1 Introduzione

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

2 Minizinc

2.1 Il modello

Ad ogni stanza viene associato un intero partendo da 0 ed incrementando di 1. Il numero totale di stanze è $2 * K * H$, perché abbiamo K corridoi e per ciascun corridoio ci sono H stanze per lato (sinistro o destro). Con questa numerazione ogni K numeri si cambierà 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 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 per 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 camere sono a distanza *Vicinato 2* se condividono una camera a distanza *Vicinato 1*. In pratica, presa una camera, un elemento nel suo *Vicinato 2* può essere raggiunto in due passi selezionando prima un *Vicinato 1* adeguato e poi un *Vicinato 1* della camera appena selezionata. Nel vincolo si richiede l'esistenza di questa terza camera 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)
;
```

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)
]);
```

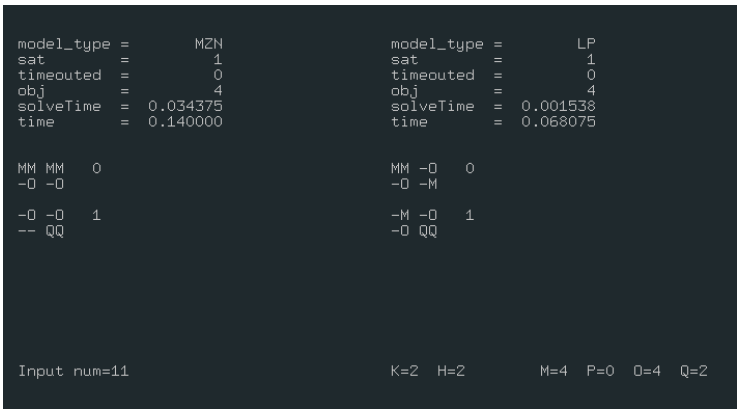

Figura 1: Schermata di *in_out_visualizer.py*

2.2 *Symmetry Breaking*

Osservando che le camere all'inizio ed alla fine dei corridoi hanno un numero inferiore di camere 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 camera 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. Le stanze vengono definite con

```
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 definiscono i vincoli

```
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

```
% 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 (esclusi i positivi), 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 Qui vengono riportati i primi due

TODO here

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

```
  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: adiacenti 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).

```
  %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
```

allungare?

3.2 *Symmetry Breaking*

Poiché non ci sono differenze tra malati dello stesso tipo è possibile fissare un ordinamento arbitrario. In questo caso ad ospiti con un indice inferiore vengono assegnate camera ai piani più bassi.

```
  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
```

giustificare meglio?

commentare la configurazione migliore

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 per poterle valutare manualmente e confrontare meglio, questo è risultato molto utile per trovare errori durante la fase di modellazione.

4.1 La libreria

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.

add image

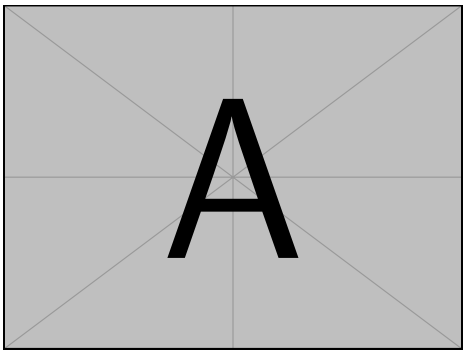

Figura 2: TODO img in_out_visualizer