Progetto Ragionamento Automatico

Studente: Tristano Munini

ANNO ACCADEMICO 2019-2020

1 Il problema

Si riporta il testo della consegna

14 Epidemia 3

Si vuole ridurre la diffusione dell'epidemia di CoronaVirus2020 isolando le persone in una nave da crociera messa a disposizione. La nave ha camere doppie. Vi sono k corridoi con h camere per lato corridoio (h a sinistra e h a destra). I corridoi sono uno sopra l'altro. I condotti di aerazione potrebbero in linea di massima diffondere appena il virus.

Le N persone sono etichettate come malate, positive (sane), in osservazione (per probabile contatto con positivo, ma non ancora positive), in quarantena precauzionale.

Possono essere in camera assieme (non stiamo a pensare a sesso, simpatia etc) malati con malati, positivi sani con positivi sani, precauzionali con precauzionali. Invece quelli in osservazione meglio lasciarli soli.

le camere hanno una nozione di vicinato. Vicinato 1 per le due camere adiacenti, per quella di fronte (il corridoio a ha camere in ogni lato), e per quella sopra e sotto. Ovviamente le ultime camere del corridoio hanno meno vicini. Così come le camere di corridoio 1 e k. Vicinato 2 applicando due volte la regola di vicinato 1.

Si vuole trovare una allocazione delle persone nella nave in modo tale da minimizzare il numero di malati a distanza ≤ 2 dai precauzionali e distanza ≤ 1 dai positivi sani.

Se ti pare sensato, aggiungi pure degli altri vincoli/preferenze (ad esempio per i positivi sani).

1. Encode it in Minizinc **and** an ASP.
2. Prepare your benchmark suite changing the values on N, h and k. Prepare 100 benchmarks.
3. Run both the Minizinc and the ASP encoding on all the instances. Use a timeout of 5 minutes for each test ("configuration" option in Minizinc, –time-limit option in clingo, in both cases you can use linux tricks)
4. Choose one configuration that runs in a pair of minutes and try alternative seach primitives on it. Use the most promising ome for the more difficult instances.
5. Write a 6–10 pages report containing your models (and the reasons for some choices) and a presentation of the execution results. Try to infer some complexity considerations. Prepare the programs and the benchmark instances used in a unique zip file.

2 Minizinc

2.1 Il modello

Il modello riceve in input i valori K, H, M, O, P e Q. I primi due indicano numero di corridoi e numero di stanze per lato, mentre i restanti il numero di ospiti per ogni tipo (anche zero per alcuni). Si suppone $M \geq 1$ altrimenti ogni assegnamento di stanza sarebbe lecito, purché ci sia un numero sufficiente di stanze. Nella codifica utilizzata nel modello ad ogni stanza viene associato un intero partendo da 0 ed incrementando di 1 fino a raggiungere il numero totale di stanze, cioè $2 * K * H$, perché abbiamo K corridoi e per ciascun corridoio ci sono H stanze per lato (sinistro o destro). Con questa numerazione ogni K numeri si cambia lato, mentre ogni $2 * K$ numeri si indica un corridoio ad un piano superiore. La numerazione permette di modellare abbastanza facilmente le relazioni spaziali tra stanze, ad esempio: le prime H stanze appartengono al

lato sinistro del primo corridoio; le stanze dalla H alla $2 * H - 1$ appartengono al lato destro del primo corridoio; le stanze dalla $2 * H$ alla $3 * H - 1$ sono sul secondo piano a sinistra; etc...

```
include "globals.mzn";

int: K; % corridoi/piani
int: H; % stanze per lato

int: M; % malati
int: P; % positivi
int: O; % osservazione
int: Q; % quarantena precauzionale

set of int: stanze = 0..2*H*K-1;

array[1..M] of var stanze: malati;
array[1..P] of var stanze: positivi;
array[1..O] of var stanze: osservazione;
array[1..Q] of var stanze: quarantena;
```

Le limitazioni sul numero di ospiti per stanza sono date dai seguenti vincoli. Notare come questa modellazione permette di usare *alldifferent* sui numeri delle stanze degli ospiti in osservazione.

```
constraint
  % Al piu' due Malati nella stessa stanza
  forall(i in 1..M)
    (sum(j in i+1..M)
      (if (malati[i]==malati[j]) then 1 else 0 endif) <= 1)
  /\
  % Al piu' due Positivi nella stessa stanza
  forall(i in 1..P)
    (sum(j in i+1..P)
      (if (positivi[i]==positivi[j]) then 1 else 0 endif) <= 1)
  /\
  % In Osservazione isolati uno per stanza
  alldifferent(osservazione)
  /\
  % Al piu' due Quarantena nella stessa stanza
  forall(i in 1..Q)
    (sum(j in i+1..Q)
      (if (quarantena[i]==quarantena[j]) then 1 else 0 endif) <= 1)
  /\
  % No stanze miste
  forall(i in 1..M)
    (forall(j in 1..P)(malati[i] != positivi[j]    ) /\
     forall(j in 1..O)(malati[i] != osservazione[j]) /\
     forall(j in 1..Q)(malati[i] != quarantena[j]  ))
  /\
  forall(i in 1..P)
    (forall(j in 1..O)(positivi[i] != osservazione[j]) /\
     forall(j in 1..Q)(positivi[i] != quarantena[j]  ))
  /\
  forall(i in 1..O)
     (forall(j in 1..Q)(osservazione[i] != quarantena[j]))
;
```

La relazione di *Vicinato 1* verifica se due stanze sono adiacenti secondo una delle cinque direzioni: sopra, sotto, destra, sinistra e di fronte.

```
predicate vicini1(var stanze: s1, var stanze: s2) =
  % Di fronte
  (
    % Sono sullo stesso piano
    (s1 div H == s2 div H + if ((s1 div H) mod 2 == 0) then -1 else 1 endif)
    /\
    % Sono di fronte
    (s1 == s2+H \/ s1 == s2-H)
  )
  \/
  % Sopra/Sotto
  (
    % Sono sopra-sotto
    (s1 == s2+2*H \/ s1 == s2-2*H)
  )
  \/
  % Lato dx/sx
  (
    % Sono sullo stesso lato dello stesso piano
    (s1 div H == s2 div H)
    /\
    % Sono adiacenti
    (s1 == s2+1 \/ s1 == s2-1)
  )
;
```

Due stanze sono a distanza *Vicinato 2* se condividono una stanza a distanza *Vicinato 1*. In pratica, presa una stanza, un elemento nel suo *Vicinato 2* può essere raggiunto in due passi selezionando prima un *Vicinato 1* adeguato e poi un *Vicinato 1* della stanza appena selezionata. Nel vincolo si richiede l'esistenza di questa terza stanza che faccia da "perno" per lo spostamento.

```
predicate vicini2(var stanze: s1, var stanze: s2) =
  (s1 != s2 /\
   exists (s3 in stanze)
    ( s3 != s1 /\ s3 != s2
      /\
      (vicini1(s1,s3) /\ vicini1(s3,s2))
    )
  );
```

Si vuole minimizzare un intero c calcolato contando tutte le coppie di ospiti con tipologia scomoda (malato, quarantena precauzionale, positivo) in *Vicinato 1* oppure *Vicinato 2*, a seconda della tipologia. Notare che c ha un valore più grande se nelle stanze in vicinato ci sono più ospiti, come si vede in Figura 1, in cui $c = 4$ a causa dei malati nella stanza 1 in *Vicinato 2* con gli ospiti in quarantena della stanza 7. Le stanze si contano partendo da quella in alto a sinistra, stanza numero 0 e prima stanza a sinistra del primo corridoio, e procedendo per righe. Ogni due righe si sale al corridoio superiore, come indicato dai numeri sulla destra.

```
var int: c =
  sum(i in 1..M, j in 1..Q)
```

```
    (if (vicini1(malati[i], quarantena[j]) \/
         vicini2(malati[i], quarantena[j]))
     then 1 else 0 endif)
  +
  sum(i in 1..M, j in 1..P)
    (if (vicini1(malati[i], positivi[j]))
     then 1 else 0 endif)
;
```

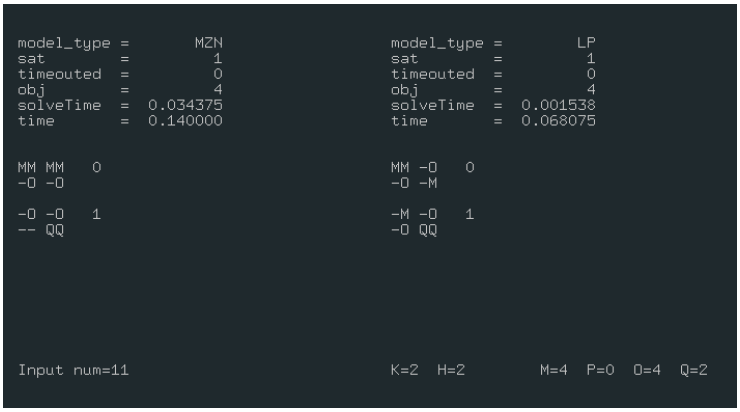

Figura 1: Schermata di *in_out_visualizer.py*

La configurazione che ha dato i risultati migliori è riportata nel listato che segue. Notare come si tenti fin da subito di raccogliere ospiti malati o in osservazione nelle prime stanze e sui piani più bassi, mentre per ospiti in quarantena o positivi si cerca una stanza nei piani superiori.

```
ann: search_ann;
solve :: search_ann
      minimize c;

search_ann = seq_search([
  int_search(malati,       input_order, indomain_min, complete),
  int_search(osservazione, input_order, indomain_min, complete),
  int_search(positivi,     input_order, indomain_max, complete),
  int_search(quarantena,   input_order, indomain_max, complete)
]);
```

2.2 *Symmetry Breaking*

Osservando che le stanze all'inizio ed alla fine dei corridoi hanno un numero inferiore di stanze in *Vicinato 1* (e quindi anche in *Vicinato 2*), risulta sensato fissare il primo malato nella prima stanza disponibile, ossia nella prima stanza del primo corridoio. Inoltre, poiché non ci sono differenze tra malati dello stesso tipo è possible fissare un ordinamento arbitrario. In questo caso ad ospiti con un indice inferiore vengono assegnate stanze ai piani più bassi.

```
constraint
  (malati[1] == 0)
  /\
  increasing(malati)
  /\
  increasing(positivi)
  /\
  increasing(osservazione)
  /\
  increasing(quarantena)
;
```

3 ASP

3.1 Il modello

Similmente a quanto visto per Minizinc, in *covid19.lp* si ha una prima definizione delle stanze

```
stanza(S) :- corridoi(K), stanze_per_lato(H), S=0..2*K*H-1.
```

Poiché gli ospiti devono essere assegnati esattamente ad una stanza, si definisce il vincolo

```
1 { in_stanza(P,T,S) : stanza(S) } 1 :- soggetto(P,T).
```

in cui *soggetto(P,T)* indica il numero identificativo P associato ad un ospite e la sua tipologia: 0 significa malato; 1 significa positivo; 2 significa osservazione; 3 significa quarantena. I predicati vengono forniti nell'input e si suppone siano corretti, quindi non è possibile che un P soddisfi la relazione con due T differenti. Poiché le stanze hanno capacità limitate, che dipendono dal tipo di ospite, risulta necessario definire i vincoli

```
% Ogni stanza ha capacita' limitata (2 ospiti per stanza)
:- T!=2, in_stanza(P,T,S), in_stanza(P1,T,S),
   in_stanza(P2,T,S),
   P!=P1, P!=P2, P1!=P2.

% Nel caso osservazione limito di piu' (1 ospite per stanza)
:- T=2, in_stanza(P,T,S), in_stanza(P1,T,S),
   P!=P1.
```

Inoltre una stanza può essere condivisa solo da ospiti dello stesso tipo, quindi vengono vietate tutte le coppie illecite

```
:- in_stanza(P,T,S), in_stanza(P1,T1,S), T!=T1.
```

Due camere sono a distanza *Vicinato 1* se rispettano uno dei cinque vincoli di vicinanza. Per motivi di efficienza vengono calcolati soltanto le vicinanze tra ospiti che danno un contributo nel calcolo della funzione di costo, quindi malati, quarantena e positivi. Qui vengono riportati i primi due vincoli

```
% Vicini1 tra malato e (quarantena o positivo)
scomodo(P,P1) :-
  P<P1,
```

```
  in_stanza(P,T,S), in_stanza(P1,T1,S1),
  T!=T1, T!=2, T1!=2, % Sono diversi e nessuno e' osservazione
  T*T1==0,            % almeno uno dei due e' malato
  % Vicini1 % CASO: di fronte
  stanza(S), stanza(S1), S!=S1,
  stanze_per_lato(H),
  S/H == S1/H + (-1)**(1+ (S/H)\2), % sullo stesso piano
  S == S1 + H*(-1)**(1+ (S/H)\2).   % sono di fronte

scomodo(P,P1) :-
  P<P1,
  in_stanza(P,T,S), in_stanza(P1,T1,S1),
  T!=T1, T!=2, T1!=2, % Sono diversi e nessuno e' osservazione
  T*T1==0,            % almeno uno dei due e' malato
  % Vicini1 % CASO: sopra
  stanza(S), stanza(S1), S!=S1,
  stanze_per_lato(H),
  S == S1 +2*H.
```

Due camere sono a distanza *Vicinato 2* se condividono una camera a distanza *Vicinato 1*. Diversamente da quanto fatto in Minizinc, qui è stato definito un vincolo per ciascuno dei 12 possibili vicini 2 di una stanza, considerando soltanto le coppie di stanze contenenti malati od ospiti in quarantena. Nei listati seguenti sono riportati due vincoli di questo tipo

```
% Vicini2 tra malato e quarantena
scomodo(P,P1) :-
  P<P1,
  in_stanza(P,T,S), in_stanza(P1,T1, S1),
  |(T-T1)|==3, % Uno dei due e' quarantena l'altro e' malato
  % Vicini2 % CASO: sx sx
  stanza(S), stanza(S1), S!=S1,
  stanza(S2), S2!=S, S2!=S1,
  stanze_per_lato(H),
  S/H == S1/H, S2/H == S1/H,        % tutti stesso lato dello stesso piano
  S - 2 == S1,                      % sx
  S - 1 == S2.                      % sx sx

scomodo(P,P1) :-
  P<P1,
  in_stanza(P,T,S), in_stanza(P1,T1, S1),
  |(T-T1)|==3, % Uno dei due e' quarantena l'altro e' malato
  % Vicini2 % CASO: dx front
  stanza(S), stanza(S1), S!=S1,
  stanza(S2), S2!=S, S2!=S1,
  stanze_per_lato(H),
  S/H == S1/H + (-1)**(1+ (S/H)\2), % sullo stesso piano
  S + 1 == S2,                      % a dx
  S2 == S1 + H*(-1)**(1+ (S2/H)\2). % di fronte al dx
```

La funzione da minimizzare viene calcolata contando che ospiti di che stanze soddisfano la relazione *scomodo*. Due ospiti non possono soddisfare la relazione se occupano stanze non in vicinato tra loro, oppure se appartengono a tipologie non problematiche (es. in osservazione).

```
cost(C) :- C == #count { P,P1 : scomodo(P,P1) }.
#minimize {C : cost(C)}.
```

3.2 *Symmetry Breaking*

Come nel modello Minizinc, il primo malato viene assegnato alla prima stanza e viene fissato un ordinamento arbitrario.

```
% Symmetry
in_stanza(1,0,0).
% Ai primi assegno prime stanze
:- in_stanza(P,T,S), in_stanza(P1,T1,S1), T=T1, P<T1, S>S1.
```

4 Infrastruttura di supporto

Per agevolare la raccolta dati e l'analisi delle soluzioni fornite dai due modelli ho sviluppato una piccola libreria python sfruttando i pacchetti *minizinc* e *clingo* che permettono di caricare, configurare e lanciare i rispettivi modelli. Inoltre, ho implementato un visualizzatore delle soluzioni con *ncurses* in python per potere valutare manualmente e confrontare meglio le soluzioni, questo è risultato molto utile per trovare errori durante la fase di modellazione.
I file che permettono l'esecuzione dei modelli e salvano i loro output sono:

- *input_generator.py* genera le istanze e le salva nei formati *.lp* e *.dzn* nelle relative cartelle *inputs_mzn* ed *inputs_lp*. Per fare ciò si appoggia a *my_lib.py*, istanziando la classe *InputGenerator* e passando gli argomenti per gli intervalli di H e K. Gli input vengono creati in modo casuale considerando il numero massimo di posti disponibili e poi saturandolo fino ad un livello casuale. In questo modo è possibile generare input difficili (molti ospiti "scomodi") oppure facili (se il numero di ospiti è basso rispetto al numero di posti).

- *run.py* avvia l'esecuzione di entrambi i modelli su tutti gli input generati, salva gli output man mano che vengono trovate le soluzioni. Anche questo file si appoggia a *my_lib.py*, in particolare usa le classi *RunnerMzn* e *RunnerLp* che permettono di caricare i modelli indicati e di utilizzare le configurazioni migliori per ciascuno. Una barra di progresso indica la percentuale di input elaborati;

- *run_mzn.py* e *run_lp.py* permettono di eseguire un particolare modello su un input, specificandolo con il numero relativo. Ad es. python run_mzn.py 10 tornerà un *pretty print* della soluzione dell'istanza *inputs_mzn/input_10.dzn*;

- *my_gloabals.py* contiene tutte le globali utili, come ad esempio il tempo di timeout per le esecuzioni dei modelli, oppure nomi di cartelle e file;

- *batch_saver.py* è risultato utile in fase di sviluppo dei modelli perché crea una copia di input, output e modelli in una cartella apposita. In questo modo è stato possibile confrontare le prestazioni di modelli diversi. Anche questo file si appoggia a *my_lib.py* ed utilizza la classe *BatchCoordinator*;

- *my_lib.py* contiene un insieme di classi che permettono di astrarre modelli, istanze, soluzioni oppure semplificare la creazione e gestione dei file.

Eseguendo python run.py verranno create due cartelle, una per modello, contenenti i file *.json* delle soluzioni, è qui riportato un esempio di una soluzione generata in questo modo.

```
{
 "model_type": "MZN", "sat": true, "obj": 1,
 "timeouted": false, "solveTime": 0.000832, "time": 0.09,
 "sol": {
    "K": 1, "H": 3,
    "M": [ 0 ],
    "P": [ 2, 2, 4, 4 ],
    "O": [ 1 ],
    "Q": [ 3, 5, 5 ]
 }
}
```

Infine, eseguendo il file *in_out_visualizer.py* è possibile scorrere tra le soluzioni prodotte dai due modelli. In Figura 2 si può osservare il programma in esecuzione: a sinistra si ha la soluzione di Minizinc con rappresentazione grafica di camere e dei posti occupati; a destra il modello ASP non ha generato alcuna soluzione perché è andato in timeout; in basso si hanno informazioni riguardo all'input di partenza. Il visualizzatore permette di scorrere avanti ed indietro tra le soluzioni contenute nelle cartelle *outputs_mzn* ed *outputs_lp*.

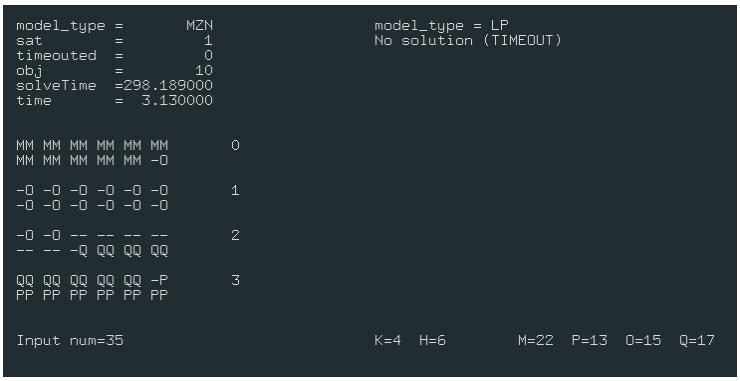

Figura 2: Schermata di *in_out_visualizer.py*

5 Tempi

I grafici che seguono sono stati generati usando R e *ggplot*.
In Figura 3 si vede come il tempo totale di risoluzione degli input dati. Notare come in genere il modello Minizinc sia migliore del modello ASP. I pallini indicano se il modello è andato in timeout. L'andamento del grafico è dato dal modo in cui sono stati generati gli input, quindi dal loro ordinamento: durante

la generazione venivano effettuati due cicli for annidati uno per il numero di corridoi ed uno per il numero di stanze. Confrontando Figura 4, Figura 3 e Figura 5 si osserva come l'andamento delle dimensioni degli input sia in linea con la complessità dell'istanza, misurata in tempo di risoluzione dei modelli. In Figura 5 si può osservare il tempo di risoluzione dei due modelli. I pallini indicano per quali istanze i modelli sono andati in timeout.

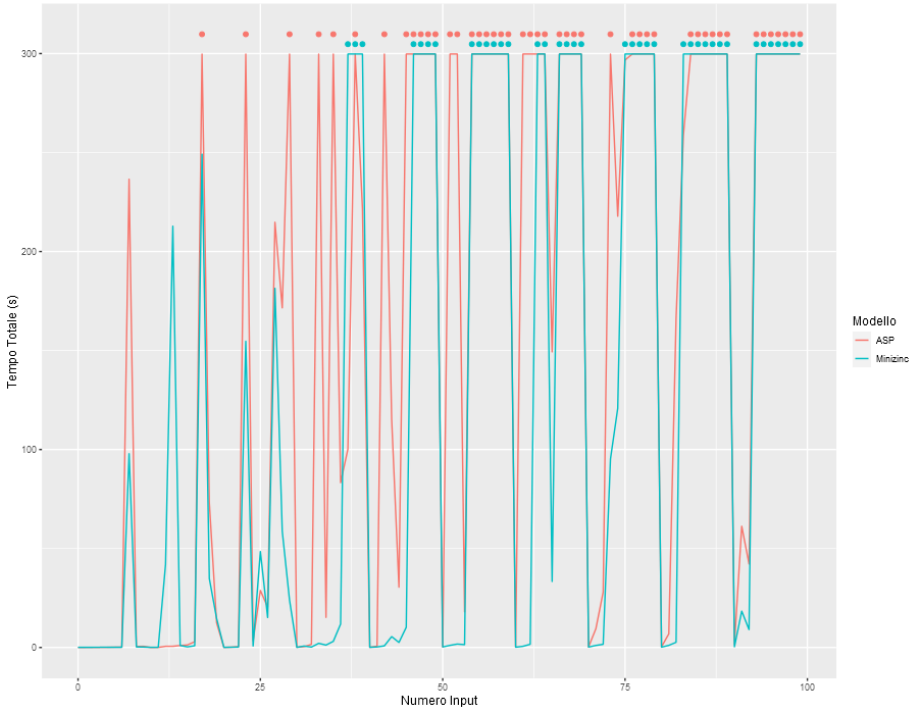

Figura 3: Input in ordine di numerazione

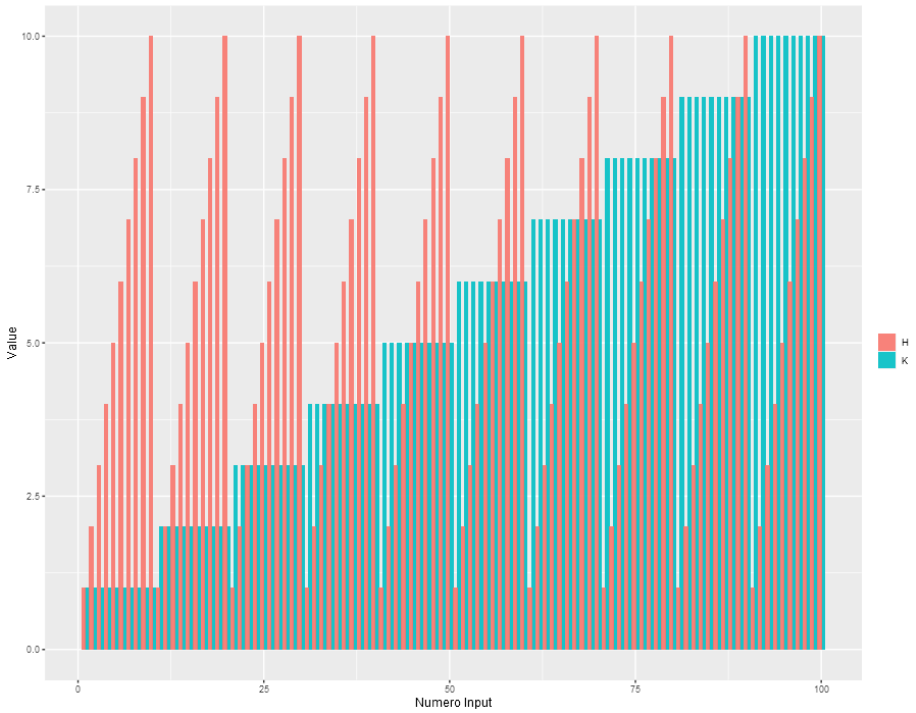

Figura 4: Dimensioni H e K degli input in ordine di numerazione

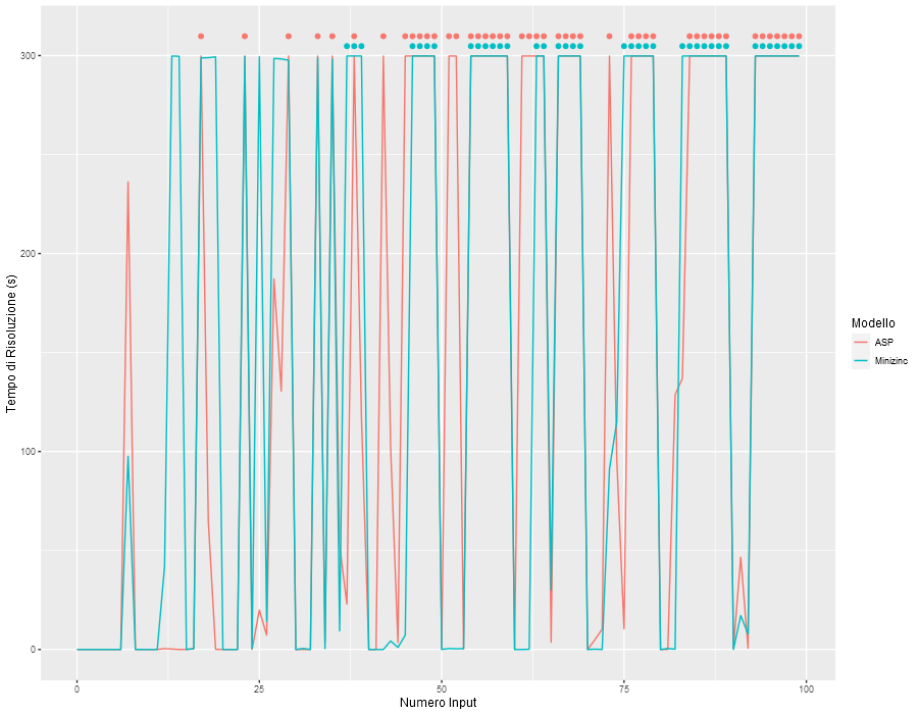

Figura 5: Input in ordine di numerazione

Osservando soltanto le soluzioni per le quali non si è raggiunto il timeout ed ordinando secondo tempo totale di risoluzione del modello ASP si ottiene il grafico in Figura 6. Notare come in più occasioni, su uno stesso input, un

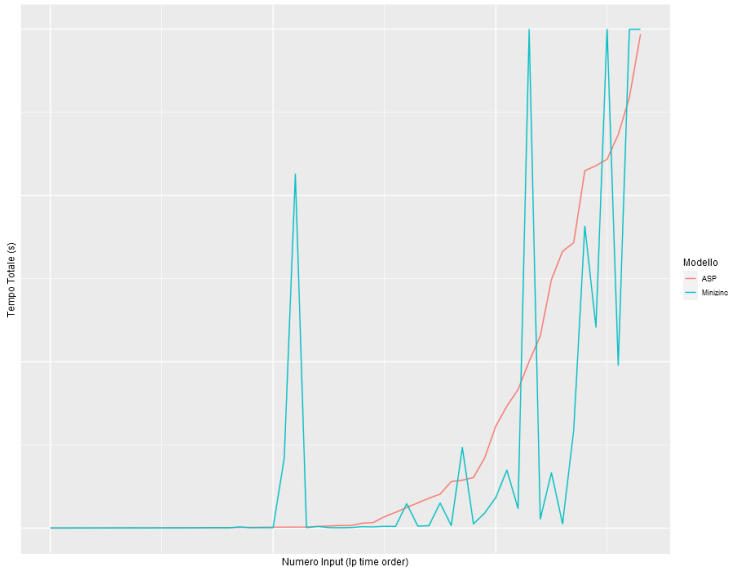

Figura 6: Input in ordine crescente di tempo totale per il modello ASP

modello performa molto bene mentre l'altro ha difficoltà. Una delle cause di questo fenomeno è il modo in cui il modello Minizinc cerca le soluzioni, cioè iniziando a collocare gli ospiti in quarantena nelle stane più lontane e dei piani più alti. Ciò non viene fatto nel modello ASP. Le osservazioni appena fatte valgono anche per il tempo di risoluzione, come si vede in Figura 7.
Confrontando i valori ottenuti dai due modelli si ha che

- Minizinc batte sul tempo complessivo ASP 47 volte su 100;
- Minizinc batte sul tempo di risoluzione ASP 30 volte su 100.

Possiamo quindi dire che i modelli hanno capacità confrontabili e che il tempo di grounding inficia in modo notevole sulle prestazioni del modello ASP. Infatti, considerando soltanto i tempi di risoluzione, il modello ASP è da preferirsi.

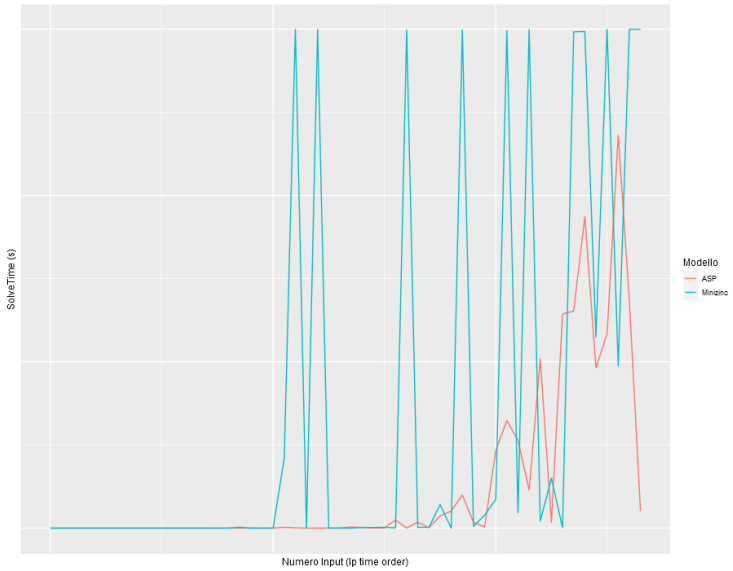

Figura 7: Input in ordine crescente di tempo totale per il modello ASP