Università degli Studi di Perugia

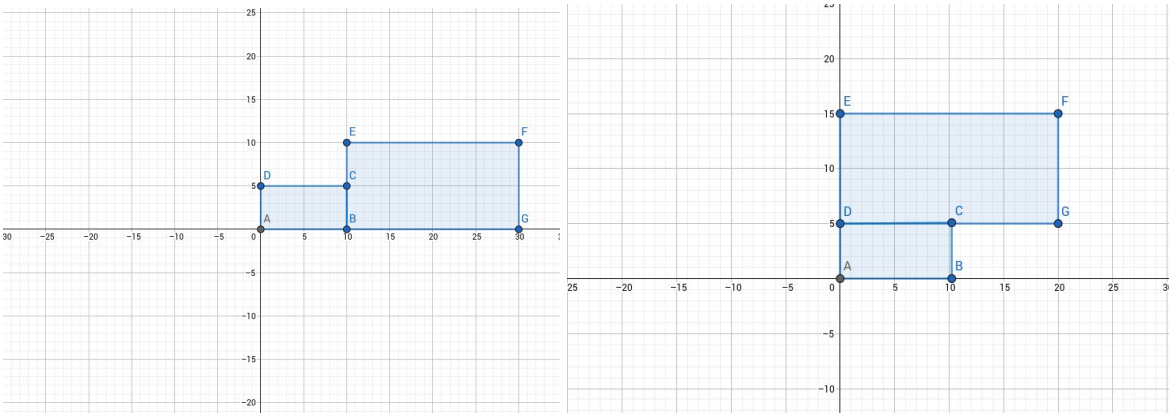
Progetto di Programmazione Dichiarativa

Rectangle packing

Studente
Lorenzo Ranucci (276655)

Descrizione del problema

Quello del rectangle packing è un problema che riguarda il posizionamento ottimale e senza sovrapposizioni di un insieme di rettangoli di dimensioni variabili all'interno di un piano bidimensionale di coordinate discrete. I rettangoli possono essere ruotati o capovolti in qualsiasi maniera. L'ottimalità del posizionamento è caratterizzata dalla sua compattezza, il posizionamento migliore è quello con un perimetro minore rispetto agli altri.



Nelle immagini precedenti ci sono due esempi di posizionamento, solamente il secondo è ottimo.

Analisi del problema

Il problema è stato affrontato utilizzando due linguaggi di programmazione dichiarativa: Prolog ed ASP. L'idea alla base di entrambe le soluzioni proposte è quella di enumerare tutti i possibili posizionamenti dei rettangoli avvalendosi di semplici accorgimenti per limitarne la generazione ai soli casi non banali.

Le osservazioni fatte sono:

- é indifferente la scelta, la posizione e l'orientamento del primo rettangolo posizionato
- il posizionamento ottimo è tale da avere ogni rettangolo adiacente ad almeno un altro rettangolo, questo significa che, una volta posizionato il primo rettangolo arbitrariamente, i successivi posizionamenti possono essere fatti lungo il perimetro dei rettangoli già posizionati

- un rettangolo può essere posizionato, dopo aver fissato il primo vertice in un punto, in otto maniere diverse
- la soluzione ottima è il posizionamento con perimetro minore e quindi quella con maggiori coincidenze di punti perimetrali tra i rettangoli
- scegliendo il posizionamento ottimo localmente con una logica *greedy* si ottiene la soluzione globalmente ottima

Considerate le precedenti osservazioni è stato progettato un algoritmo che:

1. posiziona il primo rettangolo in maniera arbitraria
2. sceglie il posizionamento ottimo localmente:
    2.1. posiziona tutti i rettangoli rimanenti da posizionare, in ogni punto perimetrale dei rettangoli già posizionati, in ognuno degli otto modi possibili
    2.2. scarta i posizionamenti non validi a causa di sovrapposizioni
    2.3. sceglie tra tutti i posizionamenti validi generati quello con maggiore sovrapposizione perimetrale

Implementazione ASP

ASP è un linguaggio di programmazione dichiarativa che si basa sulla generazione di modelli stabili (o answer set). Il programma implementato fa in modo che il solver ASP generi l'answer set composto da tutti i posizionamenti validi e scelga grazie all'istruzione *maximize*, il modello stabile che ha una maggiore quantità di coincidenze perimetrali.

I rettangoli da posizionare sono rappresentati dal predicato *rect(indiceRettangolo, dimensione1, dimensione2).*
I posizionamenti dei rettangoli (e quindi l'output del programma) sono espressi sotto forma di predicati *vertex(indiceRettangolo, indiceVertice, posizioneAsseX, posizioneAsseY).*
Il predicato *range* simboleggia le dimensioni della griglia in cui i rettangoli vengono posizionati.
I predicati *perimeterPoint* e *areaPoint* sono utilizzati per indicare rispettivamente i punti perimetrali e i punti area dei rettangoli posizionati.

Il vertice con indice zero viene posizionato in ogni modello stabile al centro della griglia.
Per qualsiasi altro rettangolo viene posizionato il vertice con indice zero in un *perimeterPoint* e di seguito posizionati i restanti vertici in ognuno degli otto modi possibili e rispettando i vincoli determinati dalle dimensioni del rettangolo. La regola *vertexElsewhere* assicura il posizionamento di ogni vertice una volta sola in ogni modello stabile.
La regola:

```
:-rect(Ra,Ba,Ha),rect(Rb,Bb,Hb), range(X;Y), Ra!=Rb,
     areaPoint(Ra,X,Y), areaPoint(Rb,X,Y),
     areaPoint(Ra,X+1,Y), areaPoint(Rb,X+1,Y),
     areaPoint(Ra,X,Y+1), areaPoint(Rb,X,Y+1),
     areaPoint(Ra,X+1,Y+1), areaPoint(Rb,X+1,Y+1).
```

permette di scartare tutte le soluzioni che hanno rettangoli sovrapposti. La sovrapposizione è presente quando due rettangoli posizionati hanno in comune almeno quattro *areaPoint* che determinano i vertici di un'unità quadrata della griglia. In parole semplici, due rettangoli sono sovrapposti se hanno un quadrato 1x1 della griglia in comune.

La regola:

```
commonPerimeterPoint(R1,R2,X,Y):-range(X;Y),rect(R1,B1,H1),rec
t(R2,B2,H2), R2>R1, perimeterPoint(R1,X,Y),
perimeterPoint(R2,X,Y).
```

e l'istruzione:

```
maximize{commonPerimeterPoint(R1,R2,X,Y):rect(R1,B1,H1):rect(R
2,B2,H2):range(X;Y): R1<R2}.
```

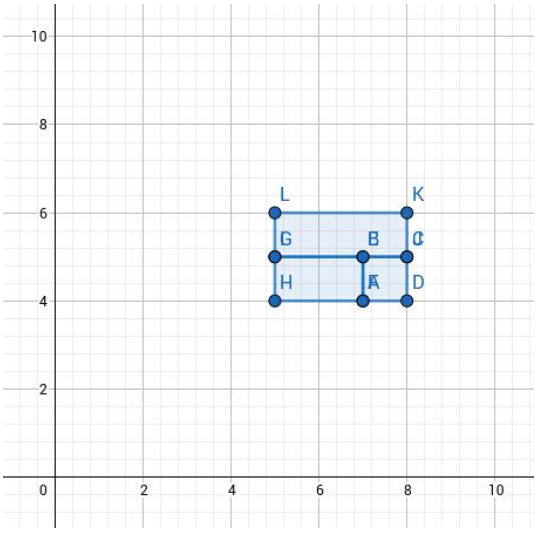
fanno in modo che il solver scelga le soluzioni che massimizzano le coincidenze di punti perimetrali dei rettangoli.

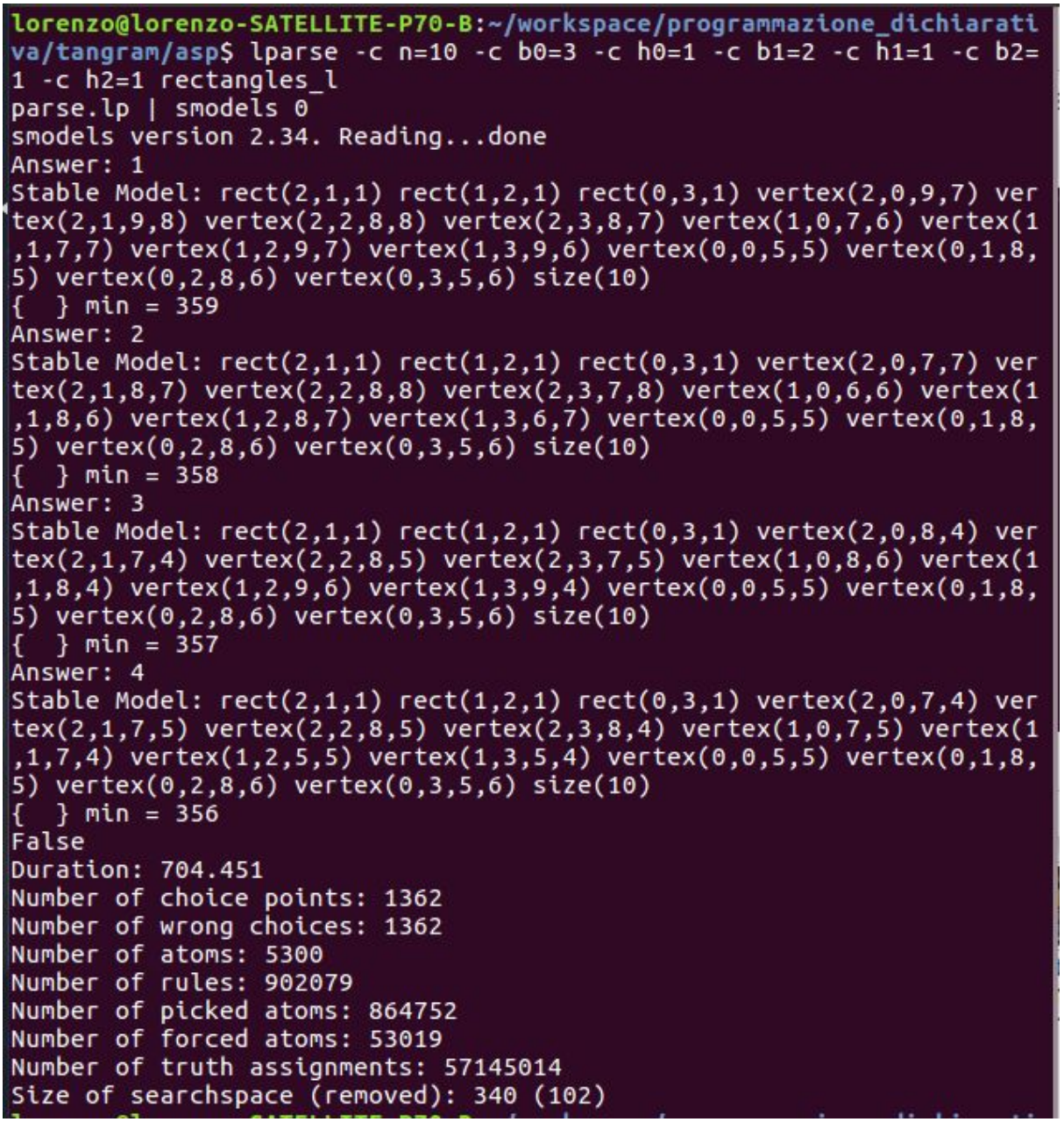
Test

Un test su un input di tre rettangoli di dimensioni 1x1, 2x1 e 3x1 può essere eseguito invocando lparse e smodels nel seguente modo:

```
lparse -c n=10 -c b0=3 -c h0=1 -c b1=2 -c h1=1 -c b2=1 -c h2=1
rectangles_lparse.lp | smodels 0
```

Lanciando il solver su un laptop con processore i7-4720HQ, quad-core a 2,60 GHz e 16 GB di memoria, l'esecuzione termina dopo 704 secondi trovando una soluzione ottima rappresentata nell'immagine seguente.





Implementazione Prolog

Il programma Prolog implementato può essere eseguito dal solver chiedendo un goal del tipo:

```
solution([rect(0,vertex(0,_,_),vertex(1,_,_),vertex(2,_,_),vertex(3,_,_), 1, 1),
rect(1,vertex(0,_,_),vertex(1,_,_),vertex(2,_,_),vertex(3,_,_), 1, 1)],Solution).
```

dove al primo termine abbiamo la lista dei rettangoli da posizionare e al secondo termine la variabile risultato che il solver cerca di valorizzare con una lista di rettangoli analoga a quella passata come input al primo termine, ma con le coordinate variabili dei vertici anch'esse valorizzate.
La regola *solutionAux* viene unificata ricorsivamente dal solver. Nel corpo della regola solutionAux la regola *bestLocalPositioning* viene utilizzata per trovare il miglior posizionamento locale per il primo rettangolo della lista in input e in funzione dei rettangoli già posizionati nelle precedenti chiamate ricorsive.
Nel corpo della regola bestLocalPositioning c'è la regola *validPositionings* che permette di ottenere una lista di tutti i posizionamenti validi per il rettangolo corrente e in funzione dei rettangoli già posizionati.
La regola *bestLocalPositioningAux* scorre ricorsivamente tutti i posizionamenti validi individuando quello con *score* massimo.
La regola *score* permette di contare il numero di punti perimetrali che i rettangoli posizionati condividono, maggiore è il numero di punti perimetrali condivisi, migliore è il posizionamento. Tale regola si avvale della regola *removeDuplicates* che prevede come primo termine una lista (i punti perimetrali del posizionamento valido), come secondo termine la stessa lista senza duplicati e al terzo termine il numero di duplicati (lo score).
La regola *validPositionings* ha al terzo termine tutti i posizionamenti validi del rettangolo al primo termine, in funzione dei rettangoli già posizionati al secondo termine. Nel corpo di questa regola abbiamo la regola *rectPositionings* che restituisce tutti gli otto posizionamenti possibili del rettangolo in ogni punto perimetrale dei rettangoli già posizionati. La regola *validPositioningAux* scorre ricorsivamente tutti questi posizionamenti e scarta quelli non validi. Un posizionamento non valido viene scartato se almeno un *areaPoint* o *perimeterPoint* dell'ultimo rettangolo posizionato si sovrappone con almeno un areaPoint dei rettangoli posizionati in precedenza.
La regola *rectPositioning* posiziona un rettangolo nel punto di origine (1000,1000) se nessun rettangolo è stato ancora posizionato, altrimenti si ottengono tutti i punti perimetrali del posizionamento con la regola *positioningPerimetersPoints* e si posiziona in maniera ricorsiva il rettangolo negli otto modi possibili in ognuno di tali punti tramite la regola *rectPositioningsInPoint*.

Differentemente dalla versione ASP del programma, questa implementazione non è completa, infatti è dipendente dall'ordine iniziale dei rettangoli in input. La ricerca del miglior posizionamento locale in funzione dei rettangoli già posizionati non avviene considerando ogni rettangolo rimasto da posizionare, ma solamente il primo della lista.

Andrebbe indagato il motivo per cui l'esecuzione continua indefinitamente senza apparentemente mai terminare per alcuni input.

Test

Un test su un input di tre rettangoli di dimensioni 1x1, 1x1 e 2x2 può essere eseguito eseguendo GNU Prolog:

```
gprolog
```

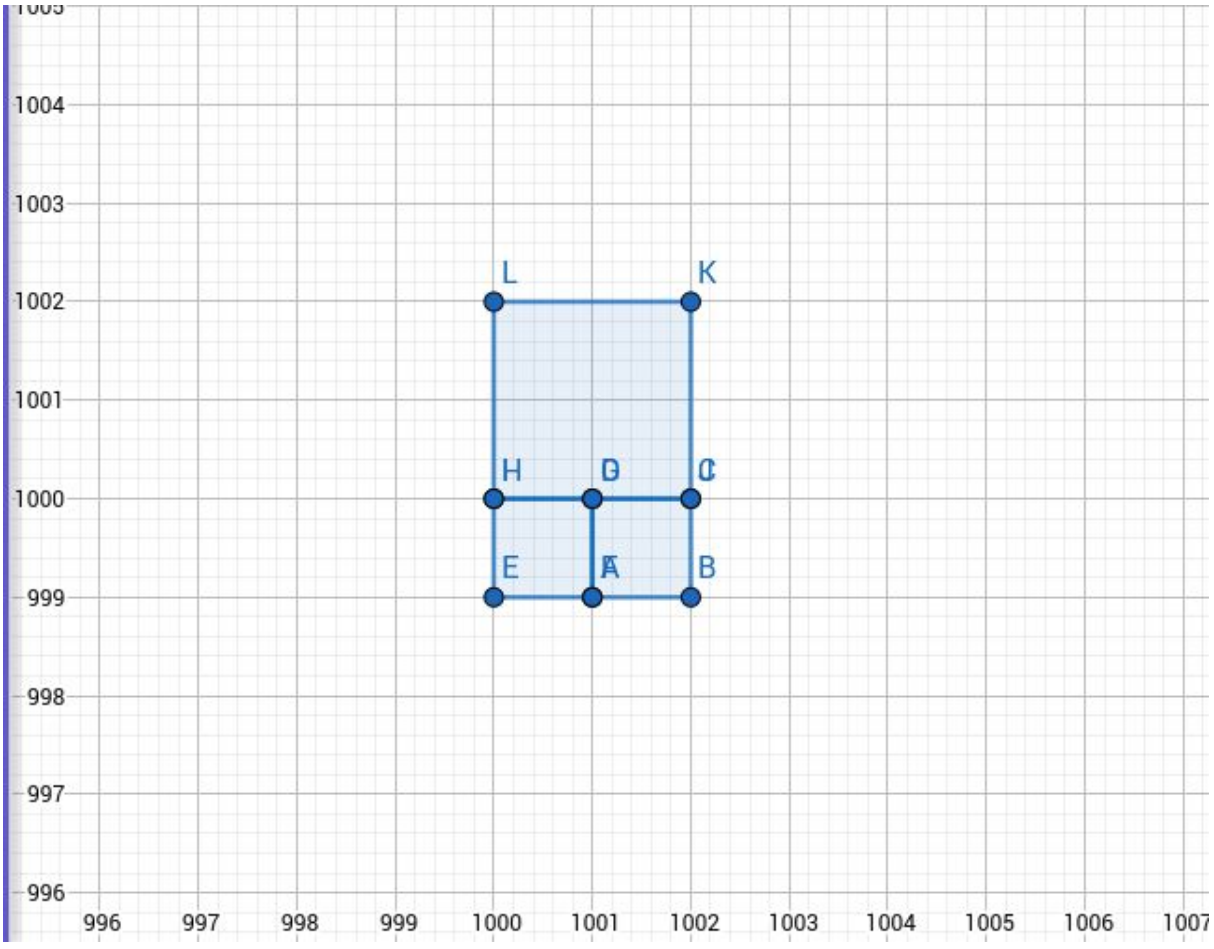
compilando il file principale del progetto *solution.pl:*

```
[solution].
```

e facendo la query:

```
solution([rect(0,vertex(0,_,_),vertex(1,_,_),vertex(2,_,_),ver
tex(3,_,_),2,2),rect(1,vertex(0,_,_),vertex(1,_,_),vertex(2,_,
_),vertex(3,_,_),1,1),rect(2,vertex(0,_,_),vertex(1,_,_),verte
x(2,_,_),vertex(3,_,_), 1, 1)],Solution).
```

Lanciando il solver su un laptop con processore i7-4720HQ, quad-core a 2,60 GHz e 16 GB di memoria, l'esecuzione termina dopo 2 secondi trovando una soluzione ottima rappresentata nell'immagine seguente.




```
| ?- solution([rect(0,vertex(0,_,_),vertex(1,_,_),vertex(2,_,_),vertex(3,_,_), 2, 2),
 rect(1,vertex(0,_,_),vertex(1,_,_),vertex(2,_,_),vertex(3,_,_), 1, 1), rect(2,vertex
(0,_,_),vertex(1,_,_),vertex(2,_,_),vertex(3,_,_), 1, 1)],Result).

Result = [rect(2,vertex(0,1001,999),vertex(1,1002,999),vertex(2,1002,1000),vertex(3,1
001,1000),1,1),rect(1,vertex(0,1000,999),vertex(1,1001,999),vertex(2,1001,1000),verte
x(3,1000,1000),1,1),rect(0,vertex(0,1000,1000),vertex(1,1002,1000),vertex(2,1002,1002
),vertex(3,1000,1002),2,2)] ? ;

(1980 ms) no
```