



Politecnico di Torino

Corso di Laurea Triennale in Ingegneria Gestionale

Classe L8 – Ingegneria dell’Informazione

A.A. 2023/2024

Turni 24H

Software per la gestione di turni

Relatore:

Prof. Fulvio Corno

Candidato:

Ravalli Lorenzo

INDICE

CAPITOLO 1: PROPOSTA DI PROGETTO	3
1.1 Studente proponente.....	3
1.2 Titolo della proposta.....	3
1.3 Descrizione del problema proposto	3
1.4 Descrizione della rilevanza gestionale del problema	3
1.5 Descrizione dei data-set per la valutazione	4
1.6 Descrizione preliminare degli algoritmi coinvolti	4
1.7 Descrizione preliminare delle funzionalità previste per l'applicazione software.....	4
CAPITOLO 2: DESCRIZIONE DETTAGLIATA DEL PROBLEMA	5
2.1 L'importanza dell'assegnazione dei turni	5
2.2 Input, output, criticità e rilevanza	6
2.3 Caso in esame	6
CAPITOLO 3: DESCRIZIONE DEL DATA-SET.....	7
3.1 Descrizione Data-set iniziale	7
3.2 Modifiche apportate al database iniziale	7
3.3 Diagramma ER	8
CAPITOLO 4: STRUTTURE DATI E DEGLI ALGORITMI UTILIZZATI.....	9
4.1 Strutture dati	9
4.2 Algoritmi principali.....	9
4.3 Diagramma classi principali	16
CAPITOLO 5: OUTPUT GRAFICO	17
5.1 Link YouTube.....	17
5.2 Videata all'apertura dell'applicazione.....	17
5.3 Videata risultati.....	18
5.4 Videata turni	19
CAPITOLO 6: RISULTATI E CONCLUSIONI	20
6.1 Risultati ottenuti	20
6.2 Limiti del programma	20
CAPITOLO 7: LICENZA	21

CAPITOLO 1: PROPOSTA DI PROGETTO

1.1 Studente proponente

Lorenzo Ravalli

1.2 Titolo della proposta

Turni 24H. Software per la gestione di turni.

1.3 Descrizione del problema proposto

La proposta mira a risolvere il problema algoritmico dell'assegnazione a turno mensile di vari dipendenti. Nello specifico il contesto lavorativo riguarderà turnazioni di 24 ore continuative con differenti richieste per ogni turno.

1.4 Descrizione della rilevanza gestionale del problema

Il problema ha un elevata rilevanza gestionale, essendo un problema di "scheduling". Un'adeguata turnazione offre numerosi vantaggi. Innanzitutto, permette di ottimizzare l'allocazione delle risorse umane, garantendo che vi siano abbastanza dipendenti presenti nei momenti di maggiore affluenza o durante le fasi cruciali del processo produttivo. Ciò migliora l'efficienza operativa e la soddisfazione dei clienti.

Inoltre, lo scheduling considera le preferenze dei dipendenti, contribuendo alla loro soddisfazione e alla riduzione del turnover.

Nel settore dei servizi, come ristoranti o alberghi, lo scheduling può influenzare direttamente la qualità del servizio offerto. In ambiti complessi come l'assistenza sanitaria, la gestione accurata dei turni è cruciale per garantire una copertura continua e la sicurezza dei pazienti.

1.5 Descrizione dei data-set per la valutazione

Il database utilizzato sarà uno di quelli proposti dal sito:
["http://www.schedulingbenchmarks.org/".](http://www.schedulingbenchmarks.org/) Verrà preso in esame uno dei dataset riguardanti 18 infermieri, che si alterneranno in 4 settimane, per 3 turni di lavoro al giorno. Il dataset avrà una tabella riguardante i dipendenti, con orari di lavoro richiesti min e max, una tabella sui turni effettivi durante il mese, una tabella per la gestione dei giorni "off" di non disponibilità, e giorni di richiesta turno per i vari dipendenti.

1.6 Descrizione preliminare degli algoritmi coinvolti

Il software si baserà su una ricorsione che, analizzando i vari vincoli, cercherà la migliore soluzione possibile per gestire i turni. I vincoli saranno relativi al sovraccarico di lavoro (es. se notte smonto, se pomeriggio non mattina, ogni 5 giorni 2 di riposo ecc.). Si cercherà anche di rispettare i day off e le varie richieste dei dipendenti ove possibile e richiesto. Inoltre, tutti dovranno rispettare il numero di ore min/max cercando di evitare le disparità.

La ricorsione terminerà in caso di soluzione ottima o dopo un numero di tentativi massimi. Le soluzioni saranno calcolate in base ai turni coperti e alle ore rispettate per i dipendenti. I dipendenti disponibili verranno segnalati da un metodo che valuta i vincoli.

1.7 Descrizione preliminare delle funzionalità previste per l'applicazione software

Il programma proporrà una finestra di Home per il calcolo dei turni, con alcune opzioni come la possibilità di escludere a priori le richieste di turni dei dipendenti, una finestra riepilogativa con gli eventuali turni calcolati per dipendente e, infine, una parte dedicata ai responsi con alcune valutazioni, come ad esempio se con i turni richiesti tutti lavorano le ore necessarie, vi è una carenza di personale ecc.

CAPITOLO 2: DESCRIZIONE DETTAGLIATA DEL PROBLEMA

2.1 L'importanza dell'assegnazione dei turni

L'assegnazione a turni è una pratica essenziale che va oltre il mondo delle aziende e coinvolge anche settori pubblici come la sanità. In entrambi i contesti, l'efficace pianificazione dei turni gioca un ruolo cruciale nell'ottimizzare le risorse umane, garantendo che i servizi siano erogati in modo efficiente e continuo.

Nelle aziende l'assegnazione a turni consente di massimizzare la produzione e minimizzare i costi. Le aziende devono affrontare fluttuazioni nella domanda e gestire le risorse in base a picchi e cali di attività. Una pianificazione dei turni accurata garantisce che vi siano sempre abbastanza dipendenti disponibili nei momenti di maggiore necessità, evitando al contempo sovrapposizioni di personale quando la richiesta è bassa.

Ma l'importanza dei turni non si ferma qui. Nella sanità, ad esempio, la pianificazione dei turni è vitale per garantire la copertura 24/7 nei reparti ospedalieri e nei servizi di emergenza. In questo settore la vita delle persone è in gioco, quindi una corretta assegnazione dei turni può fare la differenza.

Inoltre, nell'ambito pubblico, come nella pubblica amministrazione o nei servizi di emergenza, i dipendenti spesso operano in turni per garantire la continuità dei servizi. Questo vale anche per le forze dell'ordine e i vigili del fuoco, che devono essere pronti a rispondere alle emergenze in qualsiasi momento.

In generale, una pianificazione dei turni ben gestita porta a una maggiore soddisfazione dei dipendenti, a una migliore qualità dei servizi e alla riduzione dei costi operativi. È un elemento chiave per garantire che le risorse umane siano utilizzate in modo efficiente, indipendentemente dal settore in cui si opera. Pertanto, l'assegnazione a turni è fondamentale per garantire l'efficienza, la continuità e la qualità delle prestazioni in molteplici contesti, sia aziendali che pubblici.

2.2 Input, output, criticità e rilevanza

Per la divisione in turni sono necessari diversi input chiave. Innanzitutto, occorre comprendere le esigenze operazionali dell'organizzazione, tra cui la quantità di lavoro da svolgere in determinati momenti e le competenze richieste. Gli input includono anche la disponibilità dei dipendenti, tenendo conto delle loro preferenze, dei limiti normativi sulle ore di lavoro e delle richieste di ferie o permessi.

Gli output della pianificazione dei turni includono la creazione di orari dettagliati che indicano quando e dove ciascun dipendente deve lavorare. Questi orari dovrebbero garantire una distribuzione equa del lavoro e una copertura adeguata, mantenendo al contempo un equilibrio tra lavoro e vita privata per i dipendenti. Gli output possono essere sotto forma di calendari o sistemi digitali di gestione dei turni, che facilitano il monitoraggio e la comunicazione tra i dipendenti.

Tuttavia, ci sono alcune criticità da considerare. La pianificazione dei turni può diventare complicata quando si devono soddisfare molteplici requisiti, come le leggi sul lavoro o le preferenze dei dipendenti.

Le sovrapposizioni dei turni o la mancanza di copertura possono causare problemi operativi. Per questo un algoritmo che si avvicina ad una buona soluzione per la pianificazione dei turni è di cruciale importanza in quanto migliora l'efficienza, la soddisfazione dei dipendenti e la qualità dei servizi.

2.3 Caso in esame

L'obiettivo della proposta è affrontare l'importante sfida algoritmica legata alla pianificazione mensile dei turni per una varietà di dipendenti. La situazione specifica riguarda un ambiente lavorativo in cui i dipendenti devono essere assegnati a turni che durano 8 ore, consecutivi nell'arco della giornata e che presentano diverse richieste ed esigenze specifiche per ciascun turno. In altre parole, si cerca di sviluppare un algoritmo che consenta di organizzare in modo efficiente e soddisfacente le turnazioni mensili per il personale coinvolto in un contesto operativo complesso, come quello ospedaliero, tenendo conto delle diverse variabili e delle esigenze specifiche legate a ciascun turno.

CAPITOLO 3: DESCRIZIONE DEL DATA-SET

3.1 Descrizione Data-set iniziale

IL Data set iniziale è stato preso dal sito <http://www.schedulingbenchmarks.org/>, nella categoria “nurse”, alla voce “Instance6”. Il database prevede l’assegnazione a turno di 18 infermieri, in 3 turni giornalieri continui di 8 ore ciascuno, per una durata complessiva di 4 settimane.

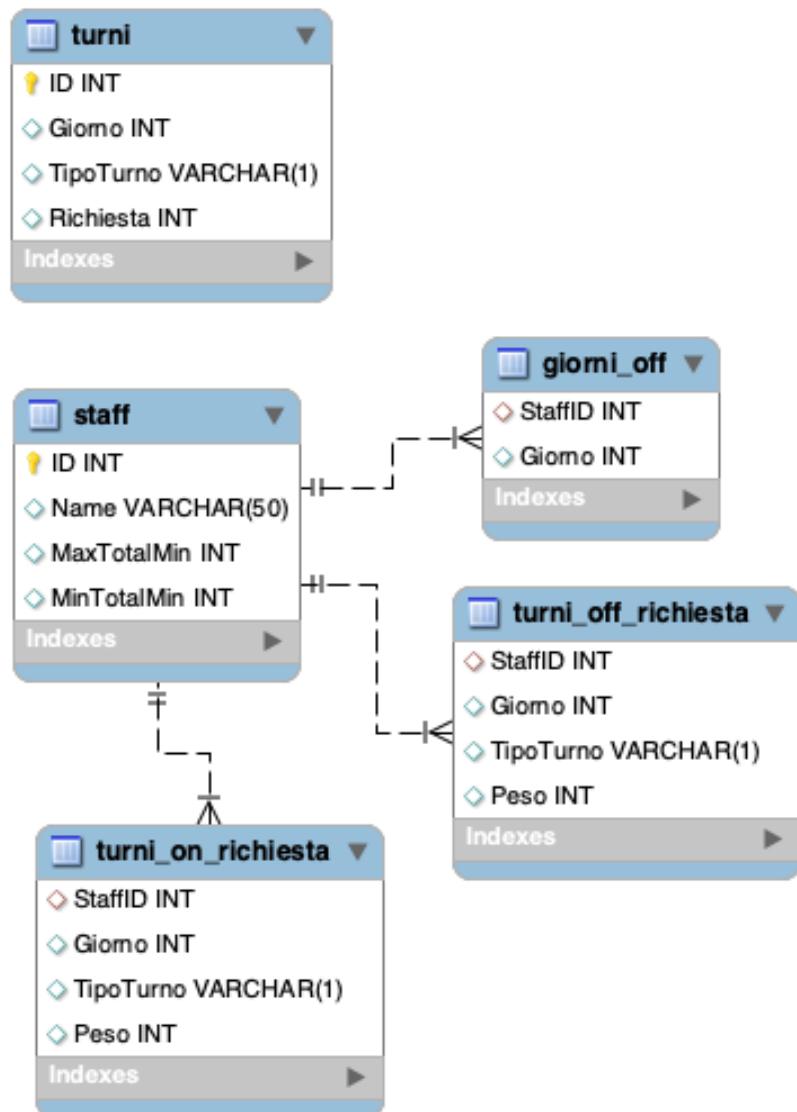
In ordine dal data set proposto sono state prese in considerazioni le seguenti informazioni:

- I dati dei 18 dipendenti con un ID numerico assegnato a ciascuno, il numero massimo e minimo di minuti lavorativi per il quale sono disponibili nel corso delle 4 settimane.
- I giorni Off per ciascun dipendente, come ad esempio ferie o giorni in cui il dipendente non è disponibile.
- Le richieste a turno On e Off da parte degli infermieri, che indicano rispettivamente preferenze di essere messi in uno specifico turno o se possibile non partecipare a determinati turni. Questi dati erano divisi in ordine di importanza, che viene considerata dall’algoritmo; dunque, anche il dato sul peso viene considerato.
- Gli ID e i giorni relativi agli 84 turni, del quale vengono considerati anche le esigenze relative al numero dei dipendenti richiesti dall’ospedale per turno.

3.2 Modifiche apportate al database iniziale

Nel data set, prima di essere trasformato da foglio di testo a database SQL, sono state apportate alcune modifiche. Gli ID dei turni sono stati riordinati da 0 ad 83 (prima erano divisi solo per giorni). Ai dipendenti è stato assegnato un nome di fantasia per rendere più realistico il caso e di più facile lettura.

3.3 Diagramma ER



CAPITOLO 4: STRUTTURE DATI E DEGLI ALGORITMI UTILIZZATI

4.1 Strutture dati

L'applicazione è stata realizzata in linguaggio Java e implementa il pattern MVC (Model View Controller) e il pattern DAO (Data Access Object).

Per realizzare l'interfaccia grafica in JavaFX è stato utilizzato il software SceneBuilder.

Il progetto si compone di tre package:

- *it.polito.Turni24H*: qui sono presenti tre classi, il *Main*, da cui viene eseguita l'applicazione, l'Entry Point e l'*FXMLController*, in cui sono definiti i metodi necessari per garantire una corretta interazione con l'utente.
- *it.polito.Turni24H.db*: in questo package è compresa la classe *ConnectDB*, che implementa la connessione al database e la classe *Turni24HDao*, che contiene i metodi necessari ad ottenere informazioni dal data-set, come in turni o il personale.
- *it.polito.Turni24H.model*: in questo package sono presenti quattro classi rappresentanti gli oggetti con cui abbiamo a che fare nel programma, ovvero i dipendenti (classe *Staff*), i turni (classe *Turno*), le richieste turni On e Off (classe *RichiestaTurni*) e i giorni Off (classe *GiorniOff*). Il package contiene la classe *Model*, in cui sono implementati tutti i metodi che effettuano operazioni sui dati, tra questi vi è l'algoritmo ricorsivo che permette all'applicazione di assegnare i turni. Inoltre, troviamo una classe *Accoppiamenti* che permette di accoppiare turni e dipendenti, una classe di nome *TurniAssegnatiSettimanali*, che divide i turni in settimane e infine una classe *Risultati* che è utile al programma per la fase video riguardanti risultati.

4.2 Algoritmi principali

Il principale algoritmo dell'applicazione è la ricorsione. Gli algoritmi ricorsivi semplificano la soluzione di problemi complessi, decomponendoli in problemi più piccoli, facilitando la comprensione, la manutenzione e la riusabilità del codice. In questo caso si divide il problema dell'assegnazione a turni in tante piccole ricerche, esse mirano ad assegnare ad un turno un dipendente disponibile. Fondamentale è il ruolo svolto dal metodo *isDisponibile* che controlla se il dipendente è disponibile ad entrare nel turno per il quale è stato richiesto. Assegnato un dipendente la ricorsione

va avanti tentando di assegnare altri dipendenti ai turni per il quale vi è richiesta. Ovviamente per cercare la soluzione migliore si utilizza il famoso principio del backtracking, una tecnica di ricerca ricorsiva che esplora tutte le opzioni possibili per risolvere un problema, tornando indietro quando si incontra un ostacolo o si trova una soluzione. Le soluzioni trovate sono iscritte in una lista di Accoppiamenti (turni e dipendenti). Un ruolo fondamentale è svolto dal metodo *calcolaPunteggio* che calcola un punteggio per ogni soluzione trovata cercando di rendere i turni equi per tutti i dipendenti.

La ricorsione è definita dai seguenti metodi:

```

//prima parte metodo ricorsivo
public List<Accoppiamenti> cercaLista(int profondita, boolean richiestiOff, boolean richiestiOn, int weekend, int straordinario){

    List<Accoppiamenti> parziale = new ArrayList<>();
    miglioriAccoppiamenti = new ArrayList<>();
    solTrovata = false;
    migliorPunteggio = 1000000000;

    if(richiestiOn) {
        aggiungiTurni(parziale, richiestiOff, weekend, straordinario);      //aggiungo i turni richiesti
    }

    //prove in base alla richiesta dell'utente per il calcolo turni
    for(int i = 0; i < profondita ; i++) {
        contatore = 0;
        Collections.shuffle(turni);      //ordino casualmente i turni
        cerca(parziale, richiestiOff, weekend, straordinario, turniTualiRichiesti, migliorPunteggio);
        System.out.println(i+" = "+ miglioriAccoppiamenti.size()+" di punteggio: " + this.migliorPunteggio);
    }

    System.out.println("BEST = "+ miglioriAccoppiamenti.size() + " di punteggio: " + this.migliorPunteggio);
    return miglioriAccoppiamenti;
}

//metodo che aggiunge i turni su eventuale richiesta
private void aggiungiTurni(List<Accoppiamenti> parziale,boolean richiestiOff, int weekend, int straordinario){
    for(Turno t : turni) {
        if(turniCompletati(t, parziale)==false) {
            for(RichiestaTurni r : this.richiesteTurni) {
                if(r.isOffDay()==false && r.getGiorno() == t.getGiorno() &&
                    r.getTipoTurno().compareTo(t.getTipoTurno()) == 0) {
                    for(Staff s : allStaff) {
                        if(s.getId() == r.getIdStaff()) {
                            if(isDisponibile(parziale, s, t, richiestiOff, weekend, straordinario)
                                && turniCompletati(t, parziale)==false) {
                                Accoppiamenti a = new Accoppiamenti(s,t);
                                parziale.add(a);
                                break;
                            }
                        }
                    }
                }
            }
        }
    }
}

```

```

//ricorsione
private void cerca(List<Accoppiamenti> parziale, boolean richiestiOff, int weekend, int straordinario,int turniDisponibili, int punteggio) {
    //cond Terminazione con soluzione ottima o al raggiungimento di un limite di tentativi con un contatore
    if(solTrovata || contatore == 15000)
        return;

    contatore++;
    int p = calcolaPunteggio(parziale);

    if(p==0) {
        this.miglioriAccoppiamenti.clear();
        this.miglioriAccoppiamenti.addAll(parziale);
        this.solTrovata = true;
        return;
    }

    if(p < punteggio) {
        this.miglioriAccoppiamenti.clear();
        this.miglioriAccoppiamenti.addAll(parziale);
        this.migliorPunteggio = p;
        punteggio = p;
    }
}

//prima vengono aggiunti dipendenti senza straordinario
for(Turno t : turni) {
    if(turniCompletati(t, parziale)==false) {
        for(Staff s : allStaff) {
            if(isDisponibile(parziale, s, t, richiestiOff, weekend, 0)
                && turniCompletati(t, parziale)==false) {
                Accoppiamenti a = new Accoppiamenti(s,t);
                parziale.add(a);
                cerca(parziale, richiestiOff, weekend, straordinario, turniDisponibili, punteggio);
                parziale.remove(a);
            }
        }
    }
    //nel caso in cui gli straordinari siano attivi e i turni non sono completati
    if(straordinario !=0){
        for(Turno t : turni) {
            if(turniCompletati(t, parziale)==false) {
                for(Staff s : allStaff) {
                    if(isDisponibile(parziale, s, t, richiestiOff, weekend, straordinario)
                        && turniCompletati(t, parziale)==false) {
                        Accoppiamenti a = new Accoppiamenti(s,t);
                        parziale.add(a);
                        cerca(parziale, richiestiOff, weekend, straordinario, turniDisponibili, punteggio);
                        parziale.remove(a);
                    }
                }
            }
        }
    }
}
}

```

Il primo metodo, *cercaLista*, è il metodo che poi verrà effettivamente richiamato per ottenere i turni. Esso restituisce una lista di accoppiamenti (turni-staff). Nella prima parte del metodo vengono eseguiti alcuni settaggi. Successivamente se l'utente richiede di rispettare le richieste a turno, il metodo le assegna in automatico tramite il metodo *aggiungiTurni*. Il metodo poi effettua un'iterazione nel quale prova su più tentativi l'effettivo metodo ricorsivo cercando la migliore soluzione (il numero dei tentativi viene impostato dall'utente). Prima di ogni iterazione vengono ordinati in modo casuale i turni, in modo da non partire ad occuparli in ordine sequenziale ma casuale, questo aumenta la precisione trovando sempre soluzioni molto diverse tra loro. Infine, il metodo *cercaLista* da una lista di accoppiamenti, *best*, con il migliore punteggio.

La seconda parte, chiamata *cerca*, è l'effettivo metodo ricorsivo. La condizione di terminazione del metodo è basata su un contatore, che tiene conto di un numero di prove sufficienti a dare una buona soluzione. Altra condizione di terminazione è la soluzione ottima, ma per questo tipo di problema è praticamente impossibile trovarla.

Il punteggio assegnato alla lista è calcolato dal metodo *calcolaPunteggio* che vedremo in seguito.

In sostanza l'algoritmo prova ad assegnare un dipendente ad ogni turno, verifica se esso è disponibile, se sì lo mette a turno e va avanti con la ricorsione, altrimenti cerca qualcun altro disponibile. Se nessun dipendente è disponibile si passa al turno successivo. Dopo queste iterazioni, se l'utente ha dato disponibilità agli straordinari, il programma proverà ad inserire nei turni non completati i dipendenti che sono disponibili a fare straordinari.

Il metodo che calcola il punteggio invece è:

```
//metodo che confronta le soluzioni della ricorsione
private int calcolaPunteggio(List<Accoppiamenti> lista) {
    int punteggio = 0;

    for(Turno t : turni) {
        int personaleATurno = 0;
        for(Accoppiamenti a : lista) {
            if(t.equals(a.getTurno())) {
                personaleATurno++;
            }
        }
        if(t.getRichiesta()>personaleATurno)
            punteggio += (Math.pow(10, (t.getRichiesta()-personaleATurno)) + 10); //se turno non completato
    }
    for(Staff s : allStaff) {
        int turniFatti = 0;
        for(Accoppiamenti a : lista) {
            if(s.equals(a.getStaff())) {
                turniFatti++;
            }
        }
        if(((s.getMinTotalMin() /60) /8) > turniFatti) //se staff fa meno turni del dovuto
            punteggio += (Math.pow(10, (((s.getMinTotalMin() /60)/8)-turniFatti)) + 10);
        if(((s.getMaxTotalMin() /60) /8) < turniFatti) //se staff fa straordinari
            punteggio += Math.pow(10, (turniFatti - ((s.getMaxTotalMin() /60) /8)));
    }
    return punteggio;
}
```

Questo metodo è cardine del programma in quanto valuta le varie soluzioni trovate. Il metodo calcola un punteggio che verrà assegnato alla lista. Più basso è il punteggio migliore sarà la soluzione trovata. Si da priorità ai turni, se non sono completati si alza di molto il punteggio, ogni dipendente che manca al turno aumenta il punteggio esponenzialmente. Poi si valutano i dipendenti se fanno turni in meno dei richiesti o fanno straordinari, il metodo alza il punteggio anche qui in maniera esponenziale per ogni turno mancante o in più.

Il metodo che invece controlla i vari vincoli e ci dice se il dipendente è disponibile per il turno è:

```

//metodo che controlla se il dipendente è disponibile a entrare a far parte del turno richiesto
private boolean isDisponibile(List<Accoppiamenti> parziale, Staff s, Turno t, boolean richiestiOff, int weekend, int straordinari) {
    List<Turno> turniAssegnati = new ArrayList<>();
    for(Accoppiamenti a : parziale) {
        if(a.getStaff().equals(s))
            turniAssegnati.add(a.getTurno());
    }

    //supero le ore massime
    int minutiFatti = turniAssegnati.size() * 60 * 8;
    if(minutiFatti >= (s.getMaxTotalMin() + (60*8*straordinari))) {
        return false;
    }

    //se turno lo stesso giorno non assegno
    for(Turno tt : turniAssegnati) {
        if(tt.getGiorno() == t.getGiorno())
            return false;
    }

    //max 5 gg a settimana (5 su 7) // comprende max 5 gg e poi pausa
    int ggPrimaSett = 0;
    for(Turno tt : turniAssegnati) {
        if(tt.getGiorno() == t.getGiorno() - 1 ||
           tt.getGiorno() == t.getGiorno() -2 ||
           tt.getGiorno() == t.getGiorno() -3 ||
           tt.getGiorno() == t.getGiorno() -4 ||
           tt.getGiorno() == t.getGiorno() -5 ||
           tt.getGiorno() == t.getGiorno() -6 ||
           tt.getGiorno() == t.getGiorno() -7 )
            ggPrimaSett++;
    }
    if(ggPrimaSett==5)
        return false;

    //max 2 notti di fila
    if(t.getTipoTurno().compareTo("S")==0 ) {
        int ggPrimaN = 0;
        int ggDopoN = 0;
        int dueggPrimaN = 0;
        int dueggDopoN = 0;

        for(Turno tt : turniAssegnati) {
            if(((tt.getGiorno() == (t.getGiorno() - 1) && tt.getTipoTurno().compareTo("S") == 0)))
                ggPrimaN++;
            if(((tt.getGiorno() == (t.getGiorno() + 1) && tt.getTipoTurno().compareTo("S") == 0)))
                ggDopoN++;
            if(((tt.getGiorno() == (t.getGiorno() - 2) && tt.getTipoTurno().compareTo("S") == 0)))
                dueggPrimaN++;
            if(((tt.getGiorno() == (t.getGiorno() + 2) && tt.getTipoTurno().compareTo("S") == 0)))
                dueggDopoN++;
        }
        if((ggPrimaN == 1 && dueggPrimaN == 1) ||
           (ggPrimaN == 1 && ggDopoN == 1) ||
           (ggDopoN ==1 && dueggDopoN == 1))
            return false;
    }
}

```

```

//se notte prima no mattina o pomeriggio
if(t.getTipoTurno().compareTo("M")==0 || t.getTipoTurno().compareTo("P")==0) {
    for(Turno tt : turniAssegnati) {
        if((tt.getGiorno() == t.getGiorno() - 1 && tt.getTipoTurno().compareTo("S") == 0))
            return false;
    }
}
if(t.getTipoTurno().compareTo("S")==0 ) { //aggiungere se già assegnato
    for(Turno tt : turniAssegnati) {
        if((tt.getGiorno() == t.getGiorno() + 1 && tt.getTipoTurno().compareTo("M") == 0))
            return false;
        if((tt.getGiorno() == t.getGiorno() + 1 && tt.getTipoTurno().compareTo("P") == 0))
            return false;
    }
}

//se pomeriggio prima no mattina
if(t.getTipoTurno().compareTo("M")==0) {
    for(Turno tt : turniAssegnati) {
        if(tt.getGiorno() == t.getGiorno() - 1 && tt.getTipoTurno().compareTo("P") == 0)
            return false;
    }
}
if(t.getTipoTurno().compareTo("P")==0) {
    for(Turno tt : turniAssegnati) {
        if(tt.getGiorno() == t.getGiorno() + 1 && tt.getTipoTurno().compareTo("M") == 0)
            return false;
    }
}

// weekend (quanti giorni di weekend a riposo si vogliono minimo su 8 gg)
if(t.isWeekEnd()) {
    int weekendTot = 0;
    for(Turno tt : turniAssegnati) {
        if(tt.isWeekEnd())
            weekendTot++;
    }
    if(weekendTot >= (8 - weekend)) {
        return false;
    }
}

//Rispetto giorni off
for(GiorniOff gg : this.giorniOff) {
    if(gg.getIdStaff()==s.getId() && gg.getGiorno() == t.getGiorno())
        return false;
}
//Rispetto richieste
if(richiestiOff) {
    for(RichiestaTurni r : this.richiesteTurni) {
        if(r.isOffDay() && r.getIdStaff()==s.getId()
           && r.getGiorno() == t.getGiorno() &&
           r.getTipoTurno().compareTo(t.getTipoTurno())==0)
            return false;
    }
}

```

```

//Non supero 1/3 dei turni in notti
if(t.getTipoTurno().compareTo("S")==0) {
    int sere = 0;
    for(Turno tt : turniAssegnati) {
        if(tt.getTipoTurno().compareTo("S") == 0)
            sere++;
    }
    if((sere* 60 * 8) >= ((s.getMaxTotalMin() + (60*8*straordinari))/3))
        return false;
}
//Non supero 1/2 dei turni in mattina
if(t.getTipoTurno().compareTo("M")==0) {
    int mattine = 0;
    for(Turno tt : turniAssegnati) {
        if(tt.getTipoTurno().compareTo("M") == 0)
            mattine++;
    }
    if((mattine* 60 * 8) >= ((s.getMaxTotalMin() + (60*8*straordinari))/2))
        return false;
}
//Non supero 1/2 dei turni in pomeriggio
if(t.getTipoTurno().compareTo("P")==0) {
    int pomeriggi = 0;
    for(Turno tt : turniAssegnati) {
        if(tt.getTipoTurno().compareTo("P") == 0)
            pomeriggi++;
    }
    if((pomeriggi* 60 * 8) >= ((s.getMaxTotalMin() + (60*8*straordinari))/2))
        return false;
}
return true;
}

```

Questo è l'ultimo metodo di fondamentale importanza. Questo metodo, *isDisponibile*, valuta, ogni volta che un dipendente è richiamato nella ricorsione, se esso è disponibile ad entrare a far parte del turno. Praticamente tiene conto di tutti i vincoli.

Il suo funzionamento è semplice, controlla se rispetta un vincolo, se non lo rispetta termina il metodo e da un valore booleano *false*. I vincoli sono in ordine di importanza e in base alla facilità con cui possono essere violati, così da ottimizzare il tempo di calcolo.

I vincoli da rispettare sono:

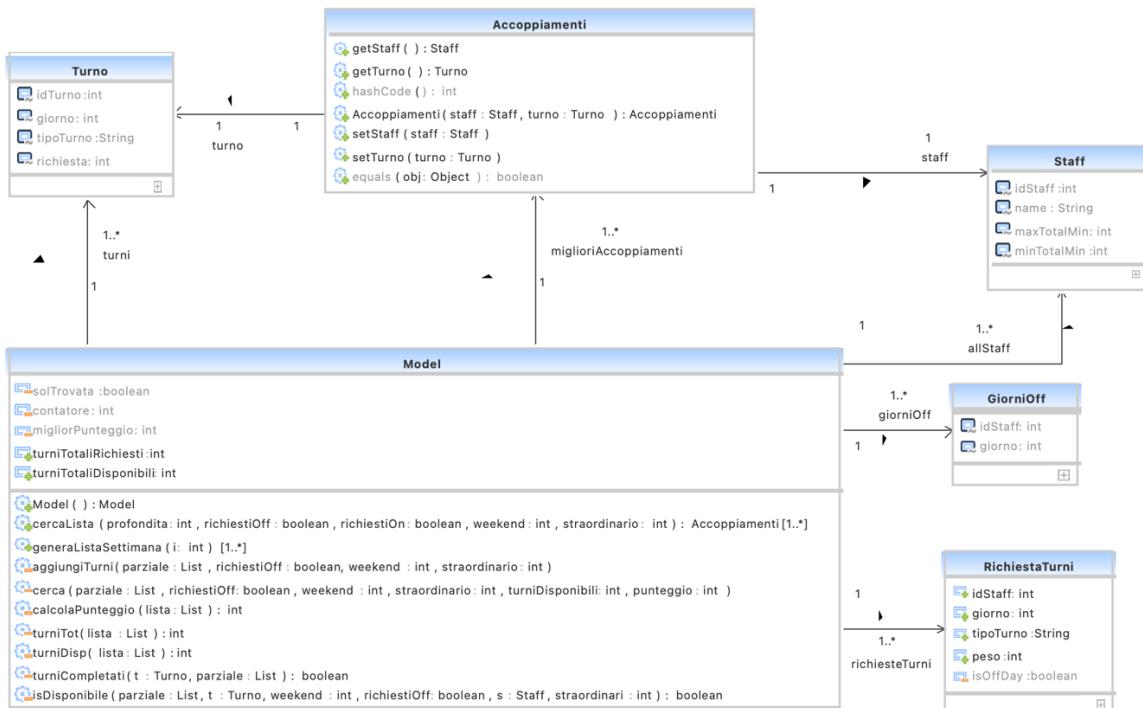
- Non superare le ore massime;
- Non assegnare due o tre turni lo stesso giorno;
- Non superare i 5 giorni a settimana;
- Vietato superare le due notti di fila;
- Rispettare lo smonto dopo la notte (no mattina o pomeriggio il giorno dopo);
- Mattina vietata se il giorno prima il dipendente ha fatto pomeriggio;

- Rispettare i giorni di weekend massimi per dipendente (dato impostato dall'utente);
- Rispettare i giorni Off (ferie, indisponibilità);
- Rispettare le richieste di giorni Off (se impostato dall'utente).

Altri tre vincoli sono stati aggiunti per bilanciare il carico dei turni tra i vari dipendenti:

- Non superare un terzo dei propri turni in notti;
- Non superare la metà dei propri turni in mattine;
- Non superare la metà dei propri turni in pomeriggi.

4.3 Diagramma classi principali

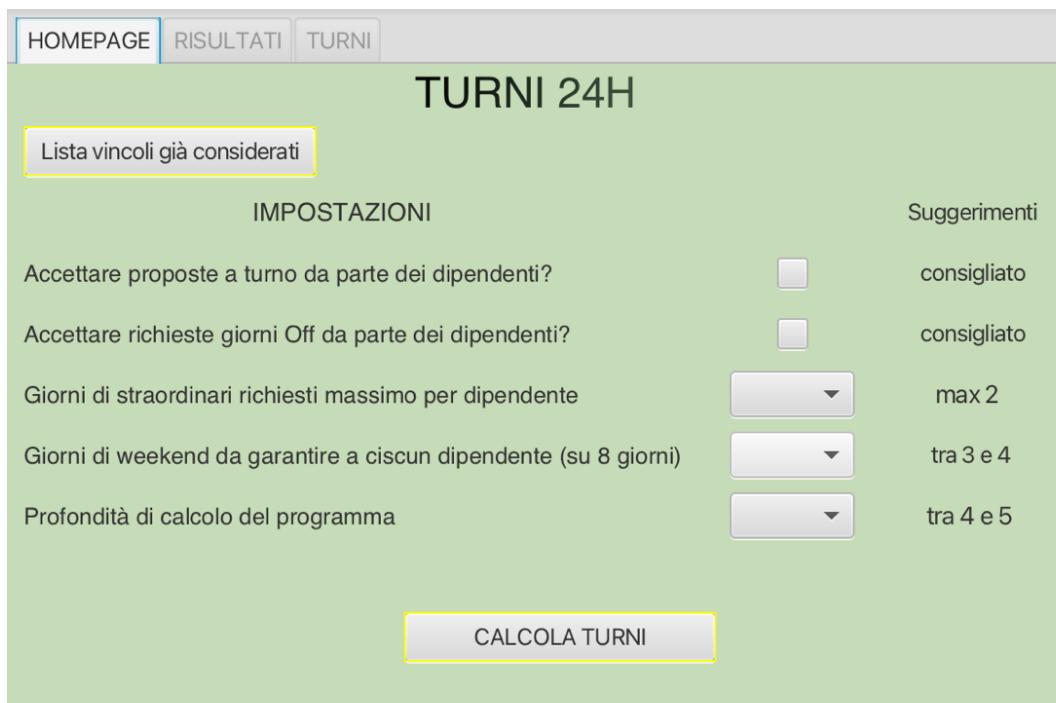


CAPITOLO 5: OUTPUT GRAFICO

5.1 Link YouTube

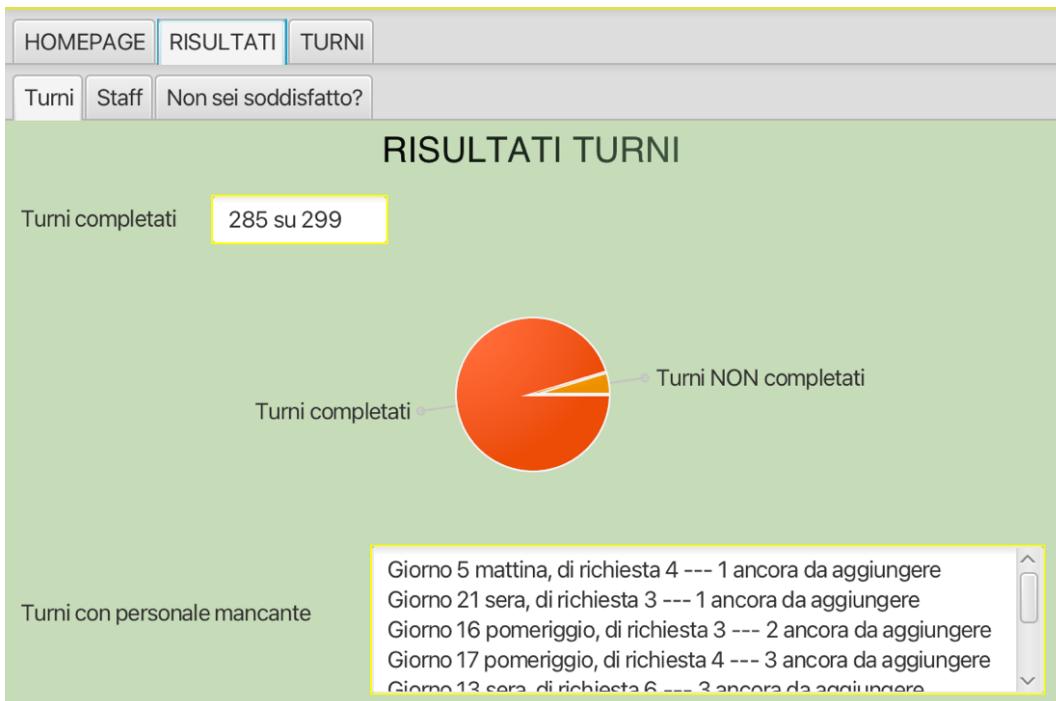
La spiegazione dell'utilizzo del programma ed una piccola simulazione sono presenti sul video di presentazione YouTube al link: <https://youtu.be/2AwoMV82ah8>.

5.2 Videata all'apertura dell'applicazione

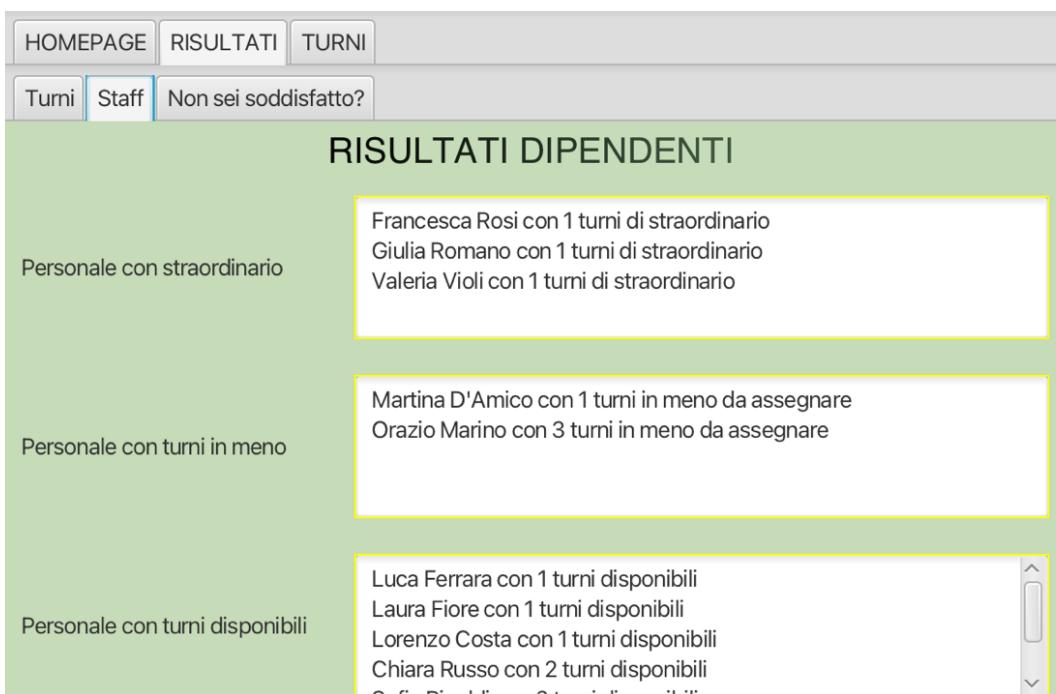


La homepage del programma presenta alcune opzioni modificabili relative ai vincoli nella richiesta dei turni. Chi utilizza l'app, infatti, può scegliere se accettare le proposte a turno dei dipendenti e/o rispettare le richieste dei giorni off, quanti giorni di straordinario richiedere massimo a ciascun dipendente, quanti giorni di weekend garantire e quanti tentativi far provare all'algoritmo per trovare una buona soluzione. Nella homepage inoltre troviamo un link che indirizza ai vari vincoli già considerati e ovviamente il pulsante per calcolare i turni.

5.3 Videata risultati



La sezione sui risultati, accessibile solo dopo aver calcolato i turni, si divide principalmente in due parti. La prima espone i risultati sui turni. Questa prima parte indica quanti turni l'algoritmo è riuscito a ricoprire ed un grafico a torta che ne indica la percentuale. Nella parte inferiore troviamo invece quali sono i turni nel quale mancano personale e di quanti dipendenti si ha bisogno in quel turno.



La seconda parte espone i risultati relativi ai dipendenti. Indica quali componenti dello staff fanno rispettivamente, straordinari, turni in meno rispetto ai previsti e quanti invece avrebbero dei turni disponibili non avendo raggiunto le ore massime.

Inoltre, è presente una sezione con alcuni suggerimenti sui settaggi iniziali che l'utente utilizzatore del programma imposta.

5.4 Videata turni

Staff	1	2	3	4	5	6	7
Mario Rossi	R	S	S	R	M	R	P
Giovanni Verdi	P	R	M	S	R	R	P
Matteo Neri	S	R	S	R	P	R	R
Simona Bianchi	R	S	S	R	P	P	P
Valeria Violi	R	P	P	P	P	P	P
Matilde Ferrari	R	R	S	S	R	R	M
Chiara Russo	M	M	M	M	M	M	R
Michele Marchetti	M	M	R	M	M	P	R
Simone Ricci	S	R	S	R	M	R	S

Dopo aver calcolato i turni si può aprire la finestra relativa alle liste dei turni. Le liste sono ordinate per settimane e sotto vi sono tutti i dipendenti con i vari turni già assegnati. Sulle colonne vi sono i sette giorni della settimana. Con R si indica il riposo, con M, P e S rispettivamente mattina, pomeriggio e sera.

CAPITOLO 6: RISULTATI E CONCLUSIONI

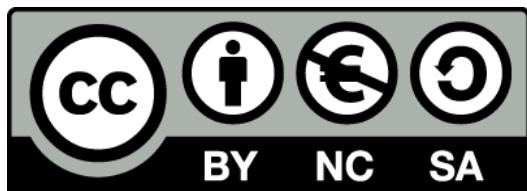
6.1 Risultati ottenuti

Il risultato atteso, ovvero una buona soluzione per il famoso problema dell'assegnazione a turno, è stato sicuramente raggiunto. Il programma ha un'ottima adattabilità a vari contesti lavorativi, con qualche piccola modifica ai vincoli si può stravolgere il risultato. Il pregio principale del programma è il sapersi adattare ai molteplici vincoli trovando comunque una buona soluzione. L'output grafico rispecchia ciò che serve ad un eventuale utilizzatore, risultati pratici e lista dei turni.

6.2 Limiti del programma

Il limite principale del programma è il non completamento totale dei turni. Trovare una soluzione ottima per questi tipi di problemi ad oggi è quasi impossibile e questo programma ovviamente non riesce a trovare, per un numero elevato di dipendenti e vincoli, la soluzione ottima. Altro limite è la durata temporale dei turni, il programma si adatterebbe bene ad un altro database con dipendenti diversi, ma se la richiesta dei turni variasse in termine di giorni (diverso da quattro settimane), risulterebbe necessaria qualche piccola modifica all'interno del codice.

CAPITOLO 7: LICENZA



Quest'opera è stata rilasciata con licenza Creative Commons Attribuzione - Non commerciale - Condividi allo stesso modo 4.0 Internazionale.

Copia della licenza consultabile al sito web: <http://creativecommons.org/licenses/by-nc-sa/4.0/>.