Model-driven Decision-making Methods 2019/2020 Report

2D Bin Packing problem

Daniele Atzeni, matricola: 568740

April 2020

Abstract

Report del progetto realizzato per il corso di Model-driven Decision-making Methods 2019/2020 presso Università di Pisa. Il progetto nasce con l'obiettivo di risolvere, mediante programmazione lineare intera, il problema del 2D Bin Packing. La prima sezione presenta la traccia proposta; la seconda descrive la modellazione matematica del problema; nella terza sezione è discussa la fase sperimentale e l'analisi dei risultati ottenuti.

1 Definizione del problema

A clothes firm in Prato has to organize the cutting of parts of clothes from a given roll of (costly) tissue, so that they can be later sewn up into complete clothes. The roll has given (fixed) height, and potentially unlimited length. Each part is represented by a convex 2D polygon with at most k edges (with k not too large, but not too small either), and it must be cut from the roll a given number of times. Because of the markings on the tissue, the orientation of the parts w.r.t. the roll is fixed, but each part can be placed at any point. The problem is to place all parts on the shortest possible stretch of the roll

2 Modellazione matematica

2.1 Dati del problema

1. N, numero di poligoni convessi;
2. H, altezza del roll;
3. K, numero massimo di lati dei poligoni;
4. per ogni $n \in \{1, ..., N\}$
   (a) K_n, numero di lati del poligono n;
   (b) $\forall v \in \{1, ..., K_n\}$ $(x_{n,v}, y_{n,v})$, vertici del poligono n ognuno adiacente al successivo;

Si può far coincidere l'origine del sistema di riferimento utilizzato per definire le coordinate dei vertici con i minimi rispettivamente delle ascisse e delle ordinate dei vertici senza perdere la generalità. Infatti, se così non fosse, è sufficiente traslare i vertici di un poligono di $-min_{n,v}(x_{n,v})$ e $-min_{n,v}(y_{n,v})$. Questa operazione renderà più leggibile la notazione utilizzata per i vincoli.

Utilizzando i vertici di ciascun poligono è possibile inoltre ricavare $\forall n \in \{1, ..., N\}$ e $\forall v \in \{1, ..., K_n\}$ l'espressione della retta su cui giace il lato che unisce i vertici v e $v + 1$[1] nella forma $l_{n,v}(x, y) = a_{n,v}x + b_{n,v}y + c_{n,v}$, in modo tale che il vettore $(a_{n,v}, b_{n,v})$, cioè il vettore normale della retta, punti verso l'interno del poligono. Così facendo i punti che fanno parte del poligono n sono tutti e soli quelli che soddisfano per ogni v il vincolo $l_{n,v}(x, y) \geq 0$.

2.2 Variabili decisionali

Prima di introdurre le variabili decisionali è necessario fare chiarezza sul modo in cui vengono modellati i vincoli di non sovrapposizione dei poligoni. Si utilizzerà la seguente proposizione

[1] nel caso in cui $v = K_n$ il vertice $v + 1$ sarebbe il vertice 1

> Siano P_1 e P_2 due poligoni convessi nel piano e siano L_1 ed L_2 gli insiemi delle rette su cui giacciono i lati di P_1 e P_2 rispettivamente. Allora condizione necessaria e sufficiente affinché $P_1 \cap P_2 = \emptyset$ è che esista $l \in L_1 \cup L_2$ tale che, detti l^+ ed l^- i semipiani generati da l, si abbia $P_1 \subset l^+$ e $P_2 \subset l^-$.

Bisogna inoltre osservare che per controllare che un poligono convesso giaccia su un semipiano è sufficiente controllare che i suoi vertici facciano parte del semipiano.
Il roll viene modellato come un piano cartesiano la cui origine è posta nel vertice in basso a sinistra del roll e con gli assi paralleli ai lati del roll.
Detto ciò, le variabili decisionali sono:

1. $\forall n \in \{1, ..., N\}$ Δx_n e Δy_n, rispettivamente traslazione orizzontale e verticale del poligono n rispetto all'origine degli assi;
2. $\forall n \in \{1, ..., N\}$, $\forall m \in \{n+1, ..., N\}$ se $n \neq N$,
   $\forall v \in \{1, ..., K_n, K_n + 1, ..., K_n + K_m\}$, $z_{n,m,v}$. Queste variabili sono binarie e valgono 0 se il lato passante per il vertice v del poligono n o il vertice $v - K_n$ del poligono m separa i poligoni n ed m, 1 altrimenti.
3. w, variabile ausiliaria per la funzione obiettivo.

2.3 Vincoli

Di seguito vengono descritti i vincoli, in cui si utilizza la notazione
$h_n = max_v(y_{n,v})$ e $w_n = max_v(x_{n,v})$ per rappresentare rispettivamente l'altezza e la larghezza del poligono n.

1. Vincoli di posizionamento all'interno del roll:
   - $\forall n \in \{1, ..., N\}$
     $\Delta x_n \geq 0$
     $0 \leq \Delta y_n \leq H - h_n$
2. Vincoli sulla variabile della funzione obiettivo:
   - $\forall n \in \{1, ..., N\}$
     $w \geq w_n + \Delta x_n$

   Questi vincoli fanno in modo che questa variabile w rappresenti il massimo delle ascisse dei punti dei poligoni. Lo scopo del risolutore sarà quindi trovare $\min(w)$.

3. Vincoli di non sovrapposizione:

   - $\forall n \in \{1, ..., N\}$, $\forall m \in \{n+1, ..., N\}$ se $n \neq N$, $\forall v \in \{1, ..., K_n\}$, $\forall v' \in \{1, ..., K_m\}$
     $l_{n,v}(x_{m,v'} + \Delta x_m - \Delta x_n,\, y_{m,v'} + \Delta y_m - \Delta y_n) - M z_{n,m,v} \leq 0$
     $l_{m,v'}(x_{n,v} + \Delta x_n - \Delta x_m,\, y_{n,v} + \Delta y_n - \Delta y_m) - M z_{n,m,K_n+v'} \leq 0$
   - $\forall n \in \{1, ..., N\}$, $\forall m \in \{n+1, ..., N\}$ se $n \neq N$
     $\sum_v z_{n,m,v} \leq K_n + K_m - 1$

   Come già detto precedentemente, per esprimere i vincoli di non sovrapposizione si utilizza il fatto che deve esistere un lato di uno dei due poligoni che li separa.
   Detto più formalmente, supponendo che i poligoni siano n ed m e che il lato separatore faccia parte del poligono n, si ha che deve esistere $v \in \{1, ..., K_n\}$ tale che il valore dell'espressione del lato traslato $l_{n,v}(x - \Delta x_n, y - \Delta y_n)$ calcolata in ogni vertice traslato di m, cioè in $(x_{m,v'} + \Delta x_m, y_{m,v'} + \Delta y_m)$ per ogni v', sia negativo.

   In questo modello l'esistenza di un vincolo viene modellata attraverso la tecnica del "big-M". Supponendo infatti che il valore M presente nei vincoli sia sufficientemente grande, si ha che le disequazioni sono banalmente verificate se $z_{n,v,m}$ vale 1, cioè nel caso in cui il lato del poligono n che unisce i vertici v e $v+1$ non separi il poligono n dal poligono m; nel caso in cui $z_{n,v,m} = 0$ si ottengono esattamente le disequazioni che esprimono la separazione dei poligoni appena discusse. L'ultimo vincolo infine impone che almeno una variabile z sia pari a 0, cioè impone effettivamente l'esistenza di un lato separatore.

   Per ottenere un valore di M sufficientemente grande si può osservare che per ogni vertice (x, y) si ha:

   (a) $0 \leq y \leq H$

   (b) $0 \leq x \leq \sum_n w_n$

   dove (a) e il limite inferiore di (b) valgono per i vincoli, mentre il limite suoeriore di (b) rappresenta il caso pessimo in cui tutti i poligoni sono allineati.
   Allora, per rendere le disequazioni superflue quando $z_{n,v,m} = 1$, basta imporre $M = max_{n,v}\{l_{n,v}(x, y) \mid 0 \leq y \leq H,\ 0 \leq x \leq \sum_n w_n\}$. Per rendere trattabile il calcolo di questo massimo è sufficiente notare che il massimo di un'espressione lineare in un rettangolo si ottiene se si calcola l'espressione in uno dei vertici, quindi per calcolare M basta calcolare il massimo dei valori assunti dalle espressioni delle rette su cui giacciono i lati dei poligoni calcolati nei punti $(0, 0), (0, H), (\sum_n w_n, 0)$ e $(\sum_n w_n, H)$.

3 Fase sperimentale

3.1 Implementazione e creazione di istanze

Per l'implementazione del modello è stato utilizzato il linguaggio di programmazione Python. Le librerie utilizzate sono:

- PuLP per la creazione di modelli MILP e la loro risoluzione;
- NumPy per la manipolazione dei dati;
- matplotlib per la creazione di grafici;
- SciPy per la creazione dei dati del problema

Per la generazione dei poligoni è stato utilizzato l'algoritmo seguente, in cui $N, H, Kmax, xmax, ymax$ rappresentano rispettivamente il numero di poligoni, l'altezza del roll, il numero massimo di lati per ciascun poligono e il massimo valore delle ascisse e delle ordinate dei vertici di ciascun poligono.

```
def create_data(N, H, Kmax, xmax, ymax, seed):

    polygons = []
    # set the seed of pseudo-random number generator
    numpy.random.seed(seed)

    while len(polygons) < N:
        # generate Kmax 2d points in the given range
        xs = numpy.random.uniform(0, xmax, Kmax)
        ys = numpy.random.uniform(0, ymax, Kmax)
        points = numpy.stack((xs, ys), axis=1)

        # compute ys' maximum
        # it will be the height of the polygon
        height = numpy.max(points, axis=0)[1]

        # check if height is less than H, if yes:
        # compute convex hull and append the result
        if height <= H:
            pol = scipy.spatial.ConvexHull(points)
            polygons.append(pol.points[pol.vertices])

    return polygons
```

3.2 Analisi dei risultati

Per la risoluzione del problema sono stati utilizzati due solver, CPLEX e CBC. Tutti i valori riportati sono la media ottenuta su tre esperimenti, per mitigare la randomicità del processo di ricerca di una soluzione ottimale. Nonostante il tempo impiegato dipenda fortemente dall'istanza del problema, CPLEX si dimostra già negli esperimenti di prova molto più efficiente di CBC. Perciò si è optato per una grid search tra i parametri di CBC, con lo scopo di trovare la miglior configurazione possibile. I parametri presi in considerazione sono:

- CUTS $\in$ {'on', 'off', 'root', 'ifmove', 'forceOn'}
- PRESOLVE $\in$ {'on', 'off', 'more', 'file'}
- STRONG $\in$ {0, 5}
- HEUR $\in$ {'on', 'off'}

In tabella 1 sono riportate le 3 configurazioni migliori e le 3 peggiori ottenute in un'istanza con $N = 5$, $H = 12$, $Kmax = 6$, $xmax = 10$ e $ymax = 10$.

CUTS	PRESOLVE	STRONG	HEUR	Tempo (sec)
forceOn	on	5	off	0.6085
forceOn	file	5	off	0.6240
ifmove	file	5	on	0.6579
ifmove	more	0	off	112.7
ifmove	on	0	off	120.0
root	off	0	off	120.1

Table 1: Migliori e peggiori 3 combinazioni dei parametri di CBC.

Innanzitutto bisogna apprezzare l'importanza dei parametri algoritmici di CBC, grazie ai quali si può passare da 2 minuti, nel caso peggiore, a 0.6 secondi per ottenere una soluzione ottimale. Inoltre si può notare l'importanza del parametro STRONG, che vale 5 nelle prime 16 configurazioni, ordinate per velocità. Per questo motivo si è deciso di ripetere il procedimento altre due volte, su istanze diverse dalla precedente, facendo variare esclusivamente STRONG tra i valori 5, 10 e 20, e tenendo costante STRONG e faendo variare gli altri parametri. Da questi esperimenti non si ottengono differenze significative finché il parametro STRONG sia maggiore di 5 ed il parametro CUT valga 'ifmove' oppure 'forceOn'.

Una spiegazione per i risultati della grid search si può ottenere se si analizzare più dettagliatamente il log di output di CBC. Di seguito sono mostrati degli estratti del log, la versione completa si trova in Appendice.

```
...
Problem MODEL has 507 rows, 133 columns and 2538 elements
Coin0008I MODEL read with 0 errors
Continuous objective value is 9.70573 - 0.00 seconds
Cgl0004I processed model has 141 rows, 133 columns (120 integer
(120 of which binary)) and 732 elements
...
Cbc0012I Integer solution of 33.057801 found by feasibility
pump after 0 iterations and 0 nodes (0.16 seconds)
Cbc0038I Full problem 141 rows 133 columns, reduced to 126
rows 13 columns
Cbc0031I 73 added rows had average density of 17.506849
Cbc0013I At root node, 73 cuts changed objective from
9.7057281 to 14.953123 in 56 passes
...
Cbc0010I After 0 nodes, 1 on tree, 33.057801 best solution,
best possible 14.953123 (0.64 seconds)
...
Cbc0010I After 1000 nodes, 200 on tree, 24.196337 best
solution, best possible 14.953123 (10.38 seconds)
...
Cbc0010I After 26000 nodes, 55 on tree, 23.703044 best
solution, best possible 23.466354 (61.45 seconds)
Cbc0001I Search completed - best objective 23.7030436701
, took 1777130 iterations and 214469 nodes (61.92 seconds)
...
Cbc0032I Strong branching done 79142 times (633297
iterations), fathomed 774 nodes and fixed 3928 variables
Cbc0041I Maximum depth 50, 779 variables fixed on reduced
cost (complete fathoming 1491 times, 187904 nodes taking
858725 iterations)
...
Objective value:                23.70304367
Enumerated nodes:               214469
Total iterations:               1777130
Time (CPU seconds):             61.93
Time (Wallclock seconds):       61.93
```

Nel primo blocco si può osservare come il PREPROCESS sia in grado di ridurre notevolmente il numero di righe del problema, ma non riesca a ridurre il numero di colonne, quindi il numero di variabili.
Nel secondo blocco si può apprezzare l'importanza dei CUT, che, già alla radice, migliorano il lower bound di circa il 54%.
Nel terzo e quarto blocco si nota che il solver non sembra avere problemi a trovare un buon upper bound, già piuttosto vicino alla soluzione ottimale dopo appena 1000 nodi, al contrario del lower bound. Questo è un trend ricorrente

in tutti gli esperimenti effettuati, che giustifica l'importanza dei CUT e dello STRONG branching e la quasi ininfluenza delle euristiche.

Nonostante l'utilizzo della miglior configurazione di parametri possibili, CBC risulta essere ancora inferiore rispetto a CPLEX. Infatti, già sulle stesse piccole istanze su cui è stata effettuata la grid search, CPLEX ottiene una soluzione ottimale anche 10 volte più velocemente di CBC, come mostrato in tabella 2.

Istanza	CBC	CPLEX
1	1.019 sec	0.300 sec
2	3.024 sec	0.254 sec
3	1.111 sec	0.254 sec

Table 2: Confronto tra CBC e CPLEX.

Per poter effettuare un confronto più approfondito si può analizzare anche il log di CPLEX. Analogamente a quanto fatto precedentemente, di seguito ne è mostrato un estratto, la versione integrale si può trovare in Appendice.

```
...
Tried aggregator 1 time.
MIP Presolve eliminated 366 rows and 0 columns.
Reduced MIP has 141 rows, 133 columns, and 732 nonzeros.
Reduced MIP has 120 binaries, 0 generals, 0 SOSs, and 0 indicators.
Presolve time = 0.00 sec. (0.72 ticks)
Probing time = 0.00 sec. (0.11 ticks)
...
Root relaxation solution time = 0.02 sec. (0.86 ticks)

        Nodes                                         Cuts/
   Node  Left     Objective  IInf  Best Integer    Best Bound    ItCnt
...
      0     2        9.7057    29       26.3238        9.7057      229
...
*  1537+  584                           24.9911       13.0983

Performing restart 1

Repeating presolve.
Tried aggregator 1 time.
...
MIP Presolve modified 109 coefficients.
Reduced MIP has 130 rows, 122 columns, and 666 nonzeros.
Reduced MIP has 109 binaries, 0 generals, 0 SOSs, and 0 indicators.
...
   4607     0       14.5477    28       24.9911      Cuts: 48    17926
...
```

```
  11942  1298       23.4380     6       23.7030       20.0327    54778
...
MIP - Integer optimal solution:  Objective =  2.3703043670e+01
Solution time =    1.20 sec.  Iterations = 82513  Nodes = 17398
Deterministic time = 350.04 ticks  (290.97 ticks/sec)
```

Si nota che il primo PRESOLVE produce gli stessi risultati del PRESOLVE di CBC, mentre è migliore l'upper bound alla radice che passa da 33.06 a 26.32. Sembra essere abbastanza efficace il secondo PRESOLVE, grazie al quale si riescono ad eliminare altre 11 righe ed 11 colonne.
Le differenze più grandi tra i due solver sono il numero di iterazioni ed il numero di nodi visitati, rispettivamente più di 20 volte e più di 12 volte più grande in CBC rispetto a CPLEX.

Un ruolo importante nella ricerca di un buon lower bound è ricoperto dai CUT, perciò riscontrare differenze tra i solver potrebbe in parte giustificare la differenza di performance. In tabella 3 sono mostrati il numero CUTS effett da ciascun solver per ogni categoria.

CUTS	CBC		CPLEX
—	Generati	Attivi	—
Probing	370185	18821	-
Gomory	179186	6776	2
Knapsack	50	0	-
Clique	0	0	1
MixedIntegerRounding2	60875	291	22
FlowCover	0	0	9
TwoMirCuts	46285	0	-
ImplicationCuts	56	0	-
ImpliedBound	-	-	42

Table 3: Confronto tra i cut generati da CBC e CPLEX.

Si può osservare come CBC impieghi molti sforzi a generare un gran numero di CUT, sebbene una piccola percentuale risulti effettivamente attiva.
Inoltre CPLEX sembra utilizzare molto gli Implied Bound Cuts, tipologia di CUT non supportata da CBC.

3.3 Riformulazione del modello

Una possibile strada da percorrere per ottenere miglioramenti nella ricerca della soluzione è quella della riformulazione del problema. Nel modello in analisi i vincoli più difficili da risolvere sono quelli di non sovrapposizione, dal momento che, a differenza degli altri vincoli, coinvolgono le variabili binarie. Una formulazione alternativa dei vincoli di non sovrapposizione coinvolge la somma di Minkowski e il no-fit polygon.

Dati due insiemi di punti A e B, si definisce somma di Minkowski l'insieme $A \oplus B = \{\mathbf{a} + \mathbf{b} | a \in A.b \in B\}$. Se A e B sono due poligoni, si definisce no-fit polygon di A e B l'insieme $U_{A,B} = A \oplus -B$. Come mostrato da Z.Li in "Compaction Algorithms for NonConvex Polygons and Their Applications", si può dimostrare che:

> Dati due poligoni A e B e detti $\mathbf{p}$ e $\mathbf{q}$ due loro punti di riferimento, si ha che $A \cap B = \emptyset$ se e solo se $\mathbf{q} - \mathbf{p} \notin U_{A,B}$.

Utilizzando questo risultato e denotanto con $K_{n,m}$ il numero di lati del no-fit polygon dei poligoni n ed m e con $l_{n,m,v}$ il lato di $U_{n,m}$ che parte dal vertice v in modo tale che i punti interni del poligono appartengano al semipiano positivo, si possono riscrivere i vincoli di non sovrapposizione come:

- $\forall n \in \{1, ..., N\}$, $\forall m \in \{n+1, ..., N\}$ se $n \neq N$, $\forall v \in \{1, ..., K_{n,m}\}$, $l_{n,m,v}(\Delta x_m - \Delta x_n, \Delta y_m - \Delta y_n) - M(1 - z_{n,m,v}) \leq 0$

- $\forall n \in \{1, ..., N\}$, $\forall m \in \{n+1, ..., N\}$ se $n \neq N$ $\sum_v z_{n,m,v} = 1$

Dal momento che il no-fit polygon di due poligoni convessi è a sua volta un poligono convesso, per far sì che il vettore $(\Delta x_m - \Delta x_n, \Delta y_m - \Delta y_n)$ non sia all'interno di $U_{n,m}$, deve esistere un lato $l_{n,m,v}$ di $U_{n,m}$ per cui si abbia $l_{n,m,v}(\Delta x_m - \Delta x_n, \Delta y_m - \Delta y_n) \leq 0$; in questo caso si avrà $z_{n,m,v} = 1$.

Per ottenere un valore di M sufficientemente grande è necessario che $M \geq max_{n,v}(l_{n,m,v}(\Delta x_m - \Delta x_n, \Delta y_m - \Delta y_n))$ per ogni possibile valore delle variabili Δx e Δy. Analogamente a quanto fatto nella prima formulazione del problema, considerando come caso pessimo $W = \sum_n w_n$, si ha che $\forall n \in \{1, ..., N\}$, $0 \leq \Delta x_n \leq W - w_n$ e $0 \leq \Delta y_n \leq H - h_n$. Quindi $\forall n \in \{1, ..., N\}$, $\forall m \in \{n+1, ..., N\}$ se $n \neq N$ si ha che $-(W - w_n) \leq \Delta x_m - \Delta x_n \leq W - w_m$ e $-(H - h_n) \leq \Delta y_m - \Delta h_n \leq H - h_m$. Allora basta calcolare il valore di $l_{n,m,v}$ nei vertici di questo rettangolo, quindi definire M come il massimo valore ottenuto al variare di n ed m.

Si noti che attraverso questa nuova formulazione si ottiene lo stesso numero di variabili della formulazione precedente, dal momento che il numero di vertici del no-fit polygon tra due poligoni A e B è dello stesso ordine della somma del numero di vertici di A e B. Ciononostante non si può escludere l'eventualità che i nuovi vincoli siano più facili da risolvere per il solver.

In tabella 4 sono mostrati i risultati ottenuti da CPLEX su 3 istanze con $N = 6$, $H = 12$, $Kmax = 7$, $xmax = 10$ e $ymax = 10$ usando il modello appena descritto e quello descritto inizialmente.

Si può notare che questa nuova formulazione sembra avere all'incirca le stesse performanece della formulazione originale. Per questo motivo nelle fasi successive si è deciso di continuare ad utilizzare il modello originale.

Istanza	Primo Modello	Secondo Modello
1	4.326 sec	4.554 sec
2	13.63 sec	16.42 sec
3	10.21 sec	6.956 sec

Table 4: Confronto tra i due modelli analizzati.

3.4 Variazioni su "big-M"

Una delle problematiche inerenti i modelli che utilizzano la tecnica del "big-M" è proprio il calcolo del valore di M. Nel problema in analisi, durante la fase di soluzione del rilassamento del problema si potrebbero ottenere soluzioni in cui il valore delle variabili $z_{n,m,v}$ è molto basso, seppure il lato che diparte dal vertice v non separi i poligoni n ed m. Questo fenomeno è incrementato se il valore di M è molto grande, in quanto i vincoli di non sovrapposizione sono banalmente validi anche per valori piccoli delle variabili z.Per questo motivo può essere utile un approfondimento sul valore da assegnare ad M.

Nella presentazione iniziale del modello il valore di M è stato calcolato come $max_{n,v}\{l_{n,v}(x,y) \mid 0 \leq y \leq H,\ 0 \leq x \leq \sum_n w_n\}$, in cui il limite superiore delle x è stato ottenuto come valore della soluzione del problema nel caso in cui tutti i poligoni fossero affiancati. Questo upper bound può essere diminuito non appena si scopre una soluzione ammissibile del problema migliore. Quindi, nel caso in cui il massimo nella definizione di M sia ottenuto in uno dei vertici con ascissa non nulla, il valore di M può essere diminuito ed i vincoli di non sovrapposizione possono essere rafforzati.

Purtroppo PuLP non dispone di un metodo che consente all'utente di generare cuts personalizzati in fase di esecuzione dell'algoritmo. Per superare questo ostacolo è stato dapprima risolto il problema con un limite di tempo di 10 secondi, per poter ottenere una soluzione ammissibile del problema, e successivamente è stato calcolato il nuovo valore di M utilizzando come limite superiore delle ascisse il valore della funzione obiettivo ottenuto.

In tabella 5 sono riportati i risultati ottenuti da CPLEX su 5 istanze con 7 poligoni utilizzando il valore originale di M ed quello ottenuto con la procedura appena descritta.

M originale	M ricalcolato
91.772 sec	64.514 sec
26.606 sec	67.024 sec
179.66 sec	103.09 sec
5.2614 sec	11.244 sec
146.17 sec	118.98 sec

Table 5: Confronto tra soluzioni al variare di M.

Si vede che nelle istanze più complicate il metodo appea descritto sembra

migliorare la velocità dell'algoritmo. Questo trend è confermato anche da altri esperimenti effettuati, ad esempio su un'istanza con 8 poligoni si è passati da più di 900 sec a 360 sec. Bisogna inoltre considerare come i miglioramenti potrebbero essere ancora più evidenti se questo procedimento fosse ripetuto più volte durante l'algoritmo, dal momento che il valore di M può essere ridotto ogni volta che migliora l'upper bound.

4 Conclusioni

Il problema del bin packing bidimensionale si è dimostrato essere piuttosto complicato da affrontare utilizzando modelli MILP ed efficace esclusivamente con istanze contenenti non più di 8 poligoni. Anche la riformulazione del problema e la diminuzione del valore del big-M non sono riusciti ad apportare miglioramenti sostanziali.

Per quanto riguarda i solver, CPLEX ha dimostrato una netta superiorità rispetto a CBC, nonostante siano stati utilizzati solamente i parametri di deafult del primo, mentre è stata effettuata una grid search tra i parametri del secondo

Per concludere, in figura 1 è mostrata la soluzione ottenuta con un'istanza creata con $N = 8$, $H = 12$, $Kmax = 6$, $xmax = 10$ e $ymax = 10$, per la quale CPLEX 300 secondi.

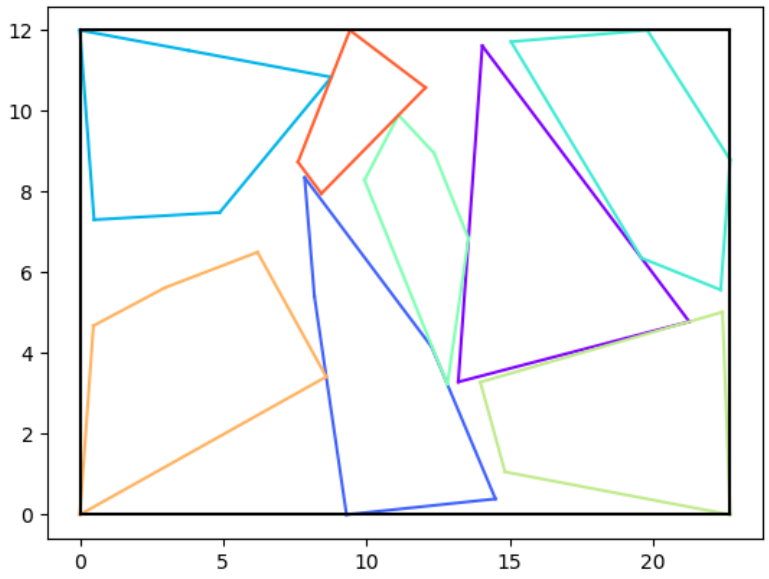

Figure 1: Soluzione di un'istanza.

5 Appendice

Esempio di log di output di CBC, ottenuto su un'istanza ottenuta con $N = 6$, $H = 12$, $Kmax = 6$, $xmax = 10$ e $ymax = 10$.

```
Welcome to the CBC MILP Solver
Version: 2.9.0
Build Date: Feb 12 2015

Problem MODEL has 507 rows, 133 columns and 2538 elements
Coin0008I MODEL read with 0 errors
Option for cutsOnOff changed from on to forceOn
Continuous objective value is 9.70573 - 0.00 seconds
Cgl0004I processed model has 141 rows, 133 columns (120 integer
(120 of which binary)) and 732 elements
Cbc0038I Initial state - 120 integers unsatisfied sum - 14.0704
Cbc0038I Pass   1: suminf.   14.07038 (114) obj. 15.7191
iterations 14
Cbc0038I Pass   2: suminf.   10.70459 (96) obj. 19.66
iterations 28
Cbc0038I Pass   3: suminf.    7.00541 (70) obj. 29.169
iterations 33
Cbc0038I Pass   4: suminf.    4.13475 (44) obj. 38.9705
iterations 36
Cbc0038I Pass   5: suminf.    0.44928 (11) obj. 48.0424
iterations 36
Cbc0038I Solution found of 48.0424
Cbc0038I Relaxing continuous gives 46.4579
Cbc0038I Before mini branch and bound, 0 integers at bound
fixed and 3 continuous
Cbc0038I Full problem 141 rows 133 columns, reduced to 140 rows
130 columns - 22 fixed gives 140, 108 - still too large
Cbc0038I Mini branch and bound did not improve solution (0.03
seconds)
Cbc0038I Freeing continuous variables gives a solution of
46.4579
Cbc0038I Round again with cutoff of 42.7827
Cbc0038I Pass   6: suminf.   14.07038 (114) obj. 15.7191
iterations 0
Cbc0038I Pass   7: suminf.   10.02015 (90) obj. 21.5629
iterations 31
Cbc0038I Pass   8: suminf.    4.04331 (44) obj. 38.737
iterations 79
Cbc0038I Pass   9: suminf.    0.58568 (18) obj. 42.4841
iterations 35
Cbc0038I Pass  10: suminf.    0.10627 (5) obj. 42.4406
```

```
iterations 24
Cbc0038I Pass  11: suminf.    0.06690 (5) obj. 42.3611
iterations 4
Cbc0038I Pass  12: suminf.    0.03528 (4) obj. 42.387
iterations 11
Cbc0038I Pass  13: suminf.    0.01888 (2) obj. 42.3925
iterations 5
Cbc0038I Pass  14: suminf.    0.00215 (1) obj. 42.4012
iterations 4
Cbc0038I Solution found of 42.4012
Cbc0038I Relaxing continuous gives 41.0178
Cbc0038I Before mini branch and bound, 0 integers at bound
fixed and 2 continuous
Cbc0038I Full problem 141 rows 133 columns, reduced to 140
rows 131 columns - 73 fixed gives 135, 57 - still too large
Cbc0038I Mini branch and bound did not improve solution
(0.04 seconds)
Cbc0038I Freeing continuous variables gives a solution of
41.0178
Cbc0038I Round again with cutoff of 34.7554
Cbc0038I Pass  15: suminf.   14.07038 (114) obj. 15.7191
iterations 0
Cbc0038I Pass  16: suminf.    6.90269 (69) obj. 25.97
iterations 76
Cbc0038I Pass  17: suminf.    2.15185 (35) obj. 34.7554
iterations 47
Cbc0038I Pass  18: suminf.    0.10357 (7) obj. 34.7554
iterations 33
Cbc0038I Pass  19: suminf.    0.15339 (4) obj. 34.7554
iterations 18
Cbc0038I Pass  20: suminf.    0.02640 (2) obj. 34.7554
iterations 3
Cbc0038I Pass  21: suminf.    0.04831 (1) obj. 34.7554
iterations 3
Cbc0038I Solution found of 34.7554
Cbc0038I Relaxing continuous gives 33.0578
Cbc0038I Before mini branch and bound, 0 integers at bound
fixed and 0 continuous
Cbc0038I Full problem 141 rows 133 columns, reduced to 141
rows 133 columns - 68 fixed gives 141, 65 - still too large
Cbc0038I Mini branch and bound did not improve solution
(0.06 seconds)
Cbc0038I Round again with cutoff of 26.0522
Cbc0038I Pass  22: suminf.   14.07038 (114) obj. 15.7191
iterations 0
Cbc0038I Pass  23: suminf.    9.85791 (90) obj. 19.5979
```

```
iterations 29
Cbc0038I Pass  24: suminf.    4.37851 (51) obj. 26.0522
iterations 52
Cbc0038I Pass  25: suminf.    1.36058 (27) obj. 26.0522
iterations 34
Cbc0038I Pass  26: suminf.    0.63099 (9) obj. 26.0522
iterations 29
Cbc0038I Pass  27: suminf.    0.58099 (14) obj. 26.0522
iterations 10
Cbc0038I Pass  28: suminf.    0.43616 (9) obj. 26.0522
iterations 25
Cbc0038I Pass  29: suminf.    0.33876 (9) obj. 26.0522
iterations 16
Cbc0038I Pass  30: suminf.    0.22667 (7) obj. 26.0522
iterations 19
Cbc0038I Pass  31: suminf.    0.22667 (7) obj. 26.0522
iterations 1
Cbc0038I Pass  32: suminf.    0.40380 (6) obj. 26.0522
iterations 14
Cbc0038I Pass  33: suminf.    0.21271 (6) obj. 26.0522
iterations 11
Cbc0038I Pass  34: suminf.    4.79643 (40) obj. 26.0522
iterations 55
Cbc0038I Pass  35: suminf.    1.37568 (16) obj. 26.0522
iterations 35
Cbc0038I Pass  36: suminf.    1.21399 (19) obj. 26.0522
iterations 6
Cbc0038I Pass  37: suminf.    0.46866 (8) obj. 26.0522
iterations 30
Cbc0038I Pass  38: suminf.    0.44763 (9) obj. 26.0522
iterations 4
Cbc0038I Pass  39: suminf.    0.22299 (6) obj. 26.0522
iterations 15
Cbc0038I Pass  40: suminf.    0.38869 (6) obj. 26.0522
iterations 9
Cbc0038I Pass  41: suminf.    0.41425 (8) obj. 26.0522
iterations 14
Cbc0038I Pass  42: suminf.    0.51810 (9) obj. 26.0522
iterations 19
Cbc0038I Pass  43: suminf.    0.40892 (6) obj. 26.0522
iterations 10
Cbc0038I Pass  44: suminf.    0.32558 (8) obj. 26.0522
iterations 12
Cbc0038I Pass  45: suminf.    2.96951 (23) obj. 19.66
iterations 54
Cbc0038I Pass  46: suminf.    2.84836 (26) obj. 19.66
```

```
iterations 3
Cbc0038I Pass  47: suminf.    1.59445 (12) obj. 26.0522
iterations 49
Cbc0038I Pass  48: suminf.    0.59694 (9) obj. 26.0522
iterations 6
Cbc0038I Pass  49: suminf.    0.31896 (6) obj. 26.0522
iterations 15
Cbc0038I Pass  50: suminf.    0.59793 (7) obj. 26.0522
iterations 14
Cbc0038I Pass  51: suminf.    2.82173 (23) obj. 26.0522
iterations 36
Cbc0038I No solution found this major pass
Cbc0038I Before mini branch and bound, 0 integers at bound
fixed and 0 continuous
Cbc0038I Full problem 141 rows 133 columns, reduced to 141
rows 133 columns - 68 fixed gives 141, 65 - still too large
Cbc0038I Mini branch and bound did not improve solution
(0.16 seconds)
Cbc0038I After 0.16 seconds - Feasibility pump exiting with
objective of 33.0578 - took 0.14 seconds
Cbc0012I Integer solution of 33.057801 found by feasibility
pump after 0 iterations and 0 nodes (0.16 seconds)
Cbc0038I Full problem 141 rows 133 columns, reduced to 126
rows 13 columns
Cbc0031I 73 added rows had average density of 17.506849
Cbc0013I At root node, 73 cuts changed objective from
9.7057281 to 14.953123 in 56 passes
Cbc0014I Cut generator 0 (Probing) - 5928 row cuts average
5.9 elements, 1 column cuts (1 active)  in 0.055 seconds -
new frequency is 1
Cbc0014I Cut generator 1 (Gomory) - 1164 row cuts average
24.2 elements, 0 column cuts (0 active)  in 0.022 seconds -
new frequency is 1
Cbc0014I Cut generator 2 (Knapsack) - 0 row cuts average
0.0 elements, 0 column cuts (0 active)  in 0.004 seconds -
new frequency is 1
Cbc0014I Cut generator 3 (Clique) - 0 row cuts average 0.0
elements, 0 column cuts (0 active)  in 0.002 seconds - new
frequency is 1
Cbc0014I Cut generator 4 (MixedIntegerRounding2) - 483 row
cuts average 3.2 elements, 0 column cuts (0 active)  in
0.020 seconds - new frequency is 1
Cbc0014I Cut generator 5 (FlowCover) - 0 row cuts average
0.0 elements, 0 column cuts (0 active)  in 0.017 seconds -
new frequency is 1
Cbc0014I Cut generator 6 (TwoMirCuts) - 784 row cuts average
```

```
13.2 elements, 0 column cuts (0 active)  in 0.023 seconds -
new frequency is 1
Cbc0010I After 0 nodes, 1 on tree, 33.057801 best solution,
best possible 14.953123 (0.64 seconds)
Cbc0012I Integer solution of 31.449894 found by DiveCoefficient
after 5443 iterations and 11 nodes (1.05 seconds)
Cbc0038I Full problem 141 rows 133 columns, reduced to 136
rows 44 columns - 14 fixed gives 131, 30 - ok now
Cbc0038I Full problem 141 rows 133 columns, reduced to 131
rows 30 columns
Cbc0012I Integer solution of 29.088472 found by RINS after
8331 iterations and 41 nodes (1.21 seconds)
Cbc0012I Integer solution of 26.572374 found by rounding
after 9392 iterations and 69 nodes (1.30 seconds)
Cbc0038I Full problem 141 rows 133 columns, reduced to 132
rows 28 columns
Cbc0038I Full problem 141 rows 133 columns, reduced to 138
rows 45 columns - 14 fixed gives 132, 25 - ok now
Cbc0038I Full problem 141 rows 133 columns, reduced to 137
rows 42 columns - 11 fixed gives 132, 27 - ok now
Cbc0038I Full problem 141 rows 133 columns, reduced to 132
rows 27 columns
Cbc0012I Integer solution of 26.066912 found by rounding
after 26449 iterations and 451 nodes (2.28 seconds)
Cbc0016I Integer solution of 25.286221 found by strong branching
after 28762 iterations and 500 nodes (2.44 seconds)
Cbc0016I Integer solution of 24.306239 found by strong branching
after 28926 iterations and 502 nodes (3.37 seconds)
Cbc0016I Integer solution of 24.196337 found by strong branching
after 30556 iterations and 521 nodes (4.27 seconds)
Cbc0038I Full problem 141 rows 133 columns, reduced to 132
rows 31 columns
Cbc0038I Full problem 141 rows 133 columns, reduced to 138
rows 53 columns - 13 fixed gives 136, 38 - still too large
Cbc0038I Full problem 141 rows 133 columns, reduced to 136
rows 38 columns - too large
Cbc0010I After 1000 nodes, 200 on tree, 24.196337 best
solution, best possible 14.953123 (10.38 seconds)
Cbc0038I Full problem 141 rows 133 columns, reduced to 132
rows 29 columns
Cbc0010I After 2000 nodes, 95 on tree, 24.196337 best
solution, best possible 14.953123 (13.29 seconds)
Cbc0010I After 3000 nodes, 71 on tree, 24.196337 best
solution, best possible 14.953123 (15.34 seconds)
Cbc0010I After 4000 nodes, 61 on tree, 24.196337 best
solution, best possible 14.953123 (17.02 seconds)
```

```
Cbc0010I After 5000 nodes, 57 on tree, 24.196337 best
solution, best possible 14.953123 (18.90 seconds)
Cbc0010I After 6000 nodes, 47 on tree, 24.196337 best
solution, best possible 14.953123 (20.38 seconds)
Cbc0010I After 7000 nodes, 29 on tree, 24.196337 best
solution, best possible 14.953123 (21.64 seconds)
Cbc0010I After 8000 nodes, 27 on tree, 24.196337 best
solution, best possible 14.953123 (23.32 seconds)
Cbc0038I Full problem 141 rows 133 columns, reduced to 134
rows 32 columns
Cbc0010I After 9000 nodes, 25 on tree, 24.196337 best
solution, best possible 14.953123 (24.86 seconds)
Cbc0010I After 10000 nodes, 30 on tree, 24.196337 best
solution, best possible 14.953123 (26.85 seconds)
Cbc0038I Full problem 141 rows 133 columns, reduced to 133
rows 32 columns
Cbc0010I After 11000 nodes, 16 on tree, 24.196337 best
solution, best possible 14.953123 (28.24 seconds)
Cbc0010I After 12000 nodes, 367 on tree, 24.196337 best
solution, best possible 17.609705 (33.39 seconds)
Cbc0010I After 13000 nodes, 634 on tree, 24.196337 best
solution, best possible 18.85473 (37.23 seconds)
Cbc0016I Integer solution of 24.004302 found by strong
branching after 457991 iterations and 13004 nodes
(37.31 seconds)
Cbc0010I After 14000 nodes, 666 on tree, 24.004302 best
solution, best possible 18.85473 (39.79 seconds)
Cbc0038I Full problem 141 rows 133 columns, reduced to 141
rows 63 columns - 17 fixed gives 138, 43 - still too large
Cbc0010I After 15000 nodes, 858 on tree, 24.004302 best
solution, best possible 20.477339 (43.54 seconds)
Cbc0010I After 16000 nodes, 985 on tree, 24.004302 best
solution, best possible 21.252626 (46.40 seconds)
Cbc0010I After 17000 nodes, 1099 on tree, 24.004302 best
solution, best possible 21.660655 (49.02 seconds)
Cbc0010I After 18000 nodes, 919 on tree, 24.004302 best
solution, best possible 21.660655 (50.67 seconds)
Cbc0010I After 19000 nodes, 956 on tree, 24.004302 best
solution, best possible 22.047162 (52.76 seconds)
Cbc0010I After 20000 nodes, 959 on tree, 24.004302 best
solution, best possible 22.41342 (54.61 seconds)
Cbc0010I After 21000 nodes, 886 on tree, 24.004302 best
solution, best possible 22.73564 (56.11 seconds)
Cbc0012I Integer solution of 23.703044 found by rounding
after 815940 iterations and 21015 nodes (56.18 seconds)
Cbc0010I After 22000 nodes, 513 on tree, 23.703044 best
```

```
solution, best possible 22.735782 (57.03 seconds)
Cbc0010I After 23000 nodes, 428 on tree, 23.703044 best
solution, best possible 23.039698 (58.54 seconds)
Cbc0010I After 24000 nodes, 328 on tree, 23.703044 best
solution, best possible 23.319244 (59.71 seconds)
Cbc0010I After 25000 nodes, 224 on tree, 23.703044 best
solution, best possible 23.456464 (60.59 seconds)
Cbc0010I After 26000 nodes, 55 on tree, 23.703044 best
solution, best possible 23.466354 (61.45 seconds)
Cbc0001I Search completed - best objective 23.7030436701
, took 1777130 iterations and 214469 nodes (61.92 seconds)
Cbc0032I Strong branching done 79142 times (633297
iterations), fathomed 774 nodes and fixed 3928 variables
Cbc0041I Maximum depth 50, 779 variables fixed on reduced
cost (complete fathoming 1491 times, 187904 nodes taking
858725 iterations)
Cuts at root node changed objective from 9.70573 to 14.9531
Probing was tried 29559 times and created 370185 cuts of
which 18821 were active after adding rounds of cuts
(6.618 seconds)
Gomory was tried 23295 times and created 179186 cuts of
which 6776 were active after adding rounds of cuts (2.063
seconds)
Knapsack was tried 23295 times and created 50 cuts of which
0 were active after adding rounds of cuts (1.292 seconds)
Clique was tried 23295 times and created 0 cuts of which 0
were active after adding rounds of cuts (0.209 seconds)
MixedIntegerRounding2 was tried 23295 times and created
60875 cuts of which 291 were active after adding rounds of
cuts (2.740 seconds)
FlowCover was tried 23295 times and created 0 cuts of which
0 were active after adding rounds of cuts (4.464 seconds)
TwoMirCuts was tried 23295 times and created 46285 cuts of
which 0 were active after adding rounds of cuts (1.867 seconds)
ImplicationCuts was tried 788 times and created 56 cuts of
which 0 were active after adding rounds of cuts (0.015 seconds)

Result - Optimal solution found

Objective value:                23.70304367
Enumerated nodes:               214469
Total iterations:               1777130
Time (CPU seconds):             61.93
Time (Wallclock seconds):       61.93

Option for printingOptions changed from normal to all
```

```
Total time (CPU seconds):       61.96
(Wallclock seconds):       61.96

time = 62.021741500000005
status = Optimal
objective = 23.703044
```

Esempio di log di output di CPLEX, ottenuto sulla stessa istanza.

```
Welcome to IBM(R) ILOG(R) CPLEX(R) Interactive Optimizer 12.10.0.0
  with Simplex, Mixed Integer & Barrier Optimizers
5725-A06 5725-A29 5724-Y48 5724-Y49 5724-Y54 5724-Y55 5655-Y21
Copyright IBM Corp. 1988, 2019.  All Rights Reserved.

Type 'help' for a list of available commands.
Type 'help' followed by a command name for more
information on commands.

Read time = 0.00 sec. (0.05 ticks)
CPLEX> Version identifier: 12.10.0.0 | 2019-11-26 | 843d4de2ae
Tried aggregator 1 time.
MIP Presolve eliminated 366 rows and 0 columns.
Reduced MIP has 141 rows, 133 columns, and 732 nonzeros.
Reduced MIP has 120 binaries, 0 generals, 0 SOSs, and 0 indicators.
Presolve time = 0.00 sec. (0.72 ticks)
Probing time = 0.00 sec. (0.11 ticks)
Tried aggregator 1 time.
Detecting symmetries...
Reduced MIP has 141 rows, 133 columns, and 732 nonzeros.
Reduced MIP has 120 binaries, 0 generals, 0 SOSs, and 0 indicators.
Presolve time = 0.00 sec. (0.45 ticks)
Probing time = 0.00 sec. (0.11 ticks)
MIP emphasis: balance optimality and feasibility.
MIP search method: dynamic search.
Parallel mode: deterministic, using up to 8 threads.
Root relaxation solution time = 0.02 sec. (0.86 ticks)

        Nodes                                         Cuts/
   Node  Left     Objective  IInf  Best Integer    Best Bound    ItCnt     Gap

      0     0        9.7057    23                    9.7057      118
      0     0        9.7057    23                  Cuts: 37      162
      0     0        9.7057    23                  Cuts: 43      196
      0     0        9.7057    23                  Cuts: 26      229
*     0+    0                          35.6968        9.7057            72.81%
*     0+    0                          26.6251        9.7057            63.55%
```

```
*     0+    0                          26.3238        9.7057           63.13%
      0     0  -1.00000e+75     0       26.3238        9.7057      229  63.13%
Detecting symmetries...
      0     2        9.7057    29       26.3238        9.7057      229  63.13%
Elapsed time = 0.19 sec. (21.13 ticks, tree = 0.02 MB, solutions = 3)
*     3+    2                          26.3238        9.7057           63.13%
*    34+   20                          26.1930        9.7057           62.95%
*    39+    3                          26.1478        9.7057           62.88%
*   349+  130                          25.2707       12.5531           50.33%
*   719+  344                          25.0075       13.0983           47.62%
*  1537+  584                          24.9911       13.0983           47.59%

Performing restart 1

Repeating presolve.
Tried aggregator 1 time.
MIP Presolve eliminated 10 rows and 10 columns.
MIP Presolve modified 110 coefficients.
Reduced MIP has 131 rows, 123 columns, and 672 nonzeros.
Reduced MIP has 110 binaries, 0 generals, 0 SOSs, and 0 indicators.
Presolve time = 0.00 sec. (0.38 ticks)
Tried aggregator 1 time.
MIP Presolve eliminated 1 rows and 1 columns.
MIP Presolve modified 109 coefficients.
Reduced MIP has 130 rows, 122 columns, and 666 nonzeros.
Reduced MIP has 109 binaries, 0 generals, 0 SOSs, and 0 indicators.
Presolve time = 0.00 sec. (0.48 ticks)
Represolve time = 0.02 sec. (1.38 ticks)
   4607     0       14.5477    28       24.9911      Cuts: 48    17926   41.79%
   4607     0       14.5477    28       24.9911      Cuts: 39    17960   41.79%
   4607     0       14.5477    28       24.9911      Cuts: 43    17988   41.79%
   4607     0       14.5477    28       24.9911      Cuts: 77    18017   41.79%
   4607     0       14.5477    28       24.9911      Cuts: 50    18052   41.79%
*  4611+    1                          24.9911       14.5477           41.79%
*  4616+    2                          24.9911       14.5477           41.79%
*  4989+  203                          23.7030       14.5477           38.63%
  11942  1298       23.4380     6       23.7030       20.0327    54778   15.48%

Clique cuts applied:  1
Implied bound cuts applied:  42
Flow cuts applied:  9
Mixed integer rounding cuts applied:  22
Gomory fractional cuts applied:  2

Root node processing (before b&c):
  Real time             =    0.19 sec. (21.04 ticks)
```

```
Parallel b&c, 8 threads:
  Real time             =    1.02 sec. (328.99 ticks)
  Sync time (average)   =    0.19 sec.
  Wait time (average)   =    0.00 sec.
                          ------------
Total (root+branch&cut) =    1.20 sec. (350.03 ticks)

Solution pool: 13 solutions saved.

MIP - Integer optimal solution:  Objective =  2.3703043670e+01
Solution time =    1.20 sec.  Iterations = 82513  Nodes = 17398
Deterministic time = 350.04 ticks  (290.97 ticks/sec)

CPLEX> MILP problem relaxed to LP with fixed integer variables using
incumbent solution.
CPLEX> Version identifier: 12.10.0.0 | 2019-11-26 | 843d4de2ae
Tried aggregator 1 time.
LP Presolve eliminated 381 rows and 120 columns.
Reduced LP has 126 rows, 13 columns, and 492 nonzeros.
Presolve time = 0.00 sec. (0.34 ticks)

Iteration log . . .
Iteration:     1   Dual objective     =             0.000000

Dual simplex - Optimal:  Objective =  2.3703043670e+01
Solution time =    0.00 sec.  Iterations = 12 (0)
Deterministic time = 0.66 ticks  (662.91 ticks/sec)

CPLEX> time = 1.3048028000000045
status = Optimal
objective = 23.703043670125375
```