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. Nonostante il tempo impiegato per ottenere una soluzione ottimale dipenda fortemente dall'istanza del problema, con i valori dei parametri algoritmici di default cplex si dimostra molto più efficiente di cbc. In tabella 1 è riportato il tempo impiegato dal solutore per trovare una soluzione ottimale al variare del numero di poligoni, con $H = 15$, $Kmax = 5$, $xmax = 10$ e $ymax = 10$.

N	Cbc	Cplex
3	0.40896 sec	0.21322 sec
4	2.5556 sec	0.26914 sec
5	36.933 sec	0.62200 sec
6	611.48 sec	2.7290 sec
7	701.74 sec	40.640 sec

Table 1: Confronto tra cbc e cplex al variare di N.

Quindi, per poter migliorare l'efficienza di cbc, è stata effettuata una grid search tra i parametri che la libreria PuLP mette a disposizione. In tabella 2 sono mostrate le 5 migliori combinazioni in termini di tempo dei parametri in un'istanza creata con $N = 6$, $H = 15$, $Kmax = 5$, $xmax = 10$ e $ymax = 10$. Questi risultati non sono da confrontare con quelli in tabella 1 dal momento che l'istanza del problema è differente e, come precedentemente detto, il tempo necessario a trovare una soluzione ottimale dipende fortemente dall'istanza.

CUTS	PRESOLVE	DUAL	STRONG	Tempo (sec)
True	False	False	None	0.04863
True	False	False	2	0.04894
True	False	False	5	0.04899
True	False	False	9	0.04899
True	False	True	9	0.05564

Table 2: Migliori 5 combinazioni dei parametri di cbc.

Innanzitutto bisogna apprezzare l'importanza dei parametri algoritmici di cbc, grazie ai quali si può passare da più di 700 secondi, nel caso peggiore, a 5 centesimi di secondo per ottenere una soluzione ottimale. Si può inoltre osservare l'importanza del parametro CUTS e l'effetto opposto a quello desiderato che provocano i parametri DUAL e PRESOLVE. Per capire il motivo di questi risultati si può analizzare più dettagliatamente l'output prodotto dal solver durante la risoluzione di un'istanza del problema, di cui ne è mostrato un estratto iniziale e uno finale in figura 1.

Si può notare come il solver sia in grado di trovare un upper bound alla soluzione ottimale rapidamente, con solo 1.6% di errore dopo 1000 nodi visitati.

```
Cbc0016I Integer solution of 13.645675 found by strong branching after 18423 iterations and 508 nodes (6.99 seconds)
Cbc0016I Integer solution of 13.164889 found by strong branching after 18515 iterations and 509 nodes (7.74 seconds)
Cbc0038I Full problem 146 rows 138 columns, reduced to 144 rows 49 columns - 12 fixed gives 140, 33 - still too large
Cbc0038I Full problem 146 rows 138 columns, reduced to 140 rows 33 columns - too large
Cbc0010I After 1000 nodes, 248 on tree, 13.164889 best solution, best possible 9.8895457 (9.77 seconds)
Cbc0010I After 2000 nodes, 167 on tree, 13.164889 best solution, best possible 9.8895457 (14.44 seconds)
Cbc0010I After 3000 nodes, 154 on tree, 13.164889 best solution, best possible 9.8895457 (15.68 seconds)

Cbc0010I After 153000 nodes, 917 on tree, 12.948624 best solution, best possible 12.896394 (251.55 seconds)
Cbc0010I After 154000 nodes, 725 on tree, 12.948624 best solution, best possible 12.896394 (252.96 seconds)
Cbc0010I After 155000 nodes, 534 on tree, 12.948624 best solution, best possible 12.913402 (254.39 seconds)
Cbc0010I After 156000 nodes, 362 on tree, 12.948624 best solution, best possible 12.922085 (255.72 seconds)
Cbc0010I After 157000 nodes, 251 on tree, 12.948624 best solution, best possible 12.93211 (257.36 seconds)
Cbc0010I After 158000 nodes, 73 on tree, 12.948624 best solution, best possible 12.93211 (258.96 seconds)
Cbc0010I After 159000 nodes, 19 on tree, 12.948624 best solution, best possible 12.942225 (260.77 seconds)
Cbc0001I Search completed - best objective 12.948623788025, took 5007107 iterations and 481780 nodes (261.11 seconds)
```

Figure 1: Log di cbc durante la ricerca di una soluzione del problema.

Da ciò si può dedurre che lo sforzo che il solver impiega nel cercare una soluzione accettabile se il parametro PRESOLVE è True sia solamente una perdita di tempo. Al contrario, l'utilizzo di CUTS migliora notevolmente l'incremento del lower bound, che dopo 1000 nodi ha un errore del 24%, garantendo così una riduzione molto più rapida del gap, quindi una notevole diminuzione del tempo necessario ad ottenere una soluzione ottimale.

Nonostante il notevole miglioramento dopo il tuning dei parametri, cbc continua ad impiegare più tempo nel risolvere una data istanza rispetto a cplex (3.40 sec cbc contro 0.38 sec di cplex in un'istanza con 6 poligoni). Purtroppo non è possibile analizzare dettagliatamente il funzionamento di cplex, tuttavia, data l'importanza dei cut, può essere interessante esaminare quali tipi di cut vengono generati. Un esempio è mostrato in figura 2.

```
GUB cover cuts applied:  1
Clique cuts applied:  4
Cover cuts applied:  1
Implied bound cuts applied:  132
Flow cuts applied:  38
Mixed integer rounding cuts applied:  90
Gomory fractional cuts applied:  9
```

Figure 2: Cuts generati da cplex.

Si può notare che i cut più utilizzati sono gli Implied bound cuts e i Mixed integer rounding cuts. Dagli esperimenti è risultato che quest'ultimo tipo di cuts è utilizzato molto anche da cbc, al contrario degli Implied bound cuts che cbc attualmente non supporta. Quindi è ragionevole pensare che questi cuts giochino un ruolo importante durante la risoluzione di un problema, insieme ovviamente a tutti gli altri dettagli che differenziano gli algoritmi utilizzati da cbc e da cplex.

3.3 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 3 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 3: 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, quando si otteniene una soluzione ammissibile molto migliore rispetto alla precedente. Per concludere, in figura 3 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 ha impiegato 241 secondi.

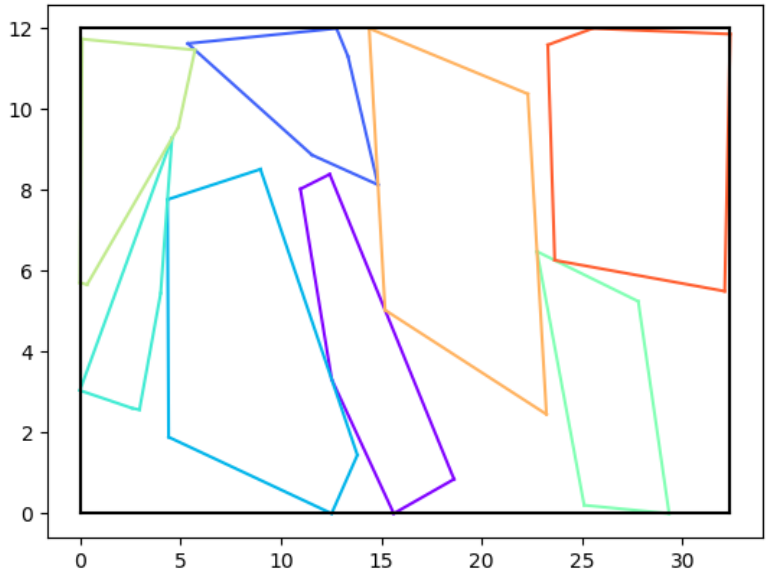

Figure 3: Soluzione di un'istanza.