Progetto del corso
Automated Reasoning 2017/18
Esercizio 9

Gabriele Venturato (125512)

23 luglio 2018

1 Materiale Prodotto

Nel materiale consegnato è possibile trovare uno script `create_instances.py` per generare le istanze di input per i due modelli in ASP e MiniZinc — le istanze sono le stesse per entrambi. Lo script è stato scritto in Python (versione 3.6) e genera le istanze nella cartella `instances/`. Ogni file di input generato ha il nome $in\_n\_k\_h\_i.ext$, dove n, k, e h sono i parametri di input, e i è un indice per distinguere le diverse istanze con gli stessi parametri di input. Le tessere del domino di ogni istanza sono generate casualmente quindi ogni istanza è (idealmente) diversa dalle altre.

Sono presenti poi le cartelle `asp/` e `minizinc/`. All'interno di ognuna si trovano:

- una cartella `input/` che contiene una copia delle istanze del rispettivo modello generate (sono in una cartella diversa da quella di generazione per evitare sovrascritture involontarie).
- una cartella `output/` che contiene gli output dei test effettuati. Il nome dei file è analogo a quello degli input. É presente una sottocartella *sample* che contiene una porzione di output (vedi punto successivo).
- uno script bash `benchmark.sh` che può essere lanciato senza specificare niente, oppure con l'opzione `-s` che testa il modello su 10 delle istanze disponibili scelte in maniera pseudo-casuale e distribuita tra tutte le possibilità.
- e infine i modelli del problema.

Inoltre, nella cartella di minizinc è possibile trovare un modello *max-sequence.mzn* che è un'estrapolazione dal modello completo, che permette di trovare la sequenza più lunga possibile di tessere, non vincolate dalla scacchiera.

2 Il problema

Oltre a quanto descritto nel testo del problema, ho assunto che una tessera può essere ruotata in verticale solo di 90 gradi in senso orario. In questo modo il valore destro della tessera può essere o a destra o in alto.

2.1 Difficoltà

Ho dovuto modificare i parametri di input dopo aver testato diverse soluzioni. Ho lasciato le dimensioni di n invariate, ho invece definito $k = n/2$ invece che $n/5$, e ho diminuito leggermente le dimensioni della scacchiera testando $h = 6, 7, 8, 9, 10$.

Tale scelta è frutto di un'analisi dei risultati prodotti dai modelli. Il modello ASP non ha presentato gravi problemi. Quello con cui ho avuto più difficoltà è stato il modello MiniZinc che non è stato performante fin da subito e mi ha fornito le indicazioni su come diminuire i parametri. Prima di tutto la dimensione della scacchiera è stata diminuita per abbassare il numero di possibilità in cui si possono disporre le tessere ovviamente. Per quanto riguarda k invece ho voluto abbassarlo per diminuire il numero di diverse sequenze di tessere possibili. Questo mi ha permesso di "allontanarmi" dal valore $n!$ a cui tende se le tessere sono spesso ripetute, cioè se $k << n$.

3 Modelli

Per la produzione dei modelli sono partito da una versione iniziale che poi ho affinato man mano che verificavo le performance. Qui riporto le versioni finali che raccolgono tutti i miglioramenti considerati.

Innanzitutto ho cercato di scrivere dei modelli tenendo conto alcuni aspetti:

- ho limitato il numero di variabili il più possibile e in ogni caso ho cercato di restringere il dominio il più possibile con i mezzi forniti dai linguaggi (vincoli e domain predicate).

- ho cercato di utilizzare il minor numero di vincoli possibili evitando di inserirne di ridondanti ma comunque cercando di vincolare il problema il più possibile in modo diretto e preciso.

Oltre a questo nello specifico ho provato due approcci per migliorare l'efficienza, uno ha funzionato e uno no.

- Quello che non ha funzionato: nei modelli parlo di "scacchiera" e di "sequenza" (in MiniZinc in modo esplicito, in ASP attraverso i predicati `placed_on` e `next`). Ho pensato di fare a meno di uno dei due in qualche modo, per ridurre il numero di variabili e predicati utilizzati, ma non sono riuscito a modellare il problema senza tenerli entrambi.

- Quello che ha funzionato e mi ha dato lo speed-up maggiore rispetto a tutte le migliorie fatte, è stato notare che se esiste una sequenza di lunghezza l posizionabile nella scacchiera, allora esiste una sequenza di lunghezza l che inizia dall'angolo in alto a sinistra con la tessera posizionata in orizzontale. Questo ha eliminato molte situazioni simmetriche, e i vincoli relativi a questo si trovano in entrambi i modelli nelle sezioni "symmetry breaking".

Nelle due prossime sottosezioni ci sono i codici dei modelli ricchi di commenti per rendere più chiare le idee di ogni definizione e vincolo.

3.1 Modello ASP

Oltre a quanto detto, e a quanto scritto nei commenti del codice, per il modello ASP non c'è molto altro da dire. L'idea che ho seguito è stata quella di definire una sequenza, con il predicato `next`, imponendo in aggiunta i vincoli di spazio imposti dalla presenza della scacchiera.

In questo modello non è presente il vincolo inserito nel testo dell'esercizio relativo alla richiesta di avere una sequenza di lunghezza $l > h$ dove h è il lato della scacchiera, solamente perchè ho notato rallentamenti inserendolo, e dato che è facile verificare, una volta fornito il risultato, se l'ottimo individuato è maggiore del lato della scacchiera, ho preferito non inserirlo.

Una nota: nella scrittura del modello mi è venuto naturale utilizzare la X per le colonne, e la Y per le righe (pensando al piano cartesiano), anche se poi scrivendo il modello MiniZinc mi sono reso conto che forse era più intuitivo fare il contrario, ma comunque è un dettaglio che si può aggiustare facilmente in base alle preferenze, e non porta miglioramenti nell'efficienza.

```
%% Exercise 9 - (Custom) Domino
%% Gabriele Venturato (125512)
%% Automated Reasoning 2017/2018
%% tested with clingo 5.3.0

value(1..k).
tilename(1..n).
coord(1..h).

% Input: k, n, h, tile(n,l,r)

% ------------------------------------------------------------------------------ %
%% Placed
% a tile can be "placed on" the board at position (x,y)
% a tile is placed, if it is placed on a position
placed(T) :- placed_on(T,_,_).

% Left and Right values
% the value in (x,y) is the left (or top) one of the tile T
leftval(T,X,Y) :- placed_on(T,X,Y), placed_on(T,X+1,Y), coord(X+1).
leftval(T,X,Y) :- placed_on(T,X,Y), placed_on(T,X,Y+1), coord(Y+1).

% the value in (x,y) is the right (or bottom) one of the tile T
rightval(T,X,Y) :- placed_on(T,X,Y), not leftval(T,X,Y).

%% Compatible
% two tiles are compatible if and only if they are not the same tile
% and have the same value in position left and right
compatible(T1,T2) :- tile(T1,_,R), tile(T2,L,_), T1 != T2, R == L.
:- compatible(T,T).
:- compatible(T1,T2), tile(T1,_,R), tile(T2,L,_), R != L.


% ----------------------------- CONSTRAINTS ---------------------------------- %
%% Placed
% and can occupy exactly zero or exactly two positions
0 { placed_on(T,X,Y) : coord(X), coord(Y) } 2 :- tilename(T).
{ placed_on(T,X,Y) : coord(X), coord(Y) } != 1 :- tilename(T).

%% a tile can be only placed in two consecutive (no diagonal) cells of the board
:- placed_on(T,X1,Y1), placed_on(T,X2,Y2), X1 != X2, Y1 != Y2.
:- placed_on(T,X,Y1), placed_on(T,X,Y2), Y1 < Y2, Y2 != Y1 + 1.
:- placed_on(T,X1,Y), placed_on(T,X2,Y), X1 < X2, X2 != X1 + 1.

%% tiles can't overlap
0 { placed_on(T,X,Y) : tilename(T) } 1 :- coord(X), coord(Y).

%% Next
% define when a tile can be next to another
0 { next(T1,T2) : tilename(T2) } 1 :- tilename(T1), compatible(T1,T2).
0 { next(T1,T2) : tilename(T1) } 1 :- tilename(T2), compatible(T1,T2).
:- next(T1,T2), not compatible(T1,T2).
:- next(T,T).

% a tile can be next only to one another tile
:- next(T,T1), next(T,T2), T1 != T2.
:- next(T1,T), next(T2,T), T1 != T2.

%% Next and Placed
% a tile can't be placed and not next to anything
:- placed(T), placed(T1), T != T1, not next(T,_), not next(_,T).

% a tile can't be next to another if it is not placed
```

```
:- next(T,_), not placed(T).
:- next(_,T), not placed(T).

%% Where to place (if two tiles are next)
% if two tiles are on the same column the row distance must be 1
:- placed_on(T1,X1,Y1), placed_on(T2,X2,Y2),
    T1 != T2, next(T1,T2),
    rightval(T1,X1,Y1), leftval(T2,X2,Y2),
    X1 == X2,
    |Y1 - Y2| != 1.

% if two tiles are on the same row the column distance must be 1
:- placed_on(T1,X1,Y1), placed_on(T2,X2,Y2),
    T1 != T2, next(T1,T2),
    rightval(T1,X1,Y1), leftval(T2,X2,Y2),
    Y1 == Y2,
    |X1 - X2| != 1.

% they can't be on a different row and different column
:- placed_on(T1,X1,Y1), placed_on(T2,X2,Y2),
    T1 != T2, next(T1,T2),
    rightval(T1,X1,Y1), leftval(T2,X2,Y2),
    |Y1 - Y2| != 0,
    |X1 - X2| != 0.

% both different from 1 is not necessary because overlap is not allowed

%% Symmetry breaking
% if a sequence is long l, then there exist a sequence of length l which starts
% from top left corners
placed_on(T,1,1) :- firsttile(T).
placed_on(T,2,1) :- firsttile(T).

% firsttile is T if there are at least two tiles and T is the first
% or if T is the only tile placed.
firsttile(T) :- next(T,_), not next(_,T), placed(T1), T1 != T.
firsttile(T) :- not next(_,_), placed(T).

%% Sequence definition
sequence(S+1) :- S = #count{ T: next(T,_) }.

#maximize { S : sequence(S) }.
```

3.2 Modello MiniZinc

Prima di tutto: mi aspettavo di riuscire a produrre un modello più efficiente, invece nonostante le diverse analisi e i miglioramenti introdotti, c'è stato un problema che non sono riuscito a risolvere: ho notato che se ho una scacchiera in cui ci sta al massimo una sequenza di lunghezza l, però considerando i valori delle tessere riuscirei a formare una sequenza più lunga di l, il modello prima di dirmi che non ci sta nella scacchiera una sequenza di lunghezza $l + 1$ (e quindi che l è il, massimo), prova tutte le permutazioni di tessere per vedere se riesce a farle stare. Detto in altre parole, avrei voluto esprimere che: se viene individuata una sequenza di lunghezza $l + 1$ che non ci sta nella scacchiera (quindi che non rispetta i vincoli di spazio),

è inutile provare tutte le altre possibili sequenze di tessere, perchè tanto non ci staranno allo stesso modo della prima. Però purtroppo non sono riusicto a modellare un vincolo di questo tipo. Ho provato anche a valutare di cambiare completamente l'idea del modello, ma non sono riuscito a pensare a una modellazione alternativa. Se fosse possibile esprimere tale vincolo, senza dubbio il modello MiniZinc otterrebbe uno speed-up rilevante. Si può aggirare il problema "giocando" con il modello aggiuntivo che ho descritto all'inizio ("maxsequence"), andando a controllare se esiste una sequenza di una certa lunghezza, e se esiste, si può quindi modificare l'input per ottenere velocemente un risultato che attesta l'ottimo sulla lunghezza della sequenza.

Sebbene non sia riuscito a risolvere il problema appena descritto, nel cercare di risolverlo ho trovato una rappresentazione di "board" leggermente più efficiente: inizialmente avevo definito "board" come mi è venuto più naturale, ovvero $board_{xy} = t$ significava che la tile t si trovava in posizione (x, y) nella scacchiera. Poi l'ho cambiato, scrivendo in posizione (x, y) la posizione della tile in sequenza, invece che il suo "nome" (cioè un numero da 1 a n). In questo modo riduco il domino di ogni cella di board a $1..s$ invece che $1..n$. Ovviamente al crescere della sequenza possibile, s si avvicina ad n e le prestazioni ritornano come nella rappresentazione iniziale di board.

Oltre al problema descritto ho cercato di migliorare l'efficienza in diversi modi. In primo luogo ho cercato di sfruttare i vincoli globali, come suggerito dal manuale di MiniZinc, vista la loro efficiente implementazione, però non sono riuscito ad inserirli nel mio modello. Ho poi inserito dei vincoli aggiuntivi nella sezione "symmetry breaking", oltre a quello descritto all'inizio della sezione 3:

- Ho ridotto le dimensioni della sequenza massima vincolando dall'alto con $(h \times h)/2$, dato che comunque più di questo numero di tessere non posso inserirlo. Poi ho ridotto dal basso con il vincolo imposto dal testo dell'esercizio ($l > h$). Come scritto nel commento al codice, questo porta a trovare alcuni modelli come insoddisfacibili, ma d'altro canto permette di escludere dei valori possibili, riducendo il numero di tentativi da fare, quindi ho deciso di tenerlo. Inoltre con i parametri per i benchmark sono rarissimi i casi di qeusto tipo.

- Ho notato poi, con alcuni test, che potevo rimuovere delle simmetrie escludendo le permutazioni delle tessere con gli stessi valori. Il vincolo è però disattivato perchè porta vantaggi solo se $k << n$, cioè se sono presenti molte tessere ripetute, altrimenti causa rallentamenti inutili. C'è da dire inoltre che se $k << n$, nonostante si possa attivare quel

vincolo, le possibili permutazioni iniziano a diventare un numero fattoriale rispetto a k^2 (possibili tessere), quindi comunque diventa difficile da risolvere. Non ho riportato questo vincolo nel modello ASP appunto per la sua inutilità in relazione ai test richiesti dall'esercizio.

Infine, come ultimo aspetto per migliorare l'efficienza ho considerato la possibilità di aggiungere annotazioni alle variabili di ricerca. Queste annotazioni si trovano nell'istruzione `solve` e sono quelle con cui ho ottenuto i migliori risultati. La scelta di tenere l'euristica `indomain_min` sulla variabile s che è quella massimizzata, è dovuta alla decisione di voler comunque arrivare ad avere un parametro di approssimazione dell'ottimo. Inizialmente l'avevo impostata `indomain_max`, che lo rendeva efficiente per istanze in cui la sequenza massima è vicina ad n, però eccessivamente lento in casi in cui la sequenza massima è più corta.

```
% Exercise 9 - (Custom) Domino
% Gabriele Venturato (125512)
% Automated Reasoning 2017/2018
% tested with mzn-gecode v2.0.2 (i.e. fzn-gecode 5.1.0)

% ------------------------------- DATA ------------------------------------- %
% Input parameters
int: h; % board dimension
int: n; % number of tiles
int: k; % max value on tiles

% tiles[t,1] is the left value of tile t (2 is the right one)
array[1..n, 1..2] of 1..k: tiles;

% board[x,y] = p means that the tile t is in position p on the sequence,
% and that it is placed on position (x,y) on board
% (you can forget about this detail and think that a tile is placed on board,
% and not it's corresponding position in the sequence)
array[1..h, 1..h] of var 0..n: board;

var 1..n: s; % length of the sequence
array[1..n] of var 0..n: sequence;


% ---------------------------- CONSTRAINTS ---------------------------------- %
% SEQUENCE
% two tiles to be in sequence must respect values in them
constraint forall (i in 1..s-1) (
  tiles[ sequence[i] , 2 ] = tiles[ sequence[i+1] , 1 ]
);

% after last tile, can't have other tiles in the sequence
constraint forall (i in s+1..n) (
  sequence[i] = 0
);

% at least one tile in the sequence (avoid empty sequence)
constraint sequence[1] > 0;

% can use each tile at most once in the sequence
constraint forall(t in 1..n) (
```

```
  sum( [ bool2int( sequence[i] == t ) | i in 1..n ] ) <= 1
);

% BOARD
% a tile placed on the board must occupy at least two consequent cells
constraint forall( i,j in 1..h, p in 1..s ) (
  board[i,j] = p -> exists( a,b in {-1,0,1} where
      i+a >= 1 /\ j+b >= 1 /\
      i+a <= h /\ j+b <= h /\
      abs(a) + abs(b) = 1
    ) ( board[i+a,j+b] = p )
);

% for each row a tile can occupy at most two consequent cells
constraint forall( p in 1..s ) (
  sum( [ bool2int(board[i,j] == p) | i,j in 1..h ] ) <= 2
);

% defining where a tile can be placed on board
constraint forall( i in 1..s-1 ) (
  exists( t1x,t1y,t2x,t2y in 1..h ) (
    board[t1x,t1y] = i /\ rightval(board[t1x,t1y],t1x,t1y) /\
    board[t2x,t2y] = i+1 /\ leftval(board[t2x,t2y],t2x,t2y) /\
  abs(t1y - t2y) <= 1 /\
  abs(t1x - t2x) <= 1 /\
  abs(t1y - t2y) + abs(t1x - t2x) = 1)
);

% a tile can be on board iff it is in the sequence
constraint forall( p in 1..s ) (
  exists(x,y in 1..h) (board[x,y] = p)
);

% SYMMERTRY BREAKING
% the longest sequence will start from top left corner with the tile
% placed horizontally
constraint board[1,1] = 1 /\ board[1,2] = 1;

% try only sequences with length compatible with the board dimension
constraint s <= (h*h div 2);

% sequence must be > h (imposed by assignment)
% this could lead to unsatifiable situation if there are not enough tiles
% compatilble to build a sequence length at least h
constraint s > h;

% don't consider permutation of tiles with the same values
% this can be activated with high "n" and low "k", otherwise it is not worth it
%constraint forall( i in 1..s-1, j in i+1..s ) (
%  ( tiles[ sequence[i], 1 ] ==  tiles[ sequence[j], 1 ] /\
%  tiles[ sequence[i], 2 ] ==  tiles[ sequence[j], 2 ] )
%  -> sequence[i] < sequence[j]
%);

constraint forall( i,j in 1..h ) ( board[i,j] <= s );

% ------------------------------ OUTPUT ------------------------------------ %
solve :: seq_search([
    int_search([s], input_order, indomain_min, complete),
    int_search(sequence, first_fail, indomain_min, complete),
    int_search([ board[i,j] |i,j in 1..h], first_fail, indomain_min, complete)])
  maximize s;

output [
```

```
106   "Sequence length: " ++ show(s) ++
107   "\nSequence: " ++ show(sequence) ]
108 ++ ["\nBoard:\n"] ++
109 [
110   show(board[i,j]) ++ "\t" ++
111   if (j mod h == 0) then "\n" else "" endif
112   | i,j in 1..h
113 ];
114
115
116 % ----------------------------- PREDICATES ---------------------------------- %
117 % true if position (x,y) is the right (resp. bottom) value of tile t
118 predicate rightval(var int: p, 1..h: x, 1..h: y) =
119   exists(i,j in 1..h) (
120     board[i,j] == p /\ board[x,y] == p /\ ( x > i \/ y > j )
121   );
122
123 % true if position (x,y) is the left (resp. top) value of tile t
124 predicate leftval(var int: p, 1..h: x, 1..h: y) =
125   exists(i,j in 1..h) (
126     board[i,j] == p /\ board[x,y] == p /\ ( x < i \/ y < j )
127   );
```

4 Risultati

Riporto in tabella i risultati dei benchmark. Non ho prodotto grafici perchè i risultati non si prestano ad una visualizzazione di questo tipo. Ogni tabella rappresenta una dimensione diversa di n (e di conseguenza di k), che ho lasciato nelle prime due colonne per comodità di lettura. I parametri in input identificano univocamente anche il nome del file da cui sono stati prelevati. Oltre alle prime colonne che indicano i parametri già descritti nella sezione 1, per entrambi i modelli ci sono: il tempo di risoluzione, una colonna che indica se è stato individuato il fatto di aver raggiunto l'ottimo, e l'ottimo individuato alla fine dell'esecuzione. Nota che anche se il timeout di cinque minuti è scaduto, è possibile che l'ottimo sia stato comunque raggiunto, anche se non segnalato, perchè lo spazio di ricerca non è stato completamente esplorato.

In generale, come ci si può aspettare, si nota che per parametri in input minori, si riesce a raggiungere l'ottimo più spesso e con tempi molto ridotti. Man mano che cresce n invecce, risulta sempre più difficile individuare un ottimo, e comunque richiede tempi maggiori. Si può comunque notare che i tempi variano molto anche in base a come sono state generate le tessere del domino. Per cui, as esempio, per $n = 40$ si trovano casi in cui il timeout è scaduto, e casi in cui il tempo è di qualche secondo.

Come anticipato, il modello ASP risulta notevolmente più performante. Le ragioni individuate sono già descritte nella sezione relativa al modello MiniZinc.

				ASP			MiniZinc		
n	k	h	i	Time (s)	Opt	Opt val	Time (s)	Opt	Opt val
20	10	6	1	0.08	y	9	0.1	y	9
20	10	6	2	2.12	y	14	23.1	y	14
20	10	6	3	0.63	y	12	4.35	y	12
20	10	6	4	0.59	y	12	3.78	y	12
20	10	7	1	1.02	y	11	1.35	y	11
20	10	7	2	1.9	y	11	1.72	y	11
20	10	7	3	9.52	y	14	21.03	y	14
20	10	7	4	0.59	y	12	4.95	y	12
20	10	8	1	0.09	y	9	0.34	y	9
20	10	8	2	10.64	y	15	62.18	y	15
20	10	8	3	15.95	y	15	45.19	y	15
20	10	8	4	2.86	y	12	2.91	y	12
20	10	9	1	8.86	y	14	21.67	y	14
20	10	9	2	0.55	y	12	2.4	y	12
20	10	9	3	1.41	y	15	48.18	y	15
20	10	9	4	2.87	y	13	7.21	y	13
20	10	10	1	0.39	y	10	0.44		0
20	10	10	2	5.62	y	15	8.6	y	15
20	10	10	3	64.89	y	13	8.79	y	13
20	10	10	4	1.34	y	10	1.99		0

Da questi risultati si può notare che quando l'ottimo viene individuato esplicitamente, esso ovviamente coincide in entrambi i modelli. Quando non viene individuato invece, in generale, il modello ASP riesce ad individuare un ottimo maggiore (e quindi migliore) rispetto al modello MiniZinc.

Può essere che si riescano a trovare ottimi migliori anche con il modello MiniZinc, cambiando le annotazioni di ricerca, però va valuto istanza per istanza, e per un test su tutte gli input generati ho preferito mantenere un comportamento uniforme, come descritto nell'apposita sezione.

				ASP			MiniZinc		
n	k	h	i	Time (s)	Opt	Opt val	Time (s)	Opt	Opt val
30	15	6	1	299.71		15	300		15
30	15	6	2	299.7		15	300		15
30	15	6	3	59.6	y	15	300		15
30	15	6	4	299.71		15	300		15
30	15	7	1	4.08	y	16	117.08	y	16
30	15	7	2	20.8	y	16	111.19	y	16
30	15	7	3	4.76	y	17	300		16
30	15	7	4	10.48	y	16	108.65	y	16
30	15	8	1	7.13	y	15	54.89	y	15
30	15	8	2	5.68	y	14	16.97	y	14
30	15	8	3	39.42	y	17	300		16
30	15	8	4	6.01	y	18	300		16
30	15	9	1	83.79	y	20	300		17
30	15	9	2	20.91	y	16	174.34	y	16
30	15	9	3	67.65	y	19	300		17
30	15	9	4	14.96	y	16	156.32	y	16
30	15	10	1	8.04	y	17	257.48	y	17
30	15	10	2	130.02	y	20	300.01		17
30	15	10	3	71.86	y	19	300.01		17
30	15	10	4	8.31	y	15	21.13	y	15

				ASP			MiniZinc		
n	k	h	i	Time (s)	Opt	Opt val	Time (s)	Opt	Opt val
40	20	6	1	299.69		15	300.01		15
40	20	6	2	152.07	y	15	300.01		15
40	20	6	3	299.68		15	300.02		15
40	20	6	4	54	y	15	300.01		15
40	20	7	1	108.18	y	19	300.01		16
40	20	7	2	8.71	y	15	56.4	y	15
40	20	7	3	15.91	y	18	300.01		16
40	20	7	4	47.5	y	18	300.01		16
40	20	8	1	227.08	y	23	300.01		16
40	20	8	2	299.03		22	300.01		16
40	20	8	3	8.56	y	12	2.3	y	12
40	20	8	4	51.39	y	18	300.01		16
40	20	9	1	85.73	y	23	300.01		16
40	20	9	2	205.46	y	20	300.01		16
40	20	9	3	54.89	y	11	3.26	y	11
40	20	9	4	298.43		12	6.49	y	12
40	20	10	1	2.65	y	14	6.26	y	14
40	20	10	2	296.95		25	300.01		17
40	20	10	3	126.14	y	19	300.01		17
40	20	10	4	18.98	y	17	238.49	y	17

				ASP			MiniZinc		
n	k	h	i	Time (s)	Opt	Opt val	Time (s)	Opt	Opt val
50	25	6	1	299.45		15	300.01		15
50	25	6	2	299.57		15	300.01		15
50	25	6	3	299.56		15	300.01		15
50	25	6	4	299.48		15	300.01		15
50	25	7	1	299.19		21	300.01		16
50	25	7	2	299.22		20	300.01		16
50	25	7	3	299.16		21	300.01		16
50	25	7	4	299.15		21	300.01		16
50	25	8	1	298.43		21	300.01		16
50	25	8	2	298.67		24	300.01		16
50	25	8	3	24.24	y	17	300.01		16
50	25	8	4	298.69		21	300.01		16
50	25	9	1	297.83		23	300.01		16
50	25	9	2	297.68		24	300.01		16
50	25	9	3	92.58	y	22	300.01		16
50	25	9	4	297.76		11	300.01		16
50	25	10	1	28.06	y	22	300.01		17
50	25	10	2	296.22		23	300.01		17
50	25	10	3	296.77		21	300.01		17
50	25	10	4	296.55		23	300.01		17

				ASP			MiniZinc		
n	k	h	i	Time (s)	Opt	Opt val	Time (s)	Opt	Opt val
60	30	6	1	299.54		15	300.01		15
60	30	6	2	299.49		15	300.01		15
60	30	6	3	299.52		15	300.01		15
60	30	6	4	299.48		15	300.01		15
60	30	7	1	47.34	y	17	300.01		16
60	30	7	2	299.09		21	300.01		16
60	30	7	3	298.98		17	300.01		16
60	30	7	4	91.7	y	20	300.01		16
60	30	8	1	298.66		25	300.01		16
60	30	8	2	298.28		23	300.01		16
60	30	8	3	298.15		22	300.01		16
60	30	8	4	298.3		21	300.01		16
60	30	9	1	297.53		23	300.01		16
60	30	9	2	297.52		26	300.01		16
60	30	9	3	297.03		20	300.01		16
60	30	9	4	297.48		19	300.01		16
60	30	10	1	295.65		25	300.01		17
60	30	10	2	295.88		19	300.01		17
60	30	10	3	296.05		3	300.01		17
60	30	10	4	295.8		24	300.01		17