535.54)	-	(418;407.833)	-
	CCPX14	(994;973.5)	-	(568;529.5)	-	(436;412.694)	-
	CCPX15	1091	2595.8	(608;573.841)	-	(509;489.333)	-
	CCXP16	954	64.3	(603;521.767)	-	(435;424.3)	-
	CCPX17	(1041;1025.77)	-	(546;536.325)	-	(449;430)	-
	CCPX18	(1053;1031.9)	-	508	1416.4	(475;433.4)	-
	CCPX19	1031	899.2	(583;530.667)	-	(494;439.05)	-
	CCPX20	(1089;973.656)	-	(716;522.628)	-	(502;475.667)	-

Tabella 5.8: Branch dicotomico - Depth First

Il bound calcolato grazie all'algoritmo column generation è molto stretto, anche se computazionalmente oneroso: anche imponendo una condizione poco restrittiva come questa l'effetto sul bound duale è rilevante.

Branching di tipo "forbid"

Sia, per ogni nodo i, nel problema avente insieme di soluzioni S ed insieme di candidati mediane $\mathcal{M}^S$

$$H_i = \{j \in \mathcal{M}^S \mid x_{ij} \neq 0\} = \{h_i^1 \dots h_i^{F_i}\}$$

l'insieme degli indici delle mediane per le quali non è stato impedito l'assegnamento. In generale, inizialmente $H_i = \mathcal{M}\ \forall i \in \mathcal{N}$, successivamente, ponendo condizioni del tipo $x_{ij'} = 0$ è necessario rimuovere j' da H_i.

Come proposto da Savelsbergh in [Save95] è possibile ottenere uno schema di branching binario

1. selezionando un nodo i^b per cui $H_{i_b} \neq \emptyset$
2. selezionando un intero $j^* \simeq \frac{|H_{i^b}|}{2}$
3. ponendo

$$S_l = S_0 \cap \{(\boldsymbol{x^j}, y_j) \mid \textstyle\sum_{k=1}^{j^*} x_{i^b h_{i^b}^k} = 0\}$$

e

$$S_r = S_0 \cap \{(\boldsymbol{x^j}, y_j) \mid \textstyle\sum_{k=j^*+1}^{|H_{i^b}|} x_{i^b h_{i^b}^k} = 0\}$$

Si assicura, in tal modo, che i^b non venga assegnato alle prime j^* mediane in H_{i^b} (S_l) o alle rimanenti $|H_{i^b}| - j^*$ (S_r). La scelta può essere specializzata per CPMP, definendo l'insieme M_{i^b} come nel branching guidato dal rilassamento lineare e partizionandolo in due insiemi $M_{i^b}^l$ e $M_{i^b}^r$, in modo che $|M_{i^b}^l| \simeq \frac{|M_{i^b}|}{2}$ e $M_{i^b}^r = M_{i^b} \setminus M_{i^b}^l$. Anche l'insieme $R_{i^b} = H_{i^b} \setminus M_{i^b}$ può essere partizionato nello stesso modo negli insiemi $R_{i^b}^r$ ed $R_{i^b}^l$:

$$S_l = S_0 \cap \{(\boldsymbol{x^j}, y_j) \mid \textstyle\sum_{k \in M_{i^b}^l \cup R_{i^b}^l} x_{i^b k} = 0\}$$

e

$$S_r = S_0 \cap \{(\boldsymbol{x^j}, y_j) \mid \textstyle\sum_{k \in M_{i^b}^r \cup R_{i^b}^r} x_{i^b k} = 0\}$$

Confrontando i risultati riportati nella tabella 5.9, relativi alla politica non specializzata, con quelli in tabella 5.10 è evidente l'incremento di prestazioni dovuto a questa specializzazione della politica di branch: partizionare separatamente l'insieme M_{i^b} e R_{i^b} elimina il rischio di ottenere una soluzione frazionaria identica alla precedente, dato che in ogni caso alcune variabili di valore non nullo nel rilassamento di S_0 vengono poste a 0. Anche in

problema		p = 5		p = 12		p = 20	
N = 50	CCPX1	713	27.9	383	19.0	266	12.8
	CCPX3	751	12.7	405	38.3	311	7.5
	CCPX10	829	60.3	461	2315.1	(458;456.5)	-
N = 100	CCPX16	954	108.9	(538;527.083)	-	428	433.4
	CCPX19	1031	163.9	(553;542.5)	-	(454;443.5)	-

Tabella 5.9: Forbid branch - partizionamento semplice

questo caso è indicato scegliere l'imposizione di vincoli sulla variabile "più frazionaria" (cioè assegnata al maggior numero di mediane nella soluzione del rilassamento lineare). In tal modo si ha maggior probabilità che il bound duale nei due sottoproblemi-figlio sia più stringente. I risultati presentati nelle tabelle (5.10) e (5.11) mostrano come questa strategia offra prestazioni sensibilmente migliori delle precedenti.

Problema		classe A		classe B		classe C	
		$Z^*(\omega_{CG})$	tempo (s)	$Z^*(\omega_{CG})$	tempo (s)	$Z^*(\omega_{CG})$	tempo (s)
N = 50	CCPX1	713	17.1	383	16.1	266	9.9
	CCPX2	740	4.8	412	8.4	298	23.8
	CCPX3	751	11.9	405	25.2	311	6.7
	CCPX4	651	10.0	384	1469.9	277	6.7
	CCPX5	664	5.7	429	106.1	356	5.5
	CCPX6	778	5.4	482	261.8	370	5.2
	CCPX7	787	20.2	445	2578.2	358	6.5
	CCPX8	(822;813.778)	-	403	102.2	312	196.6
	CCPX9	715	12.9	436	113.4	412	107.2
	CCPX10	829	43.0	461	1317.7	(458;456.833)	-
N = 100	CCPX11	1006	146.3	(545;538.667)	-	(415;410.5)	-
	CCPX12	966	153.9	504	2500.9	(388;369.92)	-
	CCPX13	1026	76.6	(561;542.25)	-	(412;410.5)	-
	CCPX14	982	1125.1	(544;537.75)	-	421	3337.2
	CCPX15	1091	1296.9	583	1984.3	(497;493.5)	-
	CCPX16	954	65.0	(537;527.4)	-	428	247.3
	CCPX17	1034	353.7	542	134.1	(440;434.833)	-
	CCPX18	1043	1385.6	508	195.0	(450;447.5)	-
	CCPX19	1031	136.4	(556;542.667)	-	(453;446.25)	-
	CCPX20	(1025;987.667)	-	(584;531.556)	-	(486;484)	-

Tabella 5.10: Forbid branch - Best First

Problema		classe A		classe B		classe C	
		$Z^*(\omega_{CG})$	tempo (s)	$Z^*(\omega_{CG})$	tempo (s)	$Z^*(\omega_{CG})$	tempo (s)
N = 50	CCPX1	713	16.7	383	14.1	266	9.8
	CCPX2	740	4.7	412	14.2	298	26.0
	CCPX3	751	12.0	405	24.6	311	6.4
	CCPX4	651	10.1	384	1735.3	277	7.6
	CCPX5	664	5.7	429	728.8	356	5.6
	CCPX6	778	5.6	482	378.0	370	5.3
	CCPX7	787	19.7	(445;431)	-	358	6.9
	CCPX8	(820;775.515)	-	403	238.0	312	277.3
	CCPX9	715	12.1	436	89.3	412	99.1
	CCPX10	829	165.6	461	1792.9	(460;441.8)	-
N = 100	CCPX11	1006	100.0	(550;529.583)	-	(430;405.883)	-
	CCPX12	966	409.4	(514;496)	-	(419;364.381)	-
	CCPX13	1026	75.9	(561;534.438)	-	(421;405.5)	-
	CCPX14	982	3029.8	(548;530.167)	-	(423;413.333)	-
	CCPX15	1091	1312.0	583	2020.5	(499;488.833)	-
	CCPX16	954	80.2	(557;520.156)	-	428	1325.6
	CCPX17	1034	895.0	542	143.2	(447;430.5)	-
	CCPX18	1043	2117.1	(522;502.167)	-	(450;434.4)	-
	CCPX19	1031	283.5	(567;532.583)	-	(455;439.5)	-
	CCPX20	(1029;973.85)	-	(622;522.794)	-	(499;476.333)	-

Tabella 5.11: Forbid Branch - Depth First

Branching sulle penalties

In alternativa si può ricorrere ad una strategia simile a quella proposta da Pirkul [Pirk87], scegliendo prima quali nodi debbano essere considerati mediane ed, in un secondo momento, a quali mediane assegnare ogni nodo. Come descritto nella sezione 3, al termine delle iterazioni di column generation si dispone dei valori delle penalties (π_j) corrispondenti alla scelta ottima dei moltiplicatori. Si può dunque scegliere un nodo j^b e porre

$$S_l = S_0 \cap \{(\boldsymbol{x^j}, y_j) \mid y_{j^b} = 0\}$$
e
$$S_r = S_0 \cap \{(\boldsymbol{x^j}, y_j) \mid y_{j^b} = 1\}$$

fino a quando p nodi siano stati designati come mediane.

j^b può essere scelto in corrispondenza di uno dei valori estremi dei coefficienti di penalty ($j^b = argmax_j\{\pi_j\}$ oppure $j^b = argmin_j\{\pi_j\}$).

Di seguito (tabelle 5.12 e 5.13) è presentato un confronto tra questa strategia (Tipo A) ed un'altra strategia simile (Tipo B): scelto j^b come descritto si opera branching dicotomico ponendo

$$S_l = S_0 \cap \{(\boldsymbol{x^j}, y_j) \mid x_{j^b j^b} = 0\}$$
e
$$S_r = S_0 \cap \{(\boldsymbol{x^j}, y_j) \mid x_{j^b j^b} = 1\}$$

La seconda condizione comporta comunque che j^b sia definito mediana, ma la prima non lo impedisce.

L'albero è visitato in maniera depth first; una volta scelti p nodi come mediane, è stata utilizzata la strategia "forbid".

Nei test effettuati la strategia "Tipo B" si dimostra migliore: sui grafi proposti in letteratura ed utilizzati come test $d_{jj} = 0$ (per attinenza ai casi reali); quindi, se j è mediana, con molta probabilità si ottiene $x_{jj} = 1$. Lo svantaggio della strategia "Tipo A" è che imporre condizioni forti sull'appartenenza di una mediana j^b ($y_{j^b} = 1$) alla soluzione richiede di modificare la struttura di RMP, disaggregando il vincolo (5.16)

$$\sum_{k \in Z} z^k \leq p$$

problema (N = 50)	p = 5	p = 12	p = 20
CCPX1	2293.6	-	-
CCPX3	133.8	-	-
CCPX10	-	-	-

Tabella 5.12: Penalty Branch - Tipo A - Best First

problema (N = 50)	p = 5	p = 12	p = 20
CCPX1	27.9	767.0	3035.2
CCPX3	24.6	171.3	240.4
CCPX10	125.7	-	2275.4

Tabella 5.13: Penalty Branch - Tipo B - Best First

nei vincoli

$$\sum_{k\in Z}(1-\gamma^k_{j^b})z^k \leq p-1$$

e

$$\sum_{k\in Z}\gamma^k_{j^b}z^k = 1$$

oppure, in alternativa, portare il vincolo

$$\sum_{k\in Z}\gamma^k_{j^b}z^k \leq 1$$

della famiglia (5.17) in forma di uguaglianza:

$$\sum_{k\in Z}\gamma^k_{j^b}z^k = 1$$

Nelle tabelle 5.15 e 5.14 sono paragonati i tempi di calcolo per la strategia presentata (Tipo C), modificando la struttura di RMP, con i risultati relativi alle due strategie precedenti. Il costo computazionale necessario per operare sulla struttura di RMP è ampiamente compensato dal miglioramento del bound duale.

Problema		classe A		classe B		classe C	
		$Z^*(\omega_{CG})$	tempo (s)	$Z^*(\omega_{CG})$	tempo (s)	$Z^*(\omega_{CG})$	tempo (s)
N = 50	CCPX1	713	20.1	383	852.7	(266;265)	-
	CCPX2	740	4.9	412	93.8	298	3715.1
	CCPX3	751	18.6	405	380.0	311	199.0
	CCPX4	651	10.1	(386;377.714)	-	277	2004.7
	CCPX5	664	5.9	429	353.4	356	9.9
	CCPX6	778	5.4	(492;481)	-	370	18.4
	CCPX7	787	43.7	(449;442)	-	358	77.5
	CCPX8	820	1849.1	403	864.3	(316;303.1)	-
	CCPX9	715	48.3	436	802.3	(414;406.333)	-
	CCPX10	829	93.9	(468;457.056)	-	(509;445.833)	-
N = 100	CCPX11	1006	769.2	(590;531.5)	-	(480;407.667)	-
	CCPX12	966	950.3	(533;499.16)	-	(452;364.833)	-
	CCPX13	1026	264.1	(597;538.167)	-	(424;408)	-
	CCPX14	982	3330.9	(584;533.033)	-	(454;412.694)	-
	CCPX15	1091	2887.3	(633;577.312)	-	(554;492.05)	-
	CCPX16	954	318.7	(604;524.438)	-	(451.424.3)	-
	CCPX17	1034	3281.5	(552;541.75)	-	(475;430.25)	-
	CCPX18	(1047;1041.19)	-	(524;506.333)	-	(480;434.412)	-
	CCPX19	1031	392.0	(609;538.667)	-	(529;442.25)	-
	CCPX20	(1102;987.14)	-	(693;529.3)	-	(506;475.333)	-

Tabella 5.14: Penalty Branch - Tipo C - Best First

Problema		classe A		classe B		classe C	
		$Z^*(\omega_{CG})$	tempo (s)	$Z^*(\omega_{CG})$	tempo (s)	$Z^*(\omega_{CG})$	tempo (s)
N = 50	CCPX1	713	20.0	383	542.9	266	3095.7
	CCPX2	740	4.8	412	53.8	(307;292.5)	-
	CCPX3	751	18.7	405	561.1	311	715.9
	CCPX4	651	10.2	(386;364.929)	-	(308;275.5)	-
	CCPX5	664	5.9	429	448.7	356	11.8
	CCPX6	778	5.5	(492;481)	-	370	25.8
	CCPX7	787	47.5	(445;431.778)	-	358	32.0
	CCPX8	820	1493.1	403	2811.2	(332;297.071)	-
	CCPX9	715	48.2	436	695.2	(423;403.4)	-
	CCPX10	829	100.5	(461;447.053)	-	(519;441.167)	-
N = 100	CCPX11	1006	778.9	(559;528.917)	-	(476;405.833)	-
	CCPX12	966	1024.2	(525;496)	-	(437;364.25)	-
	CCPX13	1026	237.9	(594;534.04)	-	(424;404.833)	-
	CCPX14	982	3495.6	(589;529.5)	-	(454;412.694)	-
	CCPX15	1091	1857.0	(657;572.683)	-	(542;488.833)	-
	CCPX16	954	423.4	(567;521.112)	-	(451;424.3)	-
	CCPX17	(1038;1026.74)	-	(552;535.835)	-	(469;430)	-
	CCPX18	(1043;1034.9)	-	(524;501)	-	(480;433.4)	-
	CCPX19	1031	508.5	(610;530.667)	-	(526;439.05)	-
	CCPX20	(1058;974.68)	-	(695;522.628)	-	(506;475.333)	-

Tabella 5.15: Penalty Branch - Tipo C - Depth First

5.6.3 Branching con condizioni logiche

Già diversi autori hanno proposto schemi di branching per problemi di clustering in cui vengono generati sottoproblemi imponendo condizioni logiche sulla *compatibilità* di alcuni nodi [SaveEA]. Scelti due nodi i' e i'' si ottengono due sottoproblemi come:

$$S_l = S_0 \cap \{(\boldsymbol{x^j}, y_j) \mid (x_{i'j} = 1 \Leftrightarrow x_{i''j} = 1)\ \forall j \in \mathcal{M}\}$$

i' e i'' devono appartenere allo stesso cluster e

$$S_r = S_0 \cap \{(\boldsymbol{x^j}, y_j) \mid \neg(x_{i'j} = 1 \wedge x_{i''j} = 1)\ \forall j \in \mathcal{M}\}$$

i' e i'' non possono appartenere allo stesso cluster

Condizioni di questo tipo sono molto restrittive (e quindi ideali per generare alberi di branching binari). Tuttavia non è possibile risolvere in modo simmetrico i problemi S_l ed S_r: la condizione posta in S_l equivale semplicemente ad accorpare i' ed i'' in un unico nodo s, avente peso e coefficienti pari alla somma dei pesi e dei coefficienti dei due nodi

$w_s = w_{i'} + w_{i''}$ e

$d_{sj} = d_{i'j} + d_{i''j}\ \forall j \in \mathcal{M}$

La condizione posta in S_r, invece, modifica la struttura del pricing problem. Definito

$$E = \{(i', i'') \mid \neg(x_{i'j} = 1 \wedge x_{i''j} = 1)\ \forall j \in \mathcal{M}\}$$

ad ogni iterazione di column generation è necessario risolvere M problemi nella forma
KP-ISC ($j \in \mathcal{M}$))

$$min \sum_{i \in \mathcal{N}} [(d_{i,j} - \pi_i)x_{ij}] + \delta_j + \nu$$

s.t.

$$\sum_{i \in \mathcal{N}} w_i x_{ij} \leq Q_j$$

$$x_{i'j} + x_{i''j} \leq 1 \ \forall (i', i'') \in E$$

$$x_{ij} \in \{0, 1\} \ \forall i \in \mathcal{N}, \ \ \forall j \in \mathcal{M}$$

ovvero problemi di knapsack con *vincoli di independent set*. Per ogni livello dell'albero di branching viene aggiunto un vincolo di independent set in uno dei figli. Altri vincoli dello stesso tipo possono essere successivamente dedotti con tecniche di *constraint propagation*. Essendo maggiormente vincolati, questi problemi potrebbero essere risolti in modo più efficiente rispetto ai tradizionali problemi di knapsack. Tuttavia sarebbe necessario lo sviluppo di appositi algoritmi.

Un possibile criterio per la scelta delle variabili su cui imporre le condizioni descritte può essere quello di individuare i' e i'' aventi minima o massima correlazione $\sigma_{i'i''}$ nella soluzione del rilassamento lineare di S_0. La correlazione può essere calcolata come:

$$\sigma_{i'i''} = \sum_{k \in Z} z^k a^k_{i'} a^k_{i''}$$

ossia la somma del valore delle variabili in L-RMP che si riferiscono a clusters in cui compaiono sia i' che i''.

5.7 Column management

Come affermato in precedenza, non è necessario inserire in RMP tutte le colonne di costo ridotto negativo individuate tramite la soluzione del pricing problem: la base corrispondente al valore ottimo del rilassamento lineare può variare anche a causa dell'inserimento di una sola nuova colonna di costo ridotto negativo. Inserendo ad ogni iterazione tutte le colonne generate, è necessario un minor numero di iterazioni di column generation per la convergenza all'ottimo. Per contro, inserendo solo *alcune* colonne, fra quelle generate, in RMP si rende possibile risolvere rilassamenti lineari su problemi più semplici e ridurre il numero di chiamate all'algoritmo di pricing.
Anche la rimozione di colonne da RMP richiede particolare attenzione: come descritto, il processo di column generation è eseguito per ogni sottoproblema; clusters vantaggiosi per un sottoproblema potrebbero esserlo o meno per altri. Per questo rimuovere clusters poco vantaggiosi permette di ridurre le dimensioni di RMP, ma cancellare clusters desiderabili aumenta il numero di chiamate all'algoritmo di pricing. Sorge dunque la necessità di sviluppare appositi algoritmi per la gestione dei clusters rimossi da RMP ed il loro eventuale reinserimento per sottoproblemi diversi.

5.7.1 Scelta delle colonne iniziali

Per dare inizio al processo di column generation è necessario che in RMP vi sia un insieme di colonne che costituiscano una soluzione di base ammissibile; questo problema può essere risolto inserendo in RMP un opportuno insieme di colonne *dummy*. Nel caso di CPMP possono essere M colonne nella forma $(\forall j \in \mathcal{M})$

$$(\underbrace{1 \ldots 1}_{N},\ \gamma_1^j, \gamma_2^j, \ldots, \gamma_M^j,\ 1)^T$$

con

$$\gamma_j^j = 1 \text{ e } \gamma_k^j = 0\ \forall k \neq j$$

con costo molto alto, per garantire che non siano mai utilizzate nella soluzione ottima (è sufficiente porre $c^j \geq \sum_{j \in \mathcal{M}} \sum_{i \in \mathcal{N}} d_{ij}$), avendo cura di conservare questo insieme di colonne nel RMP di ogni sottoproblema.

Savelsbergh in [Save95] propone di aggiungere al RMP iniziale alcune colonne generate in modo euristico: come descritto nelle sezioni successive, avere a disposizione sin dalle prime iterazioni di column generation un set di colonne che rappresentino dei clusters vantaggiosi riduce notevolmente i tempi di calcolo.

Di seguito è riportato un confronto tra le prestazioni dell'algoritmo per alcune scelte dell'insieme di colonne iniziali sul nodo radice. Vengono confrontati il numero di iterazioni ed il tempo necessario a terminare il processo di column generation sul nodo radice per cinque istanze diverse nei casi in cui RMP contenga inizialmente:

1. sole colonne *dummy*

2. colonne *dummy* e colonne corrispondenti a soluzioni generate dall'euristica primale scegliendo i coefficienti di desiderabilità $f_{ij} = -d_{ij}$ o $f_{ij} = \frac{1}{d_{ij}}$ (come descritto nella sezione 5) e selezionando le mediane mediante estrazione casuale da distribuzione uniforme; l'estrazione delle mediane viene iterata per 10 volte e, per ogni iterazione, si inseriscono le eventuali soluzioni primali individuate dall'euristica per entrambe le scelte dei coefficienti

3. colonne *dummy* e colonne corrispondenti a soluzioni generate dall'euristica primale per le scelte dei coefficienti descritte sopra ed estraendo le mediane casualmente in accordo con i coefficienti $\psi_j = \sum_{i \in \mathcal{N}, i \neq j} \frac{1}{d_{ij}}$, per entrambe le scelte dei coefficienti f_{ij}; in questo caso si compiono 5 estrazioni diverse delle mediane

4. colonne *dummy* e colonne generate come al punto 3, ma compiendo 10 estrazioni diverse

Per eseguire il confronto, il processo di column generation è stato condotto eseguendo solo l'algoritmo esatto per la soluzione del pricing problem,

inserendo tutte le colonne di costo ridotto negativo individuate, senza ricorrere ad analisi di sensitività. Nella tabella 5.16 sono riportati il tempo ed il numero di iterazioni necessari per risolvere all'ottimo il rilassamento lineare sul nodo radice. Nei casi in cui siano state aggiunte colonne generate tramite euristica è indicato (tra parentesi) anche il tempo necessario per l'esecuzione dell'euristica primale.

Problema		*Solo colonne Dummy*		*Mediane estratte (2) 10 iterazioni*		*Mediane estratte (3) 5 iterazioni*		*Mediane estratte (4) 10 iterazioni*	
		Iteraz.	Tempo	Iteraz.	Tempo	Iteraz.	Tempo	Iteraz.	Tempo
N = 50	CCPX1	106	4.596	106	5.236 (0.047)	107	4.615 (0.024)	108	4.623 (0.050)
	CCPX2	108	4.765	112	5.759 (0.047)	108	4.829 (0.027)	112	5.801 (0.049)
	CCPX3	122	6.554	118	6.584 (0.047)	116	6.180 (0.026)	119	6.064 (0.049)
	CCPX10	107	4.295	99	3.982 (0.046)	103	4.368 (0.023)	99	3.947 (0.049)
N = 100	CCPX11	175	46.851	178	45.860 (0.308)	180	46.645 (0.162)	175	43.980 (0.329)
	CCPX12	172	41.927	172	40.517 (0.317)	172	39.737 (0.161)	174	41.256 (0.318)

Tabella 5.16: Scelta delle colonne iniziali

Aggiungere colonne generate come descritto conduce a miglioramenti di rilievo: le politiche (3) e (4) forniscono in generale prestazioni migliori, anche se il maggior guadagno registrabile, in termini di tempo di calcolo, è circa del 6%. Il miglioramento nei valori duali iniziali è scarso: il numero totale di iterazioni di column generation rimane quasi invariato. Confrontando i risultati con quelli riportati nella sezione 5 (tabella 5.4 inerente la scelta dei coefficienti per l'euristica primale) e nel paragrafo riguardante il reinserimento di clusters in RMP (tabella 5.21) si conclude che lo scarso miglioramento nei tempi di calcolo introducendo le colonne come descritto è conseguenza delle scarse prestazioni dell'euristica primale per le scelte $f_{ij} = -d_{ij}$ e $f_{ij} = \frac{1}{d_{ij}}$ dei coefficienti.

5.7.2 Inserimento di colonne

La soluzione dei problemi di knapsack (uno per ogni potenziale mediana $j \in \mathcal{M}$) consente di individuare, ad ogni iterazione, fino a $M = |\mathcal{M}|$ colonne di costo ridotto negativo.

First Negative

E' possibile inserire nel RMP la prima colonna di costo ridotto negativo generata ad ogni iterazione. Supponendo che tale colonna sia generata grazie alla valutazione del problema di knapsack per il k-esimo nodo, non è necessario risolvere i rimanenti $M - k$ problemi di knapsack. Tuttavia, se la colonna inserita avesse costo ridotto poco vantaggioso, risolvere nuovamente il rilassamento lineare di RMP potrebbe condurre ad una soluzione di base molto vicina alla precedente, senza sostanziali miglioramenti. Inoltre, al convergere del valore delle variabili duali al valore dei moltiplicatori ottimi, le nuove colonne generate hanno costi ridotti sempre minori in modulo. Per questo motivo è conveniente iniziare la generazione di colonne partendo da mediane diverse ad ogni iterazione.

Best Negative

Dopo aver risolto tutti gli M problemi di knapsack, alle colonne del RMP può essere aggiunta la colonna di minor costo ridotto generata. Anche in questo caso possono essere necessarie molte iterazioni per la convergenza all'ottimo del rilassamento lineare, anche se la dimensione del problema su cui risolvere il rilassamento lineare cresce più lentamente.

All Negative

Tutte le colonne di costo ridotto negativo generate vengono inserite nel RMP. Si punta a ridurre il numero di iterazioni di column generation complessivamente necessarie per la convergenza, a scapito delle dimensioni di RMP.

5.7.3 Rimozione di colonne

Inserendo di volta in volta nuove colonne, il numero di variabili in RMP può diventare troppo elevato per poter essere gestibile efficientemente da un MIP Solver ed, in ogni caso, dover risolvere il rilassamento lineare di problemi

dalle dimensioni elevate può peggiorare sensibilmente le prestazioni dell'algoritmo.
Per contro, avere a disposizione un insieme di clusters "molto buoni" prima di dare inizio alla generazione di nuove colonne può accelerare la convergenza all'ottimo del rilassamento lineare, riducendo il numero di iterazioni di column generation, e quindi il numero di chiamate all'algoritmo per la soluzione del pricing problem. La cancellazione di clusters da RMP deve essere regolata con cura: in generale, politiche che prevedano un compromesso tra i due aspetti possono fornire le prestazioni migliori, anche se il corretto bilanciamento dei tempi di calcolo dipende dall'efficienza relativa del MIP solver utilizzato per la soluzione del rilassamento lineare e dell'algoritmo per il pricing problem.

Reset delle colonne

Un approccio banale è rimuovere da RMP *tutte* le colonne generate (ad eccezione delle colonne "dummy", necessarie per assicurare la presenza di una soluzione ammissibile in ogni sottoproblema), per generarne di nuove corrispondenti solo ai clusters ammissibili per ogni sottoproblema in esame.

Rimozione delle sole colonne non ammissibili

Al contrario, si può scegliere di mantenere in RMP tutte le colonne precedentemente generate, compatibili con i vincoli del sottoproblema in esame.

Analisi del costo ridotto delle colonne

Se il costo ridotto di una colonna è molto positivo, difficilmente tale colonna può, nelle iterazioni successive, entrare in base. Una possibile politica è la rimozione delle colonne aventi, al termine delle iterazioni di column generation, costo ridotto maggiore di una soglia prefissata. Inoltre, più il bound duale si avvicina al primale, minore è il numero di clusters candidati ad entrare in base: se il costo ridotto di una colonna risulta maggiore del gap tra bound primale e bound duale in un problema, il cluster corrispondente non è candidato a far parte della soluzione ottima per i sottoproblemi del

problema in esame. Invece di confrontare i costi ridotti delle colonne con un valore costante, è possibile conservare in RMP solo le colonne aventi costo ridotto inferiore ad una soglia prefissata $r(Z^*, \omega_{CG})$, funzione della differenza tra bound primale (Z^*) e bound duale (ω_{CG}).

Analisi del numero di colonne

E' possibile utilizzare l'informazione derivante dall'analisi dei costi ridotti, ottenuti nel corso delle iterazioni di column generation, per stimare la "bontà" dei clusters. Ad ogni nodo si può scegliere di mantenere un numero fisso C di colonne, corrispondenti ai C clusters aventi i migliori coefficienti di costo ridotto. Le considerazioni esposte per il caso della rimozione delle colonne in funzione del loro costo ridotto rimangono valide: al diminuire del gap tra bound primale e duale diminuisce il numero di clusters candidati ad entrare in base; è quindi possibile mantenere in RMP un *numero* di clusters proporzionale al coefficiente $r(Z^*, \omega_{CG})$.

Algoritmi adattativi

Generalizzando gli approcci proposti, sarebbe possibile sviluppare algoritmi adattativi con il compito di bilanciare il carico computazionale tra le chiamate all'algoritmo per il pricing problem e soluzione del rilassamento lineare.

Le prime due politiche non offrono particolari vantaggi per quanto riguarda la qualità dei clusters in RMP, ma non richiedono praticamente alcuna implementazione. L'operazione di rimozione delle colonne può essere effettuata in maniera molto efficiente.
Le politiche orientate all'analisi del costo ridotto di ogni colonna garantiscono di mantenere clusters di una certa qualità in RMP: si privilegia la riduzione del numero di iterazioni di column generation necessarie, a scapito del numero di colonne su cui valutare il rilassamento lineare. In particolare, è naturale associare la politica di rimozione per confronto del costo ridotto della colonna con la soglia $r(Z^*, \omega_{CG})$ ad esplorazioni best first dell'albero di branching: il gap tra bound duale e bound primale non cresce durante le valutazioni dei

classe A

Problema		Reset		Keep		keepNumber		rcGap	
N = 50	CCPX1	713	113.4	713	25.4	713	24.9	713	16.9
	CCPX3	751	14.8	751	8.6	751	12.1	751	12.3
N = 100	CCPX16	954	927.3	954	122.7	954	102.8	954	66.2
	CCPX19	1031	1880.6	1031	153.2	1031	198.9	1031	77.7

classe B

Problema		Reset		Keep		keepNumber		rcGap	
N = 50	CCPX1	383	142.8	383	25.5	383	26.1	383	16.9
	CCPX3	405	267.4	405	49.2	405	49.5	405	23.3
N = 100	CCPX16	(554;523)	-	(538;526.5)	-	(538;526.577)	-	(538;527.391)	-
	CCPX19	(575;536)	-	(555;541.69)	-	(555;542.4)	-	(555;542.5)	-

classe C

Problema		Reset		Keep		keepNumber		rcGap	
N = 50	CCPX1	266	84.6	266	14.4	266	17.6	266	10.2
	CCPX3	311	25.1	311	5.6	311	14.2	311	6.9
N = 100	CCPX16	(428;425.9)	-	428	915.8	428	448.0	428	221.0
	CCPX19	(465;443.5)	-	(453;445.667)	-	(467;443.333)	-	(452;446.25)	-

Tabella 5.17: Cancellazioni da RMP

successivi sottoproblemi, quindi una colonna avente costo ridotto superiore al gap non è più candidata ad entrare in base. Anche in questo caso, tuttavia, è possibile optare per condizioni più strette. Le politiche improntate all'analisi del numero di colonne garantiscono, in maniera simmetrica, di contenere le dimensioni di RMP, agevolando il compito del MIP solver, anche a prezzo di rimuovere da RMP clusters potenzialmente vantaggiosi. Nella tabella 5.17 viene presentato un confronto tra queste politiche. Il confronto tiene anche conto della possibilità, descritta di seguito, di reinserire clusters in RMP.

Nell'implementazione realizzata della politica di analisi del costo ridotto delle colonne $r(Z^*, \omega_{CG}) = \frac{Z^* - \omega_{CG}}{p}$ (questa scelta è indicata successivamente con **rcGap**): al termine delle iterazioni di column generation vengono rimossi da RMP i clusters che hanno costo ridotto superiore a $r(Z^*, \omega_{CG})$. Per quanto riguarda l'implementazione della politica di analisi del numero di colonne, vengono conservati in RMP i clusters di minor costo ridotto in numero pari a $r(Z^*, \omega_{CG}) = N * p * (Z^* - \omega_{CG})$ (**keepNumber**). Vengono proposti anche i tempi di calcolo adottando la politica di cancellazione dell'intero insieme di colonne (**reset**) e di conservazione dell'intero insieme (**keep**).

Cancellare l'intero insieme di colonne risulta sempre svantaggioso. Conservare tutte le colonne generate in RMP è talvolta vantaggioso per i problemi su grafi di 50 nodi: il vantaggio derivante dalla gestione del pool di clusters non è sufficiente a bilanciarne il costo computazionale. Per problemi dalle dimensioni maggiori rimuovere da RMP clusters poco promettenti, analizzandone il costo ridotto, consente di ridurre i tempi di calcolo, a volte in maniera significativa (problemi CCPX16 e CCPX19 per $p = 10$ e $p = 40$).

Conservare un numero di colonne proporzionale a $r(Z^*, \omega_{CG})$, e ancor più conservare le colonne secondo la politica **rcGap** consente *comunque* di migliorare le prestazioni del metodo, consentendo, in alcuni casi, di dimezzare o *ridurre ad un quarto* i tempi di calcolo.

5.7.4 Pool di colonne

In generale non ci sono garanzie che clusters rimossi da RMP non siano vantaggiosi per altri sottoproblemi. E' conveniente, allora, conservare le colonne rimosse in apposite strutture dati (pool) e valutarne nuovamente il costo ridotto durante le iterazioni di column generation in altri sottoproblemi prima di deciderne l'eliminazione definitiva.

Reinserimento

Ad ogni iterazione di column generation si può esaminare il pool dei clusters precedentemente rimossi, valutandone il costo ridotto. Se vengono trovati clusters di costo ridotto negativo, il loro reinserimento in RMP può accelerare la convergenza all'ottimo del rilassamento lineare.

Come per l'inserimento in RMP di colonne generate tramite pricing problem, possono essere adottate le politiche first, best e all negative. L'analisi del costo ridotto delle colonne nel pool, tuttavia, è poco impegnativa in termini di tempo, quindi è stata esplorata solo la politica all negative.

Eliminazione dal pool

Anche la dimensione del pool deve essere ben calibrata: il vantaggio di reinserire clusters in RMP deve bilanciare il carico computazionale della gestione

del pool.

Per cercare di conservare clusters vantaggiosi senza dover gestire pools dalle dimensioni elevate si è deciso di utilizzare una politica *n-th chance* per la gestione delle cancellazioni dal pool: clusters il cui costo ridotto non sia negativo per n_c ispezioni consecutive vengono cancellati definitivamente.

Nella tabella 5.18 sono comparate diverse scelte del parametro n_c, regolando la rimozione da RMP (ed inserimento nel pool) secondo la politica rcGap. Osservando i risultati presentati si può notare come diverse scelte del parametro n_c influenzino, anche se in modo non radicale le prestazioni dell'algoritmo.

classe A

Problema		$n_c = 3$		$n_c = 6$		$n_c = 9$	
N = 50	CCPX1	713	18.6	713	16.9	713	16.9
	CCPX3	751	12.4	751	12.3	751	12.3
N = 100	CCPX16	954	69.2	954	66.2	954	66.1
	CCPX19	1031	90.4	1031	77.7	1031	85.4

classe B

Problema		$n_c = 3$		$n_c = 6$		$n_c = 9$	
N = 50	CCPX1	383	18.2	383	16.9	383	16.2
	CCPX3	405	25.2	405	23.3	405	26.9
N = 100	CCPX16	(535;527.362)	-	(538;527.391)	-	(535;527.517)	-
	CCPX19	(555;545.333)	-	(555;542.5)	-	(555;542.5)	-

classe C

Problema		$n_c = 3$		$n_c = 6$		$n_c = 9$	
N = 50	CCPX1	266	9.6	266	10.2	266	9.8
	CCPX3	311	6.8	311	6.9	311	7.2
N = 100	CCPX16	428	232.9	428	221.0	428	232.1
	CCPX19	(453;446.083)	-	(452;446.25)	-	(453;445.889)	-

Tabella 5.18: Scelte 3, 6 e 9 del parametro n_c

5.8 Risultati sperimentali

La serie di test della quale sono riportati i risultati di seguito è volta a comparare le politiche first, best e all negative di inserimento delle colonne in RMP, in termini assoluti ed in relazione alla gestione del pool di clusters.

Prestazioni sul nodo radice

Si esaminino i risultati sul nodo radice per diversi problemi, senza utilizzo di euristiche (tabella 5.19) ed inserendo tutte le colonne di costo ridotto negativo generate da algoritmi euristici per il pricing problem (tabella 5.20)

Problema		*First Negative*		*Best Negative*		*All Negative*	
		iterazioni di CG	tempo	iterazioni di CG	tempo	iterazioni di CG	tempo
N = 50	CCPX1	826	9.594	496	6.186	106	4.504
	CCPX2	978	12.651	607	8.135	108	4.739
	CCPX3	895	11.16	585	8.385	122	6.415
	CCPX10	660	7.885	449	7.031	107	4.202
N = 100	CCPX11	2004	116.634	1268	69.212	175	47.548
	CCPX12	1687	92.945	1045	53.701	172	41.636
	CCPX16	1835	104.611	1300	71.517	178	43.600

Tabella 5.19: Prestazioni delle politiche di inserimento sul nodo radice

Problema		*First Negative*		*Best Negative*		*All Negative*	
		iterazioni di CG	tempo	iterazioni di CG	tempo	iterazioni di CG	tempo
N = 50	CCPX1	121	4.812	123	4.985	115	4.726
	CCPX2	125	5.556	122	5.525	120	5.462
	CCPX3	139	6.952	139	7.110	132	6.719
	CCPX10	136	5.548	145	6.182	132	5.634
N = 100	CCPX11	1874	108.724	1255	82.313	185	47.478
	CCPX12	1616	95.304	1084	71.205	189	43.927
	CCPX16	1808	106.132	1294	88.165	196	48.106

Tabella 5.20: Prestazioni sul nodo radice con euristiche

- Considerando esclusivamente le prestazioni sull'analisi del nodo radice, senza utilizzo di euristiche per il pricing problem, le politiche first e best negative forniscono risultati peggiori di all negative.

- L'introduzione di colonne generate tramite euristiche fornisce risultati contrastanti: i tempi di calcolo della politica di inserimento first negative generalmente migliorano, per la politica all negative si ha un peggioramento (non decisivo). I tempi di calcolo delle tre politiche sono quasi gli stessi per problemi su grafi di 50 nodi; all negative si dimostra migliore per problemi dalle dimensioni maggiori (con scarti anche superiori al 50 % rispetto ai tempi di calcolo utilizzando le altre politiche).

Inserimento e reinserimento

La politica di gestione del pool di clusters interagisce, durante l'esplorazione dell'albero di branching, con la politica di inserimento delle colonne generate in RMP.

Nelle tabelle seguenti (5.21) sono indicati i tempi impiegati dall'algoritmo per risolvere diverse istanze di CPMP, al variare della politica di inserimento (first, best ed all negative) ed a quella di re-inserimento di clusters del pool in RMP.

L'algoritmo utilizza tecnica di branching "forbid" abbinata ad esplorazione best first dell'albero di ricerca, analisi di sensitività e preprocessing. I clusters poco vantaggiosi vengono rimossi da RMP secondo la politica rcGap, le cancellazioni dal pool sono gestite, con politica *n-th chance* (con $n_c = 3$). Tutti i clusters di costo ridotto negativo trovati nel pool ad ogni ispezione vengono reinseriti in RMP.

Problema		*senza reinserimento*			*con reinserimento durante CG*			*CG ed iniziale*		
					classe A					
		first	best	all	first	best	all	first	best	all
N = 50	CCPX1	25.5	52.2	22.0	28.4	26.5	16.0	42.8	26.0	20.8
	CCPX3	15.8	10.2	12.1	15.8	10.4	12.6	15.8	10.1	12.6
	CCPX10	132.8	118.9	65.5	125.1	71.0	51.9	162.6	77.6	63.6
N = 100	CCPX11	953.7	528.6	236.7	452.2	240.0	139.6	712.8	377.1	175.7
	CCPX16	333.7	139.9	122.8	219.6	134.9	68.6	169.8	151.8	92.3
	CCPX19	195.1	286.0	118.2	200.2	274.7	139.6	208.2	268.3	144.9
					classe B					
		first	best	all	first	best	all	first	best	all
N = 50	CCPX1	55.1	60.8	64.9	107.7	56.8	65.1	69.4	30.0	71.5
	CCPX3	277.5	243.0	112.6	243.3	156.7	96.1	224.0	237.5	113.6
N = 100	CCPX11	- (3.584%)	- (2.243%)	- (1.587%)	- (2.024%)	- (2.070%)	- (1.556%)	- (1.651%)	- (2.087%)	- (2.313%)
	CCPX16	- (2.696%)	- (2.152%)	- (1.991%)	- (2.049%)	- (2.122%)	- (2.527%)	- (2.055%)	- (2.116%)	- (2.023%)
					classe C					
		first	best	all	first	best	all	first	best	all
N = 50	CCPX1	63.3	31.1	26.8	55.9	20.5	18.8	78.8	33.5	41.8
	CCPX3	8.5	8.9	9.0	5.5	9.3	8.3	5.9	7.3	17.1
N = 100	CCPX11	- (1.302%)	- (1.220%)	- (1.302%)	- (1.302%)	- (1.199%)	- (1.281%)	- (1.261%)	- (1.261%)	- (1.261%)
	CCPX16	330.0	1583.7	968.8	320.4	919.3	601.3	279.7	1040.1	1247.0

Tabella 5.21: Inserimento e Reinserimento

Ottenere rapidamente dei buoni valori duali è molto importante, poiché riduce drasticamente il numero di iterazioni necessarie per risolvere il rilassamento. A tal scopo è necessario disporre di colonne che rappresentino soluzioni "molto buone" prima di risolvere il rilassamento lineare. Bisogna, tuttavia, scegliere con cura quali clusters reinserire in RMP.

- La politica all negative consente di ottenere tempi di calcolo in media inferiori rispetto alle altre politiche. Per i problemi di classe B su grafi di 100 nodi la computazione non termina entro 3600 secondi; tuttavia il *gap* tra bound primale e bound duale dopo 3600 secondi, nei casi in cui sia stata utilizzata all negative, è generalmente inferiore.

- Osservando i risultati per i problemi di classe C, si può notare come le politiche best e first negative diventino competitive con la politica all negative; in particolare su CCPX16 first negative consente di dimezzare o ridurre ad un terzo i tempi di calcolo rispetto ad all negative.

- Inserire clusters contenuti nel pool solo durante le iterazioni di column generation in cui si utilizza l'algoritmo esatto per il pricing consente di ottenere prestazioni migliori in maniera generalmente indipendente dalla politica di inserimento utilizzata.

- In alcuni casi il guadagno derivante dall'utilizzo dei clusters conservati nel pool durante le iterazioni di column generation con algoritmo esatto è determinante. Ad esempio, per il problema CCPX16-C, inserimento all negative, si ha un guadagno del 38 % circa rispetto al caso senza reinserimento e del 51.8 % rispetto all'inserimento di clusters anche prima delle chiamate all'algoritmo per il pricing problem.

Gestione del pool ed euristiche per il pricing problem

In tabella 5.22 è invece proposto un confronto tra i tempi di calcolo necessari a terminare la computazione con strategia di branching "forbid" ed esplorazione dell'albero di ricerca best first, operando analisi di sensitività ad ogni

iterazione, inserendo colonne secondo la politica all negative, rimuovendo clusters da RMP secondo la strategia **rcGap**

1. senza utilizzare euristiche per la generazione di colonne
2. generando colonne con soli algoritmi euristici, procedendo in un secondo momento alla generazione di colonne con soli algoritmi esatti
3. utilizzando euristiche per la generazione di colonne e chiamando l'algoritmo esatto solo nel caso in cui l'euristica non individui colonne vantaggiose
4. combinando gli approcci 2 e 3

classe A

Problema		(1)		(2)		(3)		(4)	
N = 50	CCPX1	713	16.0	713	34.0	713	18.6	713	23.3
	CCPX3	751	12.6	751	14.8	751	13.7	751	14.7
N = 100	CCPX16	954	68.6	954	65.0	954	122.0	954	146.3
	CCPX19	1031	139.6	1031	155.2	1031	148.6	1031	184.8

classe B

Problema		(1)		(2)		(3)		(4)	
N = 50	CCPX1	383	65.2	383	101.0	383	88.8	383	105.7
	CCPX3	412	96.6	412	144.6	412	77.6	412	76.7
N = 100	CCPX16	(2.53%)	-	(2.13%)	-	(1.90%)	-	(2.20%)	-
	CCPX19	(3.26%)	-	(3.45%)	-	(3.24%)	-	(3.82%)	-

classe C

Problema		(1)		(2)		(3)		(4)	
N = 50	CCPX1	266	18.7	266	30.5	266	36.3	266	73.0
	CCPX3	298	8.3	298	12.4	298	11.2	298	13.7
N = 100	CCPX16	428	596.7	428	612.2	428	716.4	428	609.7
	CCPX19	(1.55%)	-	(1.39%)	-	(3.38%)	-	(3.37%)	-

Tabella 5.22: Utilizzo di euristiche per il pricing problem

Anche in questo caso, l'utilizzo delle euristiche non migliora le prestazioni dell'algoritmo. Gli algoritmi per il pricing problem utilizzati spesso falliscono nell'individuare colonne di costo ridotto negativo, ed anche in caso di successo, vengono generate colonne di costo ridotto poco vantaggioso. In modo particolare in quest'ultimo caso, inserire colonne poco vantaggiose comporta

comunque la necessità di rivalutare il rilassamento lineare di RMP, reiterare l'algoritmo euristico e ricorrere, infine alla soluzione esatta del pricing problem, con notevole costo computazionale aggiuntivo.

Capitolo 6

Conclusioni

Nei capitoli precedenti sono state prese in considerazione quattro alternative per la risoluzione di CPMP:

1. **MIP-S:** l'utilizzo di un MIP solver (come CPLEX)
2. **GAP-R:** la riformulazione di CPMP come GAP e l'utilizzo dell'algoritmo di Savelsbergh
3. **LR-A:** un algoritmo Branch & Bound basato su rilassamento Lagrangeano
4. **BP-A:** un algoritmo Branch & Price

6.1 Confronto tra gli algoritmi

Problemi risolti nel tempo limite

Un primo termine di paragone fra gli algoritmi proposti è il numero di istanze per le quali il calcolo termina entro il tempo limite di 1 ora (tabella 6.1). CPLEX permette di risolvere entro un'ora solo il 65% delle istanze analizzate. L'algoritmo **GAP-R** si è dimostrato poco efficace: riformulando CPMP come GAP solo nel 50% dei casi l'algoritmo completa il calcolo; in questi casi, inoltre, ha prestazioni paragonabili a quelle di CPLEX.

L'algoritmo basato su rilassamento Lagrangeano termina entro un'ora solo nel 50% dei casi, mentre l'utilizzo del metodo di branch & price, con strategia di branching "forbid" specializzata e ricerca best first, consente di terminare il calcolo entro il tempo limite nel $71.\overline{6}\%$ dei casi. Nei grafici 6.1 e 6.2 sono confrontati il numero di casi in cui ogni algoritmo termina in base alla classe di istanze.

grafi di cardinalità 50

Classe di istanze	**MIP-S**	**GAP-R**	**LR-A**	**BP-A**
A	100%	90%	100%	90%
B	90%	20%	60%	100%
C	80%	40%	50%	90%
Media	90.0%	50.0%	70.0%	$93.\overline{3}\%$

grafi di cardinalità 100

Classe di istanze	**MIP-S**	**GAP-R**	**LR-A**	**BP-A**
A	20%	-	90%	90%
B	50%	-	0%	40%
C	50%	-	0%	20%
Media	40.0%	-	30.0%	50.0%
Media	65.0%	-	50.0%	$71.\overline{6}\%$

Tabella 6.1: Numero di computazioni terminate entro un'ora

L'algoritmo **LR-A** dimostra di essere molto efficace per problemi in cui il rapporto $\frac{N}{p}$ è elevato (istanze di classe A), ma le sue prestazioni peggiorano sensibilmente al decrescere del rapporto (30% di successi per problemi di classe B e 25% per problemi di classe C). **BP-A** termina nel tempo limite nel 90% dei casi su istanze di classe A, 70% su istanze di classe B e 55% su istanze di classe C: l'algoritmo branch & price è sperimentalmente più *stabile* rispetto alla scelta del parametro p.

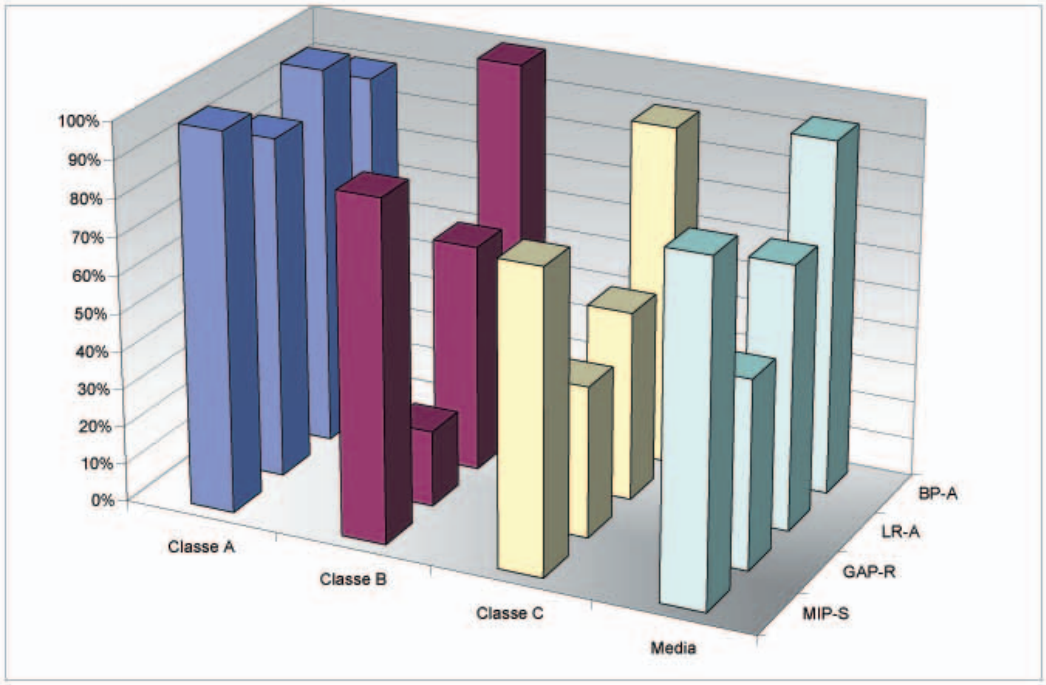

Grafico 6.1: Problemi risolti nel tempo limite (N = 50)

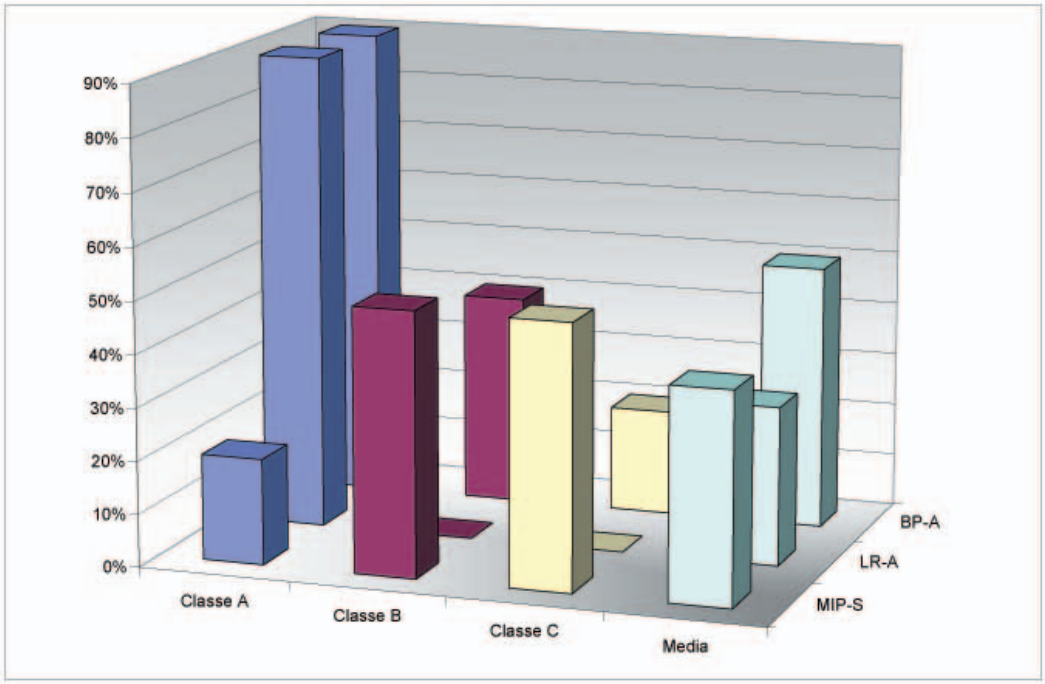

Grafico 6.2: Problemi risolti nel tempo limite (N = 100)

Anche in termini di tempi necessari a terminare il calcolo, l'algoritmo basato su rilassamento Lagrangeano è più *efficiente* per i problemi della classe A (tabella 6.2): escludendo i problemi CCPX8 e CCPX20 il tempo medio è inferiore rispetto all'algoritmo branch & price (grafico 6.3). **MIP-S** e **GAP-R** non sono competitivi sotto questo aspetto (tabelle 3.1 e 1.1).

Algoritmo	N = 50	N = 100	Medio
LR-A	6.9	361.0	183.9
BP-A	14.6	526.6	270.6

Tabella 6.2: Tempo medio per terminare il calcolo su istanze di classe A

Risultati per la classe D

Dopo aver scelto i valori dei parametri per i vari algoritmi sulla base dei risultati ottenuti per le istanze di classe A, B e C, in tabella 6.3 sono paragonati i tempi di calcolo sulle istanze di classe D (corrispondenti alla scelta $p = \frac{N}{3}$) per

1. l'algoritmo con rilassamento Lagrangeano con branching a due livelli (sulle penalties al primo livello e sulle possibili mediane al secondo), strategia di ricerca depth first
2. l'algoritmo branch & price con *la stessa politica di branching*
3. l'algoritmo branch & price con politica di branching "forbid" e ricerca best first

Anche per questa classe di problemi l'algoritmo branch & price con politica "forbid" si dimostra migliore.

Scarto tra bound primale e bound duale

Per operare un ulteriore confronto tra i quattro algoritmi è stato analizzato il gap tra bound primale e bound duale al termine del tempo limite (tabella 6.4 e grafici 6.4 e 6.5, in percentuale). Nei casi in cui l'algoritmo abbia terminato

Problema		**LR-A**		**BP-A** - Penalty Branch		**BP-A** - Forbid Branch	
		$Z^*(\omega_{CG})$	tempo (s)	$Z^*(\omega_{CG})$	tempo (s)	$Z^*(\omega_{CG})$	tempo (s)
N = 50	CCPX1	298	411.5	298	19.6	298	5.1
	CCPX2	(336;327.192)	-	(337;329.5)	-	336	55.9
	CCPX3	314	470.2	314	3100.3	314	7.6
	CCPX4	303	808.9	303	618.3	303	15.7
	CCPX5	351	186.8	351	446.7	351	9.3
	CCPX6	390	2195.8	390	148.3	390	7.4
	CCPX7	361	246.8	361	111.1	361	16.3
	CCPX8	(371;321.487)	-	(375;323.255)	-	(353;343)	-
	CCPX9	373	2427.5	373	896.1	373	27.0
	CCXP10	(391;370.68)	-	(391;376.214)	-	390	610.0
N = 100	CCPX11	(426;401.882)	-	(434;405.833)	-	414	2779.0
	CCPX12	(441;374.166)	-	(405;374.75)	-	(391;382)	-
	CCPX13	(452;436.705)	-	(463;440.5)	-	446	261.5
	CCPX14	(466;430.099)	-	(459,434)	-	(447;438.75)	-
	CCPX15	(494;465.886)	-	(493;469.375)	-	474	723.3
	CCPX16	(471;428.167)	-	(474;431.083)	-	(453;436.833)	-
	CCPX17	(432;420.462)	-	(440;423.333)	-	431	221.6
	CCPX18	(468;426.82)	-	(479;430.689)	-	(459;440.417)	-
	CCPX19	(451;426.68)	-	(474;430.667)	-	(445;440.265)	-
	CCPX20	(496;446.318)	-	(561;450.964)	-	(461;459)	-

Tabella 6.3: Prestazioni sulle istanze di classe D

la computazione entro un'ora, questo valore è stato considerato 0. Di nuovo **LR-A** è migliore per problemi con rapporto $\frac{N}{p}$ alto, **BP-A** negli altri casi.

Grafi di cardinalità 50

classe	Lagrangeano	Branch & Price
A	0.000%	0.101%
B	3.065%	0.000%
D	2.357%	0.292%
C	8.145%	0.026%

Grafi di cardinalità 100

classe	Lagrangeano	Branch & Price
A	0.826%	0.378%
B	6.190%	1.994%
D	8.097%	1.367%
C	47.162%	1.073%

Tabella 6.4: scarto medio tra bound primale e bound duale

La classe D ha un rapporto $\frac{N}{p}$ maggiore della classe C e minore della classe B. Per questo motivo i risultati relativi alle sue istanze sono rappresentati nella terza riga della tabella.

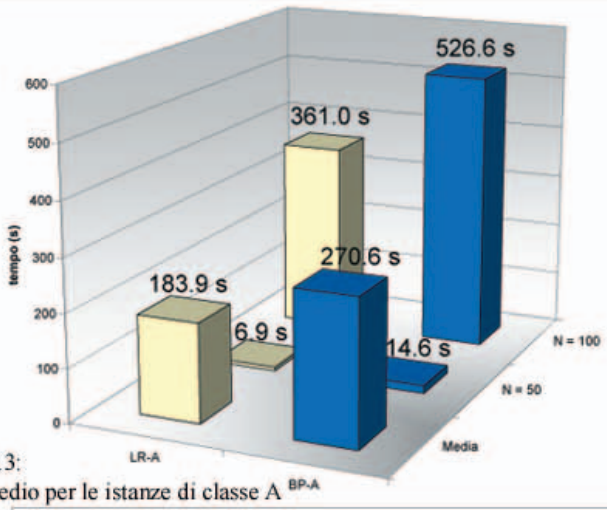

Grafico 6.3:
Tempo medio per le istanze di classe A

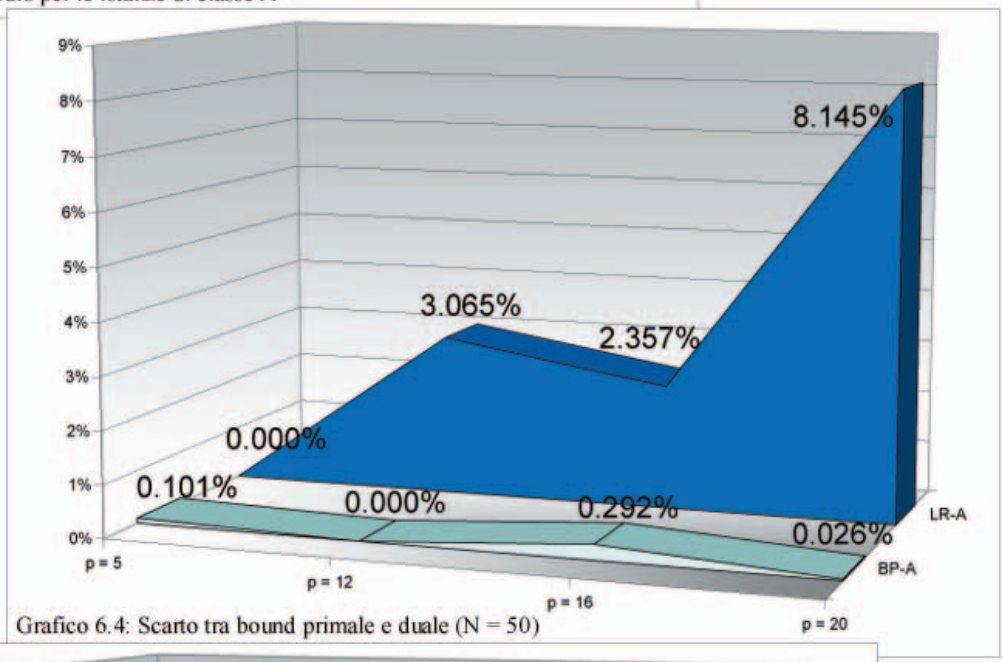

Grafico 6.4: Scarto tra bound primale e duale (N = 50)

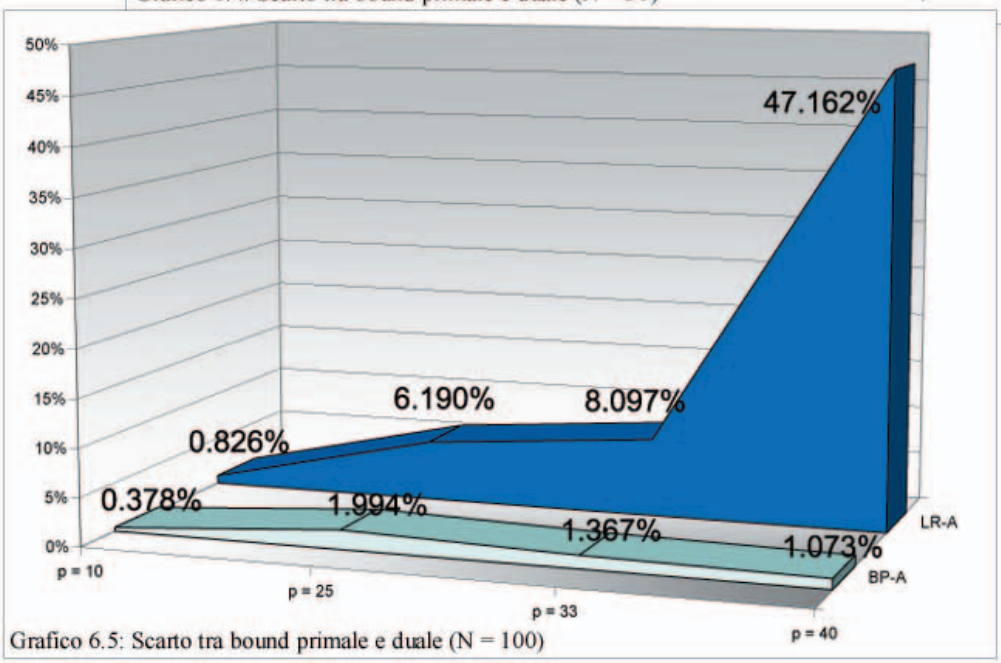

Grafico 6.5: Scarto tra bound primale e duale (N = 100)

Confronto a parità di branching

Nella tabella 5.15 sono riportati i risultati ottenuti applicando **BP-A** con politica di branching sulle penalties e ricerca depth first. E' importante notare come l'unica differenza, per tale scelta, tra gli algoritmi **BP-A** e **LR-A** è la *politica di aggiornamento dei moltiplicatori*. Infatti nel caso del rilassamento Lagrangeano i moltiplicatori vengono aggiornati con il metodo del sottogradiente, per column generation si utilizza l'informazione duale derivante dal rilassamento lineare. E' possibile, quindi, confrontare questi risultati con quelli riportati in tabella 4.4.

Sebbene per le istanze di classe A e B l'algoritmo **LR-A** risulti migliore, per le istanze di classe C **BP-A** si dimostra vantaggioso in 10 casi su 20 e nelle istanze di classe D 11 volte su 20.

Tempi di calcolo in funzione di p

Dalle osservazioni precedenti è chiaro che il rapporto $\frac{N}{p}$ gioca un ruolo fondamentale sulle prestazioni degli algoritmi presentati. In tabella 6.5 e nei grafici 6.6-6.8 sono riportati i tempi di calcolo per gli algoritmi **LR-A** e **BP-A** al variare del parametro p: ogni istanza ρ è stata ottenuta modificando le capacità delle istanze di classe A:

$$Q_j^\rho = \frac{p}{p^\rho} Q_j$$

CCPX1

Algoritmo	p=3	p=5	p=7	p=9	p=11	p=13	p=15	p=17	p=19	p=21	p=23	p=25
BP-A	13.8	16.8	17.8	54.6	52.6	30.9	54.8	27.1	8.0	5.2	5.8	92.2
LR-A	0.2	1.8	15.3	52.9	274.3	263.3	1996.4	2885.9	3600.0	24.5	343.4	3600.0

CCPX3

Algoritmo	p=3	p=5	p=7	p=9	p=11	p=13	p=15	p=17	p=19	p=21	p=23	p=25
BP-A	14.0	12.4	11.1	714.7	2.2	7.2	90.7	1147.0	18.5	7.2	2.1	6.2
LR-A	1.0	1.6	6.9	466.6	11.5	27.3	3600.0	3600.0	3600.0	78.1	29.8	3600.0

CCPX6

Algoritmo	p=3	p=5	p=7	p=9	p=11	p=13	p=15	p=17	p=19	p=21	p=23	p=25
BP-A	44.4	5.4	3.3	9.6	5.9	26.8	35.5	19.0	9.9	116.8	271.9	5.8
LR-A	0.2	1.4	10.3	30.4	12.4	549.2	1096.5	687.2	1465.5	3600.0	3600.0	3600.0

Tabella 6.5: Prestazioni in funzione del parametro p

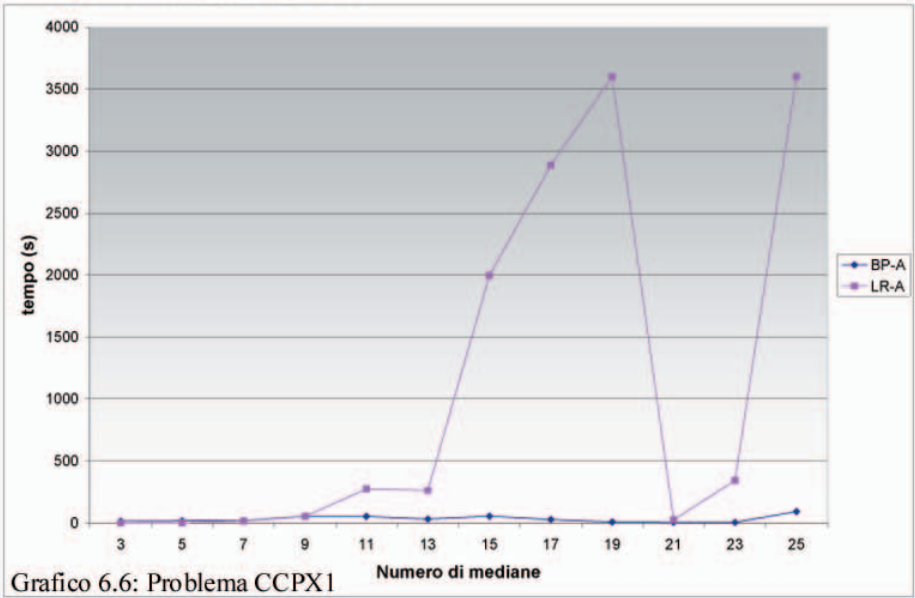

Grafico 6.6: Problema CCPX1

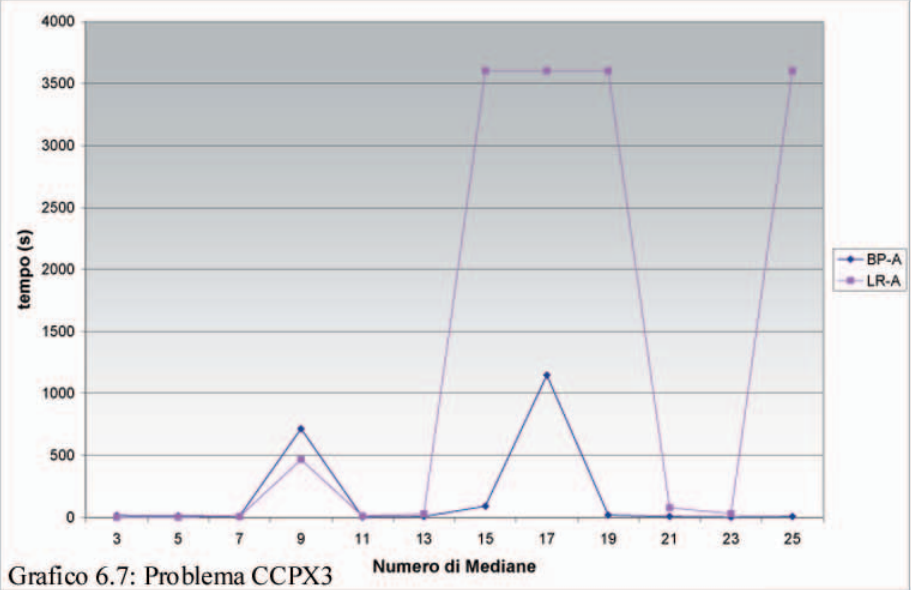

Grafico 6.7: Problema CCPX3

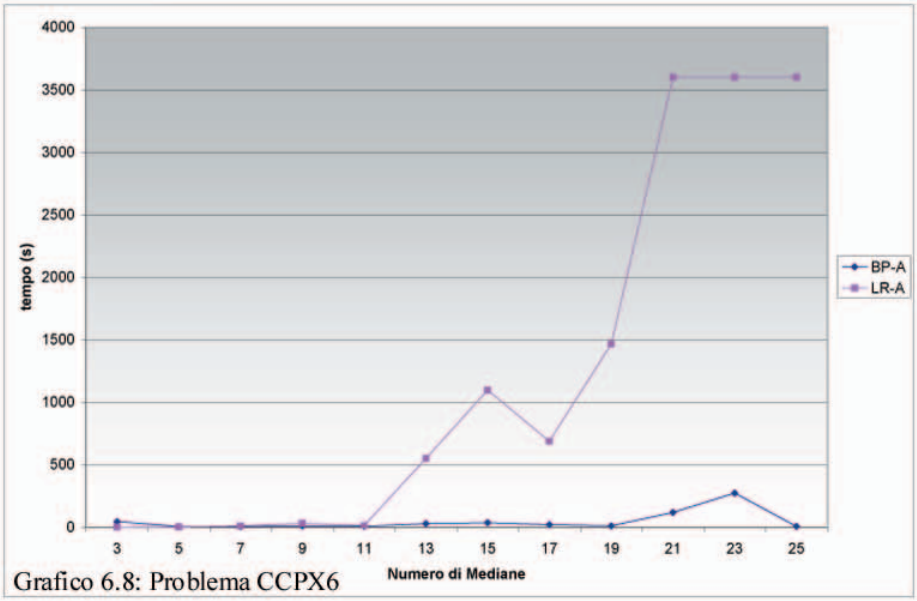

Grafico 6.8: Problema CCPX6

6.2 Osservazioni

E' infine possibile sintetizzare i risultati ottenuti:

- L'algoritmo basato su rilassamento Lagrangeano è molto efficiente per rapporti $\frac{N}{p}$ elevati, ma le sue prestazioni peggiorano al decrescere del valore di tale rapporto. Il rilassamento Lagrangeano è una tecnica ampiamente studiata e collaudata, semplice da implementare ed in cui la scelta dei parametri non gioca un ruolo fondamentale.

- L'algoritmo basato su branch & price offre un comportamento più omogeneo in funzione della variazione del rapporto $\frac{N}{p}$. Richiede l'implementazione di strutture dati piuttosto complesse; per questo l'*overhead* dovuto all'inizializzazione lo rende meno efficace dell'algoritmo **LR-A** per problemi dalle dimensioni ridotte con alto rapporto $\frac{N}{p}$. Branch & Price è una tecnica per la quale è necessaria un'accurata scelta dei parametri e delle strategie più opportune: la politica di branching "forbid" realizzata appositamente per l'algoritmo e presentata nel capitolo 5, influenza in modo determinante le prestazioni di **BP-A**. I tempi di calcolo ottenuti con **BP-A** dipendono anche dal MIP solver utilizzato.

Confronto con gli algoritmi esistenti

L'euristica presentata da Baldacci ed altri in [Bald02] (indicata di seguito come **BH**) è senza dubbio il termine di paragone più importante tra gli algoritmi proposti in letteratura per CPMP. Anche nei test proposti dagli autori per **BH** vengono utilizzate le istanze di classe A (CCPX1-A . . . CCPX-20-A), ed è stato posto un limite di 3600 secondi di utilizzo della cpu per terminare il calcolo.

Non è possibile operare un confronto diretto tra le prestazioni di **BP-A** e l'euristica **BH**, in quanto i tempi di calcolo sono riferiti a macchine con cpu diverse (MIPS R4400 200MHz per i test proposti da Baldacci e Pentium II 350MHz per quelli presentati in questo lavoro); inoltre, nel modello utilizzato in **BH** si suppone che $x_{jj} = 1 \Leftrightarrow y_j = 1$, mentre per **BP-A** vale il modello più

generale, come descritto nel capitolo 1 (queste condizioni logiche potrebbero, comunque, essere trattate in **BP-A** nella fase di preprocessing per ridurre le dimensioni del problema).

In ogni caso è possibile notare come l'algoritmo **BP-A** risolva *all'ottimo* le *stesse istanze* dell'euristica **BH** entro il tempo limite fissato, dimostrandosi anche relativamente *stabile* in funzione delle variazioni del numero p di mediane.

Possibili sviluppi

La struttura modulare permette di adattare facilmente l'algoritmo realizzato ad altri problemi di localizzazione multirisorse su grafo, ed alla gestione di problemi di CPMP con vincoli aggiuntivi. E' di interesse rilevante, infatti, la gestione di *regional constraints* [Murr97] e punti di servizio a capacità variabile [Syam97]. Come descritto nella sezione 5.6, sviluppando appositi algoritmi per il problema di *knapsack* con *independent set constraints* si potrebbero risolvere problemi con vincoli di *incompatibilità* tra utenti, di cui alcuni autori [Bald02] hanno già sottolineato l'importanza.

Bibliografia

[Bald02] Baldacci, R., E. Hadjiconstantinou, V. Maniezzo, A. Mingozzi, A new method for solving capacitated location problems based on a set partitioning approach, *Computers & Operations Research*, 29, 365-386 (2002)

[Beas85] Beasley, J.E., A note on solving large p-median problems, *European Journal of Operational Research*, 21, 270-273 (1985)

[Beas92] Beasley, J.E., Lagrangean Relaxation, *The Management School Imperial College - Technical Report* (1992)

[Chri81] Christofides, N., J.E. Beasley, A tree search algorithm for the p-median problem, *European Journal of Operational Research*, 10, 196-204 (1981)

[Chri83] Christofides, N., J.E. Beasley, Extentions to a Lagrangean relaxation approach for the capacitated warehouse location problem, *European Journal of Operational Research*, 12, 19-28 (1983)

[Corn77] Cornueojols, G., M.L. Fisher, G.L. Nemhauser, Location of bank accounts to optimize float: an analytic study of exact and approximate algorithms, *Management Science*, 23, 8, 789-810 (1977)

[Corn91] Cornuejols, G., R. Sridharan, J.M. Thizy, A comparison of heuristics and relaxations for the capacitated plant location problem, *European Journal of Operational Research*, 50, 280-297 (1991)

[Dant60] Dantzig, G.B., P. Wolfe, Decomposition principle for linear programs, *Operations Research*, 8, 101-111 (1960)

[DLT] Mirchandani, P.B., The p-median problem and generalizations, in Discrete Location Theory, Wiley & Sons, New York (1990)

[Erle82] Van Roy, T.J., D. Erlenkotter, Dual-based procedure for dynamic facility location, *Management Science* 28, 10 (1982)

[Fish86] Fisher, M.L., R. Jaikumar, L. Van Wassenhove, A multiplier adjustment method for the generalized assignment problem, *Management science*, 32/9, 1095-1103 (1986)

[FL] Zvi Drezner Ed., Facility Location, Springer Series in Operations Research, New York (1995)

[Galv79] Galvão R.D., A dual-bounded algorithm for the p-median problem, *Operations Research*, 28, 5 (1979)

[Gare79] Garey, M.R., D.S. Johnson, Computers and Intractability: a Guide to the Theory of NP-Completeness. Freeman, San Francisco, California (1979)

[Geof74] Geoffrion A.M., G.W. Graves, Multicommodity distribution system design by Benders Decomposition, *Mathematical Programming*, 17, 2, 198-228 (1974)

[Gold71] Goldman, A.J., Optimal center location in simple networks, *Transportation Science* 5, 212-221 (1971)

[Gold86] Golden, B., C. Skiscim, Using Simulated Annealing to solve routing and location problems, *Naval Research Logistic Quarterly*, 33, 261-279 (1986)

[Guig89] Guignard, M., M. Rosenwein, An improved dual-based algorithm for the generalized assignment problem, *Operations Research*, 37, 4, 658-663 (1989)

[Hanj85] Hanjoul, P., D. Peeters, A comparison of two dual-based procedures for solving the p-median problem, *European Journal of Operational Research*, 20, 387-396 (1985)

[Hans97] Hansen, P., Jaumard, B., Cluster analysis and mathematical programming, *Mathematical Programming* 79, 191-215 (1997)

[Held74] Held, M., P. Wolfe, H.P. Crowder, Validation of Subgradient Optimization, *Mathematical Programming* 6, 62-88 (1974)

[ICC] Bernasconi A., B. Codenotti, Introduzione alla Complessità Computazionale, Springer-Verlag Italia, Milano, 1998

[Jorn86] Jörnsten, K., M. Näsberg, A new lagrangean relaxation approach to the generalized assignment problem, *European journal of Operational Research*, 27, 313-323 (1986)

[Kari79] Kariv, O., S.L. Hakimi, An algorithmic approach to network location problems. Part 2. The p-median., *SIAM Journal of applied Mathematics* 37, 539-560 (1979)

[Klin86] Klincewicz, J.G, H. Luss, A Lagrangean Relaxation heuristic for capacitated facility location with single-source constraints, *Journal of Operational Research Society*, 37, 5, 495-500 (1986)

[Kueh63] Kuehn, A.A., M.J. Hamburger, A heuristic program for locating warehouses, *Management Science*, 9, 643-666 (1963)

[Lehr66] Feldman, E., F.A. Lehrer, T.L. Ray, Warehouse locations under continuous economies of scale, *Management Science*, 2 (1966)

[Mani98] Maniezzo, V., A. Mingozzi, R. Baldacci, A Bionomic approach to the capacitated p-median problem, *Journal of Heuristics*, 4, 263-280 (1998)

[MT89] Martello S., P. Toth, Knapsack Problems - Algorithms and Computer Implementations, John Wiley & Sons, (1989)

[Mulv84] Mulvey, J.M., P. Beck, Solving capacitated clustering problems, *European Journal of Operational Research*, 18, 339-348 (1984)

[Murr97] Murray, A.T., R.A. Gerrard, Capacitated service and regional constraints in location-allocation modeling, *Location Science*, 5, 2, 103-118 (1997)

[Naru76] Narula, S.C., U.I Ogbu, H.M. Samuelsson, An algorithm for the p-median problem, *Operations Research*, 25, 4 (1977)

[Neeb83] Neebe, A.W, M.R. Rao, An algorithm for the fixed charge assigning users to sources problem, *Journal of the Operational Research Society* 34, 11, 1107-1113 (1983)

[Osma94] Osman, I.H., N. Christofides, Capacitated clustering problems by hybrid Simulated Annealing and Tabu Search, *International Transactions in Operational Research* 13, 317-336 (1994)

[Pirk87] Pirkul, H., Efficient algorithms for the capacitated concentrator location problem, *Computers and Operations Research* 14, 3, 197-208 (1987).

[Pisi95] Pisinger, D., A minimal algorithm for the 0-1 knapsack problem, *Operations Research*, 46, 5, 758-767 (1995)

[Ross75] Ross, G.T., R.M. Soland, A branch and bound algorithm for the generalized assignment problem, *Mathematical Programming*, 8, 91-103 (1975)

[Ross77] Ross, G.T., R.M. Soland, Modeling facility location problems as generalized assignment problems, *Management Science*, 24, 3 (1977)

[Save95] Savelsbergh, M., A branch-and-price algorithm for the generalized assignment problem, *Operations Research*, 45, 6 (1997)

[SaveEA] Barnhart, C., E.L. Johnson, G.L. Nemhauser, M.W.P. Savelsbergh, P.H. Vance, Branch-and-price: column generation for solving huge integer programs, *Operations Research*, 46, 3 (1998)

[Srid95] Sridharan, R., The capacitated plant location problem, *European Journal of Operational Research* 87, 203-213 (1995)

[Syam97] Syam, S.S., A model for the capacitated p-facility location problem in global environments, *Computers & Operations Research*, 24, 11, 1005-1016 (1997)

[VRoy86] Van Roy, T.J., A cross decomposition algorithm for capacitated facility location, *Operations Research*, 34, 145-163 (1986)

[Wols98] Wolsey, L.A., Integer Programming. Wiley & Sons (1998)