Università degli Studi di Milano
Facoltà di Scienze Matematiche, Fisiche e Naturali
Dipartimento di Tecnologie dell'Informazione
Corso di Laurea in Informatica

Un algoritmo Branch & Price per il bidimensional level strip packing problem

Relatore: Prof. Giovanni Righini
Correlatore: Dott. Alberto Ceselli

Tesi di Laurea di:
Andrea Bettinelli
Matricola 616886

Anno Accademico 2003/2004

Indice

Capitolo 1

Bidimensional level strip packing

1.1 Introduzione

I problemi di packing bidimensionale trovano applicazione in numerosi scenari industriali. Spesso, infatti, è richiesto di collocare un insieme di elementi rettangolari in unità di stoccaggio standardizzate. Nell'industria del legno e del vetro, ad esempio, componenti rettangolari devono essere tagliate da grandi lastre di materiale. Nei magazzini, le merci devono essere posizionate sugli scaffali. Nell'impaginazione dei giornali, è necessario disporre in modo conveniente articoli e spazi pubblicitari. In altri contesti, come nell'industria tessile e in quella della carta, capita di avere un unico nastro di materiale, di cui si vuole utilizzare la minor quantità possibile per ottenere gli elementi richiesti.

In questa tesi, viene proposto un algoritmo per la soluzione *esatta* del *bidimensional level strip packing* (2LSP). Con il termine *esatta* intendiamo che l'ottimalità della soluzione trovata è garantita a priori. Di seguito viene fatta una breve panoramica sui problemi di packing bidimensionale, al fine di individuare 2LSP in tale contesto. Una bibliografia completa ed esaustiva sui problemi di *Cutting & Packing* può essere trovata in Maffioli e altri [24].

Bidimensional bin packing

Nel *bidimensional bin packing* (2BP) gli elementi rettangolari devono essere collocati all'interno di grandi unità standardizzate (*bins*), anch'esse di forma rettan-

golare, con altezza e larghezza fissate. L'obiettivo è quello di posizionare tutti gli elementi utilizzando il minor numero possibile di *bins*.

Generalmente si considera il caso in cui gli elementi hanno i lati paralleli alle pareti dei contenitori e non possono essere ruotati. Infatti, in situazioni reali, capita molto spesso che la rotazione non sia consentita. Si pensi, ad esempio, al caso in cui gli elementi debbano essere tagliati da una stoffa con una trama, che segue una certa direzione, o all'impaginazione degli articoli di un giornale.

Altra variante molto studiata è quella con *tagli a ghigliottina*, in cui i tagli devono attraversare il *bin* da un lato all'altro. Questa variante sorge in numerose applicazioni, in quanto spesso i limiti tecnologici dei macchinari di produzione impongono che i tagli avvengano in questo modo.

Per il 2BP si trovano in letteratura numerosi algoritmi di approssimazione. Chung et al. [5], ad esempio, hanno proposto una euristica per il 2BP, chiamata *Hybrid First-Fit* (HFF), che ha complessità computazionale $O(n \log n)$. È stato condotto anche uno studio del caso peggiore , in cui si dimostra che la soluzione trovata da HFF è al più $\frac{17}{8}\mathrm{OPT}(I) + 5$, dove con $OPT(I)$ indichiamo la soluzione ottima di una istanza I del problema. Altre euristiche sono state studiate da Berkey e Wang [2] e, più recentemente Lodi et al. [20, 21, 22].

Sono stati proposti anche approcci metaeuristici. In particolare, Lodi e altri [20, 21, 22] hanno sviluppato algoritmi *taboo search* per diverse varianti del 2BP. Færø e altri [8] hanno esteso al 2BP l'approccio utilizzato da Dowsland [7] per lo *strip packing*, basato su tecniche di ricerca locale .

Un approccio enumerativo per la soluzione esatta del 2BP è stato proposto, di recente, da Martello e Vigo [26] . Questo algoritmo è basato su uno schema di *branching* a 2 livelli. Prima gli elementi vengono assegnati ai *bins*, poi vengono posizionati all'interno del *bin* a cui appartengono.

Bidimensional strip packing

Nel caso del *bidimensional strip packing* (2SP), gli oggetti devono essere posizionati in un'unica striscia, di larghezza fissata ed altezza potenzialmente infinita. Lo scopo è impaccare gli elementi minimizzando l'altezza della parte di striscia

occupata dagli oggetti. In tal modo si riduce al minimo lo spazio scartato. Anche per il 2SP si possono pensare varianti simili a quelle viste per il 2BP.

Recentemente, Kenyon e Rémila [18] hanno proposto uno schema di approssimazione interamente polinomiale per il 2SP.

Dowsland [7] ha sviluppato un algoritmo basato su *simulated annealing*, mentre Jacobs [16] ha proposto un algoritmo genetico.

Martello et al. [25] hanno proposto un algoritmo *branch & bound*, riuscendo a risolvere istanze del problema contenenti fino a 200 elementi.

Bidimensional cutting stock

Un terzo importante caso di packing bidimensionale si ha quando ad ogni elemento viene associato un profitto e l'obiettivo è scegliere un sottoinsieme di elementi, da collocare in un solo *bin*, massimizzando la somma dei profitti. Tale problema è generalmene indicato come *Bidimensional Cutting Stock*, sebbene sia stato introdotto da Gilmore and Gomory[14] con il nome di *Two-dimensional (Cutting) Knapsack* (2KP).

Fekete and Schepers [9] hanno ideato una formulazione di questo problema basata su grafi, che può essere facilmente estesa a più dimensioni, e su di essa hanno costruito un approccio enumerativo per ottenere soluzioni globalmente ottime.

Un altro algoritmo esatto per il 2KP è stato proposto da Caprara e Monaci [3]. Lo schema utilizzato si basa su un rilassamento del problema in una dimensione. Gli autori hanno ottenuto risultati sperimentali paragonabili a quelli presentati in [9].

Bidimensional level packing

Esistono varianti a livelli sia per il *bin packing*, che per lo *strip packing*, che indicheremo con 2LBP e 2LSP, rispettivamente. In questo caso gli elementi non possono essere impaccati liberamente, ma devono essere organizzati in livelli, all'interno dei quali gli elementi non possono essere sovrapposti.

Il primo livello è il fondo del *bin/strip* e gli oggetti che vi vengono inseriti hanno la base che poggia su di esso. Il livello successivo è determinato dalla linea orizzontale tracciata all'altezza del più alto degli oggetti del livello sottostante e così via.

Anche per questo problema sono noti alcuni algoritmi di approssimazione. In particolare alcune euristiche costruttive con garanzie [23] sono descritte nella sezione 3.2.

Per quanto riguarda gli algoritmi esatti, Gilmore e Gomory [15] hanno proposto un approccio basato su generazione di colonne, in cui il sottoproblema veniva risolto mediante un algoritmo di programmazione dinamica in due fasi.

In [23], Lodi et al. hanno descritto una formulazione compatta del problema, che utilizza un numero polinomiale di variabili e vincoli, trattata nella sezione 1.2.2, analizzando anche le prestazioni di alcuni bound duali per i problemi di packing bidimensionale a livelli.

Scopo della tesi

Il *bidimensional level strip packing problem* è NP-hard, in quanto contiene, come caso particolare, il *bin packing* in una dimensione (quando tutti gli elementi hanno altezza unitaria). La dimostrazione che il *bin packing* sia NP-completo può essere trovata in [13]. Di conseguenza, a meno che P = NP, non è possibile trovare un algoritmo efficiente per risolvere il 2LSP all'ottimo, ma diviene necessario ricorrere all'enumerazione delle sue soluzioni. L'algoritmo, proposto in questa tesi, ne effettua una enumerazione implicita, attraverso una strategia *Branch & Price*(vedi sezione 2.3).

Struttura dell'esposizione

Nella prossima sezione di questo capitolo, viene introdotto, in modo più formale il *bidimensional level strip packing problem*, descrivendone alcune possibili formulazioni ed i relativi modelli.

Il capitolo 2 contiene dei richiami teorici, riguardanti *column generation* e scomposizione di Dantzig-Wolfe.

Nel capitolo 3, viene proposto un algoritmo *Branch & Price* per il 2LSP.

Infine, nel capitolo 4, sono descritti i risultati sperimentali ottenuti.

1.2 Formulazioni

1.2.1 Formulazione naturale

Dati:

Sia $\Omega = \{1, \ldots, N\}$ l'insieme degli elementi da posizionare.

Ad ogni elemento $i \in \Omega$ sono associati un coefficiente $w_i \in \mathbb{R}_+$, cha rappresenta la larghezza dell'elemento, ed un coefficiente $h_i \in \mathbb{R}_+$, che ne indica l'altezza.

Assumiamo, da qui in poi, che gli elementi siamo ordinati per valori non crescenti di h_i.

La striscia, in cui devono essere impaccati gli oggetti, ha larghezza fissata W.

Variabili:

Il 2LSP non pone limitazioni al numero di livelli utilizzati per impaccare gli oggetti, ma essi non potranno mai essere più di N. Qui viene proposta una formulazione naturale del problema in cui vengono, quindi, utilizzate $N * N$ variabili binarie x_{ij}. In particolare $x_{ij} = 1$ se i-esimo oggetto è assegnato al j-esimo livello, $x_{ij} = 0$ altrimenti.

N variabili $y_j \in \mathbb{R}$ indicano, invece, l'altezza di ogni livello.

Vincoli:

La somma delle larghezze degli elementi, contenuti in un livello, non deve superare la larghezza W della striscia.

Ogni oggetto deve essere inserito esattamente in un livello.

L'altezza di ogni livello deve essere maggiore o uguale a quella dell'elemento più alto collocato nel livello stesso.

Funzione obiettivo:

L'obiettivo è minimizzare l'altezza della striscia necessaria a contenere gli oggetti, ovvero minimizzare la somma delle altezze dei livelli.

Modello:

Una formulazione naturale del 2LSP è, dunque, la seguente:

FN)

$$\min \sum_{j=1}^{N} y_j$$

s.t.

$$y_j \geq h_i x_{ij} \qquad \forall i, j \in \{1, \ldots, N\}$$

$$\sum_{i=1}^{N} w_i x_{ij} \leq W \qquad \forall j \in \{1, \ldots, N\}$$

$$\sum_{j=1}^{N} x_{ij} = 1 \qquad \forall i \in \{1, \ldots, N\}$$

$$x_{ij} \in \{0, 1\} \qquad \forall i, j \in \{1, \ldots, N\}$$

1.2.2 Formulazione compatta

Una formulazione più compatta del 2LSP è stata proposta da Lodi e altri [19].

FC)

$$\min \sum_{j=1}^{N} h_j y_j \tag{1.1}$$

s.t.

$$\sum_{j=1}^{i-1} x_{ij} + y_i = 1 \qquad \forall i \in \{1, \ldots, N\} \tag{1.2}$$

$$\sum_{i=j+1}^{N} w_i x_{ij} \leq (W - w_j) y_j \qquad \forall j \in \{1, \ldots, N-1\} \tag{1.3}$$

$$y_j, x_{ij} \in \{0, 1\} \qquad \forall j \in \{1, \ldots, N-1\},\ \forall j \in \{j+1, N\} \tag{1.4}$$

Esistono N potenziali livelli, il j-esimo dei quali è associato all'oggetto j, che lo inizializza e ne determina l'altezza. Le variabili binarie y_j, con $j \in \{1, \ldots, N\}$, assumono valore 1 se l'oggetto j inizializza un livello, 0 altrimenti.

Come nella formulazione precedente, la composizione dei livelli è descritta dalle variabili x_{ij}, che assumono valore 1 se l'elemento i è inserito nel livelo j, 0 altrimenti. Sfruttando l'assunzione che gli elementi siano ordinati in base all'altezza e che il primo elemento inserito in ogni livello sia il più alto del livello stesso, le variabili x_{ij} con $i \leq j$ possono essere fissate a 0 e rimosse dal modello.

Infine, il blocco di vincoli (1.3) permette di inserire elementi in un livello j solo se il livello stesso è stato inizializzato dal corrispondente elemento ($y_j = 1$).

1.2.3 Formulazione come CCLP

Il 2LSP può essere riformulato come *capacitated concentrator location problem* (CCLP).

Su un grafo avente $|I|$ nodi, vengono individuati $|J|$ potenziali siti candidati all'apertura di concentratori (i punti di servizio). Ogni nodo $i \in I$ del grafo ha una domanda a_{ij}, che deve essere soddisfatta, ed ogni concentratore $j \in J$ ha una capacità. L'obiettivo è soddisfare la domanda di ogni nodo, aprendo dei concentratori in alcuni dei siti candidati ed assegnando i nodi ai vari concentratori, senza eccederne la capacità c_j. La domanda di ogni nodo può essere soddisfatta da un solo concentratore e, in particolare, ogni concentratore deve appartenere al cluster da esso descritto. Si vuole, inoltre, minimizzare il costo dell'operazione, dato dai

costi di trasporto d_{ij} (proporzionali alla distanza del nodo i dal concentratore j a cui è assegnato) e dai costi fissi f_j per l'apertura di ogni concentratore j. Una formulazione del CCLP è la seguente:

CCLP)

$$\min \sum_{i \in I} d_{ij} x_{ij} + \sum_{j \in J} f_j x_{jj} \tag{1.5}$$

s.t.

$$\sum_{j \in J} x_{jj} = 1 \quad \forall i \in I \tag{1.6}$$

$$\sum_{i \in I} a_{ij} x_{ij} \leq c_j x_{jj} \quad \forall j \in J \tag{1.7}$$

$$x_{ij} \in \{0, 1\} \quad \forall i \in I, j \in J \tag{1.8}$$

I vincoli di assegnamento (1.6) impongono che ogni nodo sia servito da uno e un solo concentratore. Il secondo blocco di vincoli, invece, fa in modo che la somma delle domande a_{ij} dei nodi, assegnati ad un concentratore, non ecceda la capacità del concentratore stesso.

Per formulare il 2LSP come CCLP bisogna, per ogni $i \in I$, $j \in J$:

- fissare a zero i costi di assegnamento d_{ij};
- assegnare come costo di apertura del concentratore j l'altezza h_j del j-esimo elemento;
- porre la capacità c_j dei concentratori uguale a W (la larghezza della striscia);
- assegnare ad ogni a_{ij} la larghezza w_i dell'i-esimo elemento se $i \leq j$, $+\infty$ altrimenti.

1.2.4 Formulazione come problema di set partitioning

L'algoritmo proposto nel capitolo 3 utilizza la seguente formulazione del 2LSP come problema di *set partitioning*. Questo modello, detto *Master Problem* (MP) si ottiene applicando al 2LSP la scomposizione di Dantzig-Wolfe (vedi sezione 2.1).

MP)

$$\min \sum c_k z_k \tag{1.9}$$

s.t.

$$\sum_k x_i^k z_k = 1 \qquad \forall i \in \{1, \ldots, N\} \tag{1.10}$$

$$z_k \in \{0, 1\} \qquad \forall k \in K \tag{1.11}$$

Il *Master problem* contiene un insieme di colonne $\begin{pmatrix} c_k \\ x^k \end{pmatrix}$, in cui il vettore $x^k \in \{0,1\}^N$ individua un sottoinsieme di elementi, la cui somma della larghezze non superi quella della striscia. Quindi, $\forall k \in K$

$$\begin{cases} \sum_{j=1}^{N} w_j x_j^k \leq W \\ x_i^k \in \{0,1\} \quad \forall i \in \{1, \ldots, N\} \end{cases}$$

$$h_k = \max_{j \in \{1,\ldots,N\} | x_j^k = 1} \{h_j\}$$

Capitolo 2

Richiami teorici

2.1 Scomposizione di Dantzig - Wolfe

La scomposizione di Dantzig Wolfe è una tecnica che consente di dividere un grosso problema in sottoproblemi più piccoli, che possano essere risolti in modo più semplice.

Affinchè questa tecnica possa essere applicata in modo vantaggioso, il problema deve avere le seguenti caratteristiche:

- variabili separate in gruppi logici distinti;
- molti dei vincoli coinvolgono un solo gruppo di variabili;
- pochi vincoli che raggruppano tutte le variabili.

Consideriamo un problema con una formulazione di questo tipo

$$\max \sum_{k=0}^{K} c^k x^k$$

s.t.

$$\begin{array}{lllll} A^1x^1 & +\ldots & +A^Kx^K & = & b \\ D^1x^1 & & & \leq & d_1 \\ & \ddots & & \vdots & \vdots \\ & & D^Kx^K & \leq & d_K \end{array}$$

$$x^1 \in \mathbb{Z}_+^{n_1}, \ldots, x^K \in \mathbb{Z}_+^{n_K}$$

Esistono, in questo caso, K insiemi $X^k = \{x^k \in Z_+^{n_k} : D^kx^k \leq d_k\}$indipendenti, per $k = 1, \ldots, K$, e solo il vincolo $\sum_{k=1}^{K} A^kx^k = b$ li lega insieme. Assumendo che ogni insieme X^k contenga un numero grande, ma finito, di punti $\{x^{k,t}\}_{t=1}^{T_k}$, si ha che

$$X^k = \{x^k \in \mathbb{R}^{n_k} : x^k = \sum_{t=1}^{T_k} \lambda_{k,t}x^{k,t}, \sum_{t=1}^{T_k} \lambda_{k,t} = 1, \lambda_{k,t} \in \{0,1\} \text{ con } t = 1, \ldots, T_k\}$$

Sostituendo x^knel problema si ottiene

$$\max \sum_{k=1}^{K} \sum_{t=1}^{T_k} (c^kx^{k,t})\lambda_{k,t}$$

s.t.

$$\sum_{k=1}^{K} \sum_{t=1}^{T_k} (A^kx^{k,t})\lambda_{k,t} = b$$

$$\sum_{t=1}^{T_k} \lambda_{k,t} = 1 \quad \forall k = 1, \ldots, K$$

$$\lambda_{k,t} \in \{0,1\} \; \forall t = 1, \ldots, T_k \; \forall k = 1, \ldots, K$$

in cui si ha una colonna $\begin{pmatrix} c^kx \\ A^kx \\ e_k \end{pmatrix}$ per ogni elemento $x \in X^k$. Chiameremo questa formulazione *Master Problem* (MP).

2.2 Column generation

Consideriamo il rilassamento lineare del *Master Problem*, descrito nel paragrafo precedente:

LMP)

$$\max \sum_{k=1}^{K} \sum_{t=1}^{T_k} (c^k x^{k,t}) \lambda_{k,t}$$

s.t.

$$\sum_{k=1}^{K} \sum_{t=1}^{T_k} (A^k x^{k,t}) \lambda_{k,t} = b \tag{2.1}$$

$$\sum_{t=1}^{T_k} \lambda_{k,t} = 1 \quad \forall k = 1, \ldots, K \tag{2.2}$$

$$\lambda_{k,t} \geq 0 \; \forall t = 1, \ldots, T_k \; \forall k = 1, \ldots, K$$

Indichiamo con $\{\pi\}_{i=1}^{m}$ le variabili duali associate ai vincoli di unione (2.1) e con $\{\mu\}_{k=1}^{K}$ le variabili duali associate ai vincoli (2.2), detti vincoli di convessità.

L'idea è di risolverlo utilizzando il simplesso primale. Tuttavia, in questa formulazione, il numero di variabili potrebbe essere enorme. Infatti, molto spesso le colonne del MP codificano oggetti combinatori, come cammini in un grafo, insiemi o permutazioni. Quindi, scegliere ad ogni iterazione del simplesso la variabile candidata ad entrare in base, mediante enumerazione, rischia di diventare computazionalmente impossibile. Anzichè valutare le colonne una ad una, si può pensare, allora, di affidarne la scelta a K problemi di ottimizzazione.

In pratica, si considera un LMP ridotto (RLMP), contenente un sottoinsieme di colonne, scelte in modo da garantire la presenza di una soluzione di base ammissibile e lo si risolve all'ottimo utilizzando, ad esempio, il simplesso primale. A questo punto, bisogna verificare se esistono colonne vantaggiose (aventi costo ridotto positivo, nel caso di un problema di massimizzazione). Anziché esamina-

re esplicitamente tutti i punti di X^k, li trattiamo implicitamente risolvendo un problema di ottimizzazione:

$$\zeta_k = \max\left\{(c^k - \pi A^k)x - \mu_k : x \in X^k\right\}$$

Se $\zeta_k > 0$ per qualche $k = 1, \ldots, K$, la colonna corrispondente alla soluzione ottima $\widetilde{x}$ del sottoproblema ha costo ridotto positivo. Inserendo la colonna $\begin{pmatrix} c^k\widetilde{x}^k \\ A^k\widetilde{x}^k \\ e_k \end{pmatrix}$ si ottiene un nuovo RLMP, che può essere nuovamente ottimizzato con facilità.

L'algoritmo termina quando $\zeta_k = 0$ per $k = 1, \ldots, K$. A quel punto, non ci sono più colonne con costo ridotto positivo e la soluzione trovata è ottima anche per LMP.

2.3 Branch & Price

Il *Branch & Price* è una variante delle tecniche Branch & Bound. Si tratta di strategie per la soluzione di problemi computazionalmente molto complessi, basate sulla enumerazione implicita delle soluzioni. Tale risultato viene ottenuto esplorando un albero in cui ogni nodo corrisponde ad un sottoproblema. In particolare, la radice di tale albero è costituita dal problema iniziale, in cui tutte le variabli sono libere. In ogni nodo, lo spazio delle soluzioni (S_0) viene partizionato in K sottoinsiemi disgiunti S_k, tali che $\bigcup_{k=1\ldots K} S_k = S_0$. Questo si realizza imponendo dei vincoli aggiuntivi, che fissano alcune variabili. Nelle foglie dell'albero si troveranno le soluzioni del problema.

Con tecniche euristiche è possibile cercare di ottenere delle soluzioni ammissibili. Esse costituiscono un *bound primale*.

Per ogni nodo esplorato, viene valutato un *bound duale*, ovvero un valore che indica un limite alla migliore soluzione ammissibile che si può trovare nel sottoalbero radicato in tale nodo. Se il bound duale risulta superiore al bound primale, il nodo viene chiuso ed i suoi figli non verranno esplorati in quanto sicuramente non vantaggiosi.

La caratteristica distintiva degli algoritmi Branch & Price è l'utilizzo di tecniche basate su generazione di colonne per la valutazione del bound duale.

Capitolo 3

Algoritmo Branch & Price

3.1 Bound duale

Master problem rilassato

Consideriamo la formulazione del 2LSP come *set partitioning* (pag. 12)

I vincoli (1.10) possono essere posti in forma di $\geq$. Infatti, data una qualsiasi soluzione in cui un elemento è assegnato a più di un livello, ne esiste una non peggiore, ottenibile eliminando tale elemento dal livello più svantaggioso.

Rilassando i vincoli di integrità sulle variabili z_k otteniamo il seguente master problem rilassato

LMP)

$$\min \sum_{k \in K} c_k z_k \tag{3.1}$$

s.t.

$$\sum_{k \in K} x_i^k z_k \geq 1 \qquad \forall i \in \{1, \ldots, N\} \tag{3.2}$$

$$0 \leq z_k (\leq 1) \qquad \forall k \in K \tag{3.3}$$

Per le considerazioni riportate sopra, è possibile anche eliminare i vincoli $z_k \leq 1$: un valore superiore ad 1, per tali variabili, non potrà mai essere vantaggioso.

Si può dimostrare [27] che il bound duale ottenuto risolvendo LMP all'ottimo, è equivalente alla soluzione del *problema duale lagrangiano*, quando i vincoli di *partitioning* vengono rilassati. Il numero di variabili di questa formulazione cresce esponenzialmente nel numero di elementi del problema ($O(2^N)$). All'aumentare di N, quindi, risolvere direttamente LMP diventa computazionalmete proibitivo.

Problema di pricing

Se indichiamo con λ il vettore delle variabili duali associate ai vincoli (3.2), il costo ridotto di ogni colonna k del MP sarà dato da

$$\pi_k = c_k - \sum_{j=1}^{N} \lambda_j x_j^k$$

dove $\lambda_j \geq 0 \, \forall j \in \{1, \ldots, N\}$, essendo associate a vincoli di $\geq$.

Risolvere LMP con il metodo del simplesso, richiederebbe la valutazione, ad ogni iterazione, del costo ridotto di $|K|$ colonne, rendendo i tempi di calcolo proibitivi. Tuttavia, come descritto nel capitolo 2, è possibile considerare un problema contenente solo un sottoinsieme delle colonne di LMP (R-LMP), scelte in modo da garantire la presenza di una soluzione di base ammissibile. Una volta risolto all'ottimo, la scelta di nuove variabili da inserire in R-LMP, in quanto candidate ad entrare in base, viene affidata ad un opportuno *problema di pricing* (PP). Nel nostro caso, PP è il seguente *knapsack penalizzato*:

PP)

$$\max \sum_{j=1}^{N} x_j^k \lambda_j \; - c_k \tag{3.4}$$

s.t.

$$\sum_{j=1}^{N} w_j x_j^k \leq W \tag{3.5}$$

$$c_k \geq h_j x_j^k \qquad \forall j \in \{1, \ldots, N\} \tag{3.6}$$

$$x_j^k \in \{0, 1\} \quad \forall j \in \{1, \ldots, N\} \tag{3.7}$$

Si può quindi inserire in R-LMP la colonna con costo ridotto più negativo, ottenuta risolvendo all'ottimo PP.

Si procede, poi, iterativamente, fino a che la soluzione di PP non individua alcuna nuova colonna potenzialmente vantaggiosa. A quel punto la soluzione di base trovata è ottima. Il valore ottenuto è un bound duale valido per MP (e quindi anche per 2LSP).

Per risolvere il *knapsack penalizzato* (PKP) è stato utilizzato il seguente algoritmo, proposto da Ceselli e Righini [4].

Chiamiamo *capofila* l'elemento che determina l'entità del termine di penalizzazione c_k (l'elemento più alto inserito nel livello) e indichiamo con S l'insieme dei candidati capofila. Inizialmente gli elementi vengono ordinati per valori non crescenti di h_j. Quindi, se ne considera il sottoinsieme $\{l, \ldots, N\}$ tale che $\sum_{j=l}^{N} w_j \leq W$ e $\sum_{j=l-1}^{N} w_j \geq W$. La soluzione ottimale del PKP sugli elementi $l, \ldots, N$ può essere calcolata in tempo lineare, in quanto il vincolo di capacità (3.5) è inattivo. Il valore ottenuto viene utilizzato come *lower bound* z_{LB} e gli elementi $l, \ldots, N$ non vengono più considerati come candidati capofila.

Altri elementi, che non possono essere capofila vengono individuati con questo test di riduzione: ogni volta che $\lambda_j \leq h_j - h_{j-1}$ l'elemento j può essere eliminato dall'insieme S dei candidati capofila. Infatti, data una soluzione con j come capofila, se ne troverebbe una migliore semplicemente eliminando j da essa. La fase successiva prevede di calcolare un *upper bound* per ogni elemento di S.

$$\mu_j = LPK_j - h_j \quad \forall i \in S$$

Gli elementi vengono ordinati in base alla loro efficienza $e_j = \frac{\lambda_j}{h_j}$. La soluzione ottima del rilassamento continuo del corrispondente problema di *knapsack*(LPK) viene calcolata inserendo gli elementi in ordine di efficienza, fino a che un elemento non eccede la capacità residua. Tale elemento viene preso con un valore

frazionario. Quindi, il capofila viene scartato e ne viene inserito uno nuovo. Questa operazione causa uno scarto (o una violazione) nel vincolo di capacità. La nuova soluzione ottima si ottiene inserendo (o rimuovendo) elementi, sempre in ordine di efficienza, fino alla saturazione della capacità.

Terminata questa fase di inizializzazione, si sceglie iterativamente il candidato capofila k per cui

$$k = argmax_{j \in S}\{\mu_j\}$$

Non appena $\mu_k \leq z_{LB}$ l'algoritmo termina.

Scelto k, viene risolto un *knapsack* binario in cui gli unici elementi disponibili sono quelli con indice non inferiore a k. Indichiamo con $\overline{x}$ la soluzione del seguente *knapsack* binario

$$\overline{x} = argmax \left\{ \sum_{j=k}^{N} \lambda_j x_j : \sum_{j=k}^{N} h_j x_j \leq W,\ x_j \in 0,1 \forall j = k, \ldots, N \right\}$$

Se t è il capofila della soluzione $\overline{x}$ possiamo eliminare da S tutti gli elementi tra k e t.

Da $\overline{x}$ può essere ricavata una soluzione ammissibile per il PKP: $KP(k) - h_t$. Tale soluzione può migliorare il bound primale $z_L B$. Inoltre, l'informazione derivata dalla soluzione del *knapsack* viene sfruttata per calcolare un ulteriore *upper bound* v_j^k per tutti i $j > k$.

$$v_j^k = KP(k) - p_j \quad \forall j = k+1, \ldots, N$$

Vengono quindi scartati, dall'insieme dei potenziali capofila, tutti i j per cui $v_j^k < z_{LB}$.

Lo pseudo-codice dell'algoritmo è riportato in figura 3.1.

Optimization algorithm for the PKP

```
begin
  /* Inizializzazione */
  z_LB := -∞; l := N
  while (Σ_{j=l}^{N} w_l ≤ W) do
    if (Σ_{j=l}^{N} λ_j − h_l > z_LB) then
      z_LB := Σ_{j=l}^{N} λ_j − h_l; x*_j := 0 ∀j < l; x*_j := 1 ∀j ≥ l
    l := l − 1
  S := {1, ..., l}
  /* Riduzione */
  for each j ∈ S do if (λ_j ≤ h_j − h_{j+1}) then S := S \ {j}
  /* Calcolo upper bound dal rilassamento lineare */
  for each j ∈ S do μ_j := LKP_j − h_j
  /* Esamino tutti i candidati capofila */
  repeat
    /* Scalgo il candidato più promettente */
    k := argmax_{j∈S}{μ_j}; S := S \ {k}
    /* Test di terminazione*/
    if (μ_k ≤ z_LB) then goto end
    /* Risolvo un KP, salvo la soluzione ottima e il suo valore */
    x̄ := argmax{Σ_{i=k}^{N} λ_i x_i : Σ_{i=k}^{N} w_i x_i ≤ W, x_i ∈ {0,1} ∀i = k, ..., N}
    KP(k) := max{Σ_{i=k}^{N} λ_i x_i : Σ_{i=k}^{N} w_i x_i ≤ W, x_i ∈ {0,1} ∀i = k, ..., N}
    for each j = 1, ..., k − 1 do x̄_j := 0
    /* Identifico il capofila e applico il criterio di dominazione */
    t := k
    while (x̄_t = 0) do
      h := t + 1; S := S \ {t}
    /* Aggiorno il lower bound */
    if (KP(k) − h_t > z_LB) then
      z_LB := KP(k) − h_t; x* := x̄
    /* Calcolo l'upper bound ν per scartare più candidati */
    j := t + 1
    while (KP(k) − h_j ≤ z_LB) do
      j := j + 1; S := S \ {j}
  until (S = ∅)
end
```

Figura 3.1: Pseudo-codice per l'algoritmo di ottimizzazione per il PKP

3.2 Bound primale

Algoritmi di approssimazione

Per il 2LSP esistono tre euristiche classiche, derivate dal 1BP. Anch'esse si basano sull'assunzione che gli elementi siano ordinati per altezze non crescenti. Queste euristiche seguono tutte il medesimo schema di tipo *greedy*, ma differiscono per il modo in cui viene scelta la posizione degli elementi. Lo pseudo-codice di questi algoritmi è descritto in figura 3.2. E_i indica l'insieme degli elementi inseriti nel i-esimo livello.

```
begin
  /* Inizializzazione */
  s:= 0
  for each i ∈ Ω do E_i:= ∅
  /* Inserimento */
  for each i ∈ Ω do Inserisci(j)
end
```

Next-Fit Decreasing Height (NFDH):

```
Inserisci(j)
begin
  if W − Σ_{i∈E_s} w_i < w_j then
    s:= s + 1
  E_s:= E_s ∪ {j}
end
```

First-Fit DecreasingHeight (FFDH):

```
Inserisci(j)
begin
  k:= 1
  while W − Σ_{i∈E_s} w_i ≥ w_j do k:= k + 1
  if k > s then s:= s + 1
  E_s:= E_s ∪ {j}
end
```

Best-Fit Decreasing Height:

```
Inserisci(j)
begin
  r:= W
  l:= 0
  k:= 1
  while k ≤ s do
    r_k:= W − Σ_{i∈E_s} w_i
    if r_k ≥ w_j andr_k < r then
      r:= r_k
      l:= k
  if l > 0 then E_l:= E_l ∪ {j}
  else s:= s + 1; E_s:= {j}
end
```

Figura 3.2: Pseudo-codice per le euristiche NFDH, FFDH e BFDH

Quindi, se indichiamo con j l'elemento corrente e con s l'ultimo livello creato:

- *Next-Fit Decreasing Height* (NFDH): l'elemento j è impaccato, allineato a

sinistra, nel livello s, se non eccede la larghezza della striscia; altrimenti viene creato un nuovo livello ($s := s + 1$) e j viene inserito in esso;

- *First-Fit DecreasingHeight* (FFDH): l'elemento j è inserito, allineato a sinistra, nel primo livello in grado di contenerlo; se non ve ne sono, viene inizializzato un nuovo livello, come nel caso del NFDH;
- *Best-Fit Decreasing Height*: l'elemento j è inserito, allineato a sinistra, nel livello, capace di ospitarlo, per cui risulta minimo lo spazio orizzontale inutilizzato; anche in questo caso, se nessun livello può contenere j, ne viene creato uno nuovo.

Colman e altri [6] hanno analizzato le prestazioni di NFDH e FFDH per la soluzione del *bidimensional strip packing problem* e ne hanno determinato il comportamento asintotico del caso pessimo. Dati un problema di minimizzazione P ed un algoritmo di approssimazione A, indichiamo con $A(I)$ e $OPT(I)$ il valore prodotto da A e la soluzione ottima, rispettivamente, per una istanza I del problema P. Colman et al. [6] hanno provato che, se le altezze sono normalizzate in modo che $\max_j \{h_j\} = 1$, allora

$$NFHD(I) \leq 2 * OPT(I) + 1 \tag{3.8}$$

e

$$FFDH(I) \leq \frac{17}{10} OPT(I) + 1 \tag{3.9}$$

Entrambi i bounds sono stretti (nel senso che il fattore moltiplicativo è il più piccolo possibile) e, nel caso le altezze h_j non siano normalizzate, solo la costante additiva viene influenzata. Sia NFDH che FFDH possono essere implementate in modo da richiedere tempo $O(n \log n)$.

Inoltre, per il 2LSP, queste euristiche risultano notevolmente efficaci, riuscendo non di rado a trovare la soluzione ottima (vedi capitolo 4).

Il nostro algoritmo sfrutta la soluzioni fornite da FFDH. Tale euristica viene lanciata prima di valutare il nodo radice dell'albero di *branch,* con il doppio scopo di

trovare subito un buon bound primale e di generare delle colonne da inserire nel R-LMP iniziale

Euristica di arrotondamento

Questa euristica cerca di costruire una soluzione ammissibile, partendo dalla soluzione frazionaria del rilassamento di RMP. Lavora in modo *greedy,* arrotondando a 1 le variabili con il valore maggiore, fino ad aver ottenuto la copertura di tutti gli elementi, e ponendo tutte le altre a 0.

Anche a causa dei buoni risultati ottenuti con gli algoritmi di approssimazione descritti nel paragrafo precedente, questa euristica non si è dimostrata molto utile. Raramente, infatti, è riuscita a trovare soluzioni ammissibili migliori del *bound* primale già noto.

3.3 Enumerazione implicita

Come è stato detto nella sezione 2.3, l'enumerazione implicita delle soluzioni di un problema equivale alla visita di un albero di *branching*, i cui nodi rappresentano dei sottoproblemi. Possono essere adottati diversi criteri per la generazione dei sottoproblemi-figli a partire da un sottoproblema padre (schema di *branch*) e per la loro esplorazione (strategia di ricerca).

3.3.1 Strategie di ricerca

Depth first

L'esplorazione *depth first* prevede di analizzare per primi gli ultimi sottoproblemi generati, visitando l'albero di *branch.* In questo modo si raggiungono rapidamente alcune foglie, ottenendo delle soluzioni ammissibili, che costituiscono un bound primale. Ciò si rivela importante qualora non si riescano ad ottenere delle buone soluzioni intere nei nodi interni.

Inoltre, come descritto più avanti, è utile cercare di contenere le dimensioni di R-LMP, eliminando le colonne meno vantaggiose. Poichè, con l'esplorazione *depth*

first, almeno un figlio viene valutato immediatamente dopo il proprio padre, si riduce il rischio di perdere colonne vantaggiose per cancellazione. È, infatti, molto improbabile che colonne non buone per un nodo diventino vantaggiose nei suoi figli.

Considerazioni analoghe possono essere fatte per le soluzioni di base: in ogni sottoproblema figlio è possibile valutare R-LMP partendo dalla soluzione di base ottima del padre, che è, in generale, vicina all'ottimo del figlio. Infine, l'esplorazione in profondità, mantenendo pochi nodi aperti in ogni momento, consente di mantenere molto contenuto il consumo di memoria.

Best first

Con la politica *best first*, invece, viene data precedenza ai sottoproblemi che presentano un bound duale più promettente e che conterranno, quindi, con maggiore probabilità buone soluzioni intere. Il compito di trovare rapidamente soluzioni ammissibili viene lasciato alle euristiche primali.

3.3.2 Strategie di branching

Trovare una strategia di branching efficace, che mantenga invariata la struttura del problema di *pricing*, è uno dei fattori cruciali per il funzionamento degli algoritmi *Branch & Price*, come mostrato in [1].

Poichè ad ogni soluzione ammissibile di MP ne corrisponde una nella formulazione naturale, possiamo pensare di fare *branching* sulle variabili di quest'ultima, pur lavorando con MP.

Vengono qui proposte due possibili strategie di *branching*.

Branching per interdizione

In ogni nodo vengono scelti due elementi $i, j \in \Omega$ (con $i \neq j$). Nel primo figlio si impone che i due elementi si trovino sullo stesso livello; nel secondo, invece, sono forzati a trovarsi su livelli differenti. Ovvero, se indichiamo con S_0 lo spazio delle soluzioni del nodo padre, le regioni ammissibili S_1 ed S_2 dei due figli saranno

$$S_1 = S_0 \cap \{\mathbf{x^k} | x_i^k = x_j^k\}$$

$$S_2 = S_0 \cap \{\mathbf{x^k} | x_i^k + x_j^k \leq 1\}$$

Per rendere efficace il *branching*, bisogna scegliere in modo opportuno la coppia di variabili (i, j). Si valuta per ogni (i, j) in quale misura le due variabili compaiono sul medesimo livello, nelle colonne che costituiscono la soluzione del RMP. Ovvero calcoliamo un coefficiente

$$p_{i,j} = \sum_{k \in K | x_i^k = 1 \wedge x_j^k = 1} z_k$$

che misura la frazione delle colonne scelte nella soluzione del R-LMP in cui i e j compaiono insieme. Viene, quindi, selezionata la coppia (i, j) per cui $p_{i,j}$ è più vicino ad $\frac{1}{2}$.

$$(i, j) = \arg\min \left\{ \left| p_{i,j} - \frac{1}{2} \right| \right\}$$

Per fare in modo che le colonne, rese inammissibili dai vincoli di *branching*, rimangano tali, è necessario introdurre dei vincoli aggiuntivi nel problema di *pricing*. Questi vincoli modificano la struttura di PP, che non può più essere risolto con l'algoritmo [4]. Nella nostra implementazione è stato utilizzato un MIP-Solver generico.

Branching a due fasi

Nella prima fase si determina, tramite *branching* binario, quali elementi debbano *inizializzare* un livello. Tali elementi, che indicheremo con il nome di *capofila*, hanno la caratteristica di essere i più alti dei rispettivi livelli.

Anche in questo caso è importante scegliere in modo accurato l'elemento su cui fare *branch*. La strategia adottata è quella di scegliere l'elemento che, nelle colonne che costituiscono la soluzione del R-LMP, compare il più possibile nella stessa misura come capofila e come non capofila.

Indichiamo con $\Omega = \{1, \ldots, N\}$ l'insieme degli elementi e chiamiamo C_j l'insieme delle colonne inizializzate dall'elemento j.

$$C_j = \left\{ \mathbf{x^k} \mid \sum_{i \in \Omega | i<j} x_i^k = 0 \ \wedge \ x_j^k = 1 \right\}$$

Calcoliamo in che misura l'elemento j è stato scelto come capofila nella soluzione del R-LMP.

$$F_j = \sum_{k \in C_j} z_k$$

L'elemento j per cui

$$j = \arg\min \left| F_j - \frac{1}{2} \right|$$

viene scelto come variabile di *branching*. Nel primo figlio viene imposto che j sia un capofila, nel secondo che non lo sia. Le regioni ammissibili S_1 ed S_2 dei due figli saranno

$$S_1 = S_0 \cap \left\{ \mathbf{x^k} \mid x_i^k + x_j^k \leq 1 \ \ \forall i < j \right\}$$

$$S_2 = S_0 \cap \left\{ \mathbf{x^k} \mid \sum_{i=j+1}^{N} x_i^k - x_j^k \geq 0 \right\}$$

Durante l'esplorazione dell'albero, viene effettuato un test di riduzione, prima della valutazione di ogni nodo. Indichiamo con $V_n \subset \Omega$ l'insieme dei capofila fissati nel nodo n e con BP il miglior bound primale di cui siamo a conoscenza. Sappiamo che la migliore soluzione ammissibile, che possiamo trovare in n, non può essere inferiore alla somma delle altezze dei capofila fissati nel nodo stesso. Di conseguenza, se

$$h_i \geq BP - \sum_{j \in V_n} h_j$$

per qualche $i \in \Omega$, possiamo affermare con sicurezza cha tale elemento non potrà essere scelto come capofila in n. In caso contrario causerebbe la chiusura del nodo, perchè il bound duale diverrebbe maggiore del bound primale. Di conseguenza, ad i può essere immediatamente inibita la possibilità di diventare capofila, risparmiando l'esplorazione del sottoalbero corrispondente.

Terminata questa prima fase, gli elementi di Ω sono stati partizionati in due sottoinsiemi. Il primo, V, contiene gli elementi che inizializzano un livello; il secondo, U, tutti gli altri ($U = \Omega \setminus V$). Il 2LSP è, dunque, ridotto ad un problema di assegnamento generalizzato (GAP). Ciò che rimane da fare è inserire gli elementi rimasti liberi nei livelli individuati.

In modo simile a quanto avviene con il *branching* per interdizione, si calcola per ogni $i \in V$, $j \in U$

$$p_{i,j} = \sum_{k \in K | x_i^k = 1 \wedge x_j^k = 1} z_k$$

Si sceglie la coppia (i, j) per cui $p_{i,j}$ è più vicino ad $\frac{1}{2}$ e si impone, in un figlio, che l'elemento i si trovi sul livello inizializzato da j, nell'altro, che sia collocato su un livello differente.

Nelle prove sperimentali si è visto come, in molti casi, non sia necessario ricorrere alla seconda fase di branching, in quanto i nodi vengono chiusi durante l'esplorazione del primo albero.

Per ridurre il numero di iterazioni di *column generation* necessarie ad ottimizzare il *Master problem*, il codice utilizzato, nell'implementazione dell'algoritmo, non individua una sola colonna, ma una per ogni capofila fissato e una colonna aggiuntiva, avente come capofila un elemento non ancora fissato.

3.4 Cancellazione di colonne

Ad ogni iterazione di *column generation*, nuove colonne vengono inserite nel master problem. Poichè esso viene risolto utilizzando il simplesso primale, una eccessiva crescita del numero di variabili ne rende la soluzione computazionalmente assai

onerosa. Si è pensato, quindi, di cercare di limitare il numero di colonne presenti nel *master problem*.

Una scelta possibile è quella di imporre un limite massimo. Quando tale limite viene superato, le colonne con costo ridotto meno vantaggioso vengono eliminate.

In alternativa, è possibile fissare una soglia ed eliminare le colonne il cui costo ridotto supera tale soglia. Infatti, indipendentemente dalla dimensione del RMP, tali colonne difficilmente entreranno in base. Risulta assai difficile, tuttavia, individuare un unico valore, che funzioni in modo soddisfacente indipendentemente dalla particolare istanza del problema.

Un miglioramento di quest'ultima tecnica consiste nell'imporre, anzichè un limite fisso, una soglia variabile r. Tale valore viene determinato in funzione della differenza tra bound primale e bound duale. Infatti, se il costo ridotto di una colonna fosse superiore a tale differenza, entrando in base causerebbe la chiusura del nodo perchè il bound duale diverrebbe maggiore del bound primale. Questo limite può essere reso più o meno restrittivo dividendo per un fattore p.

$$r = \frac{BP - BD}{p}$$

Ad esempio, nel nostro caso si cerca di stimare il numero di livelli su cui sarà necessario disporre gli elementi, dividendo somma delle larghezze degli elementi per la larghezza della striscia.

$$p = \frac{\sum_{i \in \Omega} w_i}{W}$$

Bisogna fare attenzione a non rendere la soglia troppo restrittiva, perchè i costi ridotti sono in funzione dei singoli nodi, ma viene tenuto un singolo RMP per tutto il problema.

Capitolo 4

Risultati sperimentali

4.1 Strumenti e dati utilizzati

L'algoritmo Branch & Price è stato implementato in C++, sfruttando le caratteristiche della programmazione ad oggetti. Per la soluzione del RMP con simplesso primale è stato utilizzato ILOG CPLEX 8.1.

Per verificare le prestazioni dell'algoritmo sono state utilizzate numerose istanze proposte in letteratura.

Il primo *dataset* è costituito da 500 istanze per il 2LBP, utilizzate in [23]. Queste istanze hanno dimensioni che vanno dai 20 ai 100 elementi e sono suddivise in 10 classi. Le prime 4 sono state proposte da Martello e Vigo [26] e sono basate sulla generazione di elementi di 4 tipi:

- *Tipo 1*: w_j estratto in modo casuale da una distribuzione uniforme in $\left[\frac{2}{3}W, W\right]$, h_j estratto in modo casuale da una distribuzione uniforme in $\left[1, \frac{1}{2}H\right]$;
- *Tipo 2*: w_j estratto in modo casuale da una distribuzione uniforme in $\left[1, \frac{1}{2}W\right]$, h_j estratto in modo casuale da una distribuzione uniforme in $\left[\frac{2}{3}H, H\right]$;
- *Tipo 3*: w_j estratto in modo casuale da una distribuzione uniforme in $\left[\frac{1}{2}W, W\right]$, h_j estratto in modo casuale da una distribuzione uniforme in $\left[\frac{1}{2}H, H\right]$;

- *Tipo 4*: w_j estratto in modo casuale da una distribuzione uniforme in $\left[1, \frac{1}{2}W\right]$, h_j estratto in modo casuale da una distribuzione uniforme in $\left[1, \frac{1}{2}H\right]$.

Ogni elemento della *Classe k* appartiene con il 70% di probabilità al Tipo k, con il 10% ciascuno agli altri 3 tipi. H e W valgono sempre 100. Le altre 6 classi sono, invece, state proposte da Berkey e Wang [2]:

- Classe 5: $W = 10$, w_j e h_j estratti con distribuzione uniforme in [1, 10];
- Classe 6: $W = 30$, w_j e h_j estratti con distribuzione uniforme in [1, 10];
- Classe 7: $W = 40$, w_j e h_j estratti con distribuzione uniforme in [1, 35];
- Classe 8: $W = 100$, w_j e h_j estratti con distribuzione uniforme in [1, 35];
- Classe 9: $W = 100$, w_j e h_j estratti con distribuzione uniforme in [1, 100];
- Classe 10: $W = 300$, w_j e h_j estratti con distribuzione uniforme in [1, 100].

Altre 38 istanze, per il 2SP e altri problemi di *cutting* in due dimensioni, tratte da [25]. Sono divise in 5 classi *beng, cgcut, gcut, ht, ngcut.* Anche queste istanze hanno caratteristiche diverse nel rapporto tra altezza e larghezza degli oggetti; in particolare quelle indicate come *ht* hanno la caratteristica di essere dei *packing perfetti.* Ovvero sono state ricavate tagliando un rettangolo.

I risultati ottenuti dall'algoritmo B&P sono stati confrontati con quelli di un MIP-Solver (ILOG CPLEX 8.1), quando il problema è formulato utilizzando il modello compatto proposto da Lodi e altri, descritto nella sezione 1.2.2.

I test sono stati eseguiti su un Pentium 4 a 1.6GHz.

4.2 Risultati

4.2.1 Esperimento 1: bounds

La prima analisi che è stata fatta riguarda la qualità del bound duale, utilizzato nell'algoritmo *Branch & Price*, (L_{CG}). Questo bound è stato confrontato con il

rilassamento continuo della formulazione compatta (L_{LP}) e con il bound ottenuto mediante l'algoritmo S-CUT (L_{cut}), entrambi proposti in [23].

L'algoritmo S-CUT si basa sull'idea di dividere verticalmente ogni elemento in strisce di larghezza unitaria, ordinarle per altezze non crescenti e riempire con esse i livelli seguendo questo ordine.

Le tabelle 4.1 e 4.2 mostrano i risultati ottenuti sulle 500 istanze del primo dataset.

La prima colonna della tabella 4.1 contiene un numero, che individua la classe di istanze. Nelle altre tre colonne viene riportato il valore, in percentuale, del rapporto tra il bound duale prodotto da L_{LP}, L_{cut} e L_{CG}, rispettivamente, e la soluzione ottenuta mediante l'euristica BFDH. I valori sono medi per ogni classe di istanze.

La tabella 4.2 permette di confrontare le prestazioni dei 3 bound duali, calcolate nel modo descritto sopra, rispetto alla dimensione delle istanze, indicata nella prima colonna.

Nella tabella 4.3 si può osservare il comportamento di L_{LP}, L_{cut} e L_{CG} sul secondo dataset. Nella prima colonna è indicato il nome della classe e, tra parentesi, il numero di istanze del problema che la compongono. In questo caso, i valori delle colonne 2, 3 e 4 sono calcolati in rapporto alle soluzioni ottenute risolvendo all'ottimo le istanze.

In media, L_{CG} risulta essere il bound più stretto dei tre. Una caratteristica, che è interessante rilevare, è che L_{CG} e L_{cut} ottengono risultati buoni su classi di istanze differenti. Ad esempio, sulle istanze della classe 6 L_{cut} trova un bound più stretto rispetto a quello individuato L_{CG}. Al contrario, nella classe 5, il gap di L_{CG} è del 99,96%, mentre per L_{cut} è 82,23%. Il rilassamento continuo della formulazione compatta fornisce sempre un bound meno stretto rispetto a L_{CG}. Questo non ci stupisce in quanto L_{LP} è dominato da L_{CG}.

Rispetto alla dimensione del problema, tutti e tre i bound mostrano risultati abbastanza costanti. In particolare, per L_{cut} e L_{CG} l'oscillazione dei valori è inferiore all'1%.

Durante la realizzazione degli esperimenti sono stati calcolati anche i valori ottenuti con il rilassamento continuo della formulazione naturale. Tale bound si è però dimostrato decisamente inferiore rispetto agli altri tre.

4.2.2 Esperimento 2: enumerazione

L'algoritmo è stato testato sulle istanze del primo dataset con un tempo limite di 100 secondi. I risultati sono riportati nella tabella 4.4. La prime due colonne indicano classe e dimensione delle istanze. Quindi, ogni riga della tabella si riferisce ai valori medi su 10 istanze. Le successive colonne contengono, rispettivamente, il numero di istanze risolte, il tempo impiegato, espresso in secondi, e il numero di nodi esplorati.

L'algoritmo descritto nel capitolo 3 è riuscito a risolvere, entro il tempo limite, 365 istanze su 500. Da un confronto con i risultati riportati in [23] si può dedurre che l'algoritmo *Beanch & Price* e CPLEX, con la formulazione compatta, hanno un comportamento abbastanza simile.

Per le istanze del secondo dataset è stato fissato un tempo limite di un'ora. La tabella 4.5 mostra i dati relativi all'analisi del nodo radice. Nei primi due campi sono riportati nome e numero di elementi di ogni istanza del dataset. Le colonne successive contengono: la soluzione trovata dall'euristica FFDH e bound primale e duale al termine dell'analisi del nodo radice. I valori del campo *%gap*, indicanti il divario tra bound primale e bound duale, sono calcolati nel modo seguente

$$gap = \frac{BP - BD}{BP} * 100$$

Nella settima colonna si trova il tempo di calcolo in secondi, mentre gli ultimi due campi indicano, rispettivamente, il numero di iterazioni di column generation effettuate e il numero di colonne generate.

Le tabelle 4.6 e 4.7 riportano i dati relativi alla soluzione delle istanze del secondo dataset utilizzando differenti politiche di esplorazione dell'albero di *branch*: *best first*, nella prima, *depth first*, nella seconda. Anche in questo caso i primi tre campi indicano il nome e la dimensione dell'istanza e la soluzione trovata dall'euristica FFDH. Nella quarta colonna si trova il valore della soluzione ottima, se l'istanza è stata risolta, bound primale e duale, altrimenti. Sono, inoltre, riportati il tempo di calcolo, espresso in secondi, lo stato di terminazione ed il numero di nodi esplorati, di iterazioni di *column generation* e di colonne generate.

Per poter fare un confronto, sulle istanze del secondo dataset sono state misurate anche le prestazioni di CPLEX con la formulazione compatta del 2LSP. I risultati

sono riportati nella tabella 4.8. I campi indicano: il nome e la dimensione della istanza, il valore della soluzione ottima trovata, il *gap* tra bound primale e duale, se il problema non è stato portato a termine, il tempo in secondi e lo stato di terminazione.

Sia CPLEX che l'algoritmo *Branch & Price* non sono riusciti a risolvere solo 7 istanze su 38, tutte appartenenti al gruppo *beng*. Tali istanze hanno la caratteristica di avere le altezze degli elementi che variano in un intervallo molto ristretto (valori interi tra 1 e 12). Di conseguenza, al crescere della dimensione dell'istanza, ci saranno, inevitabilmente, molti elementi con la medesima altezza. Questo crea problemi in quanto, dopo aver fissato una variabile con un'operazione di *branch*, con alta probabilità esisterà ancora una soluzione per R-LMP con lo stesso valore. Diventa, quindi, necessario esplorare molti nodi, contenenti soluzioni equivalenti, per stringere il bound duale. Solo sulle istanze *cgcut,* l'algoritmo *Branch & Price* mostra una netta superiorità.

Come è stato detto nel capitolo precedente, l'euristica FFDH, si è dimostrata molto efficace per il 2LSP. Sulle 38 istanze del secondo gruppo, ad esempio, per 29 volte la soluzione individuata da FFDH coincide con il miglior bound primale trovato dall'algoritmo, entro il tempo limite. Di queste, almeno 23 costituiscono la soluzione ottima della rispettiva istanza.

Riguardo alle diverse politiche di esplorazione dell'albero di *branching*, dalla tabella 4.9 si nota che, mediamente, la strategia *best first* consente di esplorare un numero minore di nodi e di restringere l'enumerazione ad una profondità inferiore dell'albero rispetto alla ricerca *depth first*, fornendo migliori gap nelle istanze non completate. D'altra parte, ha anche evidenziato dei problemi di consumo della memoria, nelle istanze più complesse, a causa del numero elevato di nodi aperti.

4.3 Conclusioni

In questa tesi, sono state presentate alcune possibili formulazioni per il *bidimensional level strip packing problem* ed è stato proposto un nuovo algoritmo per la soluzione all'ottimo del 2LSP, basato su tecniche *Branch & Price*. Tale algoritmo risolve, mediante *column generation*, il rilassamento della formulazione del problema come *set partitioning* per calcolare un *bound duale*. Quindi, applicando regole

di *branching*, risolve il problema effettuando una enumerazione implicita delle sue soluzioni. Nel testo vengono descritte due possibili strategie per realizzarlo: un branching per interdizione ed un branching in due fasi.

È possibile apportare miglioramenti all'algoritmo. Da un punto di vista implementativo, si può pensare ad una migliore e più efficiente gestione del *pool* di colonne, ad euristiche primali più vantaggiose o all'utilizzo di strategie di branching alternative. Un miglioramento delle prestazioni si potrebbe ottenere anche con l'applicazione di algoritmi euristici veloci per il problema di *pricing*, anzichè calcolarne, ad ogni iterazione di column generation, la soluzione ottima. Interessante sarebbe anche modificare l'algoritmo *Branch & Price* per supportare la rotazione degli oggetti.

Classe	L_{LP}	L_{cut}	L_{CG}
1	85,43%	87,83%	99,31%
2	91,39%	94,58%	95,47%
3	80,82%	82,23%	99,96%
4	90,76%	95,20%	97,44%
5	91,27%	93,63%	97,71%
6	92,20%	99,00%	94,54%
7	88,05%	91,03%	97,04%
8	92,01%	98,45%	96,20%
9	88,10%	90,60%	97,81%
10	91,32%	98,21%	96,22%
Avg.	89,13%	93,08%	97,17%

Tabella 4.1: Confronto delle prestazioni dei bound duali calcolati con rilassamento continuo della formulazione compatta (L_{LP}), con algoritmo CUT-S (L_{cut}) e con rilassamento della formulazione come set covering (L_{CG}). I valori sono in rapporto alla soluzione ottenuta con l'euristica BFDH

N	L_{LP}	L_{cut}	L_{CG}
20	86,67%	93,69%	96,24%
40	88,19%	92,78%	97,21%
60	89,55%	92,90%	97,38%
80	89,75%	92,30%	97,34%
100	91,51%	93,71%	97,68%
Avg.	89,13%	93,08%	97,17%

Tabella 4.2: Confronto delle prestazioni dei bound duali rispetto alla dimensione del problema

Classe	L_{LP}	L_{cut}	L_{CG}
BENG (10)	93,25%	99,53%	95,31%
GCUT (4)	85,09%	89,43%	99,86%
NGCUT (12)	87,43%	95,79%	94,90%
CGCUT (3)	95,34%	95,34%	93,05%
HT (9)	92,20%	99,60%	95,91%
Avg.	90,66%	95,94%	95,81%

Tabella 4.3: Confronto delle prestazioni dei bound duali calcolati con rilassamento continuo della formulazione compatta (L_{LP}), con algoritmo CUT-S (L_{cut}) e con rilassamento della formulazione come set covering (L_{CG})

classe	**N**	**risolti**	**tempo**	**nodi**	**classe**	**N**	**risolti**	**tempo**	**nodi**
1	20	10	0,02	3,2	6	20	10	0,09	17,8
	40	10	0,26	43,0		40	10	2,88	221,2
	60	10	0,97	85,2		60	8	38,20	1 546,3
	80	10	9,53	231,2		80	5	61,69	992,9
	100	10	36,21	587,7		100	2	91,54	1 014,0
2	20	10	0,10	10,2	7	20	10	0,02	3,0
	40	5	55,12	3 128,2		40	10	0,57	101,8
	60	2	87,59	1 167,1		60	10	0,91	102,2
	80	0	100,16	662,2		80	10	9,13	403,4
	100	0	100,10	430,3		100	5	63,63	1 999,0
3	20	10	0,03	7,8	8	20	10	0,07	6,6
	40	10	0,53	72,8		40	4	63,66	4 451,6
	60	10	20,87	2 472,9		60	2	92,54	1 311,1
	80	8	47,19	968,8		80	0	100,28	1 033,0
	100	9	40,38	644,9		100	0	100,36	726,0
4	20	10	0,03	6,0	9	20	10	0,02	4,2
	40	10	0,37	67,6		40	10	0,49	83,4
	60	9	21,35	795,4		60	10	1,44	121,2
	80	8	36,82	924,3		80	10	7,50	236,8
	100	6	56,93	938,3		100	8	49,92	996,7
5	20	10	0,01	3,6	10	20	10	0,12	25,0
	40	10	0,05	18,2		40	3	73,04	4 379,7
	60	10	0,31	75,0		60	1	94,43	1 313,1
	80	10	10,37	391,6		80	0	101,09	749,8
	100	10	4,97	142,8		100	0	100,15	546,9

Tabella 4.4: Risultati ottenuti dall'algoritmo *Branch & Price* sulle istanze del primo dataset

istanza	N	FFDH	bp	bd	%gap	tempo (s)	iter. CG	colonne
beng01	20	36	36	33,46	7,06	0,05	54	100
beng02	40	64	64	60,31	5,77	0,23	134	244
beng03	60	92	92	85,74	6,80	0,65	251	439
beng04	80	113	113	108,53	3,95	1,34	330	593
beng05	100	140	140	133,96	4,31	2,62	424	746
beng06	40	42	42	38,79	7,64	0,46	185	361
beng07	80	73	73	68,98	5,51	3,72	512	913
beng08	120	108	108	101,64	5,89	16,86	890	1 483
beng09	160	132	132	125,57	4,87	53,69	1 160	2 020
beng10	200	162	162	155,48	4,02	85,61	1 271	2 303
gcut01	10	1 016	1 016	1 016,00	0,00	0,01	2	10
gcut02	20	1 349	1 349	1 261,75	6,47	0,02	23	39
gcut03	30	1 810	1 810	1 810,00	0,00	0,02	28	52
gcut04	50	3 216	3 216	3 108,00	3,36	0,13	65	94
ngcut01	10	25	25	25,00	0,00	0,02	8	17
ngcut02	17	33	33	31,00	6,06	0,02	19	42
ngcut03	21	34	34	30,87	9,22	0,04	41	80
ngcut04	7	23	23	21,00	8,70	0,01	10	19
ngcut05	14	46	46	36,67	20,29	0,02	25	50
ngcut06	15	38	38	35,33	7,02	0,02	11	27
ngcut07	8	21	21	21,00	0,00	0,02	10	16
ngcut08	13	38	38	36,25	4,61	0,02	27	51
ngcut09	18	65	65	60,64	6,70	0,03	24	48
ngcut10	13	85	85	63,00	25,88	0,03	13	30
ngcut11	15	69	69	65,20	5,51	0,02	19	41
ngcut12	22	104	104	99,50	4,33	0,03	11	34
cgcut01	16	28	28	22,77	18,67	0,03	18	36
cgcut02	23	83	83	75,44	9,11	0,08	58	115
cgcut03	62	744	744	708,00	4,84	0,29	96	174
ht1	16	25	25	23,57	5,71	0,02	35	68
ht2	17	29	29	27,87	3,88	0,03	25	53
ht3	16	28	28	27,00	3,57	0,03	24	43
ht4	25	16	16	15,85	0,94	0,02	57	113
ht5	25	19	19	17,45	8,16	0,08	61	121
ht6	25	16	16	15,91	0,57	0,04	81	138
ht7	28	39	39	33,65	13,71	0,12	92	157
ht8	29	36	36	33,67	6,48	0,14	100	197
ht9	28	39	39	33,92	13,03	0,12	99	176

Tabella 4.5: Risultati ottenuti dall'algoritmo *Branch & Price* nel nodo radice sulle istanze del secondo dataset

istanza	N	FFDH	soluzione	%gap	tempo	stato	nodi	CG iter	colonne
beng01	20	36	36	0,0	0,36		71	779	1766
beng02	40	64	62	0,0	15,76		1509	6204	12686
beng03	60	92	(92 -86,89)	5,55	1007,11	(a)	40935	62897	41827
beng04	80	113	(113 - 108,99)	3,55	596,11	(a)	23920	25591	2789
beng05	100	141	(140 - 134,36)	4,03	465,27	(a)	15604	17164	2660
beng06	40	42	42	0,0	81,39		2663	53493	151608
beng07	80	73	(73 - 70,45)	3,50	3600,00	(b)	21995	196835	581628
beng08	120	108	(108 - 102,30)	5,28	1633,53	(a)	10936	15204	7552
beng09	160	132	(132 - 125,73)	4,75	1374,07	(a)	6286	8748	4290
beng10	200	162	(162 - 155,48)	4,02	1164,59	(a)	4069	6509	4466
gcut01	10	1016	1016	0,0	0,01		1	2	10
gcut02	20	1349	1262	0,0	0,02		3	26	40
gcut03	30	1873	1810	0,0	0,02		1	28	52
gcut04	50	3216	3126	0,0	9,11		1933	4924	5762
ngcut01	10	25	25	0,0	0,01		1	8	17
ngcut02	17	33	33	0,0	0,03		13	75	147
ngcut03	21	34	32	0,0	0,09		45	157	211
ngcut04	7	23	23	0,0	0,01		3	23	36
ngcut05	14	46	37	0,0	0,02		5	39	72
ngcut06	15	38	38	0,0	0,05		27	128	216
ngcut07	8	21	21	0,0	0,02		1	10	16
ngcut08	13	38	38	0,0	0,02		7	88	154
ngcut09	18	65	63	0,0	0,10		53	220	356
ngcut10	13	85	85	0,0	0,10		47	340	460
ngcut11	15	69	69	0,0	0,16		47	469	795
ngcut12	22	104	102	0,0	0,02		9	61	124
cgcut01	16	28	28	0,0	1,53		901	4474	9521
cgcut02	23	83	78	0,0	5,05		507	7919	17633
cgcut03	62	744	711	0,0	0,97		121	299	323
ht1	16	25	25	0,0	0,02		3	51	91
ht2	17	29	28	0,0	0,02		7	60	110
ht3	16	28	28	0,0	0,03		3	51	91
ht4	25	16	16	0,0	0,02		1	57	113
ht5	25	19	19	0,0	1,71		157	1947	5041
ht6	25	16	16	0,0	0,04		1	81	138
ht7	28	39	39	0,0	31,80		2685	32120	99112
ht8	29	36	36	0,0	4,64		339	4767	13477
ht9	28	39	36	0,0	0,73		87	719	1471

Tabella 4.6: Risultati sulle istanze del secondo dataset, dell'algoritmo *Branch & Price* con esplorazione best first. Lo stato (a) indica l'esaurimento della memoria, (b) il raggiungimento del limite di tempo concesso alla simulazione.

istanza	N	FFDH	soluzione	%gap	tempo	stato	nodi	CG iter	colonne
beng01	20	36	36	0,00	0,37		79	777	1715
beng02	40	64	62	0,00	3,23		411	2048	3789
beng03	60	92	(92 -85,7408)	5,76	3600,04	(b)	202229	1242663	5518182
beng04	80	113	(113 - 108,534)	5,05	3600,11	(b)	134397	589110	1892935
beng05	100	141	(139 - 134,637)	0,00	3600,09	(b)	25332	149112	685996
beng06	40	42	42	0,00	55,37		2851	43571	101237
beng07	80	73	(73 - 69,1058)	2,84	3600,16	(b)	33791	323087	1239557
beng08	120	108	(108 - 101,639)	6,87	3600,03	(b)	12766	93140	572398
beng09	160	132	(132 - 125,607)	8,44	3600,12	(b)	7906	56589	376749
beng10	200	162	(162 - 155,482)	10,56	3600,16	(b)	8316	42581	304062
gcut01	10	1016	1016	0,00	0,01		1	2	10
gcut02	20	1349	1262	0,00	0,02		3	26	40
gcut03	30	1873	1810	0,00	0,02		1	28	52
gcut04	50	3216	3126	0,00	224,06		39515	213008	410158
ngcut01	10	25	25	0,00	0,02		1	8	17
ngcut02	17	33	33	0,00	0,02		13	78	142
ngcut03	21	34	32	0,00	0,52		185	1403	2588
ngcut04	7	23	23	0,00	0,01		3	23	36
ngcut05	14	46	37	0,00	0,02		5	39	72
ngcut06	15	38	38	0,00	0,05		29	130	239
ngcut07	8	21	21	0,00	0,02		1	10	16
ngcut08	13	38	38	0,00	0,02		7	88	154
ngcut09	18	65	63	0,00	0,13		55	330	576
ngcut10	13	85	85	0,00	0,07		45	279	400
ngcut11	15	69	69	0,00	0,16		53	512	857
ngcut12	22	104	102	0,00	0,05		39	140	196
cgcut01	16	28	28	0,00	1,47		923	4603	7969
cgcut02	23	83	78	0,00	6,91		783	12175	23402
cgcut03	62	744	711	0,00	10,01		1059	4433	17012
ht1	16	25	25	0,00	0,02		3	51	91
ht2	17	29	28	0,00	0,03		7	60	110
ht3	16	28	28	0,00	0,02		3	51	91
ht4	25	16	16	0,00	0,02		1	57	113
ht5	25	19	19	0,00	1,67		165	1837	4247
ht6	25	16	16	0,00	0,04		1	81	138
ht7	28	39	39	0,00	25,04		2907	27013	72075
ht8	29	36	36	0,00	3,27		331	3590	8686
ht9	28	39	36	0,00	2,28		195	2251	5358

Tabella 4.7: Risultati, sulle istanze del secondo dataset, dell'algoritmo *Branch & Price* con esplorazione depth first. Lo stato (a) indica l'esaurimento della memoria, (b) il raggiungimento del limite di tempo concesso alla simulazione.

istanza	N	soluzione	%gap	tempo	stato
beng01	20	36	0,00	0,74	
beng02	40	62	0,00	52,15	
beng03	60	(91 - 86238,00)	5,23	1247,41	(a)
beng04	80	(114 - 108,60)	4,74	1134,65	(a)
beng05	100	(144 - 134,15)	6,84	1283,57	(a)
beng06	40	42	0,00	20,53	
beng07	80	(73 - 70,31)	3,68	1599,74	(a)
beng08	120	(109 - 102,31)	6,13	1754,96	(a)
beng09	160	(134 - 125,98)	5,99	1181,17	(a)
beng10	200	(162 - 155,57)	3,97	1735,28	(a)
gcut01	10	1016	0,00	0,01	
gcut02	20	1262	0,00	0,03	
gcut03	30	1810	0,00	0,03	
gcut04	50	3126	0,00	4,16	
ngcut01	10	25	0,00	0,01	
ngcut02	17	33	0,00	0,06	
ngcut03	21	32	0,00	0,24	
ngcut04	7	23	0,00	0,00	
ngcut05	14	37	0,00	0,04	
ngcut06	15	38	0,00	0,05	
ngcut07	8	21	0,00	0,02	
ngcut08	13	38	0,00	0,09	
ngcut09	18	63	0,00	0,13	
ngcut10	13	85	0,00	0,26	
ngcut11	15	69	0,00	0,12	
ngcut12	22	102	0,00	0,02	
cgcut01	16	28	0,00	9,74	
cgcut02	23	78	0,00	2,65	
cgcut03	62	711	0,00	188,64	
ht1	16	25	0,00	0,06	
ht2	17	28	0,00	0,07	
ht3	16	28	0,00	0,02	
ht4	25	16	0,00	0,41	
ht5	25	19	0,00	0,84	
ht6	25	16	0,00	0,14	
ht7	28	39	0,00	87,44	
ht8	29	36	0,00	2,05	
ht9	28	36	0,00	0,23	

Tabella 4.8: Risultati, sulle istanze del secondo dataset, ottenuti da CPLEX con la formulazione compatta.

			depth first		
istanza	nodi	aperti	MPA	iter. CG	colonne
gcut	9880,00	4,25	10,00	53266,00	102565,00
ngcut	36,33	2,42	6,17	253,33	441,08
cgcut	921,67	9,33	19,67	7070,33	16127,67
ht	401,44	2,11	7,89	3887,89	10101,00
			best first		
istanza	nodi	aperti	MPA	iter. CG	colonne
gcut	484,50	208,50	6,00	1245,00	1466,00
ngcut	21,50	5,58	4,25	134,83	217,00
cgcut	509,67	64,00	14,00	4230,67	9159,00
ht	364,78	31,33	8,11	4428,11	13293,78

Tabella 4.9: Confronto tra politiche di eplorazione per l'abero di branching. I valori sono medi per ogni gruppo di istanze. I capmpi indicano: il numero di nodi esplorati (nodi), il numero massimo di nodi aperti (aperti), la massima profondità raggiunta nell'esplorazione dell'albero di branching (MPA), il numero di iterazioni di column generation (iter. CG) e il numero di colonne generate (colonne).

	Branch and Price		**CPLEX 8.1**	
Instance Class	**gap**	**time (s)**	**gap**	**time (s)**
BENG (10)	4,20% (7 inst.)	32,50 (3 inst.)	4,96% (7 inst.)	24,47 (3 inst.)
GCUT (4)	0,00%	2,29	0,00%	1,06
NGCUT (12)	0,00%	0,05	0,00%	0,09
CGCUT (3)	0,00%	2,52	0,00%	67,01
HT (9)	0,00%	4,34	0,00%	10,14

Tabella 4.10: Risultati ottenuti dall'algoritmo *Branch & Price* e da CPLEX sulle istanze del secondo dataset.

Bibliografia

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

[2] J.O. Berkey, P.Y. Wang, Two dimensional nite bin packing algorithms, Journal of the Operational Research Society 38 (1987) 423 429.

[3] A. Caprara, M. Monaci, On the two-dimensional Knapsack Problem, Operational Research Letters 32 (2004) 5 14.

[4] A. Ceselli, G. Righini, An optimizayion algorithm for the penalized knapsack problem, Note del polo 68 (2005).

[5] F.K.R. Chung, M.R. Garey, D.S. Johnson, On packing two-dimensional bins, SIAM Journal of Algebraic and Discrete Methods 3 (1982) 66 76.

[6] E.G. Colman Jr., M.R. Garey, D.S. Johnson, R.E. Tarjan, Performance bounds for level-oriented two-dimensional packing algorithms, SIAM J. Comput. 9 (1980) 801 826.

[7] K. Dowsland, Some experiments withsimulated annealing techniques for packing problems, European Journal of Operational Research 68 (1993) 389 399.

[8] O. Færø, D. Pisinger, M. Zachariasen, Guided local search for the three-dimensional bin packing problem, Technical paper, DIKU, University of Copenhagen, 1999.

[9] S.P. Fekete, J. Schepers, On more-dimensional packing III: Exact algorithms, Technical paper ZPR97-290, Mathematisches Institut, Universität zu Köln (1997).

[10] W. Fernandez de la Vega, V. Zissimopoulos, An approximation scheme for strip-packing of rectangles with bounded dimensions, Technical paper, LRI, Universit e e de Paris Sud, Orsay,1991.

[11] W. Fernandez de la Vega, G.S. Lueker, Bin packing can be solved within $1+\varepsilon$ in linear time, Combinatorica 1 (1981) 349 355.

[12] J.B. Frenk, G.G. Galambos, Hybrid next- t algorithm for the two-dimensional rectangle bin-packing problem, Computing 39 (1987) 201 217.

[13] Garey M.R., Johnson D.S., Computers and intractability - A guide to the theory of NP-Completeness, Freeman, New York (1979)

[14] P.C. Gilmore, R.E. Gomory, Multistage cutting problems of two and more dimensions, Operations Research 13 (1965) 94 119.

[15] P.C. Gilmore, R.E. Gomory, A linear programming approach to the cutting stock problem, Operations Research 9 (1961) 849 859.

[16] S. Jacobs, On genetic algorithms for the packing of polygons, European Journal of Operational Research88 (1996) 165 181.

[17] N. Karmarkar, R.M. Karp, An approximation scheme for the one-dimensional bin-packing problem, in: Proceedings of the 23rd Annual IEEE Symposium on Found. Comput. Sci., 1982, pp. 312 320.

[18] C. Kenyon, E. Rémila, A near-optimal solution to a twodimensional cutting stock problem, Mathematics of Operations Research25 (2000) 645 656.

[19] A. Lodi, S. Martello, M. Monaci, Two-dimensional packing problems: A survey, European Journal of Operational Research 141 (2002) 241 252.

[20] A. Lodi, S. Martello, D. Vigo, Neighborhood search algorithm for the guillotine non-oriented two-dimensional bin packing problem, in: S. Voss, S. Martello, I.H. Osman, C. Roucairol (Eds.), Meta-Heuristics: Advances

and Trends in Local SearchParadigms for Optimization, Kluwer Academic Publishers, Boston, 1998, pp. 125 139.

[21] A. Lodi, S. Martello, D. Vigo, Approximation algorithms for the oriented two-dimensional bin packing problem, European Journal of Operational Research112 (1999) 158 166.

[22] A. Lodi, S. Martello, D. Vigo, Heuristic and metaheuristic approaches for a class of two-dimensional bin packing problems, INFORMS Journal on Computing 11 (1999) 345 357.

[23] A. Lodi, S. Martello, D. Vigo, Models and Bounds for Two-Dimensional Level Packing Problems, Journal of Combinatorial Optimization 8 (2004) 363 379.

[24] M. Maffioli, F. Dell'Amico, S. Martello, Annotated bibliographies in combinatorial optimization (1997).

[25] S. Martello, M. Monaci, D. Vigo, An Exact Approach to the Strip-Packing Problem, INFORMS Journal on computing 15 (2003) 310 319.

[26] S. Martello, D. Vigo, Exact solution of the two-dimensional bin packing problem, Management Science 44 (1998) 388 399.

[27] L. A. Wolsey, Integer Programming (1998).