UNIVERSITÀ DEGLI STUDI DI GENOVA

DIPARTIMENTO DI INFORMATICA, BIOINGEGNERIA, ROBOTICA E INGEGNERIA DEI SISTEMI

Ottimizzazione e Visualizzazione nella risoluzione del problema di schedulazione dei trattamenti chemioterapici

di

Daniele Yu

Tesi presentata per la Laurea Magistrale in *Ingegneria Informatica*

Prof. Marco Maratea	Relatore
Dott. Marco Mochi	Correlatore
Dott. Giuseppe Galatà	Correlatore

Ringraziamenti

Vorrei ringraziare la mia famiglia che mi ha sempre supportato e spinto a continuare gli studi, senza di loro non sarei mai arrivato dove sono ora, sono grato di tutti i sacrifici che hanno fatto per me.
Voglio ringraziare anche tutti i miei amici che mi sono sempre stati vicini in ogni momento.
Infine voglio ringraziare il prof. Marco Maratea, il dott. Marco Mochi e il dott. Giuseppe Galatà per avermi seguito e aiutato nello sviluppo della tesi e per la loro grande disponibilità.

Sommario

La Digital health (anche chiamata eHealth, o salute digitale in italiano) è la convergenza delle tecnologie digitali con i campi della salute e dell'assistenza sanitaria al fine di migliorare l'efficienza dell'erogazione delle cure sanitarie. In questa tesi vediamo come viene applicata la Digital health nell'ambito della pianificazione delle sedute chemioterapiche.
La pianificazione delle sedute chemioterapiche nelle cliniche oncologiche è un problema combinatorio complesso per via dei vincoli che deve rispettare la soluzione, come la disponibilità delle risorse, i diversi tipi di trattamenti e la loro durata.
In particolare, presentiamo una soluzione ai problemi di pianificazione utilizzando uno strumento di Intelligenza Artificiale basato sulla programmazione dichiarativa, l'Answer Set Programming (ASP). In seguito alla presentazione della nostra soluzione analizziamo i risultati ottenuti prendendo in considerazione diversi giorni di pianificazione.
I risultati ottenuti ci permettono di dire che i linguaggi di pianificazione di IA sono strumenti molto efficaci per affrontare il problema della pianificazione delle sedute chemioterapiche.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

In questo capitolo vedremo le motivazioni che rendono il problema della pianificazione delle sedute chemioterapiche un problema interessante e reso ancora più di attualità dalle problematiche legate all'avvento della pandemia da Covid-19.
Successivamente vedremo lo stato dell'arte, gli obbiettivi e la struttura della tesi.

1.1 Contesto e motivazioni

La Digital health (anche chiamata e-health, o salute digitale in italiano) consiste nell'uso di tecnologie informatiche e di telecomunicazione (ICT) a vantaggio della salute umana, secondo la definizione della World Health Organization (WHO), ossia l'Organizzazione Mondiale della Sanità (OMS). È un settore tecnologico molto concreto ed in forte crescita in cui l'applicazione efficace e mirata dell'innovazione tecnologia a vantaggio della salute umana è sfruttata per generare numerosi benefici in termini di nuove soluzioni salvavita, oppure di ausili personalizzati per migliorare significativamente la qualità della vita delle persone e del personale sanitario.
In questa tesi vediamo come viene applicata la Digital health nell'ambito della pianificazione delle sedute chemioterapiche con l'obbiettivo di ottenere una pianificazione ottimizzata che rispetta diversi vincoli.
Il problema della pianificazione delle sedute chemioterapiche consiste nel trovare una schedulazione dei pazienti che richiedono i trattamenti in diversi slot temporali lungo la giornata, rispettando diversi vincoli. È un problema complesso in quanto vi sono molti vincoli e aspetti da prendere in considerazione, come la disponibilità di diverse risorse, il tipo di terapia e la sua durata.

I trattamenti chemioterapici sono ciclici, e le tempistiche e il numero dei cicli dipende dal tipo di cancro, dallo stadio della malattia e dalla risposta del paziente ai trattamenti. I vari trattamenti hanno priorità diverse, che vengono prese in considerazione quando si effettua la pianificazione.
Visto tutto ciò risulta di vitale importanza per una clinica riuscire ad ottenere una pianificazione ottimale, per aumentare il livello di soddisfazione dei pazienti e degli infermieri, e sfruttare al meglio le risorse a disposizione. Inoltre, le tempistiche sono molto importanti anche per le cure dei pazienti, infatti molti studi hanno evidenziato come i ritardi nei trattamenti abbiano un effetto negativo, effetto che varia a seconda dell'aggressività e del tipo di cancro; da qui l'importanza di avere una soluzione che riesca a gestire le diverse priorità dei trattamenti rispettando i diversi vincoli.
A causa delle problematiche relative al Covid-19, è ancora più importante lo sfruttamento ottimale delle risorse messe a disposizione e la diminuzione dei tempi di attesa e di permanenza nei vari reparti dell'ospedale.

1.2 Stato dell'arte

Il problema della pianificazione delle sedute chemioterapiche è un problema che è stato affrontato utilizzando diverse tecniche.
Nella letteratura sono presenti diversi articoli che affrontano il problema tramite vari metodi. Per citarne alcuni, abbiamo Huggins et al. (2014) dove è stata utilizzata la programmazione a numeri interi ottimizzando l'utilizzo delle risorse, Turkcan et al. (2010) dove sono stati utilizzati metodi euristici per risolvere il problema, Sevinc et al. (2013) dove sono state utilizzate tecniche a 2 fasi e Amendola (2018) dove viene affrontato il problema chiamato SRP (Stable Roommates Problem). Tuttavia sono poco sfruttate soluzioni che utilizzano tecniche di Intelligenza Artificiale, ad esempio soluzioni basate sulla programmazione dichiarativa, dove l'utente specifica le proprietà che deve avere la soluzione, e solutori software si occupano di computare tale soluzione. Queste soluzioni sono particolarmente indicate per affrontare problemi combinatori complessi, come quello da noi affrontato.
Uno degli strumenti utilizzabili è l'Answer Set Programming (ASP) (Gelfond and Lifschitz (1991), Faber et al. (2011)), che è stato già utilizzato per problemi di schedulazione in diversi campi di ricerca. Per citare alcuni articoli che affrontano il problema della pianificazione utilizzando l'ASP, abbiamo:

- Alviano et al. (2017, 2018); Dodaro and Maratea (2017) dove viene affrontato il problema della pianificazione e ripianificazione dei turni degli infermieri;

- Gebser et al. (2018) dove viene affrontato il problema della pianificazione del percorso dei veicoli a guida autonoma per affrontare le attività di trasporto in maniera efficiente
- Ricca et al. (2012) dove viene affrontato il problema dell'organizzazione del personale di un porto volto ad affrontare il lavoro richiesto dalle navi in arrivo.
- Alviano et al. (2020) dove vengono affrontati diversi problemi in ambito Digital Health e presenta soluzioni che sono affrontati con l'ASP;
- Cardellini et al. (2021) dove viene proposta una soluzione in ASP del problema della pianificazione delle sessioni di fisioterapia riabilitativa per i pazienti;
- Dodaro et al. (2019) dove viene proposta una soluzione in ASP del problema relativo all'ottimizzazione della pianificazione giornaliera degli interventi chirurgici in sala operatoria.

Inoltre, esiste già una soluzione presentata nell'articolo Dodaro et al. (2021) che utilizza ASP per pianificare le sedute chemioterapiche. La soluzione di tale articolo verrà approfondita nel Capitolo 2 e corrisponde alla versione base del problema affrontato.

1.3 Obbiettivi della tesi

Nell'articolo Dodaro et al. (2021) viene presentata la soluzione relativa alla pianificazione delle sedute chemioterapiche. Lo scopo della tesi consiste nella creazione di una nuova soluzione (versione estesa) a partire dalla soluzione esistente (versione base). Per ottenere la versione estesa del problema abbiamo collezionato delle nuove specifiche e relativi dati dall'Ospedale Policlinico San Martino di Genova.
Gli obbiettivi della tesi sono i seguenti:

- studiare ed estendere la versione base del problema affrontato;
- effettuare una valutazione sperimentale del codificatore esteso e confrontarlo con il codificatore base;
- creare un'applicazione web che incorpora il codificatore esteso e ne permetta quindi il suo utilizzo tramite un'interfaccia grafica.

1.4 Struttura della tesi

Dopo questo primo capitolo introduttivo, nel Capitolo 2 vedremo la versione base del problema come già risolto nell'articolo Dodaro et al. (2021). Si partirà con una descrizione del problema nel mondo reale, poi, dopo aver presentato il problema più nel dettaglio, vedremo la codifica del problema in linguaggio ASP.
Successivamente, nel capitolo 3, presenteremo la versione estesa del problema affrontato. In particolare, vedremo i requisiti addizionali che sono stati collezionati dall'ospedale per rendere il problema più reale e infine vedremo la codifica in ASP del problema esteso.
Nel capitolo 4 eseguiremo un'analisi dei risultati ottenuti con il nuovo codificatore, mostrando diversi dati come l'utilizzo delle risorse, il numero di preferenze rispettate e la distribuzione dei pazienti lungo le giornate di pianificazione. Infine verranno confrontati i risultati del nuovo codificatore con quello base.
A seguire, nel capitolo 5, presenteremo l'applicazione web. Inizialmente vedremo la sua architettura e successivamente vedremo una demo affiancata da commenti e spiegazioni di tutte le interazioni tra il frontend e il backend.
Infine vedremo i lavori correlati, i lavori futuri e la conclusione della tesi.

Capitolo 2

Problema affrontato: versione base

In questa sezione si presenta la soluzione della versione base del problema (risolto nell'articolo Dodaro et al. (2021) che, oltre a trattare il problema di pianificazione giornaliero che presentiamo in questo capitolo, tratta altre due versioni del problema). Tale soluzione permette di assegnare i pazienti che devono essere pianificati in un determinato giorno nei vari slots temporali lungo la giornata.

2.1 Descrizione del problema

Il problema della pianificazione delle sedute chemioterapiche consiste nella schedulazione degli appuntamenti dei pazienti che lo richiedono in un determinato giorno. In particolare viene assegnato ad ogni paziente uno slot temporale relativo a un giorno fissato e una risorsa, che può essere una sedia o un letto. Più nello specifico, per ogni paziente, possiamo individuare quattro fasi:

1. registrazione del paziente
2. prelievo del sangue
3. visita medica
4. terapia

Ora andiamo a vedere i vincoli temporali che devono essere rispettati. Le fasi 1 e 4 sono obbligatorie, mentre le fasi 2 e 3 sono opzionali. Tuttavia, se un paziente fa la fase 2 allora la fase 3 diventa obbligatoria. Il numero delle fasi e la loro durata vengono definite dal percorso del paziente, che è noto al momento della pianificazione. Inoltre, alcuni pazienti potrebbero

non aver bisogno di un letto o una sedia, e in tali casi la durata della fase 4 è 0.
Gli orari dell'ospedale sono dalle 07:30 alle 13:30. Tale arco di tempo viene diviso in 72 slots temporali per ogni giorno con una durata di 5 minuti ciascuno, dove il primo slot temporale è 07:30 - 07:35, mentre l'ultimo slot è 13:25 - 13:30. Inoltre identifichiamo un insieme di 36 slots temporali disponibili dove la fase 4 può iniziare, dove 07:35 - 07:40 è il primo slot temporale, 07:45 - 07:50 è il secondo slot, e così via. Se la durata della fase 4 relativa a un paziente supera una certa durata, allora tale fase deve iniziare dopo le 11:25, ovvero dallo slot temporale 24 in poi. La fase 4 può iniziare solo se tutte le fasi precedenti sono state completate. Inoltre, la fase 1 deve iniziare dopo il primo slot temporale disponibile, questo significa che la differenza tra la fase 4 e la somma delle fasi precedenti deve essere maggiore del primo slot temporale.
L'input del problema consiste in registrazioni, dove ogni registrazione include la durata di ognuna delle 4 fasi (se una fase non è richiesta la sua durata viene impostata a 0) e la preferenza della risorsa, ovvero letto o sedia.
L'output del problema, ovvero la soluzione, consiste invece in assegnazioni, dove ogni assegnazione include lo slot temporale assegnato (che corrisponde all'inizio della fase 4), rispettando i seguenti requisiti:

- ogni paziente deve essere assegnato ad un letto o una sedia *(i)*
- ogni risorsa può essere usata solo da un paziente per ogni slot temporale *(ii)*
- se la fase 4 richiede più di uno slot temporale, allora il paziente deve usare sempre la stessa risorsa *(iii)*

Inoltre, una soluzione ottimale del problema massimizza il numero dei pazienti dove la preferenza sulla risorsa è stata rispettata e minimizza il numero dei pazienti che iniziano contemporaneamente la fase 2 in modo da minimizzare l'affollamento e per avere un uso più uniforme delle risorse durante la giornata. Per poter minimizzare il numero dei pazienti che iniziano contemporaneamente la fase 2 si minimizza il numero massimo di pazienti che iniziano la fase 2 contemporaneamente e, con una priorità minore, la differenza tra il numero massimo e il numero minino di pazienti che iniziano la fase 2.

2.2 Linguaggio ASP

L'Answer Set Programming (Gelfond and Lifschitz (1988), Gelfond and Lifschitz (1991)) è una forma di programmazione logica di tipo dichiarativo utilizzato per problemi di ri-

cerca complessi, basata sulla semantica dei modelli statici (o answer set). Per maggiori informazioni riguardanti ASP visitare https://potassco.org/ e l'articolo Calimeri et al. (2019).

2.2.1 Sintassi

La sintassi di ASP è simile alla sintassi di Prolog.
Un *atomo standard* è un'espressione $p(t_1,\ldots,t_n)$, dove p è un *predicato* con n argomenti e $t_1,\ldots,t_n$ sono i termini. Le variabili sono stringhe con la prima lettera maiuscola e le costanti sono interi non negativi o stringhe con la prima lettera minuscola. Un *termine* è una variabile o una costante. Un atomo $p(t_1,\ldots,t_n)$ è ground se $t_1,\ldots,t_n$ sono costanti. Un *set ground* è un set di coppie nella forma $\langle consts : conj\rangle$, dove $consts$ è una lista di costanti e $conj$ è una congiunzione di atomi ground. Un *set di simboli* è un set definito sintatticamente come $\{Terms_1 : Conj_1;\cdots;Terms_t : Conj_t\}$, dove $t > 0$, e per tutte le $i \in [1,t]$, ogni $Terms_i$ è una lista di termini tale che $|Terms_i| = k > 0$, e ogni $Conj_i$ è una congiunzione di atomi standard. Un *set di termini* è o un ground set o un set di simboli. Intuitivamente, un set di termini $\{X : a(X,c), p(X); Y : b(Y,m)\}$ rappresenta l'unione di due set: il primo contiene i valori X che rendono la congiunzione $a(X,c), p(X)$ vera, e il secondo contiene i valori Y che rendono la congiunzione $b(Y,m)$ vera. Una *funzione aggregato* è della forma $f(S)$, dove S è un set di termini, e f è un *simbolo di una funzione aggregato*.
Le funzioni aggregato mappano vari set di costanti su una costante. Le più comuni funzioni implementante nei sistemi ASP sono:

- *#count*, numero di termini.
- *#sum*, somma di interi.
- *#max*, valore massimo tra interi.

Un *atomo aggregato* è della forma $f(S) \prec T$, dove $f(S)$ è una funzione aggregato, $\prec \in \{<, \leq, >, \geq, \neq, =\}$ è un operatore di confronto, e T è un termine chimato guardia. Un atomo aggregato $f(S) \prec T$ è ground se T è una costante e S è un set ground. Un *atomo* è un atomo standard o un atomo aggregato. Una *regola r* ha la seguente struttura:

$$a_1 \vee \ldots \vee a_n \text{ :- } b_1,\ldots,b_k,not\ b_{k+1},\ldots,not\ b_m.$$

dove $a_1,\ldots,a_n$ sono atomi standard, $b_1,\ldots,b_k$ sono atomi, $b_{k+1},\ldots,b_m$ sono atomi standard, e $n,k,m \geq 0$. Un literal è o un atomo standard a o la sua negazione *not a*. La disgiunzione $a_1 \vee \ldots \vee a_n$ è la *testa* di r, mentre la congiunzione $b_1,\ldots,b_k,not\ b_{k+1},\ldots,not\ b_m$ è il suo

corpo. Regole senza corpo sono dette *fatti*. Regole senza testa sono dette *constraints*. Una variabile che compare solo nei termini di una regola r è detta *locale* in r, altrimenti è una variabile *globale* di r. Un programma ASP è un insieme di regole *safe*, dove una regola r è *safe* se rispetta le seguenti condizioni: *(i)* per ogni variabile globale X di r c'è un atomo standard ℓ nel corpo di r tale che X appare in ℓ; e *(ii)* ogni variabile locale di r che appare in un set di simboli $\{Terms : Conj\}$ appare anche in un atomo positivo in $Conj$.
Un *weak constraint* ω (Buccafurri et al. (2000)) è nella forma:

$$:\sim b_1, \ldots, b_k, not\ b_{k+1}, \ldots, not\ b_m.\ [w@l]$$

dove w e l sono rispettivamente il peso e la priorità di ω. Un programma ASP con weak costraints è $\Pi = \langle P, W \rangle$, dove P è un programma e W è un set di weak costraints. Un atomo standard, un literal, una regola, un programma o un weak costraints è *ground* se non vi appaiono variabili in esso.

2.2.2 Semantica

Sia P un programma ASP. L' *Herbrand universe* U_P e la *Herbrand base* B_P di P sono definite come abitualmente. L'istanza ground G_P di P è l'insieme di tutte le istanze ground delle regole di P che possono essere ottenute sostituendo le variabili con le costanti da U_P. Un' *interpretazione* I per P è il sottoinsieme I di B_P. Un literal ground ℓ (risp., *not* ℓ) è vero rispetto ad I se $\ell \in I$ (risp., $\ell \notin I$), e falso (risp., vero) altrimenti.
Un atomo aggregato è vero rispetto ad I se il valore della funzione aggregato (i.e., il risultato dell'applicazione di f negli insiemi S) rispetto ad I rispetta la guardia; altrimenti è falsa.
Una regola ground r è *soddisfatta* da I se almeno un atomo nella testa è vero rispetto ad I mentre tutte le congiunzioni del corpo di r sono vere rispetto ad I.
Un modello è un'interpretazione che risolve tutte le regole di un programma. Dato un programma ground G_P e un'interpretazione I, la *ridotta* Faber et al. (2011) di G_P rispetto ad I è il sottoinsieme G_P^I di G_P ottenuto eliminando da G_P le regole in cui un literl nel corpo è falso rispetto ad I.
Un'interpretazione I per P è un *answer set* (o modello stabile) per P se I è un modello minimo di G_P^I (i.e., I è un modello minimo G_P^I) Faber et al. (2011).
Dato un programma con weak costraints $\Pi = \langle P, W \rangle$, la semantica di Π amplia quella del caso base spiegato precedentemente. Sia $G_\Pi = \langle G_P, G_W \rangle$ l'istanza di Π; un constraint $\omega \in G_W$ è violato da un'intepretazione di I se tutti i literal in ω sono veri rispetto ad I. Un *optimum*

answer set per Π è un answer set di G_P che minimizza la somma dei pesi dei weak constraints violati in G_W con ordine prioritario.

Abbreviazione sintattica

Successivamente utilizzeremo anche le *choice rules* nella forma $\{p\}$, dove p è un atomo. Le choice rules possono essere interpretate come un'abbreviazione sintattica per la regola $p \vee p'$, dove p' è un atomo nuovo che non appare in altre parti del programma, questo significa che p può essere scelto come vero.
Nel nostro modello utilizziamo #minimize$\{p : x(p)\}$, la utilizziamo per via della sua maggiore leggibilità e serve per minimizzare la somma di tutti i valori p che rendono vera x(p), è quindi interpretabile come un weak constraint. Come le weak costraints si può assegnare a #minimize dei livelli diversi.

2.3 Metodologia di risoluzione

In questa sezione presentiamo la metodologia di risoluzione.

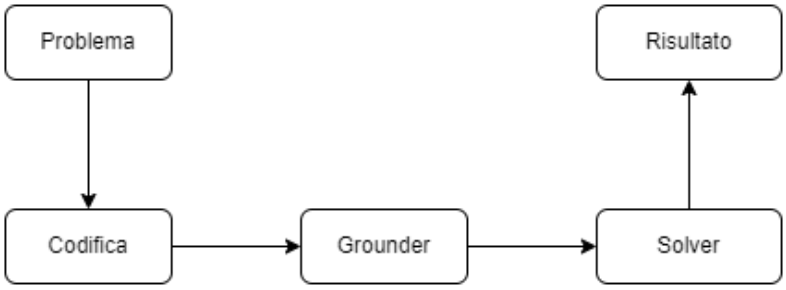


Figura 2.1 Metodologia di risoluzione

Nella figura 2.1 viene presentata la metodologia di risoluzione. In particolare, possiamo vedere i seguenti componenti:

- problema: è il problema reale da cui si parte;
- codifica: il problema reale viene modellato tramite un programma logico, ottenendo la codifica del problema in linguaggio ASP;
- grounder: il grounder esegue la fase di grounding, che consiste nel sostituire le variabili con costanti in tutti i modi possibili;

- solver: la fase di solving consiste nella ricerca della soluzione, chiamata answer set, ovvero un insieme di atomi che rispettano le regole del codificatore ASP. Possono esserci più soluzioni con ottimizzazioni diverse, a noi interessa la soluzione più ottimizzata che corrisponde all'ultima soluzione che il solver riesce a trovare;
- risultato: è il risultato, ovvero l'answer set.

Negli ultimi anni la diffusione e l'efficacia dei solver per problemi ASP è incrementata anche grazie alle diverse competizioni che sono state organizzate negli anni (Calimeri et al. (2014); Gebser et al. (2017a,b, 2020)). Esempi di solver efficienti sono Clingo (Gebser et al. (2012)) e Wasp (Alviano et al. (2019)).

2.4 Codifica ASP

In questa sezione vediamo come il problema viene codificato tramite il linguaggio ASP. Prima vedremo come viene modellato l'input, a seguire vedremo la codifica dell'output e infine verrà presentata la codifica del modello che gestisce il problema.

2.4.1 Input

Gli atomi che vengono usati come input sono i seguenti:

- reg(RID, DAY, DRUG, DURATION, VIS, COL, ACC, C): le sue istanze rappresentano le registrazioni che sono caratterizzate da un id che identifica il paziente (RID), il giorno della terapia (DAY), la terapia (DRUG), la durata della fase 4 (DURATION), la durata della fase 3, ovvero la visita (VIS), la durata della fase 2, ovvero il prelievo (COL), la durata della fase 1, ovvero l'accettazione (ACC) e infine la preferenza della risorsa (C);
- chair(ID): le sue istanze rappresentano le sedie disponibili, identificate da un id (ID);
- bed (ID): le sue istanze rappresentano i letti disponibili, identificate da un id (ID);
- mss(DAY, TS): le sue istanze rappresentano gli slots temporali disponibili (TS) relativo a un determinato giorno (DAY).

2.4.2 Output

L'output del problema, come già accennato in precedenza, consiste in assegnazioni. In particolare viene utilizzato il seguente atomo:

x(RID, DAY, TS, DRUG, DURATION, C)

tale atomo ci dice che la fase 4, di durata DURATION, relativa alla registrazione con identificativo RID, è assegnata allo slot temporale TS di un determinato giorno DAY facendo una determinata terapia DRUG e avendo una determinata preferenza C.

2.4.3 Encoding

Adesso vediamo le regole in linguaggio ASP. Di seguito viene presentato il codificatore completo:

```
1  1 {x(RID,DAY,TS,DRUG,DURATION,C) : mss(DAY,TS),TS\2 = 1} 1 :- reg(RID,_,
       DRUG,DURATION,VIS,COL,ACC,C).
2  :- x(RID,_,TS,_,DURATION,_),DURATION>50,TS<24.
3  :- x(RID,_,TS,_,_,_),reg(RID,_,_,_,VIS,COL,ACC,C),TS-COL-VIS-ACC<1.
4  res(RID,DAY,TS..TS+DURATION-1,C) :- x(RID,DAY,TS,_,DURATION,C),DURATION
       >0.
5  1 {bed(ID,RID,DAY) : bed(ID); chair(ID,RID,DAY) : chair(ID)} 1 :- x(RID,
       DAY,_,_,_,_,_).
6  chair(ID,RID,DAY,TS) :- chair(ID,RID,DAY),res(RID,DAY,TS,_).
7  bed(ID,RID,DAY,TS) :- bed(ID,RID,DAY),res(RID,DAY, TS,_).
8  :- #count{RID : chair(ID,RID,DAY,TS)} > 1, chair(ID), mss(DAY,TS).
9  :- #count{RID : bed(ID,RID,DAY,TS)} > 1, bed(ID), mss(DAY,TS).
10 supporto(RID,DAY,TS) :- x(RID,DAY,START,_,_,_),reg(RID,_,_,_,VIS,COL,_,_
       ),COL>0,START-VIS-COL=TS,mss(_,TS).
11 numbRegP(DAY,N,TS) :- N = #count{RID: supporto(RID,DAY,TS)},mss(DAY,TS).
12 numbDay(DAY,N) :- N = #count{RID: supporto(RID,DAY,_)}, mss(DAY,_).
13 numMaxP(DAY,T) :- T = #max{N: numbRegP(DAY,N,_)}, mss(DAY,_).
14 numMinP(DAY,T) :- T = #min{N: numbRegP(DAY,N,_)}, mss(DAY,_).
15 :~ x(RID,DAY,_,_,_,1), chair(_,RID,DAY). [1@4,RID]
16 :~ x(RID,DAY,_,_,_,0), bed(_,RID,DAY). [1@4, RID]
17 :~ numMaxP(DAY,M). [M@3,DAY]
18 :~ numMaxP(DAY,MAX), numMinP(DAY,MIN). [MAX-MIN@2, DAY]
```

Andiamo a vedere nel dettaglio il significato di queste regole del codificatore.
Di seguito viene riportata la prima regola:

```
1 {x(RID,DAY,TS,DRUG,DURATION,C) : mss(DAY,TS),TS\2 = 1} 1 :- reg(RID,_,
    DRUG,DURATION,VIS,COL,ACC,C).
```

Per ogni registrazione si prova ad assegnare uno slot temporale TS nel giorno DAY indicato, rappresentante l'orario di inizio terapia, tramite la metodologia Check and Guess. Una volta scelto un orario viene formato l'atomo x, che rappresenta l'assegnazione del paziente, con le informazioni presenti in reg, ovvero l'id del paziente, la terapia, la durata della terapia e la preferenza della risorsa, più la nuova informazione riguardante l'orario assegnato.

La seconda regola è un vincolo forte e dice che non è possibile assegnare un paziente ad uno slot temporale TS prima del ventiquattresimo slot se la durata della fase 4 è maggiore di 50 slots temporali:

```
:- x(RID,_,TS,_,DURATION,_),DURATION>50,TS<24.
```

Infatti, come visto precedentemente, uno dei vincoli della pianificazione è che se la durata della fase 4 relativa ad un paziente supera una certa durata, ovvero 50 slots temporali che corrispondono a 250 minuti, allora tale fase deve iniziare dopo le 11:25. Ricordando che la fase 4 può iniziare in uno slot temporale appartenente all'insieme composto da 36 slots temporali, dove 07:35 - 07:40 è il primo slot temporale, 07:45 - 07:50 è il secondo slot, e così via, in tale insieme il ventiquattresimo slot temporale corrisponde alla fascia oraria 11:25 - 11:30.

Ora vediamo la terza regola, anch'essa un vincolo forte:

```
:- x(RID,_,TS,_,_,_),reg(RID,_,_,_,VIS,COL,ACC,C),TS-COL-VIS-ACC<1.
```

Questo vincolo ci dice che non è possibile avere un'assegnazione dove lo slot temporale TS assegnato al paziente sia prima del completamento delle fasi precedenti. In particolare, dal momento che si assegna nella prima regola l'orario di inizio trattamento, si deve controllare che tutte le fasi precedenti siano effettuate non prima del primo TS disponibile. Si sottrae dunque la durata delle fasi precedenti all'orario di inizio del trattamento e si impone che sia maggiore o uguale ad 1.

La quarta regola genera un atomo ausiliario res che verrà utilizzato dalle regole successive e rappresenta intuitivamente l'utilizzazione di una risorsa in termini temporali:

```
res(RID,DAY,TS..TS+DURATION-1,C) :- x(RID,DAY,TS,_,DURATION,C),DURATION
    >0.
```

Questa regola ci dice che per ogni assegnazione x dove la durata della terapia è non nulla (quindi il paziente utilizza una risorsa) vengono generati tanti atomi res quanti sono gli slots temporali nelle quali il paziente occupa una determinata risorsa durante la terapia.

La quinta regola serve per assegnare una risorsa ad un paziente:

```
1 {bed(ID,RID,DAY) : bed(ID); chair(ID,RID,DAY) : chair(ID)} 1 :- x(RID,
   DAY,_,_,_,_,_).
```

Tale regola ci dice che per ogni assegnazione x viene assegnata una sola risorsa tra tutte le sedie e i letti disponibili.

La sesta regola genera un'altro atomo ausiliario e rappresenta intuitivamente l'occupazione della sedia in termini temporali:

```
chair(ID,RID,DAY,TS) :- chair(ID,RID,DAY),res(RID,DAY,TS,_).
```

Per ogni slot temporale occupato da un determinato paziente durante la terapia e per ogni sedia assegnata allo stesso paziente, viene generato il nuovo atomo chair che oltre a dirci l'id della risorsa, l'id del paziente e il giorno della terapia, ci dice anche in quale slot temporale tale risorsa viene utilizzata. In particolare, vengono utilizzati l'atomo ausiliario res e l'atomo chair per creare il nuovo atomo che indica che la sedia ID è utilizzata dal paziente RID nel giorno DAY e nello slot temporale TS.

La settima regola è analoga alla regola 6 ma con il letto:

```
bed(ID,RID,DAY,TS) :- bed(ID,RID,DAY),res(RID,DAY, TS,_).
```

Infatti vengono utilizzati l'atomo ausiliario res e l'atomo bed per creare il nuovo atomo che indica che il letto ID è utilizzato dal paziente RID nel giorno DAY e nello slot temporale TS.

La regola 8 è un vincolo forte e utilizza la variabile ausiliaria chair generata nella regola 6:

```
:- #count{RID : chair(ID,RID,DAY,TS)} > 1, chair(ID), mss(DAY,TS).
```

Tale vincolo ci dice che non è possibile che il numero di sedie assegnate ad un determinato paziente con identificativo RID sia maggiore di 1, quindi grazie a questa regola, ad ogni paziente viene assegnato al più una sedia.

La regola 9 è un vincolo analogo alla regola 8, ma con il letto:

```
:- #count{RID : bed(ID,RID,DAY,TS)} > 1, bed(ID), mss(DAY,TS).
```

Questo vincolo limita il numero di sedie assegnate ad un determinato paziente a 1.

La regola 10 genera un atomo ausiliario che ci dice in quale slot temporale un paziente inizia il prelievo:

```
supporto(RID,DAY,TS) :- x(RID,DAY,START,_,_,_),reg(RID,_,_,_,VIS,COL,_,
_),COL>0,START-VIS-COL=TS,mss(_,TS).
```

Viene creato l'atomo supporto contenente le informazioni riguardo lo slot temporale TS in cui il paziente RID inizia la fase 2 (prelievo del sangue). In particolare, per ogni paziente che richiede il prelievo, e quindi con durata della fase 2 maggiore di 0, viene ricavato lo slot temporale (TS) di inizio fase 2 sottraendo all'orario di inizio trattamento (START) la durata della fase 3 (VIS) e la durata della fase 2 (COL).

Le regole dalla 11 alla 14, che vediamo di seguito, generano atomi ausiliari che vengono utilizzati nelle regole finali che si occupano dell'ottimizzazione della soluzione.
La regola 11 crea l'atomo numbRegP che rappresenta il numero delle persone che iniziano il prelievo contemporaneamente (rappresentato da N) in un determinato slot temporale:

```
numbRegP(DAY,N,TS) :- N = #count{RID: supporto(RID,DAY,TS)},mss(DAY,TS).
```

Si utilizza l'atomo ausiliario supporto per ottenere il numero di pazienti che iniziano la fase 2 nel time slot TS e nel giorno DAY.

La regola 12 invece crea l'atomo numbDay che rappresenta intuitivamente il numero totale di persone che fanno il prelievo in un determinato giorno:

```
numbDay(DAY,N) :- N = #count{RID: supporto(RID,DAY,_)}, mss(DAY,_).
```

Si utilizza l'atomo ausiliario supporto per ottenere il numero di pazienti che richiedono la fase 2 in ogni giorno DAY.

Ora vediamo la regola 13:

```
numMaxP(DAY,T) :- T = #max{N: numbRegP(DAY,N,_)}, mss(DAY,_).
```

Tale regola crea l'atomo numMaxP che indica il numero massimo T di persone che iniziano il prelievo contemporaneamente in un determinato giorno DAY.

La regola 14 invece è il duale della 13:

```
numMinP(DAY,T) :- T = #min{N: numbRegP(DAY,N,_)}, mss(DAY,_).
```

Ovvero ci dice il numero minimo T di persone che iniziano il prelievo contemporaneamente in un determinato giorno DAY.

Ora vediamo le regole utilizzate per ottimizzare la soluzione, ovvero le ultime 4. Intuitivamente, le regole 15 e 16 servono per minimizzare il numero di assegnazioni dove non sono state rispettate le preferenze relative a una risorsa, mentre le regole 17 e 18 servono per distribuire al meglio le persone che iniziano il prelievo contemporaneamente durante la giornata per evitare affollamenti. Vediamo ora nel dettaglio queste 4 regole.
Iniziamo dalla regola 15:

```
:~ x(RID,DAY,_,_,_,1), chair(_,RID,DAY). [1@4,RID]
```

Tale regola minimizza il numero delle preferenze non rispettate minimizzando il numero di assegnazioni alle sedie ai pazienti che avevano come preferenza il letto.

La regola 16 è il duale della regola 15:

```
:~ x(RID,DAY,_,_,_,0), bed(_,RID,DAY). [1@4, RID]
```

Questa regola minimizza il numero delle preferenze non rispettate minimizzando il numero di assegnazioni ai letti ai pazienti che avevano come preferenza la sedia.

Ora vediamo la regola 17:

```
:~ numMaxP(DAY,M). [M@3,DAY]
```

Questa regola minimizza il numero massimo di persone che iniziano la fase 2 contemporaneamente.

Infine vediamo la regola 18:

```
:~ numMaxP(DAY,MAX), numMinP(DAY,MIN). [MAX-MIN@2, DAY]
```

Tale regola minimizza la differenza tra il numero massimo e minimo di persone che iniziano la fase 2 contemporaneamente.
Nel capitolo successivo vedremo come è stata estesa la versione base del problema appena visto, in particolare vedremo i nuovi vincoli e il nuovo codificatore.

Capitolo 3

Problema affrontato: versione estesa

In questa sezione si presenta la soluzione della versione estesa del problema. Tale soluzione è stata creata a partire dalla soluzione della versione base vista nel capitolo precedente, adattando il codificatore base alle nuove specifiche dell'ospedale per rendere il problema più reale. Di seguito vedremo prima una descrizione a parole dei requisiti addizionali e successivamente vedremo come è stata modificata la soluzione base.

3.1 Descrizione delle nuove specifiche

Le nuove specifiche che vengono mostrate in questa sezione contengono nuove informazioni, come il percorso di un paziente e la sua criticità, che verranno spiegate e approfondite nella sezione successiva. Di seguito vengono riportate le nuove specifiche del problema che sono state raccolte dall'Ospedale Policlinico San Martino:

- alcuni pazienti richiedono obbligatoriamente un letto per il trattamento
- i pazienti che hanno una terapia di durata maggiore di 180 minuti dovrebbero stare preferibilmente nei letti
- alcuni pazienti possono richiedere di essere assegnati prima di un certo orario, dopo un certo orario o in una determinata stanza a seconda della terapia

In questa versione estesa del problema, oltre ai letti e alle sedie, sono presenti una terza categoria di risorse, ovvero le risorse virtuali. Tali risorse vengono utilizzate quando non ci sono più letti o sedie disponibili. Questo si traduce in una nuova ottimizzazione aggiuntiva, ovvero la minimizzazione delle assegnazioni dei pazienti alle risorse virtuali, che è l'ottimizzazione

più importante. Inoltre, minimizziamo i pazienti che richiedevano un letto e che sono stati assegnati ad una sedia, ma non più viceversa, come vedremo nella sezione successiva.

3.2 Codifica ASP

In questa sezione presentiamo le modifiche che sono state fatte al codificatore in linguaggio ASP per passare dal problema base al problema esteso. Prima vedremo come viene modellato l'input e come vengono generati gli input, a seguire vedremo come vengono generati gli input a partire da diverse tabelle excel (con l'ausilio di uno script scritto in linguaggio Python), poi vedremo la codifica dell'output e infine verrà presentata la codifica del nuovo modello che gestisce la versione estesa del problema.

3.2.1 Input

I seguenti quattro atomi sono gli stessi della versione base del problema, alcuni con delle piccole modifiche:

- reg(RID, DAY, DRUG, DURATION, COD, C): le sue istanze rappresentano le registrazioni come già descritte nel capitolo precedente, con la differenza che al posto della durata della fase 1, 2 e 3, vi è il codice relativo al percorso del paziente (COD), che verrà approfondito nella sottosezione successiva
- chair(ID, IDSTANZA): le sue istanze, oltre a rappresentare le sedie disponibili, ci danno anche l'informazione relativa alla stanza in cui è collocata la sedia
- bed(ID, IDSTANZA): le sue istanze, oltre a rappresentare i letti disponibili, ci dicono anche la stanza in cui è collocato il letto
- mss(DAY, TS): è l'unico atomo che rimane invariato in questa versione estesa del problema

Ora presentiamo i nuovi atomi di input che sono stati introdotti:

- percorso(COD, VIS, PRE, ACC): le sue istanze rappresentano i diversi percorsi possibili che un paziente può seguire, caratterizzate dal codice del percorso (COD), dalla durata della visita (VIS), del prelievo (PRE) e dell'accettazione (ACC)
- mustBed(RID): le sue istanze rappresentano i pazienti, identificati da un id (RID), che richiedono obbligatoriamente un letto per il trattamento

- afterTs(RID, TS): questo atomo ci dice che il paziente con identificativo RID deve essere assegnato dopo lo slot temporale TS
- before(RID, TS): questo atomo invece ci dice che il paziente con identificativo RID deve essere assegnato prima dello slot temporale TS
- trasfusione(RID): le sue istanze rappresentano tutti i pazienti, identificati con RID, che fanno la trasfusione
- virtuale(ID, IDSTANZA): le sue istanze rappresentano le risorse virtuali, caratterizzate da un identificativo (ID) e dalla stanza in cui sono collocate (IDSTANZA)
- mustHaveRoom(RID, ID): le sue istanze rappresentano i pazienti, identificati con RID, che devono essere assegnati a una determinata stanza con identificativo ID
- salaTrasfusioni(ID): le sue istanze rappresentano le risorse con identificativo ID che si trovano nella sala trasfusioni

Adesso vediamo, nella sottosezione seguente, come vengono creati gli atomi di input appena presentati.

3.2.2 Creazione input

Per la creazione degli atomi di input è stato sviluppato uno script, *make_inputs.py*, scritto in linguaggio Python. Tale script prende in ingresso tre file excel, che approfondiremo di seguito, e restituisce tanti file di testo quanti sono i giorni di pianificazione. Ogni file di testo contiene tutti gli atomi di input di un determinato giorno. Di seguito presentiamo in dettaglio i tre file excel.
Il primo file excel che andiamo a vedere è pazienti.xls. Di seguito viene mostrato una parte del contenuto del file.

PAZIENTE	DATA	CRITICO	DESCRIZIONE TERAPIA	DURATA	POSTAZIONE	PERCORSO
12816	1	N	CEMIPLIMAB 350 mg	45	P04	VT
18881	1	N	IDRATAZIONE Platino > e/o = 80	0	L05	T
17567	1	S	FOLFOX 4 MOD + panitumumab	5580	P12	T
15286	1	N	Nivolumab 480 MG OGNI 4 SETTIM	75	P12	BO
14523	1	N	Xgeva (Denosumab) Q28	30	P17	AR
14176	1	N	Triangle sperimentale R-DAOX A	0	P02	T
13020	1	N	TRASFUSIONE	60	V05	AR
17812	1	N	Paclitaxel+Trastuzumab Settima	130	L08	BO
13824	1	N	Nivolumab 240 MG OGNI 2 SETTIM	75	P20	BO
13194	1	N	TRASFUSIONE	60	L10	VT
11172	1	N	Paclitaxel-albumina + gemcitab	90	P07	BO
18906	1	N	KEYTRUDA 200 mg	45	P06	VT
10993	1	N	KRD cicli 2-12	70	P09	AR
16952	1	N	Palbociclib 75 mg	25	P25	AR

Figura 3.1 Contenuto di pazienti.xls

Come possiamo vedere nella figura 3.1, attributi di questa tabella sono:

- **paziente**: rappresenta l'identificativo del paziente (è il term RID dell'atomo *reg*)
- **data**: rappresenta il giorno in cui il paziente deve effettuare il trattamento. Durante la pianificazione si considerano contemporaneamente solo i pazienti che hanno lo stesso valore di data.
- **critico**: rappresenta la criticità di un paziente, in particolare se *critico* = S allora il paziente deve essere assegnato ad un letto, altrimenti no (*critico* = N)
- **descrizione terapia**: rappresenta la terapia che un paziente deve seguire (è il term DRUG dell'atomo *reg*)
- **durata**: rappresenta la durata della fase 4 (è il term DURATION dell'atomo *reg*)
- **postazione**: rappresenta l'id della risorsa assegnata ad un paziente
- **percorso**: rappresenta il codice del percorso di un paziente (è il term COD dell'atomo *reg*)

Adesso vediamo cos'è il percorso relativo a un paziente.

COD	DESCRIZIONE	ESAMI	VISITA	TERAPIA	TERAPIA-VISITA	VISITA-ESAMI	ESAMI-ACCETT.
T	Solo Terapia	N	N	S	0	0	0
AR	Tutto in una giornata (Rapido)	S	S	S	60	30	10
A	Tutto in una giornata	S	S	S	120	30	10
BO	Terapia Oncologica	N	S	S	10	0	0
BE	Terapia Ematologica	N	N	S	0	0	0
D	Solo esami	S	N	N	0	0	0
P2	Compresse+Esami	S	N	N	0	0	0
S	farmaci sperimentali	S	S	S	120	30	10
C	Visita e Prelievi	S	S	N	0	30	10
VT	Visita e Terapia	N	S	S	10	0	0
P	Solo Prelievo	S	N	N	0	0	0
PV	Prelievo + Visita	S	S	N	0	0	0
V	Solo Visita	N	S	N	0	0	0
T	Solo Terapia	N	N	S	0	0	0
P1	Compresse+Esami+Visita	S	S	N	0	30	0
P3	Compresse+Visita	N	S	N	0	0	0

Figura 3.2 Contenuto di percorsi.xls

Nella figura 3.2 è mostrato il contenuto del secondo file, percorsi.xls. Gli attributi di questa tabella sono:

- **cod**: rappresenta il codice identificativo di un percorso (è il term COD dell'atomo *percorso*)
- **descrizione**: breve descrizione del percorso
- **esami**: indica se il paziente che segue un determinato percorso fa la fase 2, ovvero il prelievo del sangue
- **visita**: indica se il paziente che segue un determinato percorso fa la fase 3, ovvero la visita medica
- **terapia**: indica se il paziente che segue un determinato percorso fa la terapia
- **terapia - visita**: rappresenta la durata della fase 3 (è il term VIS dell'atomo *percorso*)
- **visita - esami**: rappresenta la durata della fase 2 (è il term PRE dell'atomo *percorso*)
- **esami - accett.**: rappresenta la durata della fase 1, ovvero l'accettazione (è il term ACC dell'atomo *percorso*)

Adesso vediamo l'ultimo file.

COD	DESCRIZIONE	STANZA
L01	LETTO 1	Stanza 1
L02	LETTO 2	Stanza 1
L03	LETTO 3	Stanza 2
L04	LETTO 4	Stanza 2
L05	LETTO 5	Stanza 3
L06	LETTO 6	Stanza 3
L07	LETTO 7	Stanza 3
L08	LETTO 8	Stanza 3
L09	LETTO 9	Stanza 3
L10	LETTO 10	Stanza 4
L11	LETTO 11	Stanza 4
L12	LETTO 12	Stanza 4
L13	LETTO 13	Stanza 4
L14	LETTO 14	Stanza 4
L15	LETTO 15	Stanza 5
L16	LETTO 16	Stanza 5
L17	LETTO 17	Stanza 6
L18	LETTO 18	Stanza 6
L19	LETTO 19	Stanza 6
L20	LETTO 20	Stanza 6
L21	LETTO 21	Sala Trasfusioni
L22	LETTO 22	Sala Trasfusioni
L23	LETTO 23	Sala Trasfusioni
L24	LETTO 24	Sala Trasfusioni
L25	LETTO 25	Sala Trasfusioni
P01	POLTRONA 1	Sala Terapia Arancio
P02	POLTRONA 2	Sala Terapia Arancio
P03	POLTRONA 3	Sala Terapia Arancio
P04	POLTRONA 4	Sala Terapia Arancio

Figura 3.3 Contenuto di risorse.xls

Nella figura 3.3 è mostrato una parte del contenuto del terzo file, risorse.xls. Gli attributi di questa tabella sono:

- **cod**: rappresenta il codice identificativo di una risorsa (è il term ID degli atomi *chair*, *bed* e *virtuale*)
- **descrizione**: è la descrizione di una risorsa, in particolare indica se una risorsa è un letto, una sedia o una risorsa virtuale
- **stanza**: rappresenta la stanza in cui è collocata la risorsa corrispondente (è il term IDSTANZA degli atomi *chair*, *bed* e *virtuale*)

Guardando gli attributi delle tre tabelle appena presentate possiamo notare che non c'è l'informazione relativa alla presenza o meno di un bagno nelle varie stanze. Tale informazione è stata dedotta (sempre tramite lo script *make_inputs.py*) tramite l'attributo *postazione* (vedi figura 3.1). In particolare, siccome i pazienti che utilizzano il farmaco Cisplatino devono

essere inseriti in una stanza con un bagno (è uno dei vincoli del problema), cioè devono essere assegnati ad una risorsa collocata in una stanza con un bagno, filtrando tutti i pazienti che fanno uso di Cisplatino, e guardando a quali risorse sono stati assegnati (tramite l'attributo *postazione*), possiamo risalire alla stanza tramite il file risorse.xls ottenendo quindi le stanze che hanno un bagno.

3.2.3 Output

L'output del problema consiste in assegnazioni, in particolare viene utilizzato il seguente atomo:

x(RID, DAY, TS, DRUG, DURATION, C)

ovvero è lo stesso atomo della versione base del problema, quindi rimane invariato.

3.2.4 Encoding

Adesso presentiamo il codificatore in linguaggio ASP che modella la versione estesa del problema.
Di seguito viene presentato il codificatore completo:

```
1 {x(RID,DAY,TS,DRUG,DURATION,C): mss(DAY,TS),TS\2=1} 1 :- reg(RID,_,
    DRUG,DURATION,_,C).
2 :- x(RID,_,TS,_,DURATION,_), DURATION>50, TS<24.
3 :- x(RID,_,TS,_,_,_), reg(RID,_,_,_,COD,C), percorso(COD,VIS,PRE,ACC),
    TS-PRE-VIS-ACC<1.
4 res(RID,DAY,TS..TS+DURATION-1,C) :- x(RID,DAY,TS,_,DURATION,C), DURATION
    >0.
5 1 {bed(ID,RID,DAY): bed(ID,_); chair(ID,RID,DAY): chair(ID,_); virtuale(
    ID,RID,DAY): virtuale(ID,_)} 1 :- x(RID,DAY,_,_,_,_), not trasfusione
    (RID).
6 1 {bed(ID,RID,DAY): bed(ID,_), salaTrasfusioni(ID); chair(ID,RID,DAY):
    chair(ID,_), salaTrasfusioni(ID); virtuale(ID,RID,DAY): virtuale(ID,_
    )}1 :- x(RID,DAY,_,_,_,_), trasfusione(RID).
7 chair(ID,RID,DAY,TS) :- chair(ID,RID,DAY), res(RID,DAY,TS,C).
8 bed(ID,RID,DAY,TS) :- bed(ID,RID,DAY), res(RID,DAY,TS,C).
9 virtuale(ID,RID,DAY,TS) :- virtuale(ID,RID,DAY), res(RID,DAY,TS,C).
10 :- #count{RID:chair(ID,RID,DAY,TS)}>1, chair(ID,_), mss(DAY,TS).
11 :- #count{RID:bed(ID,RID,DAY,TS)}>1, bed(ID,_), mss(DAY,TS).
12 :- #count{RID:virtuale(ID,RID,DAY,TS)}>1, virtuale(ID,_), mss(DAY,TS).
```

```
13 supporto(RID,DAY,TS) :- x(RID,DAY,START,_,_,_), reg(RID,_,_,_,COD,_),
       percorso(COD,VIS,PRE,_), PRE>0, START-VIS-PRE=TS, mss(_,TS).
14 numbRegP(DAY,N,TS) :- N=#count{RID:supporto(RID,DAY,TS)},mss(DAY,TS).
15 numbDay(DAY,N) :- N=#count{RID:supporto(RID,DAY,_)},mss(DAY,_).
16 numMaxP(DAY,T) :- T=#max{N:numbRegP(DAY,N,_)},mss(DAY,_).
17 numMinP(DAY,T) :- T=#min{N:numbRegP(DAY,N,_)},mss(DAY,_).
18 :- not bed(_,RID,_),mustBed(RID).
19 :- x(RID,_,T,_,_,_),afterTs(RID,TS),T<TS.
20 :- mustHaveRoom(RID,_),not virtuale(_,RID,_),#count{ID:bed(ID,RID,_),bed
       (ID,IDS),mustHaveRoom(RID,IDS);ID:chair(ID,RID,_),chair(ID,IDS),
       mustHaveRoom(RID,IDS)}!=1.
21 :~ x(RID,DAY,_,_,_,_),virtuale(_,RID,DAY).[1@6,RID]
22 :~ x(RID,DAY,_,_,_,1),chair(_,RID,DAY),not mustBed(RID).[1@5,RID]
23 :~ x(RID,DAY,_,_,DURATION,_),DURATION>36,chair(_,RID,DAY).[1@4,RID]
24 :~ x(RID,_,T,_,_,_),beforeTs(RID,TS),T>TS.[1@3,RID]
25 :~ numMaxP(DAY,M).[M@2,DAY]
26 :~ numMaxP(DAY,MAX),numMinP(DAY,MIN).[MAX-MIN@1,DAY]
```

Nel codificatore ci sono regole completamente nuove, regole uguali a quelle del codificatore base e regole simili ma con qualche modifica. Andiamo a vedere nel dettaglio le regole nuove e quelle che sono state modificate, le regole invariate non saranno descritte.
Iniziamo dalla terza regola:

```
:- x(RID,_,TS,_,_,_), reg(RID,_,_,_,COD,C), percorso(COD,VIS,PRE,ACC),
   TS-PRE-VIS-ACC<1.
```

Siccome l'atomo *reg* non ha più le durate delle fasi 1, 2 e 3, in questa regola viene aggiunto l'atomo *percorso* che ha le durate delle tre fasi mancanti. Il significato della regola rimane invariato.

Le regole 5 e 6 corrispondono alla modifica della regola 5 del codificatore base, ovvero la regola relativa all'assegnazione delle risorse.
Il motivo per il quale sono state utilizzate due regole per l'assegnazione di una risorsa risiede nel vincolo relativo ai pazienti che fanno la trasfusione (vedi sezione precedente). Infatti, quando si assegnano le risorse, distinguiamo chi fa la trasfusione e chi no.
Partiamo dalla quinta regola:

```
1 {bed(ID,RID,DAY): bed(ID,_); chair(ID,RID,DAY): chair(ID,_); virtuale(
   ID,RID,DAY): virtuale(ID,_)} 1 :- x(RID,DAY,_,_,_,_), not trasfusione
   (RID).
```

Questa regola differisce dalla regola 5 del codificatore base per la presenza della risorsa virtuale, infatti oltre agli atomi *bed* e *chair*, è presente anche l'atomo *virtuale* nella scelta della risorsa da assegnare. Inoltre, è presente l'atomo *trasfusione* preceduta dal simbolo di negazione *not*, quindi significa che questa regola vale solo per tutti i pazienti che non fanno la trasfusione.

Ora vediamo la sesta regola:

```
1 {bed(ID,RID,DAY): bed(ID,_), salaTrasfusioni(ID); chair(ID,RID,DAY):
   chair(ID,_), salaTrasfusioni(ID); virtuale(ID,RID,DAY): virtuale(ID,_
   )}1 :- x(RID,DAY,_,_,_,_), trasfusione(RID).
```

Questa regola vale per tutti i pazienti che fanno la trasfusione, infatti l'atomo *trasfusione* non è preceduta dal simbolo di negazione. Inoltre vi è un'altro atomo aggiuntivo, *salaTrasfusioni*, per via del vincolo sui pazienti che fanno le trasfusioni. Infatti tale atomo filtra le risorse, permettendo esclusivamente l'assegnazione a letti o sedie che sono collocate nella sala trasfusioni (oltre alle risorse virtuali).

Regola 9:

```
virtuale(ID,RID,DAY,TS) :- virtuale(ID,RID,DAY), res(RID,DAY,TS,C).
```

Tale regola è analoga alle regole 7 e 8 ma per le risorse virtuali, ovvero ci dice quale risorsa virtuale è occupata, da chi, in quale giorno e in quale slot temporale.

Regola 12:

```
:- #count{RID:virtuale(ID,RID,DAY,TS)}>1, virtuale(ID,_), mss(DAY,TS).
```

Tale regola ci dice che non è possibile che la risorsa virtuale con identificativo ID sia assegnato a più di una persona nello stesso slot temporale TS relativo al giorno DAY.

La regola 13 è analoga alla regola 10 del codificatore base:

```
supporto(RID,DAY,TS) :- x(RID,DAY,START,_,_,_),reg(RID,_,_,_,COD,_),
   percorso(COD,VIS,PRE,_), PRE>0, START-VIS-PRE=TS, mss(_,TS).
```

L'unica differenza è che siccome *reg* non ha più la durata delle fasi 1, 2 e 3, è stato aggiunto l'atomo percorso che contiene le informazioni mancanti. Il significato rimane lo stesso.

Le regole 18, 19 e 20 sono nuovi e rappresentano i nuovi vincoli della versione estesa del problema. Partiamo dalla 18:

```
:- not bed(_, RID, _), mustBed(RID).
```

Questo vincolo forte obbliga l'assegnazione di un letto a tutti i pazienti che devono necessariamente avere un letto.

Regola 19:

```
:- x(RID, _, T, _, _, _), afterTs(RID, TS), T<TS.
```

Questo vincolo forte ci dice che non è possibile assegnare un paziente (RID) ad uno slot temporale T che precede lo slot temporale TS, se tale paziente deve essere assegnato dopo lo slot temporale TS.

La regola 20 serve per assegnare i pazienti a determinate stanze:

```
:- mustHaveRoom(RID,_),not virtuale(_,RID,_),#count{ID:bed(ID,RID,_),bed
   (ID,IDS),mustHaveRoom(RID,IDS);ID:chair(ID,RID,_),chair(ID,IDS),
   mustHaveRoom(RID,IDS)}!=1.
```

Questa regola obbliga l'assegnazione dei pazienti ad una delle risorse che sono collocate in una delle stanze in cui tale paziente deve stare. In particolare, ci dice che un paziente RID, che non è stato assegnato ad una risorsa virtuale, deve essere assegnato ad una risorsa (che può essere un letto o una sedia) collocata in una delle stanze che tale paziente deve necessariamente stare.

Ora vediamo le regole dalla 21 alla 26, che sono le regole di ottimizzazione dei risultati. Vediamo la regola 21:

```
:~ x(RID,DAY,_,_,_,_),virtuale(_,RID,DAY).[1@6,RID]
```

Questa è la regola di ottimizzazione più importante e serve per minimizzare le assegnazioni di risorse virtuali.

Regola 22:

```
:~ x(RID,DAY,_,_,_,1),chair(_,RID,DAY),not mustBed(RID).[1@5,RID]
```

Questa regola di ottimizzazione riguarda tutti i pazienti che non hanno bisogno necessariamente di un letto. In particolare minimizza il numero di pazienti che sono stati assegnati alle sedie ma avevano come preferenza il letto (non vale più il viceversa).

Regola 23:

```
:~ x(RID,DAY,_,_,DURATION,_),DURATION>longDuration,chair(_,RID,DAY).[1@4
   ,RID]
```

Tale regola di ottimizzazione minimizza il numero di pazienti che hanno una terapia di lunga durata e che sono assegnati ad una sedia.

Regola 24:

```
:~ x(RID,_,T,_,_,_), beforeTs(RID,TS), T>TS.[1@3,RID]
```

Questa regola di ottimizzazione serve per minimizzare il numero di persone che iniziano la terapia dopo lo slot temporale TS nonostante sia preferibile che la inizino prima di TS.

Nel capitolo successivo vedremo la valutazione sperimentale, in particolare vedremo l'analisi dei risultati del codificatore esteso e confronteremo le sue prestazioni con il codificatore base.

Capitolo 4

Valutazione sperimentale

In questo capitolo presentiamo prima il setting sperimentale in cui abbiamo eseguito i test e successivamente analizziamo i risultati ottenuti dal vecchio e dal nuovo codificatore. Tali risultati ci permettono di dire che ASP risulta essere uno strumento utile per risolvere problemi complessi come la pianificazione delle sedute dei trattamenti oncologici anche con l'aggiunta di vincoli e dati reali come presentati in questa tesi.

4.1 Setting sperimentale e Benchmarks

In questa sezione abbiamo confrontato i tempi di grounding di entrambi i codificatori e, per il nuovo codificatore, abbiamo visto quanti secondi impiega a trovare la soluzione ottima di tutte le istanze di input. I test sono stati eseguiti utilizzando il risolutore Clingo (Gebser et al. (2012)).
Le specifiche rilevanti del computer sulla quale sono stati effettuati i test e le analisi dei risultati sono i seguenti:

- Sistema operativo Windows 10 Pro a 64 bit, processore basato su x64
- Processore AMD Ryzen 5 1600 da 3.20 GHz avente 6 core e 12 threads
- 16 GB di Ram DDR4

I risultati sono stati ottenuti passando eseguendo Clingo con i seguenti parametri:

- **–time-limit 60**: tempo limite a 60 secondi
- **–parallel-mode 12**: permette di sfruttare tutti i 12 thread durante la ricerca della soluzione ottima

- **–restart-on-model**: ogni volta che Clingo trova una soluzione, riparte da capo con la ricerca dell'ottimo

I test sono stati eseguiti sia con il vecchio che con il nuovo codificatore. Per entrambi i codificatori sono state utilizzate 22 istanze di input, che corrispondono ai 22 giorni lavorativi di un mese. I risultati per la pianificazione sono ottenuti utilizzando 60 risorse, di cui 25 letti, 26 poltrone e 8 risorse virtuali. Per ogni giorno ci sono in media 127 pazienti da pianificare, in particolare, nella figura 4.1 possiamo vedere nel dettaglio il numero di pazienti da pianificare (asse delle ordinate) per ogni istanza di input (asse delle ascisse).

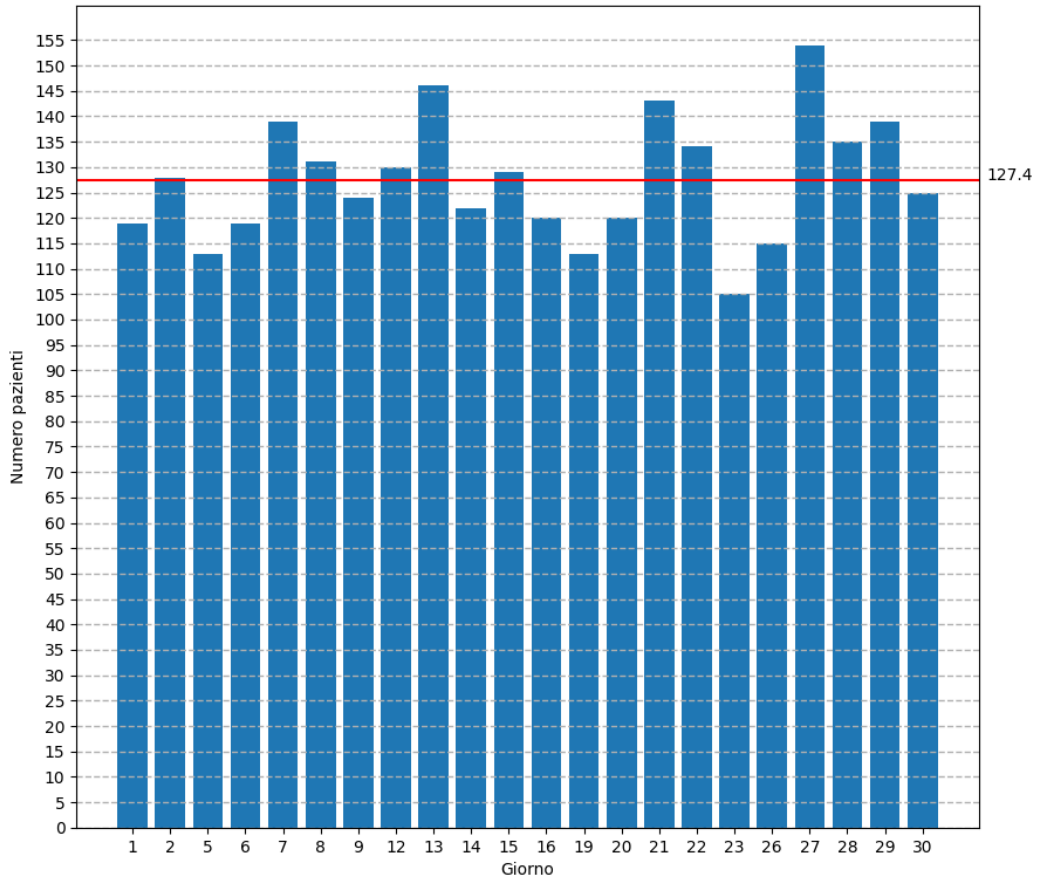


Figura 4.1 Numero dei pazienti da pianificare

Per confrontare i due codificatori siamo interessati ad analizzare i tempi di grounding di entrambi per vedere quanto sono state influenzate le performance del nuovo codificatore dai

nuovi vincoli.

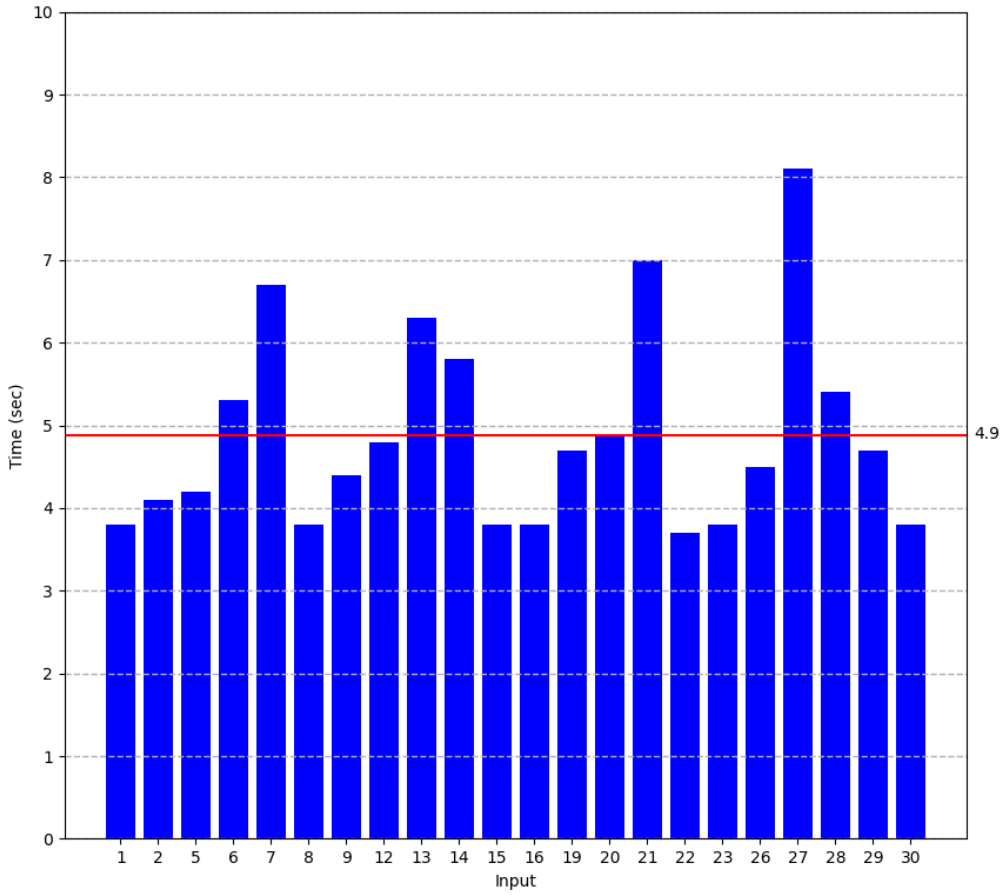


Figura 4.2 Tempo di grounding del vecchio codificatore

Nella figura 4.2 vengono presentati i tempi di grounding di tutte le istanze di input, ovvero tutti i giorni di pianificazione relativi al mese preso in considerazione, utilizzando il vecchio codificatore. Nell'asse delle ascisse abbiamo i giorni di pianificazione mentre nell'asse delle ordinate abbiamo i tempi di grounding espressi in secondi. Il tempo medio di grounding è pari a 4.9 secondi.

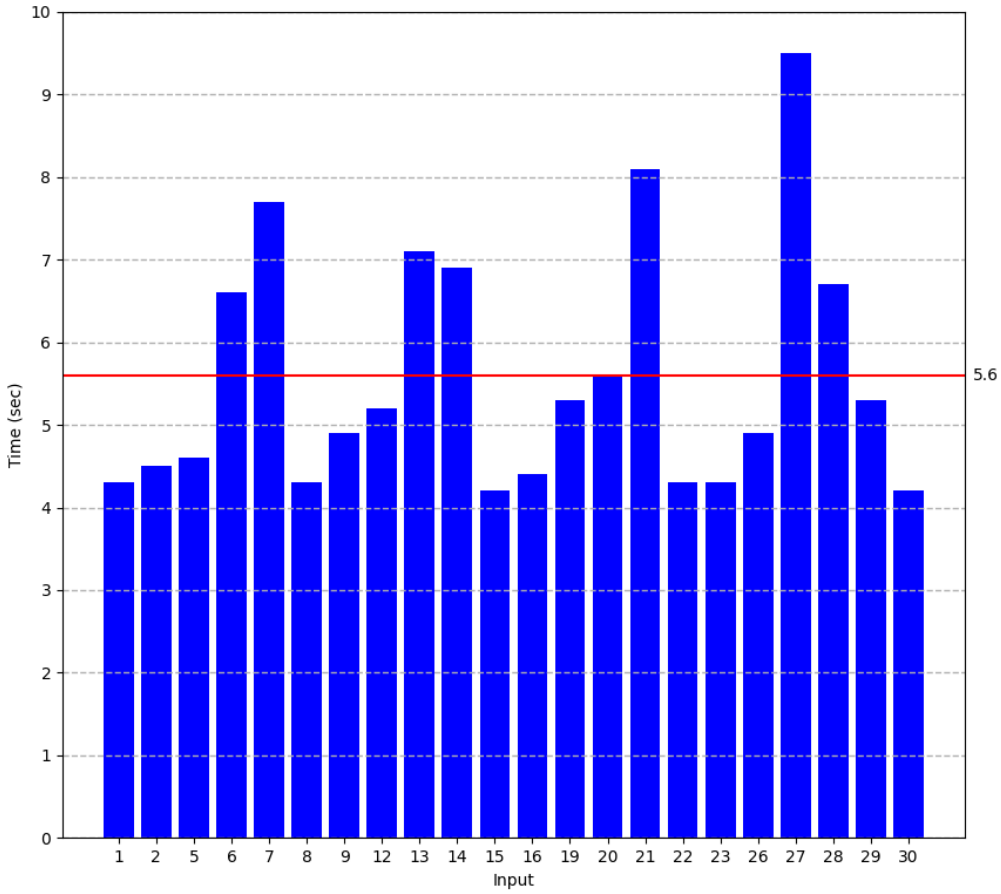


Figura 4.3 Tempo di grounding del nuovo codificatore

Nella figura 4.3 vengono presentati i tempi di grounding di tutte le istanze di input utilizzando il nuovo codificatore. Come possiamo vedere, il tempo medio di grounding è pari a 5.6 secondi. Come ci aspettavamo, il tempo di grounding del nuovo codificatore è maggiore rispetto a quello del vecchio codificatore, in particolare il tempo medio di grounding è aumentato del 14%. Questo aumento del tempo medio di grounding è causato dalla presenza di un maggior numero di atomi di input, oltre che di regole e vincoli, nel nuovo codificatore. Tuttavia siamo soddisfatti del risultato in quanto, nonostante l'aggiunta dei nuovi vincoli e la presenta di più atomi di input nel nuovo codificatore, l'aumento del 14% del tempo medio di grounding è molto piccolo.

Avendo analizzato i tempi di grounding passiamo ora all'analisi dei tempi di solving. Come detto precedentemente, il tempo limite di esecuzione per entrambi i codificatori è 60 secondi. Durante i 60 secondi di esecuzione, Clingo non riesce a dimostrare che le soluzioni ottenute sono ottime, tuttavia possiamo dire che lo sono in quanto tali soluzioni non possono essere ulteriormente migliorate. Quindi non siamo interessati a vedere i tempi di solving che sono per entrambi i codificatori 60 secondi di time out, ma siamo interessati a vedere quanto tempo ci mettono entrambi i codificatori a trovare l'ultima soluzione, ovvero quella effettivamente ottima.

Tabella 4.1 Tempi di esecuzione per trovare la soluzione ottima con entrambi i codificatori

Giorno	Nuovo codificatore	Vecchio codificatore
1	8,2	6,7
2	10,6	7,5
5	9,8	7,2
6	12,5	7,4
7	17,2	12,5
8	8,9	7,3
9	10,7	7,8
12	11	9
13	17,5	10,7
14	14	9,9
15	9	6,7
16	9	7,2
19	10,9	8,2
20	10,6	8,4
21	17,9	13,9
22	9,9	6,7
23	8	6,3
26	10,2	8,1
27	24	16,3
28	13,5	9,7
29	9,9	9,4
30	9	6,7

Nella tabella 4.1 possiamo vedere quanti secondi impiegano i due codificatori a trovare la soluzione ottima per ogni istanza di input, mentre nella tabella 4.2 vediamo la loro media. Come ci si poteva aspettare, avendo aumentato il numero di vincoli e di ottimizzazioni, il tempo che impiega il nuovo codificatore a trovare la soluzione ottima è maggiore rispetto al tempo che impiega il vecchio codificatore. In particolare, il nuovo codificatore impiega

Tabella 4.2 Riassunto delle prestazioni

Codificatore	Tempo medio esecuzione
Versione base	11,9 sec
Versione estesa	8,8 sec

in media 3,1 secondi in più per trovare l'ottimo, e anche questo è un buon risultato se consideriamo tutti i nuovi vincoli che sono stati introdotti al problema.

4.2 Analisi dei risultati

Avendo analizzato le tempistiche di risoluzione di entrambi i codificatori, in questa sezione analizzeremo i risultati ottenuti dai due codificatori per quanto riguarda le ottimizzazioni. In particolare, analizzeremo le preferenze dei pazienti sulle risorse, le assegnazioni delle risorse e la distribuzione dei pazienti che devono fare la fase 2, ovvero il prelievo del sangue. Questo perché siamo interessati a vedere se la qualità delle soluzioni diminuiscono a seguito dell'aumento della complessità del problema.

4.2.1 Analisi sulle preferenze

Nella figura 4.4 vengono presentate le percentuali relative alle preferenze rispettate per ogni istanza di input usando il vecchio codificatore. Nell'asse delle ascisse abbiamo i giorni di pianificazione mentre nell'asse delle ordinate abbiamo la percentuale delle preferenze rispettate. Come possiamo vedere, la media percentuale delle preferenze rispettate è pari al 100%, ovvero sono sempre rispettate per ogni istanza di input. Queste preferenze sono state calcolate tenendo in considerazione tutti i pazienti, visto che nella versione base del problema affrontato non vi sono vincoli che obbligano i pazienti ad essere assegnati ad una determinata risorsa.

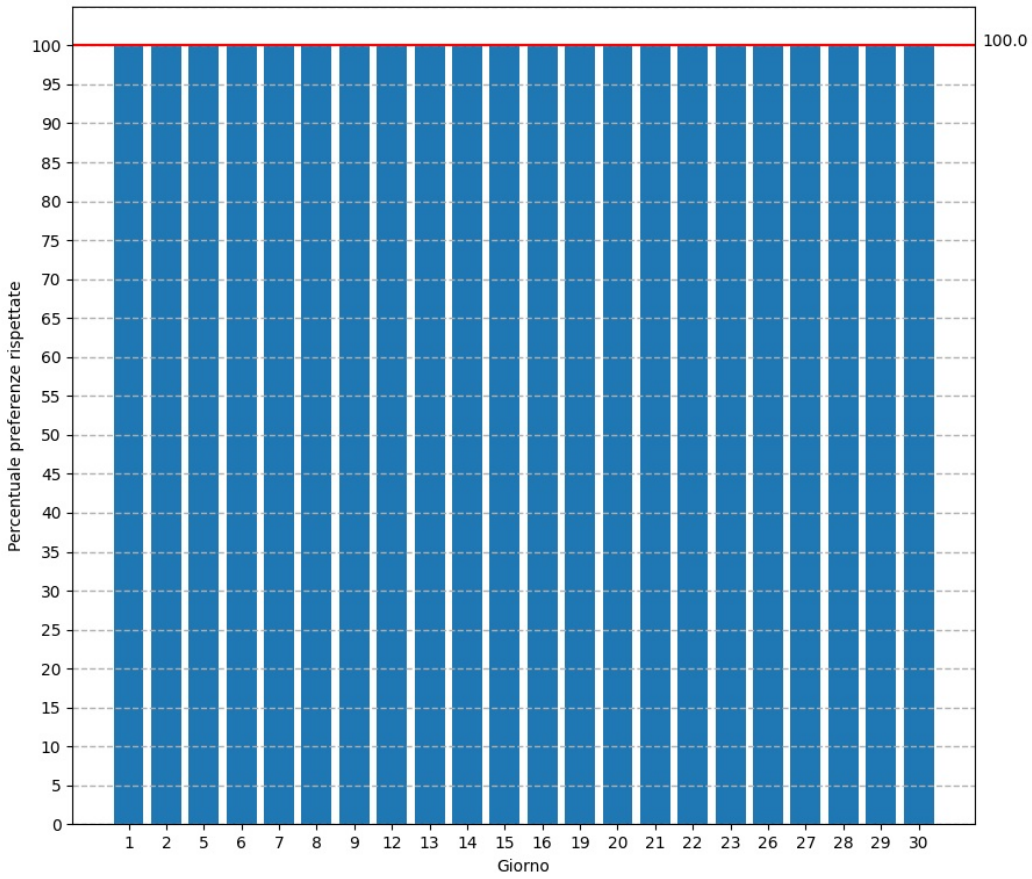


Figura 4.4 Percentuali delle preferenze rispettate usando il vecchio codificatore

Nella figura 4.5 vengono presentate le percentuali relative alle preferenze rispettate per ogni istanza di input usando il nuovo codificatore. Queste preferenze sono state calcolate tenendo in considerazione solamente i pazienti che potevano essere assegnati ad una risorsa arbitraria (cioè non avevano nessun vincolo che li obbligava ad essere assegnati ad una determinata risorsa) e avevano come preferenza il letto, perché sono gli unici pazienti a poter essere ottimizzati. Ad esempio, se un paziente è critico (quindi è obbligato ad usare un letto) e ha come preferenza la sedia, non si può fare niente per assegnargli una sedia a causa dei vincoli del problema, stessa cosa per tutti gli altri pazienti vincolati su una determinata risorsa.

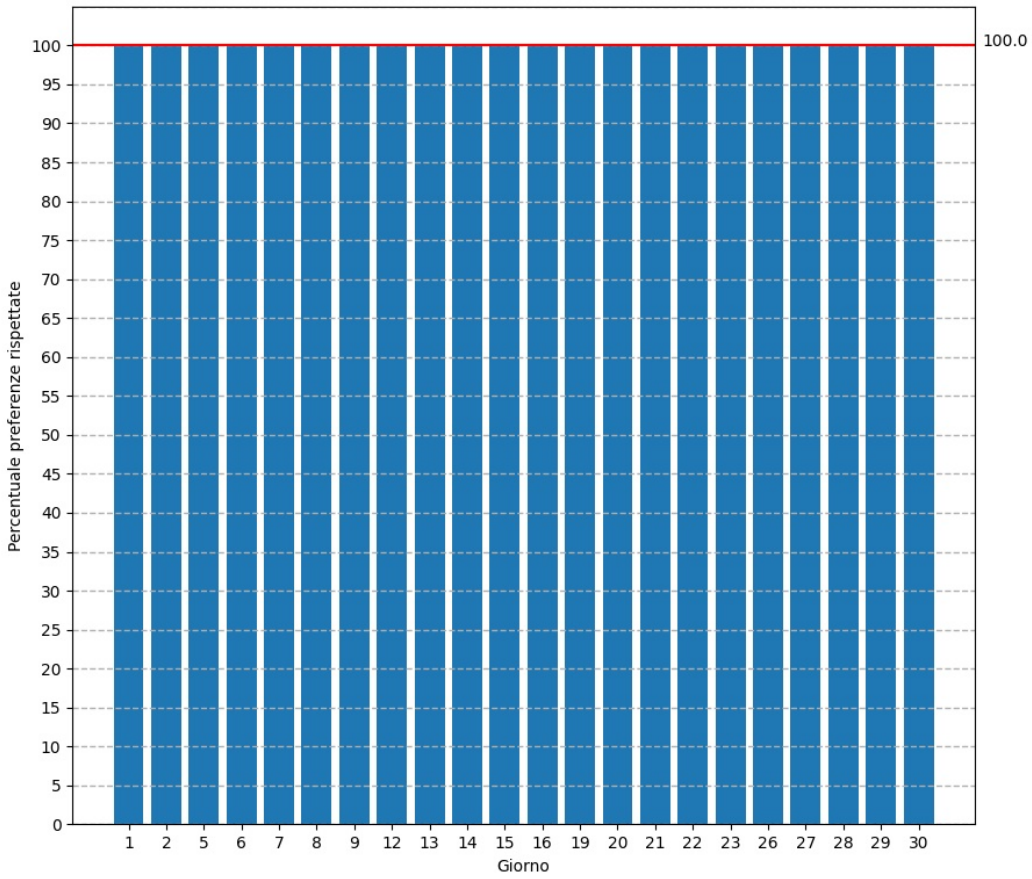


Figura 4.5 Percentuali delle preferenze rispettate usando il nuovo codificatore

Inoltre, non sono stati considerati i pazienti che devono essere assegnati alle risorse della sala Terapia Arancio, questo perché, come possiamo vedere nella figura 4.6, nella sala Terapia Arancio sono presenti solamente sedie e quindi non hanno nessuna possibilità di essere assegnati ad un letto.

L25	LETTO 25	Sala Trasfusioni
P01	POLTRONA 1	Sala Terapia Arancio
P02	POLTRONA 2	Sala Terapia Arancio
P03	POLTRONA 3	Sala Terapia Arancio
P04	POLTRONA 4	Sala Terapia Arancio
P05	POLTRONA 5	Sala Terapia Arancio
P06	POLTRONA 6	Sala Terapia Arancio
P07	POLTRONA 7	Sala Terapia Arancio
P08	POLTRONA 8	Sala Terapia Arancio
P09	POLTRONA 9	Sala Terapia Arancio
P10	POLTRONA 10	Sala Terapia Arancio
P11	POLTRONA 11	Sala Terapia Arancio
P12	POLTRONA 12	Sala Terapia Arancio
P13	POLTRONA 13	Sala Terapia Azzurra

Figura 4.6 Risorse collocate nella sala Terapia Arancio

Ritornando nella figura 4.5 possiamo vedere che la percentuale media delle preferenze rispettate usando il nuovo codificatore è pari al 100%. Questo risultato è particolarmente soddisfacente in quanto le assegnazioni delle risorse nella versione estesa del problema sono molto più vincolate e il numero di pazienti che devono essere assegnati ai letti sono maggiori rispetto alla versione base del problema e nonostante questo le preferenze rimangono sempre rispettate al 100%.

4.2.2 Analisi sulle assegnazioni delle risorse

In questa sezione siamo interessati a vedere come varia l'assegnazione delle risorse con l'aggiunta dei nuovi vincoli, in particolare il numero di letti assegnati dal nuovo codificatore. Come visto nel capitolo precedente, la versione estesa del problema ha 3 tipi di risorse, ovvero le risorse virtuali, le sedie e i letti. Il nuovo codificatore non assegna mai le risorse virtuali, infatti la regola di ottimizzazione con maggiore priorità è quella relativa alla minimizzazione delle assegnazioni alle risorse virtuali.

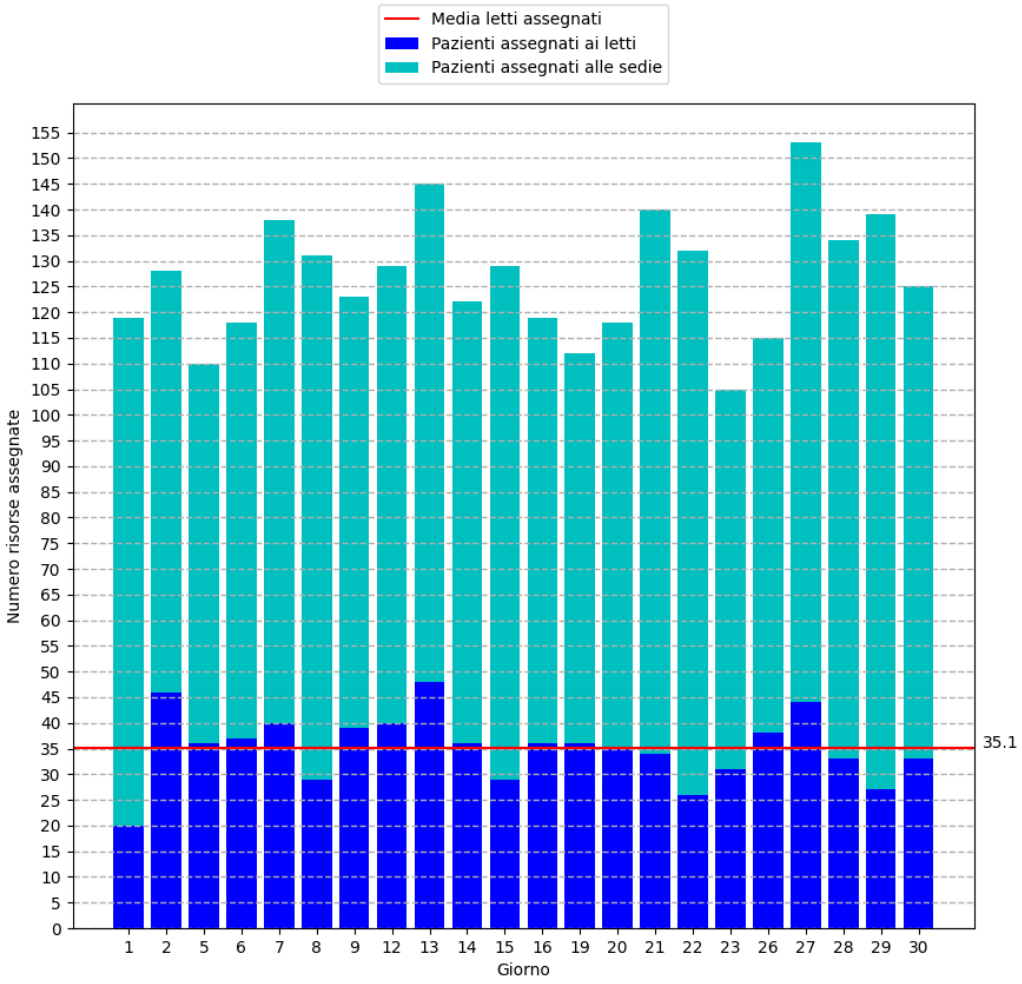


Figura 4.7 Quantità di risorse assegnate dal vecchio codificatore

Nella figura 4.7 vengono presentate il numero di risorse assegnate. Nell'asse delle ascisse abbiamo i giorni di pianificazione mentre nell'asse delle ordinate abbiamo il numero totale di risorse assegnate. Ogni barra è composta da due colori differenti, la parte blu rappresenta il numero di letti assegnati mentre la parte rimanente il numero di sedie assegnate. Come possiamo vedere sono stati assegnai molte più sedie rispetto ai letti e il numero medio di letti assegnati è 35.

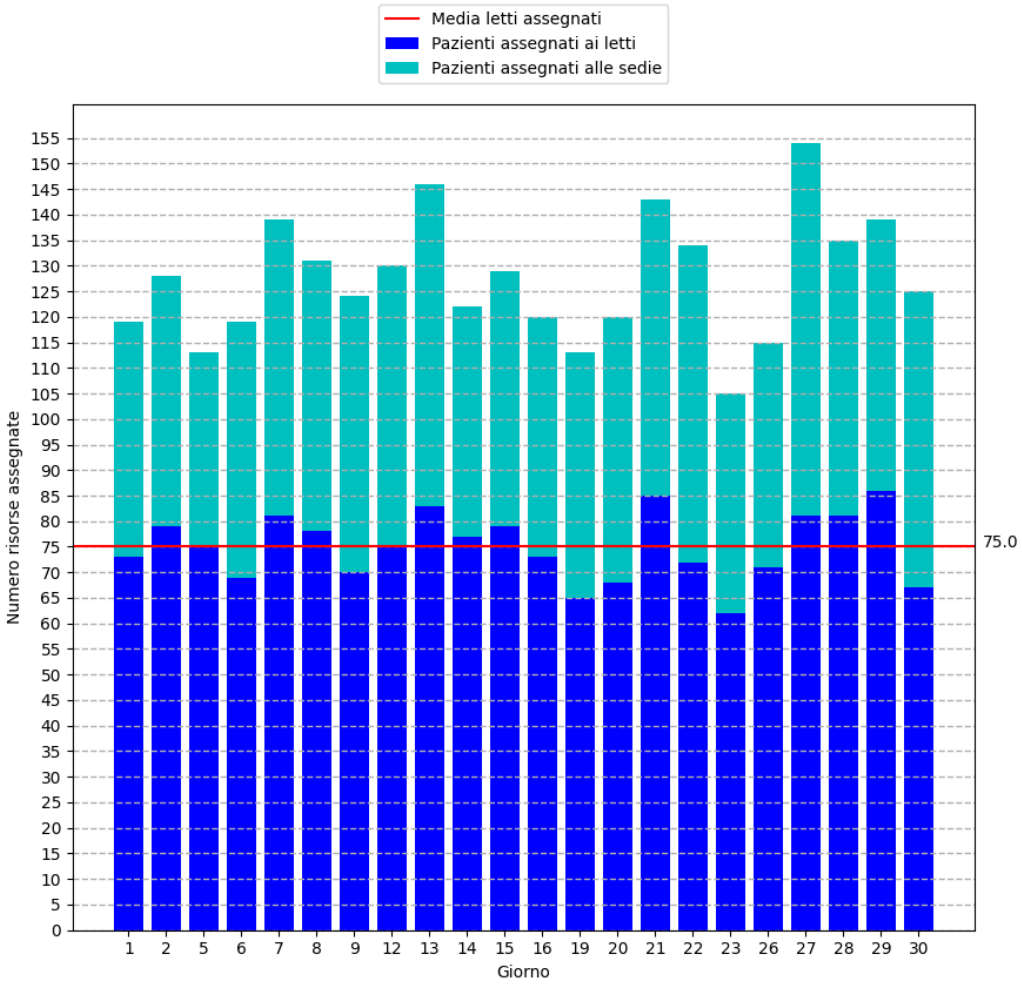


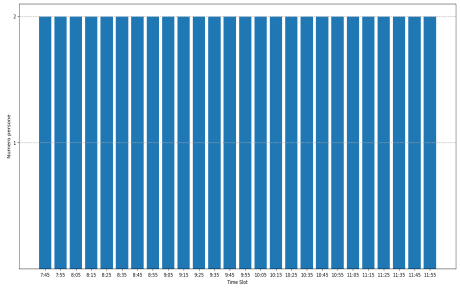
Figura 4.8 Quantità di risorse assegnate dal nuovo codificatore

Nella figura 4.8 viene sempre presentato il numero di risorse assegnate ma questa volta dal nuovo codificatore. Come possiamo vedere, il numero di letti assegnati è maggiore rispetto al numero delle sedie assegnate. Inoltre, il numero medio di pazienti assegnati ai letti da parte del nuovo codificatore (75) è molto minore rispetto al numero medio di pazienti assegnati ai letti da parte del vecchio codificatore (35,1). La causa è dovuta a tutti i nuovi vincoli (visti in precedenza) che obbligano i pazienti ad essere assegnati ai letti. Inoltre, come abbiamo visto nei capitoli precedenti, il vecchio codificatore cerca di rispettare tutte le preferenze (sia letti che sedie, dal momento che non ha tutti i vincoli aggiuntivi che ha il nuovo codificatore i quali obbligano l'assegnazione di una determinata risorsa o stanza), mentre il nuovo codificatore cerca di rispettare solamente le preferenze relative ai

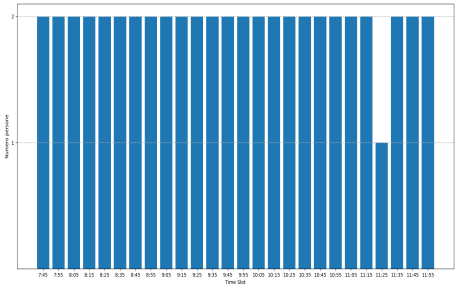
letti (per tutti i pazienti che non rientrano nei vincoli della versione estesa del problema). In conclusione, grazie a questa analisi sulle assegnazioni delle risorse, abbiamo visto che il nuovo codificatore riesce ad assegnare molti più letti rispetto al vecchio codificatore mantenendo quasi inalterate le tempistiche.

4.2.3 Analisi sulla distribuzione dei prelievi

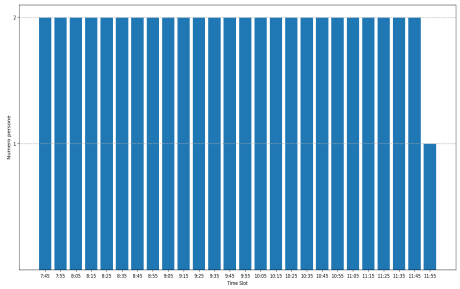
In questa sottosezione analizziamo la distribuzione dei pazienti che devono fare il prelievo durante le varie giornate di pianificazione. Dal momento che abbiamo 2 codificatori e 44 giorni di pianificazione (22 per il nuovo codificatore e 22 per quello vecchio), risulta ridondante mostrare tutti i 44 grafici relativi alle distribuzioni dei pazienti che fanno il prelievo in quanto il comportamento è molto simile e ai fini delle nostre analisi ci basta analizzare un insieme più ristretto di giorni, pertanto verranno mostrati i grafici relativi a 4 giorni di pianificazione (2 giorni per il nuovo codificatore e 2 giorni per quello vecchio).



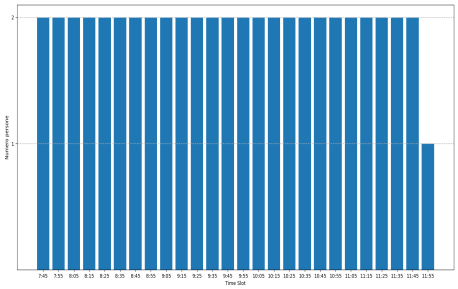
(a) nuovo codificatore: giorno 28



(b) vecchio codificatore: giorno 28



(c) nuovo codificatore: giorno 29



(d) vecchio codificatore: giorno 29

Figura 4.9 Distribuzione dei prelievi nei giorni 28 e 29

Nella figura 4.9 sono rappresentate le distribuzioni dei pazienti che fanno il prelievo nei giorni 28 e 29 utilizzando sia il nuovo codificatore che il vecchio codificatore. Per tutti i 4 grafici, nell'asse delle ordinate abbiamo il numero di pazienti che iniziano il prelievo mentre nell'asse delle ascisse abbiamo i vari orari con un intervallo di 10 minuti. Come si può vedere dai grafici, i pazienti sono molto ben distribuiti durante alla giornata grazie alle

ultime due regole del codificatore (regole 25 e 26 del codificatore presentato nel capitolo 3). Confrontando i grafici ottenuti con i due codificatori possiamo vedere che le distribuzioni sono molto simili nei stessi giorni. In particolare, possiamo notare che il numero massimo di pazienti che effettuano contemporaneamente il prelievo durante lo stesso giorno usando i due codificatori sono uguali.
Nella tabella 4.3 possiamo vedere il numero massimo di pazienti che effettuano il prelievo contemporaneamente (nello stesso slot temporale) per tutti i 22 giorni di pianificazione e per entrambi i codificatori. Infatti possiamo vedere che questi numeri sono gli stessi per ogni coppia di giorni, come accennato in precedenza.

Tabella 4.3 Numero massimo di pazienti che effettuano contemporaneamente il prelievo per ogni istanza di input

Giorno	Nuovo codificatore	Vecchio codificatore
1	2	2
2	3	3
5	3	3
6	2	2
7	2	2
8	3	3
9	3	3
12	3	3
13	3	3
14	2	2
15	3	3
16	2	2
19	3	3
20	2	2
21	2	2
22	3	3
23	2	2
26	3	3
27	3	3
28	2	2
29	2	2
30	3	3

In conclusione, possiamo affermare che nonostante la presenza di vincoli aggiuntivi nella versione estesa del problema, le modifiche al data model e l'aggiunta di regole e vincoli al nuovo codificatore, la soluzione che otteniamo è molto buona in quanto il prezzo che si paga in termini di prestazioni e risultati è molto basso se non trascurabile. In particolare,

abbiamo confrontato i due codificatori e abbiamo visto che il nuovo codificatore riesce ad assegnare molti più letti rispetto al vecchio codificatore, le preferenze sulle risorse sono sempre rispettate e i pazienti che effettuano il prelievo sono sempre ben distribuiti, il tutto mantenendo quasi inalterate le performance, ovvero i tempi di grounding e di risoluzione.

Capitolo 5

Applicazione Web

In questo capitolo presentiamo l'applicazione web che è stata sviluppata per utilizzare il codificatore relativo alla versione estesa del problema in maniera semplice e intuitiva.

5.1 Architettura

Per cominciare, vediamo l'architettura dell'applicazione web.

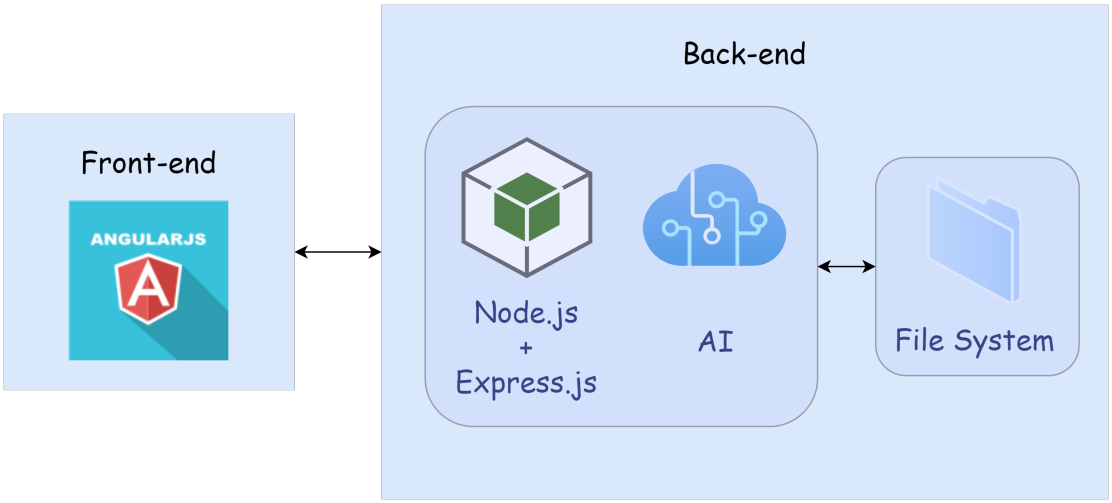


Figura 5.1 Architettura dell'applicazione web

Nella figura 5.1 viene presentata l'architettura dell'applicazione web.

Le tecnologie utilizzate per lo sviluppo dell'applicazione sono:

- Frontend:
  - AngularJS: framework JavaScript.
- Backend:
  - Express.js: framework JavaScript;
  - Node.js: runtime system orientato agli eventi che permette l'esecuzione di codice JavaScript;
  - AI: per poter utilizzare il codificatore ASP, è stato utilizzato il modulo wrapper clingojs che permette al backend di chiamare clingo.
- Storage: è stato utilizzato il file system per archiviare le giornate di pianificazione (insieme di file di testo contenenti atomi di input).

In generale, il sistema funziona nel seguente modo:

- l'utente seleziona un giorno di pianificazione e manda gli atomi di input associati a tale giorno al server;
- il server riceve gli atomi di input e li inoltra al modulo wrapper clingojs che a sua volta fa partire il codificatore ASP;
- clingojs ritorna il risultato al server che lo inoltra al client;
- il client riceve il risultato e lo mostra all'utente.

5.2 Demo

In questa sezione vediamo una demo dell'applicazione web affiancata da commenti e spiegazioni di tutte le interazioni tra il frontend e il backend. Siccome l'applicazione web è una SPA (Single Page Application), non si parlerà di pagine, ma di componenti.

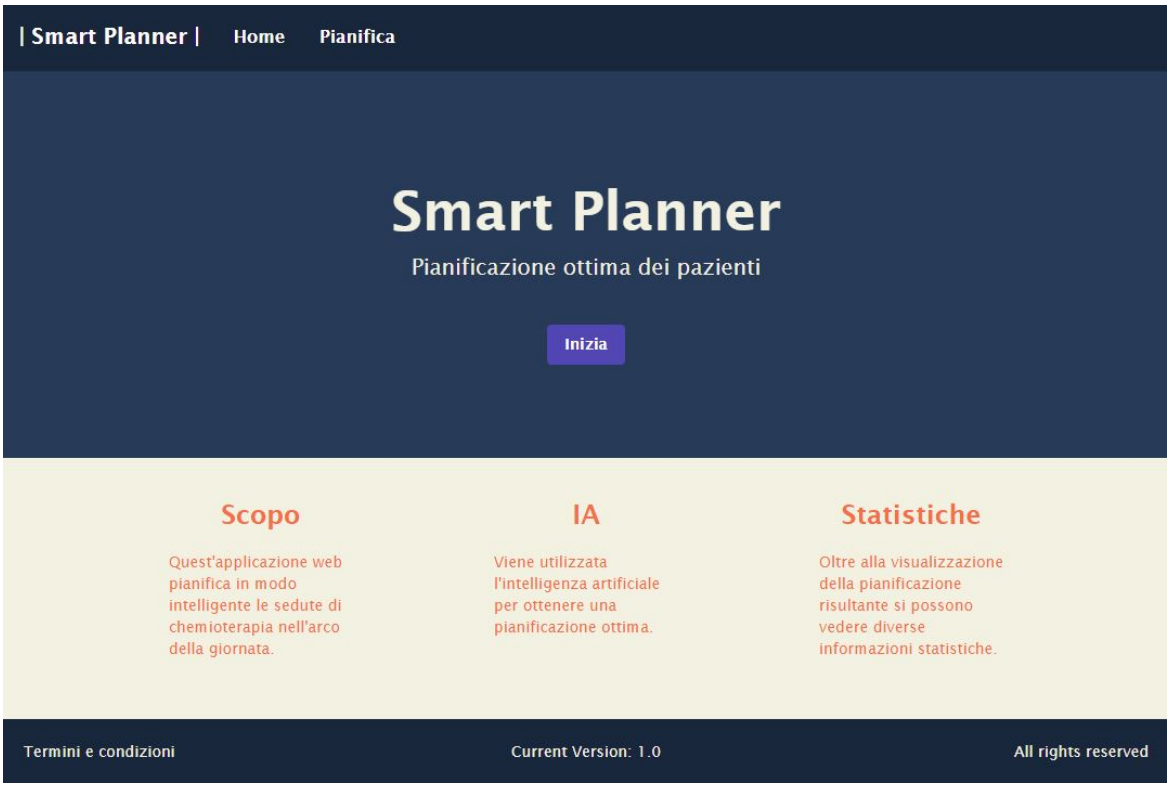


Figura 5.2 Homepage

Nella figura 5.2 viene presentata l'homepage dell'applicazione web. Cliccando su *Inizia* (o *Pianifica* nell'header) veniamo reindirizzati al componente relativo alla scelta del giorno, presentata nella figura 5.3.

Figura 5.3 Componente per la scelta del giorno

I giorni che vediamo nel dropdown list sono presi dal backend. In particolare, durante la renderizzazione di tale pagina, vengono eseguite le seguenti operazioni:

- il client manda una richiesta HTTP GET al server chiamando l'API corrispondente;
- il server risponde alla richiesta mandando al client i giorni di pianificazione disponibili; presi dal proprio file system (nella cartella input, vedi figura 5.4)
- il client riceve i giorni disponibili e li mostra all'utente attraverso il dropdown list.

In particolare, i file contenuti nella cartella *input* presentato nella figura 5.4 sono dei file di testo, i quali contengono gli atomi di input. Come visto nel capitolo 3, tali file sono stati generati dallo script Python *make_inputs.py*. I giorni che vediamo nel dropdown list sono il risultato del parsing dei nomi di tutti i file di testo presenti nella cartella *input*.

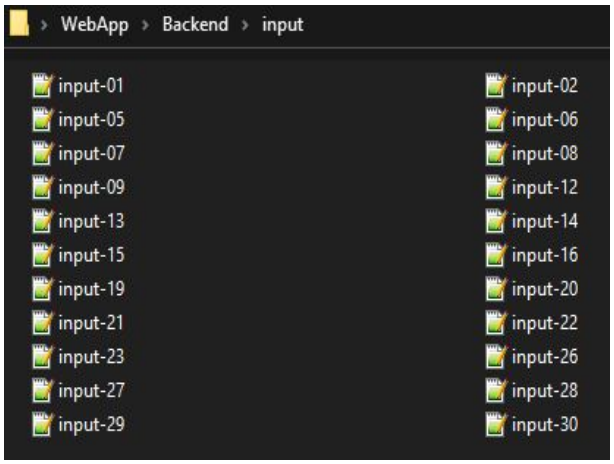


Figura 5.4 Contenuto della cartella *input* del server

Dopo aver selezionato il giorno da pianificare, veniamo reindirizzati al componente presentato nella figura 5.5.

Figura 5.5 Componente di pianificazione 1

Le principali operazioni (facoltative) che possono essere effettuate in questo componente sono:

- filtrare i pazienti tramite il loro identificativo *rid*;
- cambiare il giorno di pianificazione;
- modificare i parametri relativi ad un determinato paziente;
- modificare il tempo limite di pianificazione (60 secondi è il valore di default, come mostrato nella figura 5.6).

A questo punto, per iniziare a pianificare, si clicca il bottone *Start plan!*.

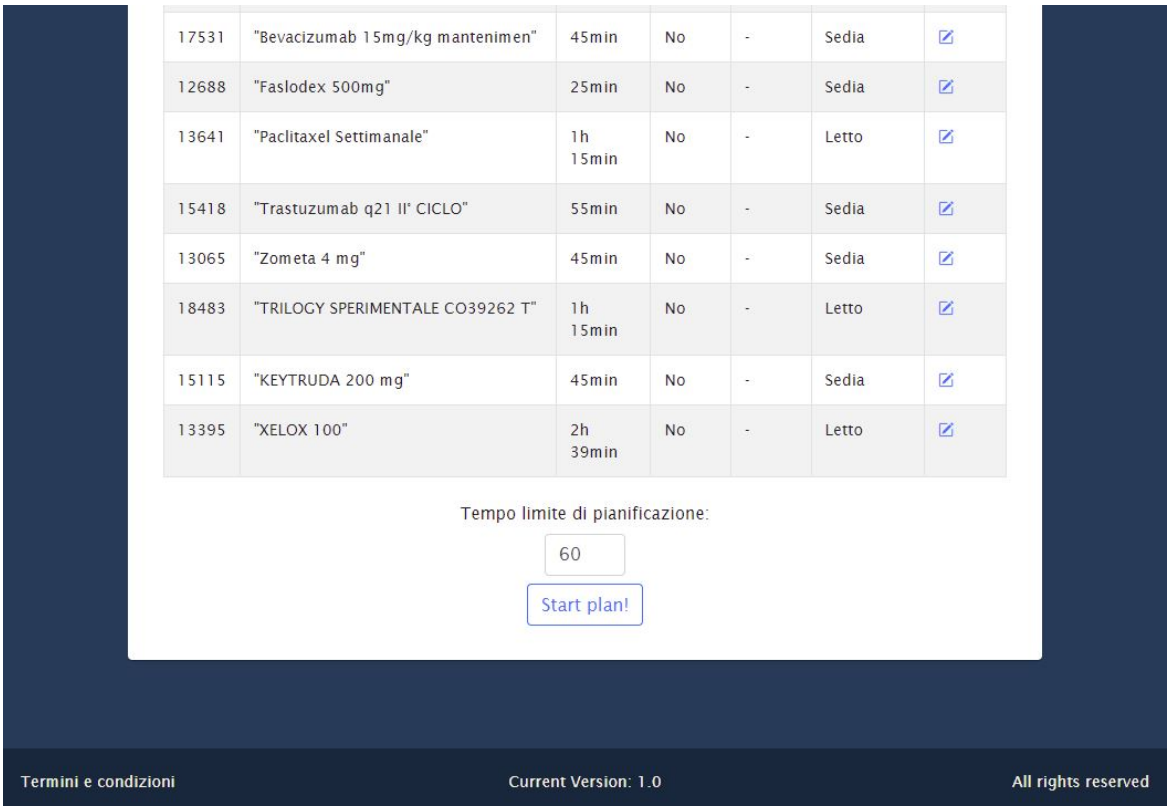


Figura 5.6 Componente di pianificazione 2

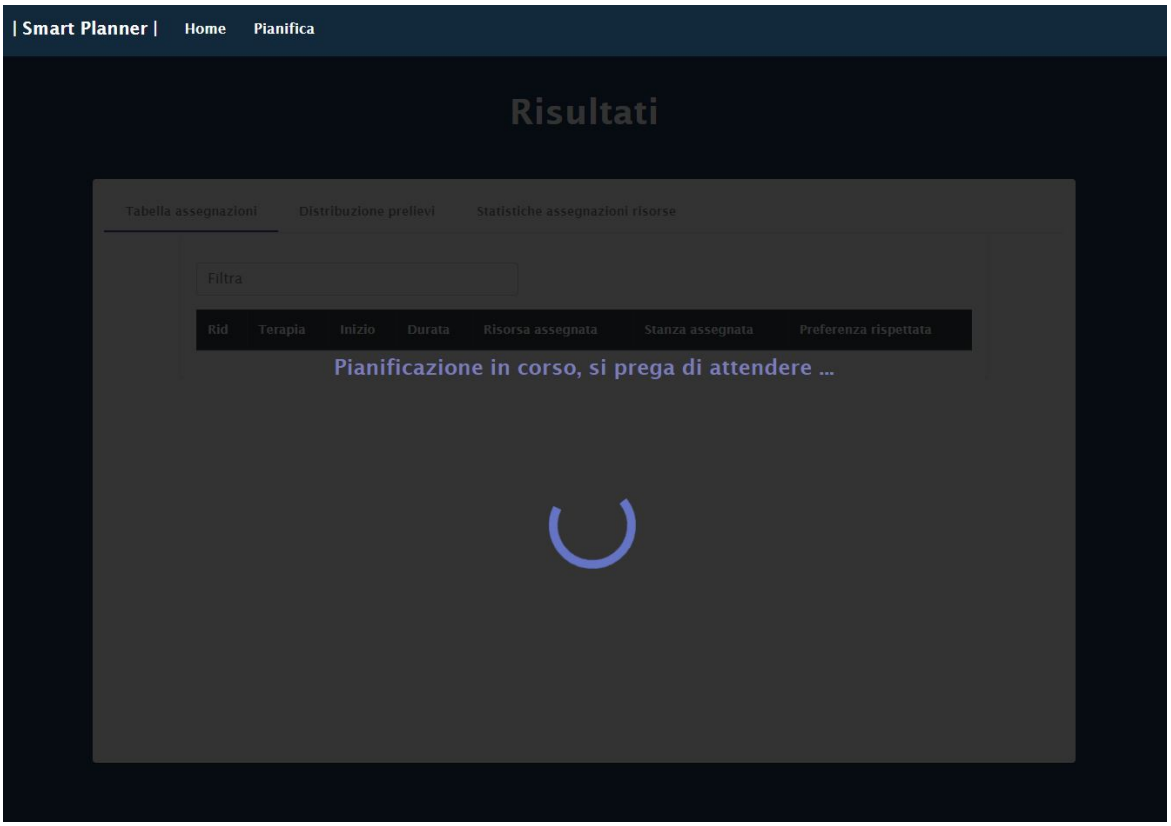


Figura 5.7 Pianificazione in corso...

Una volta iniziata la pianificazione, viene mostrato il componente presentato nella figura 5.7 con l'animazione della rotella di caricamento che gira e persiste per tutta la durata della pianificazione (a seconda del tempo limite di pianificazione scelto). Mentre la rotellina gira, il codificatore ASP è in esecuzione!
Quando viene premuto il tasto *Start plan* vengono eseguite le seguenti operazioni:

- il client manda al server un HTTP POST il quale payload contiene tutti gli atomi di input aggiornati (nel caso siano state effettuate modifiche ai parametri di qualche paziente) relativi al giorno di pianificazione scelto;
- il server riceve gli atomi di input e li inoltra al modulo wrapper clingojs ;
- il modulo wrapper fa partire il codificatore ASP con gli atomi di input ricevuti.

Quando il codificatore ASP finisce l'esecuzione e ritorna la soluzione al modulo wrapper, quest'ultimo manda tale soluzione al server che lo inoltra al client. A questo punto possiamo vedere la soluzione.

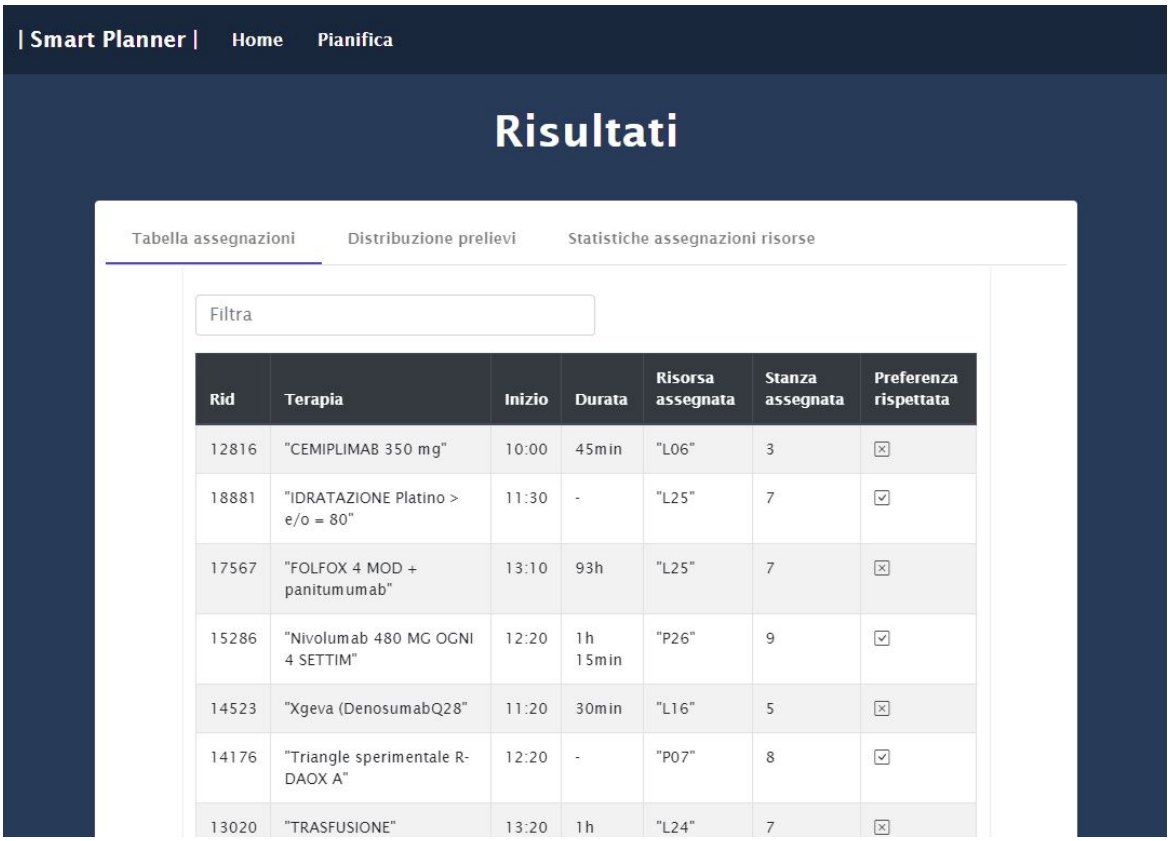


Figura 5.8 Risultato della pianificazione

Come possiamo vedere nella figura 5.8, che rappresenta il componente relativo alla visualizzazione del risultato, abbiamo 3 tabs:

- Tabella assegnazioni;
- Distribuzione prelievi;
- Statistiche assegnazioni risorse.

Nella figura 5.8 possiamo vedere la prima tab. In questa tab possiamo vedere una tabella che contiene il risultato della pianificazione di tutti i pazienti, in particolare per ogni paziente possiamo vedere:

- l'identificativo;
- la terapia;
- a che ore inizia la terapia;

- la durata della terapia;
- l'identificativo della risorsa assegnata al paziente;
- la stanza assegnata al paziente;
- un simbolo che indica se la preferenza sulla risorsa è stata rispettata o meno.

Ora andiamo a vedere la seconda tab.

Figura 5.9 Distribuzione dei pazienti che effettuano il prelievo

Nella figura 5.9 viene presentata la distribuzione dei pazienti che fanno il prelievo nella giornata di pianificazione selezionata. Nell'asse delle ascisse abbiamo i diversi orari con un intervallo di 10 minuti mentre nell'asse delle ordinate abbiamo il numero di pazienti che effettuano il prelievo.
L'ultima tab riguarda le statistiche sulle risorse.

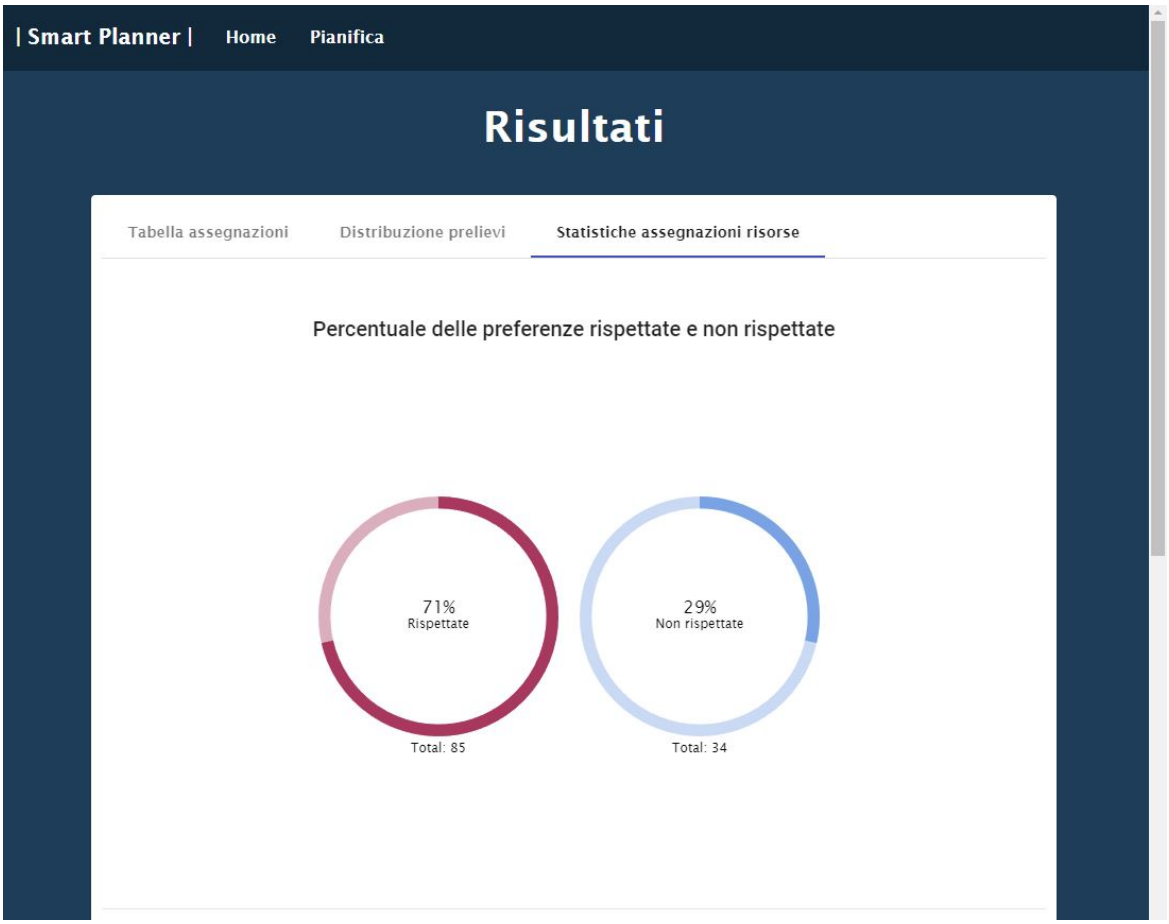


Figura 5.10 Statistiche sulle preferenze relative alle risorse

Nella figura 5.10 possiamo vedere le statistiche sulle preferenze relative alle risorse. In particolare, possiamo vedere che, in questa particolare istanza, il 71% delle preferenze relative alle risorse sono state rispettate, mentre il restante 29% delle preferenze non sono state rispettate. Inoltre possiamo anche vedere il numero esatto delle preferenze rispettate (85) e non rispettate (34). Andando in fondo alla pagina troviamo le statistiche sulle assegnazioni delle risorse.

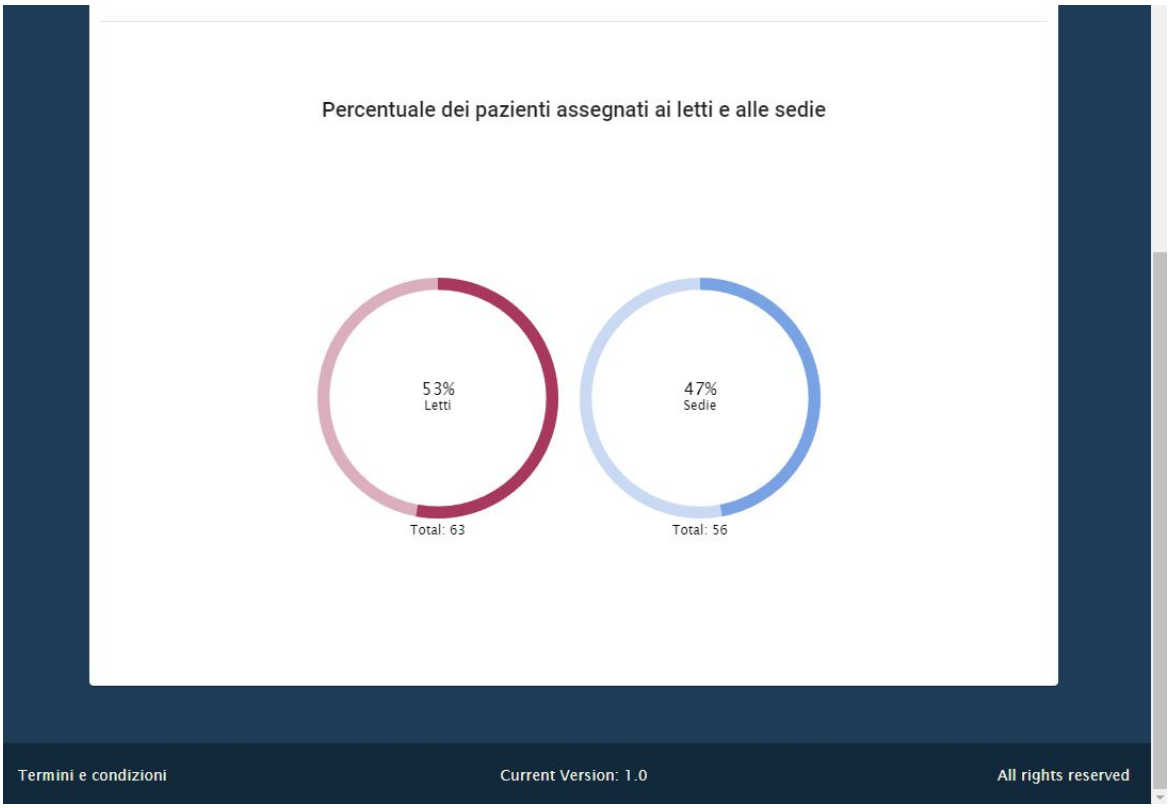


Figura 5.11 Statistiche sulle preferenze relative alle risorse

In particolare, come possiamo vedere nella figura 5.11, abbiamo la percentuale dei pazienti assegnati ai letti e alle sedie. Nel nostro caso, il 53% dei pazienti sono stati assegnati ai letti mentre il restante 47% dei pazienti sono stati assegnati alle sedie. Inoltre possiamo vedere anche il numero dei pazienti assegnati ai letti (63) e alle sedie (56).

Capitolo 6

Lavori correlati

In questo capitolo presentiamo alcuni lavori correlati che sono stati pubblicati. In particolare vediamo prima i lavori che sono stati accennati nel capitolo 1 sezione 1.2 (stato dell'arte) e successivamente vediamo i lavori correlati dove è stato utilizzato il linguaggio ASP ma su problemi non riguardanti i trattameni dei pazienti oncologici.

6.1 Lavori relativi alla pianificazione di sedute oncologiche

6.1.1 Ottimizzazione dell'utilizzo delle risorse in una clinica oncologica

L'articolo Huggins et al. (2014) citato nel primo capitolo riguarda l'ottimizzazione dell'utilizzo delle risorse in una clinica oncologica tramite l'utilizzo della programmazione a numeri interi. In particolare Huggins et al. (2014) presentano un problema di ottimizzazione mixed-integer sviluppato con l'obbiettivo di bilanciare il carico di lavoro delle risorse umane massimizzando l'utilizzo delle risorse a disposizione. In particolare bisogna tenere in considerazione aspetti come:

- la diversa durata dei trattamenti;
- l'aumento delle richieste di pazienti;
- i vincoli sulla disponibilità delle risorse.

6.1.2 Pianificazione e programmazione delle operazioni di chemioterapia

L'articolo Turkcan et al. (2010) citato nel primo capitolo riguarda la pianificazione e la programmazione delle operazioni di chemioterapia tramite l'utilizzo di metodi euristici. In particolare vengono pianificate le sedute dei pazienti oncologici tenendo in considerazione vincoli sulle risorse minimizzando sia il tempo di idle degli infermieri, sia i ritardi sui trattamenti dei vari pazienti.

6.1.3 Algoritmi per la programmazione dei piani di chemioterapia

L'articolo Sevinc et al. (2013) parla degli algoritmi per la programmazione dei piani di chemioterapia. Sevinc et al. (2013) affrontano il problema tramite un approccio a due fasi:

- fase 1: viene utilizzato un algoritmo di pianificazione adattivo a retroazione negativa per controllare il peso nel sistema;
- fase 2: vengono utilizzate due euristiche basate su "Multiple knapsack problem" (problema dello zaino multiplo) per assegnare le sedie di infusione ai pazienti.

Il sistema è stato testato in un centro di chemioterapia locale e ha restituito dei buoni risultati relativi ai tempi di attesa e di utilizzo delle sedie di infusione.

6.2 Problemi di pianificazione con ASP

L'obbiettivo dell'articolo Alviano et al. (2017, 2018); Dodaro and Maratea (2017) è la creazione di una pianificazione riguardante i turni degli infermieri, tramite l'utilizzo del linguaggio ASP, tenendo in considerazione diversi vincoli. Inoltre, per poter gestire cambiamenti inaspettati dei turni di lavoro dei vari infermieri, viene anche affrontato il problema della ripianificazione.

L'articolo Amendola (2018) propone una soluzione in ASP del problema del SRP (Stable Roommates Problem). Tale problema é una versione alternativa di un'altro problema chiamato Stable Marriage Problem. In particolare, il problema dell'articolo consiste nell' accoppiare un insieme di 2n persone tenendo in considerazione l'ordine di preferenza (relativa alle altre persone) che le persone hanno dato.

L'articolo Gebser et al. (2018) propone una soluzione in ASP del problema della pianificazione del percorso dei veicoli a guida automatizzata per affrontare le attività di trasporto in maniera efficiente nel contesto dell'assemblaggio di automobili presso Mercedes-Benz, dove i percorsi dei veicoli venivano tradizionalmente codificati a mano da ingegneri umani.

L'obbiettivo dell'articolo Ricca et al. (2012) è l'organizzazione del personale di un porto volto ad affrontare il lavoro richiesto dalle navi in arrivo, tenendo in considerazione alcuni vincoli come:

- la distribuzione bilanciata dei lavori;
- il cambio del personale nei lavori più pericolosi;
- l'assegnazione dei lavori a lavoratori che hanno le competenze adeguate al lavoro assegnato.

L'articolo Alviano et al. (2020) analizza diversi problemi in ambito Digital Health, partendo da problemi standard come pianificazione di sale operatorie e di nurse scheduling e presenta soluzioni che sono state sviluppate con l'ASP per migliorare le soluzioni giá esistenti e ulteriori problemi che sono stati affrontati con l'ASP.

L'articolo Cardellini et al. (2021) propone una soluzione in ASP del problema della pianificazione delle sessioni di fisioterapia riabilitativa per i pazienti, che consiste nell'assegnazione di operatori adeguati in una determinata fascia oraria, tenendo in considerazione diversi vincoli come il bilanciamento del lavoro degli operatori e le preferenze dei pazienti.

L'articolo Dodaro et al. (2019) propone una soluzione in ASP del problema relativo all'ottimizzazione della pianificazione giornaliera degli interventi chirurgici in sala operatoria, tenendo in considerazione diversi vincoli come l'allocazione delle risorse necessarie all'intervento, la disponibilità delle squadre chirurgiche e l'orario di inizio degli interventi chirurgici.

Capitolo 7

Conclusioni e lavori futuri

In questa tesi, dopo aver studiato la versione base del problema insieme alla sua soluzione, abbiamo studiato le nuove specifiche e dati dell'ospedale San Martino ottenendo la versione estesa del problema. Successivamente abbiamo creato il nuovo codificatore, a partire dal vecchio codificatore, utilizzando l'Answer Set Programming. In seguito abbiamo creato l'applicazione web in modo da rendere semplice e intuitivo l'utilizzo del nuovo codificatore e infine abbiamo effettuato l'analisi dei risultati.
Le analisi dei risultati sono state fatte utilizzando entrambi i codificatori, prendendo in considerazione tutti i giorni di pianificazione, per vedere se la qualità delle soluzioni diminuisse a seguito dell'aumento della complessità della versione estesa del problema.
Dalle analisi effettuate abbiamo visto che, nonostante la presenza di vincoli aggiuntivi nella versione estesa del problema, le modifiche al data model e l'aggiunta di nuovi vincoli e ottimizzazioni, la soluzione che otteniamo è ben ottimizzata. In particolare, abbiamo visto che il nuovo codificatore riesce ad assegnare molti più letti rispetto al vecchio codificatore, le preferenze sulle risorse sono sempre rispettate e i pazienti che effettuano il prelievo sono sempre ben distribuiti, il tutto mantenendo quasi inalterate le performance, ovvero i tempi di grounding e di risoluzione. Quindi, la qualità delle soluzioni del nuovo codificatore non diminuisce a seguito dell'aumento della complessità della versione estesa del problema. In conclusione, possiamo dire che l'Answer Set Programming è uno strumento di Intelligenza Artificiale valido per affrontare problemi di pianificazione complessi.
Per quanto riguarda i possibili lavori futuri, attualmente gli atomi di input vengono generati da uno script python in maniera separata: potrebbe essere utile avere la possibilità di generare gli input direttamente dalla Web Application. Inoltre si potrebbe aggiungere la funzionalità che permetta di modificare l'arco temporale di pianificazione. Come ulteriore aggiunta si potrebbe anche sviluppare la funzionalità che permetta di effettuare una ripianificazione, in

questo modo se ho dei pazienti che stanno rispondendo male ad un determinato trattamento, posso modificare la pianificazione.

Bibliografia

Alviano, M., Amendola, G., Dodaro, C., Leone, N., Maratea, M., and Ricca, F. (2019). Evaluation of disjunctive programs in WASP. In Balduccini, M., Lierler, Y., and Woltran, S., editors, *LPNMR*, volume 11481 of *LNCS*, pages 241–255. Springer.

Alviano, M., Bertolucci, R., Cardellini, M., Dodaro, C., Galatà, G., Khan, M. K., Maratea, M., Mochi, M., Morozan, V., Porro, I., and Schouten, M. (2020). Answer set programming in healthcare: Extended overview. In *IPS and RCRA 2020*, volume 2745 of *CEUR Workshop Proceedings*. CEUR-WS.org.

Alviano, M., Dodaro, C., and Maratea, M. (2017). An advanced answer set programming encoding for nurse scheduling. In *AI*IA*, volume 10640 of *LNCS*, pages 468–482. Springer.

Alviano, M., Dodaro, C., and Maratea, M. (2018). Nurse (re)scheduling via answer set programming. *Intelligenza Artificiale*, 12(2):109–124.

Amendola, G. (2018). Solving the stable roommates problem using incoherent answer set programs. In *RiCeRcA@AI*IA*, volume 2272 of *CEUR Workshop Proceedings*. CEUR-WS.org.

Buccafurri, F., Leone, N., and Rullo, P. (2000). Enhancing Disjunctive Datalog by Constraints. *IEEE Transactions on Knowledge and Data Engineering*, 12(5):845–860.

Calimeri, F., Faber, W., Gebser, M., Ianni, G., Kaminski, R., Krennwallner, T., Leone, N., Maratea, M., Ricca, F., and Schaub, T. (2019). Asp-core-2 input language format. *Theory and Practice of Logic Programming*, 20:1–16.

Calimeri, F., Gebser, M., Maratea, M., and Ricca, F. (2014). The design of the fifth answer set programming competition. *CoRR*, abs/1405.3710.

Cardellini, M., Nardi, P. D., Dodaro, C., Galatà, G., Giardini, A., Maratea, M., and Porro, I. (2021). A two-phase ASP encoding for solving rehabilitation scheduling. In Moschoyiannis, S., Peñaloza, R., Vanthienen, J., Soylu, A., and Roman, D., editors, *Proceedings of the 5th International Joint Conference on Rules and Reasoning (RuleML+RR 2021)*, volume 12851 of *Lecture Notes in Computer Science*, pages 111–125. Springer.

Dodaro, C., Galatà, G., Khan, M. K., Maratea, M., and Porro, I. (2019). An ASP-based solution for operating room scheduling with beds management. In Fodor, P., Montali, M., Calvanese, D., and Roman, D., editors, *Proceedings of the Third International Joint Conference on Rules and Reasoning (RuleML+RR 2019)*, volume 11784 of *Lecture Notes in Computer Science*, pages 67–81. Springer.

Dodaro, C., Galatà, G., Grioni, A., Maratea, M., Mochi, M., and Porro, I. (2021). An asp-based solution to the chemotherapy treatment scheduling problem. *Theory and Practice of Logic Programming*, 21:1–17.

Dodaro, C. and Maratea, M. (2017). Nurse scheduling via answer set programming. In *LPNMR*, volume 10377 of *LNCS*, pages 301–307. Springer.

Faber, W., Pfeifer, G., and Leone, N. (2011). Semantics and complexity of recursive aggregates in answer set programming. *Artificial Intelligence*, 175(1):278–298.

Gebser, M., Kaufmann, B., and Schaub, T. (2012). Conflict-driven answer set solving: From theory to practice. *Artificial Intelligence*, s 187–188:52–89.

Gebser, M., Maratea, M., and Ricca, F. (2017a). The design of the seventh answer set programming competition. In Balduccini, M. and Janhunen, T., editors, *LPNMR*, volume 10377 of *Lecture Notes in Computer Science*, pages 3–9. Springer.

Gebser, M., Maratea, M., and Ricca, F. (2017b). The sixth answer set programming competition. *Journal of Artificial Intelligence Research*, 60:41–95.

Gebser, M., Maratea, M., and Ricca, F. (2020). The seventh answer set programming competition: Design and results. *Theory Pract. Log. Program.*, 20(2):176–204.

Gebser, M., Obermeier, P., Schaub, T., Ratsch-Heitmann, M., and Runge, M. (2018). Routing driverless transport vehicles in car assembly with answer set programming. *Theory and Practice of Logic Programming*, 18(3-4):520–534.

Gelfond, M. and Lifschitz, V. (1988). The stable model semantics for logic programming. In *Proceedings of the Fifth International Conference and Symposium , Seattle, Washington, August 15-19, 1988 (2 Volumes)*, pages 1070–1080. MIT Press.

Gelfond, M. and Lifschitz, V. (1991). Classical negation in logic programs and disjunctive databases. *New Generation Comput.*, 9(3/4):365–386.

Huggins, A., Claudio, D., and Pérez, E. (2014). Improving resource utilization in a cancer clinic: An optimization model. In *IIE Annual Conference and Expo 2014*.

Ricca, F., Grasso, G., Alviano, M., Manna, M., Lio, V., Iiritano, S., and Leone, N. (2012). Team-building with answer set programming in the Gioia-Tauro seaport. *Theory and Practice of Logic Programming*, 12(3):361–381.

Sevinc, S., Sanli, U. A., and Goker, E. (2013). Algorithms for scheduling of chemotherapy plans. *Computers in Biology and Medicine*, 43(12):2103–2109.

Turkcan, A., Zeng, B., and Lawley, M. (2010). Chemotherapy operations planning and scheduling. *IIE Thansactions on Healthcare Systems Engineering*, 2.