Progetto ospedali
MMSD

Giorgio Turro, Stefano Vincenzi

Indice

Capitolo 1

Scopo e Dati

Lo scopo del progetto è quello di creare una simulazione del processo di cura di pazienti per operazioni non d'urgenza.
Per fare ciò sono stati usati dati delle operazioni della popolazione effettuate in Piemonte negli anni 2011, 2012 e 2013. Ulteriori dati utilizzati sono quelli delle distanze tra i comuni e lista degli ospedali con relative specialità e capienze.
Creato la simulazione che corrispondeva il più possibile alla realtà, sono stati aggiunti infine delle policy di riduzione e ridistribuzione delle risorse per stimare cosa sarebbe successo nel caso di situazioni particolari (chiusura di ospedali/specialità ecc..).
Tutto il progetto è scritto in python3 ed utilizza diverse librerie tra cui pyomo che verrà descritta in seguito.

1.1 Dati ed Elaborazione

Illustriamo di seguito i dati di partenza e la loro elaborazione con riferimenti al codice per capire su cosa si basa la simulazione.

1.1.1 Dati

Lista dei principali file utilizzati e loro descrizione:

- **specialtyCapacitySchedules.xlsx**: file contenente id specialità, id ospedale, per ogni giorno della settimana la capacità di ricovero e la capacità massima.
- **sdoN.csv**: con N in (1,2,3) a seconda dell'anno, file contenente informazioni di ogni paziente operato.
- **mapping_hosp_comuni.csv**: file di mapping tra id ospedale ed id del comune in cui è sito
- **distanzeComuniOspedali.csv**: file che rappresenta le distanze tra tutti i comuni. Oltre all'oggetto *distanzeComuniOspedali*, viene anche utilizzato per creare il csv *distanzeOspedali.csv* che è una matrice quadrata 69x69 con diagonale principale 0 che rappresenta le distanze tra i soli comuni che hanno un ospedale

1.1.2 Elaborazione

Di seguito elenchiamo come vengono rielaborati i dati della sezione precedente e qual'è il loro formato finale. La lista contiene anche gli oggetti python più importanti salvati con la libreria

joblib. Il salvataggio in questo formato permette un risparmio di tempo per rielaborazioni future, oltre ad essere un modo rapido per salvare e caricare strutture dati.

- **specialtyCapacitySchedules.xlsx**: questo file viene rielaborato nello script *parser_ ospedale.py* il quale produce un file chiamato *specialtyCapacitySchedules.csv* con le seguenti colonne:
  Index, codici_ospedale, codici_specialita, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY, SUNDAY, capacita_max, year
  Esempio di record: 0,res_01000300,spec_09,11,12,15,13,9,6,6,34,2011

- **sdoN.csv**: questo file viene rielaborato nello script *parser_ pazienti.py* e produce il file *ricoveri.csv* con le seguenti colonne:
  Index, anno, n_record, data_prenotazione, data_ricovero, gg_degenza, codice_struttura_erogante, disciplina_uo_ammissione, COD_BRANCA, DES_BRANCA
  Esempio di record: 0, 2011, 13978474, 2011-07-28, 2011-07-31, 2, 010611-00, CARDIOLOGIA, 8.0, CARDIOLOGIA

- **ricoveri_simulazione**: oggetto python creato con la libreria *joblib* che permette di salvare e caricare i dati di *ricoveri.csv* in modo più ottimizzato. Creato nello script *parser_ data.py*, viene prima controllato se esiste ed in tal caso caricato, altrimenti viene creato.

- **risorse_simulazione**: oggetto python creato con la libreria *joblib* che permette di salvare e caricare i dati di *specialtyCapacitySchedules.csv* in modo più ottimizzato. Creato nello script *parser_ data.py*, viene prima controllato se esiste ed in tal caso caricato, altrimenti viene creato.

- **dict_resources**: dizionario salvato con *joblib*, creato partendo dall'oggetto *risorse_ simulazione*. Il dizionario è così composto: Chiave principale "id ospedale", seconda chiave "anno" con valore un dizionario con: chiave codice specialità, valore una lista delle capacità [capacità giornaliera (7 valori), capacità massima]).

- **hosp_series**: oggetto *joblib* di lista dei codici ospedali senza duplicati

- **year_series**: oggetto *joblib* di lista degli anni senza duplicati

- **dict_mapping**: oggetto *joblib* che racchiude il dizionario del mapping del file mapping_hosp_comuni.csv. Il dizionario sarà id_c:id_h con *id_ c* l'id del comune e *id_ h* l'id dell'ospedale di quel comune. In pratica si ha che per ogni comune ho la lista delle distanze con gli altri comuni

- **dict_distances**: oggetto *joblib* con nome *distanzeComuniOspedali* che racchiude il dizionario della tabella del file distanzeComuniOspedali.csv. Il dizionario sarà id_c:id_ch : dis. *id_ c* è l'id del comune, *id_ ch* è l'id del comune dell'ospedale e *dis* è la distanza in metri

- **dict_map_residenza**: oggetto *joblib* che racchiude il dizionario di mapping tra id del record del paziente e l'id del comune di residenza, la struttura è quindi id_r:id_residenza

Capitolo 2

Componenti

In questo capitolo elenchiamo e descriviamo le strutture base e le feature presenti nel progetto.

2.1 Strutture di base

Tutta la simulazione si bassa su due classi python, *Patient* e *Hospital*, entrambe queste classi sono nel file *object_ classes.py*.
Come campi per la classe *Hospital* abbiamo:

- **id_hosp**: id univoco dell'ospedale
- **id_spec**: id della specialità
- **capacity**: lista lunga 8 campi, i primi 7 sono la capacità giornaliera dal lunedì alla domenica della specialità, l'ottavo è la capacità massima della specialità
- **waiting_queue**: lista dei pazienti che per un qualche motivo non sono stati ricoverati il giorno prestabilito
- **counter_day_cap**: counter giornaliero che conta quanti pazienti sono entrati in questa specialità
- **rest_queue**: lista di degenza dei pazienti, quanti giorni un paziente deve ancora occupare un posto letto
- **counter_day_queue**: counter che conta quanti pazienti sono entrati in lista di attesa
- **counter_max_queue**: counter che conta quanti pazienti sono entrati in lista di attesa, serve per distinguere la motivazione per cui si entra in coda

Per la classe *Patient* abbiamo:

- **id_patient**: id del paziente
- **patient_id_hosp**: id dell'ospedale in cui deve andare il paziente
- **patient_id_spec**: id specialità di cui il paziente ha bisogno
- **rest_time**: giorni di degenza del paziente
- **patient_day_recovery**: data del ricovero

- **patient_true_day_recovery**: data effettiva del ricovero
- **queue_motivation**: motivo per il quale è entrato in coda
- **counter_queue**: numero della sua posizione in coda

2.2 Lista Feature

Elenchiamo ora, con riferimenti al codice, le possibilità di tuning implementate per coprire più casistiche possibili.
Le modifiche dei parametri si fanno nel file *main_simulation.py* e sul txt *remove_info.txt*.
Nello script troviamo i seguenti campi da poter modificare:

- **spurious_days**: giorni di attesa prima che il sistema arrivi in una condizione di equilibrio. Viene usato come punto di partenza da cui si iniza ad anticipare i pazienti, con valore a 0 disabilita l'ingresso anticipato
- **forward_days**: è la finestra di giorni in cui guardare per anticipare i pazienti. Viene usata solo quando spurious_days non è 0. Più si allarga la finestra più il tempo della simulazione cresce
- **reduction_perc**: numero che indica la percentuale di riduzione della capacità giornaliera delle specialità. Con 0 non si applica alcuna riduzione
- **capacity_threshold**: numero che indica da quale capacità iniziare ad applicare la riduzione della reduction_perc. Per esempio: con reduction_perc = 5 e capacity_threshold = 8 si ridurrà di un 5% la capacità giornaliera di tutte le specialità che hanno tale capacità strettamente maggiore a 8
- **block_flag**: ha come calori *True* o *False* ed indica se utilizzare l'ottimizzatore settimanale o le policy greedy. Con *False* si utilizza l'ottimizzatore
- **mod**: indica quale modello usare per l'ottimizzatore. Con 0 usa il modello con la sommatoria.

Il file *remove_info.txt* contiene le risorse (ospedali e specialità) che da una certa data non devono più essere considerate. La prima riga del file ha i codici ospedali da rimuovere separati da uno spazio, nella seconda riga ci sono coppie nel formato *codici ospedale,specialità* separate da spazio e nella terza riga c'è la data da quando rimuovere tali risorse nel formato aaaa/mm/gg. 

2.3 Pyomo

Pyomo è uno strumento open-source che si occupa di optimization modeling ed è sviluppato in python. In Pyomo troviamo i seguenti elementi chiave:

Variables

Le variabili sono le parti sconosciute o passibili di cambiamento del modello. Solitamente i valori che assumono le variabili costituiscono la soluzione dell'ottimizzazione. Nel caso in esame le variabili sono x, δ e Δ

Parameters

I parametri rappresentano i dati che devono essere forniti per eseguire l'ottimizzazione. Nel caso in esame i parametri sono P, H, l, γ, m e d

Relations

Le relazioni rappresentano equazioni o disequazioni. Nell'implementazione del modello sono dati dai Constraint che nel caso in esame sono:

- $PatientInOnlyOneHospital$: $\sum_h x_{p,h} = 1$
- $PatientsRedistribution$: $\sum_p l_p * x_{p,h} <= \gamma_h$
- $DiscomfortCalculation$: $\delta_p >= (d_{p,h} - m_p) * x_{p,h}$
- $bigDeltaGreaterThanDelta$: $\Delta >= \delta_p$

Goals

I goal rappresentano la funzione obiettivo, nel nostro caso: $min\Delta$. La prima funzione obiettivo utilizzata era data dalla somma minima dei discomfort $min \sum_p \delta$, per questa funzione obiettivo il constraint $bigDeltaGreaterThenDelta$ non risulta necessario.

Codice

Di seguito si riporta l'estratto di codice pyomo. L'estratto non è altro che l'implementazione del modello lineare fornito:
Funzione Obiettivo: $min \sum_p \delta$ Soggetto a:

- $\sum_h x_{p,h} = 1 \ \forall p \in P_s$
- $\sum_p l_p x_{p,h} <= \gamma_h \ \forall h \in H_s$
- $\delta_p >= (d_{p,h} - m_p) x_{p,h} \ \forall p \in P_s \ h \in H_s$
- $\delta_p \in \mathbf{Z}^+$
- $x_{p,h} \in 0, 1 \ \forall p \in P_s \ h \in H_s$

Nella prima parte si definiscono parametri e variabili definendone tipo e dominio e eventuali vincoli. Si definisce quindi la funzione obiettivo e a seguire si indicano tutti i vincoli che, come detto, non sono altro che la stesura secondo la sintassi pyomo del modello lineare di cui sopra.

Code

```
# Parameter
model.P = pyo.Set(within=pyo.NonNegativeIntegers)
model.H = pyo.Set(within=pyo.NonNegativeIntegers)

model.l = pyo.Param(model.P)
model.gamma = pyo.Param(model.H)
model.m = pyo.Param(model.P)
model.d = pyo.Param(model.P, model.H)

# Variables
```

```
model.x = pyo.Var(model.P, model.H, domain=pyo.Binary)
model.delta = pyo.Var(model.P, domain=pyo.NonNegativeReals)
model.big_delta = pyo.Var(domain=pyo.NonNegativeReals)

# Goals
def obj_expression(m):
  return m.big_delta

model.OBJ = pyo.Objective(rule=obj_expression)

# Constraints
def patient_in_only_one_hospital(m, p):
  return sum(m.x[p,h] for h in m.H) == 1

model.PatientInOnlyOneHospital = pyo.Constraint(model.P,
                                  rule=patient_in_only_one_hospital)

def patients_redistribution(m, h):
  return sum(m.l[p] * m.x[p,h] for p in m.P) <= m.gamma[h]

model.PatientsRedistribution = pyo.Constraint(model.H,
                                  rule=patients_redistribution)

def discomfort_calculation(m, p, h):
  return m.delta[p] >= (m.d[p,h]-m.m[p])*m.x[p,h]

model.DiscomfortCalculation = pyo.Constraint(model.P,
                                  model.H,
                                  rule=discomfort_calculation)

def big_delta_greater_than_delta(m, p):
  return m.big_delta >= m.delta[p]

model.bigDeltaGreaterThanDelta = pyo.Constraint(model.P,
                                  rule=big_delta_greater_than_delta)
```

Capitolo 3

Software

In questo paragrafo vedremo il workflow della simulazione con riferimenti diretti al codice.

3.1 Workflow

In figura 3.1 mostriamo un diagramma di flusso dell'esecuzione della simulazione.
I quadrati in rosso sono in mutua esclusione, sono strategie diverse di assegnazione delle risorse.
Il primo rappresenta le policy, ovvero si sostituiscono le risorse del paziente applicando algoritmi greedy basati sulle distanze. Sono stati sviluppati due di questi algoritmi, in entrambi non c'è una visione d'insieme delle necessità di tutti i pazienti ma ad ognuno viene assegnata la specialità necessaria dell'ospedale più vicino:

- al comune di residenza del paziente
- all'ospedale in cui il paziente era prenotato

Queste policy vengono usate richiamando la funzione *removed_id_check()* nel file *remove_resources.py* la quale controlla se il paziente passato abbia la sua risorsa eliminata, in questo caso richiama una delle due policy: **residenza-ospedale** o **ospedale-ospedale**. In base a quale viene scelta verrà trovato l'ospedale più vicino al paziente con la specialità di interesse e distanza calcolata di conseguenza.
Il secondo si propone di controllare se è possibile anticipare i ricoveri dei pazienti presenti N giorni successivi alla data del giro. Nel caso in cui la specialità sia già piena il paziente non viene messo in coda ma sarà gestito nella sua data naturale di ricovero. Il terzo è la riassegnazione delle risorse per tutti i pazienti che ne hanno bisogno per la settimana successiva, sarà approfondito nel capitolo 4.

3.1.1 Descrizione del Workflow

In un primo momento si è provato ad usare la libreria *Simpy* come base per la simulazione. Purtroppo per i nostri scopi era necessario avere una flessibilità maggiore nell'uso delle risorse, per questo motivo non è stato usato.
Il file da cui parte tutta la simulazione è *main_simulation.py*. Le prime operazioni dello script sono di caricamento/creazione dei dati che serviranno successivamente, nello specifico troviamo:
resources, patients = parser_data.load_data()
hosp_dict = parser_data.load_hosp_dict(resources)

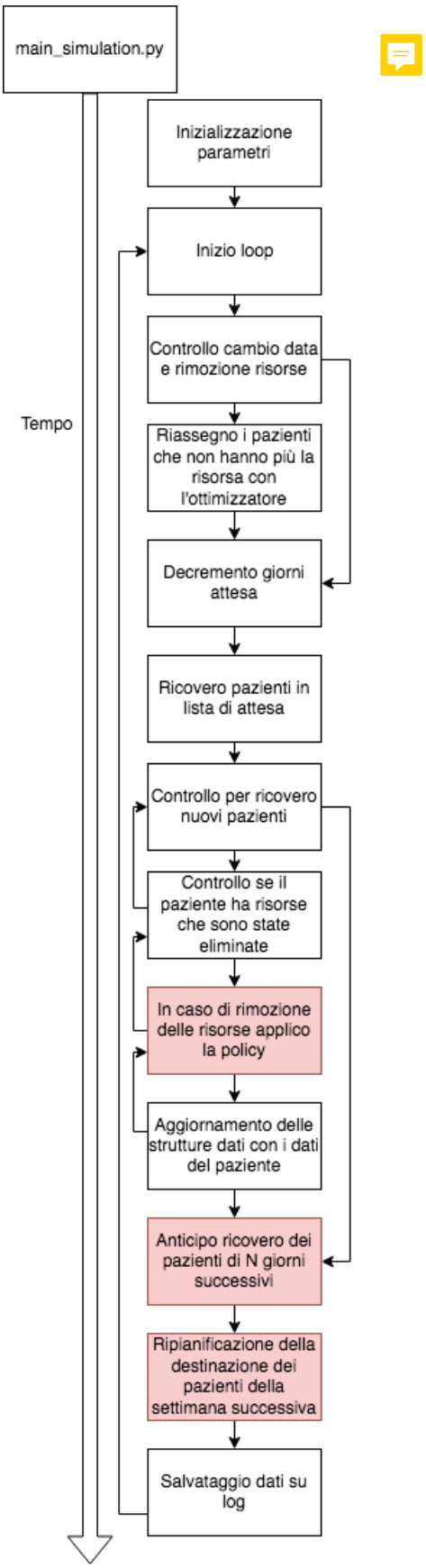

Figura 3.1: Diagramma simulazione

hosp_id_list, hosp_spec_list, date = remove_resources.read_input(file)
dict_mapping, dict_distances = parser_data.load_policy_data()

dict_map_residenza = parser_data.load_residenze()
Troviamo anche i parametri per l'ottimizzatore quali: il nome del solver da utilizzare e quanto tempo ha a disposizione lo stesso solver. Dopo questi dati viene chiamato il metodo *start_simulation()* con parametri i dati appena citati.
La simulazione lavora per "giorni" ovvero i cicli ed i calcoli delle varie policy e della simulazione stessa sono stati suddivisi in base ai dati giornalieri in nostro possesso. Prima dell'inizio di questo ciclo si trovano altre inizializzazioni ed altri parametri modificabili per poter scegliere casistiche diverse (l'elenco sarà nel prossimo paragrafo).

All'inizio del ciclo giornaliero ci sono due controlli: se è cambiato l'anno, se ci sono risorse da rimuovere; in entrambi i casi si aggiornano le liste di attesa, le capacità degli ospedali e nel caso in cui siano stati cancellati delle risorse, la funzione *optimization_reassing()* chiama l'ottimizzatore il quale riassegna una risorsa consona ai pazienti a cui era stata cancellata.
Dopo troviamo il controllo se ci sono delle risorse da rimuovere, funzione che viene sempre usata se l'ottimizzatore è attivo.
Da questo punto c'è una serie di controlli relativi alle code, in ordine: si decrementano i giorni di degenza di tutti i pazienti che occupano un posto letto e vengono rimossi quelli arrivati a 0, controllo se posso ricoverare persone nella lista di attesa ed infine controllo i nuovi pazienti; in quest'ultimo passo inizia un altro ciclo che scorre ogni singolo paziente della giornata. Per prima cosa vengono prelevate le informazioni del paziente e si controlla che l'ospedale o specialità non sia stata cancellata, in tal caso vuol dire che l'ottimizzatore non è attivo e quindi viene applicato un algoritmo greedy di riassegnazione. Dopo l'eventuale riassegnazione c'è il controllo sulle soglie (se richiesto) ed infine il controllo se la specialità del paziente nel dato ospedale ha ancora spazio, in caso negativo verrà messo in coda e saranno salvate le informazioni utili per le statistiche finali.
Terminato il ciclo di ogni paziente della giornata c'è il controllo (se richiesto) di anticipare l'ingresso dei pazienti di N giorni avanti. Nella pratica cicla su ogni giorno futuro stabilito, e per ogni giorno cicla su tutti i pazienti e controlla se le specialità in cui devono andare hanno ancora capacità di accoglierli.
Come ultimo controllo, se è uno specifico giorno della settimana (nei test si è usata la domenica) ed è richiesto, si attiva l'ottimizzatore per i pazienti della settimana successiva.
Come ultimi passi del ciclo giornaliero c'è il salvataggio dei log delle varie parti del simulatore.

3.2 Descrizione algoritmo Pyomo

Descriviamo nello specifico come funziona e com'è strutturato l'ottimizzatore in Pyomo.
La funzione da richiamare è *optimization_reassing()* nel file *reassing_hospital.py* la quale prende in input la lista dei pazienti divisi per giorni e estrapola solo quelli presenti nei successivi 7 giorni e ritorna tali giorni con le destinazioni dei pazienti modificate.
La funzione per prima cosa estrae dai 7 giorni successivi solo i pazienti che devono avere la loro destinazione aggiornata, dopodiché vengono suddivisi in gruppi in base alla specialità che devono avere. A questo punto inizia il ciclo in cui, per ogni specialità trovata, vengono costruite le strutture necessarie all'ottimizzatore, nello specifico avremo:

- dizionario id_paziente:giorni_degenza
- dizionario None:lista id_pazienti
- dizionario None:lista id_ospedale (divisi per specialità)
- dizionario id_paziente:distanza_vecchio_ospedale

- dizionario (id_paz,id_hosp):distanza ; in parole viene preso ogni paziente della specialità considerata per ogni ospedale con quella specialità

dopodiché viene richiamato il file con il modello. I file sono *optimization_model_delta.py* o *optimization_model_sum.py* e la scelta avviene all'inizio di *start_simulation()* impostando il flag *mod*.
Se il risultato del modello è 'infeasible' allora inizia un ciclo che incrementa la costante *alfa* ad ogni iterazione, tale costante è >=0 e serve per garantire un margine operativo (per esempio quando alfa=0 impongo che tutti i carichi di lavoro siano identici).
Terminato il ciclo esterno vengono modificati i pazienti nella lista con le nuove destinazione e viene restituita. L'esecuzione e la struttura dei modelli implementati è riportato al paragrafo 2.3 a pagina 4

3.3 Simple How-To: Come eseguire il software

Descriviamo ora l'ordine con cui eseguire da zero tutta la simulazione. Useremo i dati di partenza descritti nel Capitolo 1. Per usarli ci serve creare i file dati rielaborati, per questo dobbiamo eseguire gli script *parser_ospedale.py* e *parser_pazienti.py*.
A questo punto si può lanciare la simulazione dallo script *main_simulation.py*. I file necessari si trovano tutti nella cartella Dati_Elaborati, nel caso in cui mancassero verranno creati in automatico dai metodi.
Il risultato della simulazione viene salvato in tre file txt chiamati: *log_day_NOME.txt*, *queue_info_NOME.txt*, *anticipated_queue_info_NOME.txt* con NOME il nome scelto ad inizio simulazione.
C'è poi un ulteriore file chiamato *queue_statistics.txt* il quale racchiude un riassunto di varie statistiche sui pazienti entrati in coda a partire dal file *queue_info_NOME.txt*, per crearlo bisogna lanciare lo script *statistics.py* mettendo nel campo *name* il nome della simulazione lanciata.

Capitolo 4

Risultati

4.1 Lista statistiche

In questo paragrafo elenchiamo quali sono i dati salvati da ogni esecuzione.
Alla fine di ogni ciclo giornaliero il programma salva le informazioni in tre file:

- **anticipated_queue_info_NOME.txt**: Salva le informazioni dei pazienti ricoverati in anticipo (se non è richiesto sarà un txt bianco)
- **log_day_NOME.txt**: Salva le informazioni della giornata degli ospedali
- **queue_info_NOME.txt**: Salva le informazioni della situazione code

Per tutti i file i dati sono salvati in riga separati da virgola, ogni nuovo inserimento è fatto andando a capo ad inizio riga. In *anticipated_queue_info_NOME.txt* i dati sono salvati nel seguente formato: id del paziente, data di ricovero, data dell'effettivo ricovero, id ospedale, id specialità, quanti giorni ha anticipato.
In *log_day_NOME.txt* i dati sono salvati come: numero del giorno di riferimento, id ospedale, id specialità, quanti ricoveri sono stati effettuati, quanti sono i letti occupati, quanto è lunga la coda di attesa.
In *queue_info_NOME.txt* i dati sono salvati come: id paziente, data del ricovero, data dell'effettivo ricovero, id ospedale, id specialità, motivazione per cui è finito in coda, numero con cui è entrato in coda, giorni di attesa prima di essere operato.
Per il log *queue_info_NOME.txt* inoltre si può lanciare lo script *statistics.py*, inserendo il NOME del log, il quale calcola delle statistiche generali di tutto il log. Nello specifico calcola: totale dei pazienti, totale giorni di attesa, media dei giorni di attesa, totale dei pazienti che hanno avuto come motivazione "day_max", totale dei pazienti che hanno avuto come motivazione "cap_max", totale dei pazienti che hanno avuto come motivazione "all_max", numero di pazienti entrati in coda ogni giorno della settimana (nel pratico c'è un array lungo 7 che indica i pazienti dal lunedì alla domenica).
Specifichiamo cosa sono le motivazioni. Per motivazione intendiamo il motivo per cui il paziente è entrato in coda. Con motivazione "day_max" vuol dire che il paziente è stato messo in coda di attesa perché si è sforata la capacità massima giornaliera della specialità, con "cap_max" si è sforata la capacità massima dei posti letto della specialità, mentre con "all_max" sono vere entrambe le precedenti motivazioni.

Le seguenti tabelle rappresentano alcuni risultati di prove con parametri variabili descritti nei precedenti capitoli. I dati rappresentano i pazienti entrati in coda e le varie informazioni su di essa, in tutte le prove il totale dei giorni è 1092 ed il totale dei pazienti è 1661908.
(g.a. = giorni di attesa)

Simulazione base senza modifiche, tempo esecuzione 29 min						
Tot pazienti	Tot g.a.	Media g.a.	day_max	cap_max	all_max	settimana
2828	13836	4.89	159	2542	127	[485, 505, 506, 475, 451, 192, 214]

Le prove seguenti sono con **giorni anticipati**.

spurious_day = 90, forward_days = 7, tempo esecuzione 44 min						
Tot pazienti	Tot g.a.	Media g.a.	day_max	cap_max	all_max	settimana
27	27	1	2	22	3	[2, 3, 10, 11, 1, 0, 0]

spurious_day = 90, forward_days = 3, tempo esecuzione 59 min						
Tot pazienti	Tot g.a.	Media g.a.	day_max	cap_max	all_max	settimana
118	965	8.17	14	99	5	[23, 25, 23, 23, 19, 1, 4]

spurious_day = 90, forward_days = 1, tempo esecuzione 46 min						
Tot pazienti	Tot g.a.	Media g.a.	day_max	cap_max	all_max	settimana
1185	6384	5.38	64	1088	33	[284, 190, 183, 155, 177, 89, 107]

Le percentuali indicano quante risorse(gli ospedali) si sono tolte dal totale. Si è partiti a levare gli ospedali che hanno avuto meno ricoveri fino a raggiungere la percentuale scelta. I file *lista_paz_ospedale.txt* e *lista_paz_specialita.txt* che si trovano nella sottocartella *Parametri*, indicano, rispettivamente per ogni ospedale o ospedale-specialità, il totale dei pazienti ricoverati secondo i dati di partenza. Questo ci è servito per poter scegliere in maniera consapevole quali e quante delle risorse andavano eliminate inserendole nel file *remove_info.txt*.
Le prove seguenti sono con la policy distanza **ospedale-ospedale**.

Percentuale rimossa 3%, tempo esecuzione 29 min						
Tot pazienti	Tot g.a.	Media g.a.	day_max	cap_max	all_max	settimana
46639	3674175	78.77	7367	36370	2902	[9488, 8276, 8656, 8121, 7667, 2907, 1524]

Percentuale rimossa 5%, tempo esecuzione 29 min						
Tot pazienti	Tot g.a.	Media g.a.	day_max	cap_max	all_max	settimana
45422	5565927	122.53	7642	34658	3122	[9119, 8048, 8352, 7982, 7333, 2875, 1713]

Percentuale rimossa 7%, tempo esecuzione 28 min						
Tot pazienti	Tot g.a.	Media g.a.	day_max	cap_max	all_max	settimana
62288	5762018	92.5	15231	43071	3986	[12208, 11017, 11289, 10769, 10373, 3875, 2757]

Percentuale rimossa 9%, tempo esecuzione 26 min						
Tot pazienti	Tot g.a.	Media g.a.	day_max	cap_max	all_max	settimana
86706	7756582	89.45	23135	57734	5837	[16408, 15904, 15511, 14999, 14882, 5049, 3953]

Le prove seguenti sono con la policy distanza **residenza-ospedale**. Le percentuali sono come per la policy precedente.

Percentuale rimossa 3%, tempo esecuzione 27 min						
Tot pazienti	Tot g.a.	Media g.a.	day_max	cap_max	all_max	settimana
45760	3799211	83.02	5276	36933	3551	[9182, 8265, 8417, 8097, 7696, 2736, 1367]

Percentuale rimossa 5%, tempo esecuzione 28 min						
Tot pazienti	Tot g.a.	Media g.a.	day_max	cap_max	all_max	settimana
45713	5331192	116.62	5870	35918	3925	[9233, 8219, 8290, 8175, 7517, 2693, 1586]

Percentuale rimossa 7%, tempo esecuzione 28 min						
Tot pazienti	Tot g.a.	Media g.a.	day_max	cap_max	all_max	settimana
55010	5346150	97.18	10538	40417	4055	[10995, 9875, 9977, 9819, 9326, 3006, 2012]

Percentuale rimossa 9%, tempo esecuzione 26 min						
Tot pazienti	Tot g.a.	Media g.a.	day_max	cap_max	all_max	settimana
82891	7167823	86.47	18123	58936	5832	[15732, 15071, 15079, 14998, 14511, 4366, 3134]

Le seguenti prove sono con l'**ottimizzatore delta**.

Percentuale rimossa 3%, tempo esecuzione 1629 min						
Tot pazienti	Tot g.a.	Media g.a.	day_max	cap_max	all_max	settimana
42503	1730614	40.71	5815	34204	2484	[8401, 7954, 8511, 7659, 7573, 1776, 629]

Le seguenti prove sono con l'**ottimizzatore somma**.

Percentuale rimossa 3%, tempo esecuzione 1824 min						
Tot pazienti	Tot g.a.	Media g.a.	day_max	cap_max	all_max	settimana
83441	6242302	74.81	11335	67264	4842	[16789, 15611, 16153, 15123, 14981, 3510, 1274]