UNIVERSITA' DI PISA

CORSO DI LAUREA IN MATEMATICA

Tesi di Laurea Triennale

Modello di Ottimizzazione per la Schedulazione del Personale Infermieristico di una Residenza Assistita

11 aprile 2014

Relatori:
Prof. Antonio Frangioni
Prof. Maria Grazia Scutellà

Candidata:
Serena Cortopassi

ANNO ACCADEMICO 2012/2013

Indice

Introduzione

I problemi di *scheduling* interessano i più svariati ambiti della pianificazione: industriale, logistica, della forza lavoro e molti altri. In generale consistono nell'allocazione delle risorse disponibili in un certo arco di tempo al fine di svolgere una serie di compiti assegnati.

La teoria della schedulazione propone modelli e algoritmi per ottimizzare il risultato di un processo, in termini di minimizzazione di tempi e costi o massimizzazione dei profitti.

Uno dei problemi maggiormente affrontati in letteratura è quello dello *staff scheduling*, che si propone di trovare l'assegnamento ottimo dei turni lavorativi ai dipendenti di una struttura. Lo scopo di questo lavoro di tesi è proprio quello di produrre una procedura di ottimizzazione della programmazione degli orari di lavoro del personale della struttura "Il Gignoro" della Diaconia Valdese Fiorentina. Il fine ultimo è stato quindi trovare gli assegnamenti ottimi dei turni ai dipendenti per ogni giorno dell'orizzonte di pianificazione considerato.

Il primo passo di questo percorso è stato quello della raccolta dati, momento fondamentale al fine di modellizzare il problema in modo corretto e dettagliato. É seguita poi la fase di formulazione del modello matematico volto a descrivere e risolvere il problema. L'ultima fase è stata quella risolutiva, tramite un'implementazione algoritmica è stato possibile ottenere una soluzione per il problema e testare l'efficacia e l'efficienza del metodo proposto su dati reali forniti dalla Diaconia Valdese Fiorentina.

Il lavoro è stato organizzato come segue.

Nel Capitolo 1 viene fornita una panoramica dei problemi di *scheduling*, ponendo in particolare l'accento sui problemi di turnazione del personale. A tale trattazione segue una presentazione dettagliata del problema reale preso in esame, in cui vengono descritti in maniera informale i dati, i vincoli e la funzione obiettivo.

Nel Capitolo 2 si forniscono gli strumenti per la formulazione di un modello matematico, in particolare tramite programmazione lineare intera. Segue una

rassegna bibliografica completa riguardante la trattazione in letteratura della modellizzazione e implementazione algoritmica del problema dei turni della forza lavoro, sottolineando affinità o divergenze tra gli scenari proposti e quello trattato in questo lavoro di tesi. Infine viene fornita una descrizione formale dei dati in input e si procede alla formulazione vera e propria del modello in questione, definendo i vincoli del problema e i termini della funzione obiettivo.

Nel Capitolo 3 vengono riportati gli strumenti implementativi utilizzati per eseguire test utili a comprovare la reale efficienza ed efficacia del modello formulato. Vengono inoltre descritte le fasi implementative e commentati i risultati ottenuti.

Infine nell'Appendice A vengono proposti alcuni estratti del codice utilizzato per affrontare i test implementativi esposti nel Capitolo 3; nell'Appendice B vengono invece riportati i valori dei coefficienti di penalità assegnati ai termini della funzione obiettivo nei vari test effettuati.

Capitolo 1

Descrizione del problema

1.1 Problemi di scheduling

Sin dalla fine degli anni '50 si è sviluppato un certo interesse per lo studio dei problemi di *scheduling*, che vengono tutt'oggi trattati in ambito della teoria della schedulazione. Tale teoria si è sviluppata dinamicamente, proponendo modelli decisionali e algoritmi che vanno a toccare i più svariati ambiti della pianificazione: dall'organizzazione dei cicli produttivi industriali (*machine scheduling* e *project planning*), alla generazione di turni lavorativi e orari scolastici (*workforce scheduling* e *timetabling*), alla gestione del traffico e degli orari dei servizi di trasporto.

In generale i problemi di schedulazione consistono nell'allocazione delle risorse disponibili (talvolta scarse e non rinnovabili) in un certo arco di tempo al fine di svolgere una serie di compiti assegnati. Il tentativo è quello di ottimizzare il risultato, in termini di minimizzazione di tempi e costi o massimizzazione del profitto.
Esempi tipici di risorse limitate sono le macchine di un'impresa manifatturiera, il monte ore settimanale del personale di un'azienda o le piste di atterraggio e decollo di un aeroporto, così come persone, stanze, denaro ed energia. Esempi di obiettivi da ottimizzare sono invece la durata di un progetto, il costo delle ore di straordinario del personale, il tempo di attesa di un aereo in volo o di un lotto da lavorare davanti a una macchina operatrice.

Ogni assegnamento che viene incontro a tutte le richieste riguardanti il lavoro da svolgere e le risorse disponibili è detto ammissibile o realizzabile e contiene tutte le informazioni temporali che specificano l'ordine di esecuzione delle singole attività e l'istante di inizio e di fine di ciascuna. Lo scopo è quello di trovare, tra tutte le soluzioni ammissibili, quella che meglio risponde alle richieste e alle necessità.

A differenza di quanto accade per altre aree dell'ottimizzazione combinatoria, tutt'oggi per i problemi di *scheduling* più difficili non è possibile indicare un unico approccio nettamente preferibile per la loro soluzione, ma di volta in volta può essere più appropriato utilizzare euristiche, algoritmi esatti o approssimati. In generale trovare la soluzione ottima per un problema di schedulazione richiede di analizzare lo spazio delle soluzioni ammissibili e selezionare la migliore (nel senso indicato dai criteri adottati) o una che sia più vicina possibile a quella ottima (se il problema dato risulta computazionalmente intrattabile).
Alcuni problemi di *scheduling* possono essere risolti efficientemente riducendoli a problemi di ottimizzazione combinatoria risolubili in tempi polinomiali, come problemi di programmazione lineare, di flusso massimo o di trasporto. Altri possono essere affrontati con tecniche standard, come la programmazione dinamica e i metodi *branch-and-bound*.
I metodi di risoluzione si classificano in due macro-classi: i metodi esatti ed i metodi euristici.

I metodi esatti si distinguono a loro volta in:

- metodi costruttivi: si costruisce una soluzione ottima sulla base dei dati del problema seguendo una serie di semplici regole di priorità
- metodi enumerativi: si enumerano tutte le possibili soluzioni, si eliminano quelle che non sono ottimali dalla lista e si seleziona la migliore schedula

L'approccio con metodi esatti è tuttavia applicabile solo a problemi con un numero ridotto di variabili, mentre nella realtà i problemi sono spesso di larga dimensione e quindi non possono essere risolti in modo esatto con tempi di calcolo ragionevoli. Si ricorre pertanto a metodi di soluzione euristici di ricerca locale quali ad esempio *tabu search*, *simulated annealing* e algoritmi genetici; questi tuttavia non determinano una soluzione ottima del problema, ma solo una soluzione sub-ottimale ottenuta con algoritmi approssimati.

1.1.1 Problema della turnazione del personale

A causa della sua complessità e della sua rilevanza pratica, uno dei problemi di schedulazione che si incontra spesso nell'ambito della Ricerca Operativa è quello dello *staff scheduling*.
Programmare in modo ottimale i turni del personale di un'attività di qualsiasi tipo (call center, ospedali, aziende manifatturiere, ditte di trasporto, ...) può risultare un grande vantaggio, sia a livello di costi per il datore di lavoro, sia a livello di soddisfazione da parte dei dipendenti. Una buona coordinazione è inoltre fondamentale per la buona gestione aziendale, per rispettare i vincoli

sindacali e amministrativi e bilanciare i carichi di lavoro, in modo da non creare tensioni tra i lavoratori.

In questo tipo di problemi sono specificati in partenza i compiti giornalieri da svolgere e le ferie (quantificate in giorni) che spettano ad ogni impiegato nell'arco dell'orizzonte di pianificazione (solitamente si tratta di una settimana, un mese o un anno); sono invece da determinare la sequenza in cui giorni lavorativi e di ferie si alternano e il tipo di turno assegnato ad ogni dipendente nei suoi giorni lavorativi.
Spesso bisogna tenere in considerazione le qualifiche e le specializzazioni di ciascun impiegato poiché non tutti potrebbero essere adatti a svolgere determinati compiti.
L'obiettivo principale è quello di minimizzare il numero di impiegati necessari per svolgere i compiti giornalieri e determinare l'orario d'inizio e di fine del turno di lavoro di ciascuno di loro.

1.2 Problema reale: orario di lavoro del personale della struttura “Il Gignoro”

Lo scopo di questa tesi è proporre una procedura di ottimizzazione per la programmazione degli orari di lavoro del personale della struttura “Il Gignoro” della Diaconia Valdese Fiorentina (DVF).

Il problema consiste nell'assegnamento di un turno a ogni dipendente per ogni giorno dell'orizzonte di pianificazione considerato.

Con il termine turno si intende una delle possibili combinazioni di tipi di orario di servizio previsti dalla struttura. Ogni tipo di orario prevede da zero a 2 periodi di lavoro ed è caratterizzato quindi da al più due coppie di orari di entrata e uscita. Con zero periodi di lavoro si intende che non ci sono orari di entrata e uscita valorizzati; può essere il caso di ferie e riposi.
Un tipo di orario può essere virtuale o non virtuale; i tipi di orario virtuali, pur non avendo un orario effettivo di lavoro (i valori di entrata e uscita non sono valorizzati), concorrono forfettariamente alla determinazione del monte ore settimanale, mensile e annuale. Le ferie sono un esempio di orario virtuale. I restanti tipi di orario si dicono non virtuali e comprendono sia gli orari di lavoro effettivo sia le ore destinate a riunioni e corsi di formazione.
Il turno giornaliero di un operatore può quindi essere composto da più tipi di orario purché tra loro compatibili, cioè caratterizzati da intervalli temporali che non si sovrappongono. Gli orari virtuali sono da considerarsi incompatibili con tutti gli altri tipi di orario.

Un turno composto da due tipi di orario può essere ad esempio il caso della combinazione di ore lavorative e ore di formazione.

Ciascun tipo di orario, e quindi ciascun turno, richiede personale qualificato che possa svolgere le mansioni previste da quel tipo di orario (turno). Dunque ogni tipo di orario è caratterizzato non solo da un intervallo temporale (determinato dai periodi di lavoro) ma anche da una qualifica richiesta. Può accadere che due tipi di orario presentino la stessa finestra temporale ma due qualifiche diverse, ad esempio la mattina per le inservienti e la mattina per le infermiere. Tali tipi di orario sono da considerarsi distinti.
A sua volta ogni operatore ha una qualifica e può essere assegnato solamente a turni che necessitano di una qualifica pari alla sua o di grado inferiore. Il risultato è che ciascun operatore è associato a un insieme di turni per lui ammissibili. In fase di programmazione, comunque, sarà preferibile assegnare a un operatore un turno che richieda di svolgere la mansione per cui è preposto, e non turni per cui sono sufficienti qualifiche inferiori.
Inoltre possono esserci operatori che svolgono compiti che gli altri non possono svolgere; questo ci permette di schedulare i loro orari lavorativi separatamente da tutti gli altri.
Un caso può essere quello dei nottanti, operatori che lavorano su più turni nei servizi attivi 24 ore. Sono gli unici che coprono i turni di notte e i loro orari di lavoro seguono un assegnamento ciclico del tipo: pomeriggio, mattina, notte, smontante, riposo. Questa sequenza viene chiamata "turno in quinta" poiché si ripete ogni cinque giorni. Scheduleremo i nottanti separatamente dagli altri operatori considerando i loro turni in quinta come fissati a priori nella pianificazione.

L'orizzonte temporale su cui lavoreremo è mensile, ma per alcuni vincoli e per il calcolo della funzione obiettivo bisognerà tener conto di alcune scelte effettuate nei mesi precedenti a quello considerato e degli obiettivi annuali, di tipo contrattuale e amministrativo.
Ai fini della programmazione partizioneremo il mese in quattro periodi di uguale ampiezza, che assumeremo corrispondano alle quattro settimane che compongono il mese.

La struttura in questione risulta organizzata in diverse unità operative, dette moduli. A ciascuna di esse è associato un certo numero di tipi di orario in cui è attivo il servizio; questi variano dai giorni feriali a quelli festivi. Ogni operatore è a sua volta associato ad un modulo. Alcuni operatori sono tuttavia assegnati ad un modulo particolare, detto "Jolly". Quest'ultimo è un modulo fittizio e non ha quindi tipi di orario propri da coprire; i suoi operatori vengono assegnati come supporto agli altri moduli in sede di programmazione per coprire situazioni

di necessità. Analogamente è possibile che in caso di bisogno anche operatori non appartenenti al modulo “Jolly” vengano assegnati a supporto di un modulo diverso dal proprio. In fase di programmazione, comunque, sarà preferibile un assegnamento dei turni confinato a livello di modulo (eccetto il modulo “Jolly”); tale preferenza, così come le altre precedentemente citate, sarà gestita tramite un costo di assegnamento per ogni coppia operatore-turno.
L’esistenza del modulo “Jolly” e la possibilità di assegnare gli operatori anche a turni su moduli diversi da quello a cui sono associati rendono dipendente la programmazione di un modulo da quella degli altri e quindi l’ottimizzazione non può essere condotta separatamente modulo per modulo.

Ogni modulo ha una domanda di copertura da soddisfare, vale a dire che per ogni tipo di orario associato al modulo e per ogni giorno viene fatta richiesta di un dato numero di operatori da assegnare a quel tipo di orario, opportunamente qualificati. Dunque ogni tipo di orario si caratterizza anche per il modulo di riferimento a cui è associato e sarebbe preferibile che a coprirlo fossero gli operatori assegnati a quel modulo. Infatti il costo di associare un operatore a un turno che copre un tipo di orario richiesto dal suo modulo sarà più basso (diciamo zero) rispetto al costo di assegnarlo di servizio su un altro modulo.
Può capitare che un tipo di orario possa essere coperto o all’interno di un modulo o all’interno di un altro, ossia che possa essere assegnato indifferentemente a operatori di due moduli distinti. In tal caso il tipo di orario in questione sarà caratterizzato da entrambi i moduli in alternativa e il costo di assegnamento operatore-turno sarà zero per gli operatori di entrambi i moduli, invece che zero per gli operatori di un unico modulo e più alto per tutti gli altri.
Tali costi di assegnamento non riguardano il processo di ottimizzazione ma soltanto quello della creazione dei dati in input.

In linea di principio un tipo di orario può avere caratterizzazioni molteplici, che possono variare a seconda dello scenario considerato. Nel nostro caso i tipi di orario saranno definiti intrinsecamente da una fascia oraria, una qualifica e il modulo (o un’alternativa di moduli) in cui devono essere coperti.
In conclusione ciascun tipo di orario prevede un numero di operatori richiesti giornalmente e ciascuna coppia operatore-turno ha un costo di ammissibilità.
Nel caso di zero operatori richiesti per un certo tipo di orario in un certo giorno, il vincolo corrispondente per quel giorno e le relative variabili non devono essere considerate in fase di programmazione; questo permette di snellire le dimensioni del modello.

Ci sono diversi vincoli che una soluzione del problema deve soddisfare; molti di essi verranno trattati in modo *soft*, ovvero la loro violazione sarà permessa

a patto di introdurre un'opportuna penalità in funzione obiettivo. Nel nostro caso tali vincoli sono:

- coprire la domanda associata ad ogni modulo (fuorché al modulo "Jolly") per ogni giorno del mese. Il vincolo che riguarda le richieste rimaste scoperte è assai più stringente rispetto a quello che cerca di limitare un eventuale sovrannumero di operatori assegnati a un certo tipo di orario;

- assegnare a ciascun operatore almeno due giorni di riposo in ognuna delle due metà del mese (cioè almeno due giorni ogni due settimane), possibilmente prevedendone uno in ogni settimana;

- prevedere 37 ore settimanali di lavoro per ogni operatore, come stabilito da contratto; quindi $\frac{37}{6} \times n$ ore mensili (dove n è il numero di giorni lavorativi del mese e $\frac{37}{6}$ le ore medie giornaliere di lavoro) e $\frac{37}{6} \times (365 - 52) = 1930,17$ ore annuali (dove si tiene conto dei riposi settimanali assegnati). I vincoli settimanale, mensile e annuale sono via via più stringenti;

- cercare di assegnare a ogni operatore un turno per cui è richiesto di svolgere una mansione pari (e non inferiore) alla sua qualifica;

- venire incontro alle preferenze degli operatori, assegnando loro i turni più graditi;

- cercare di assegnare a ogni operatore turni che coprono richieste fatte dal modulo a cui sono associati (fatta eccezione per gli operatori associati al modulo "Jolly").

Avremo un unico vincolo *hard* da soddisfare e si tratterà di prevedere nella pianificazione i turni programmati a priori. Tale vincolo non può essere violato. Può essere il caso delle ferie, dei riposi programmati, dei turni in quinta dei nottanti o delle attività imposte dalla direzione (ad esempio la partecipazione obbligatoria ad un corso di formazione o a una riunione).
Le ferie sono rappresentate da un tipo di orario virtuale, che non prevede quindi entrate e uscite valorizzate, ma che concorre a formare il budget orario complessivo dell'operatore (settimanale, mensile e annuale). Il numero di giorni di ferie annuali è fissato a 31 giorni lavorativi. Sono programmate in anticipo su richiesta del lavoratore in sede di pianificazione annuale ed approvate dalla direzione, se ne deve pertanto tenere conto anche in fase di stesura mensile. Se la fase di programmazione dei turni dovesse evidenziare la mancanza di una soluzione (ad esempio tutti i lavoratori sono in ferie in un certo giorno), o comunque la presenza di soluzioni solo a costi troppo alti, sarà compito della direzione definire

un diverso piano ferie.
Come detto, sono prefissati anche gli orari di lavoro dei turnisti nei servizi attivi 24 ore. Ciascun nottante verrà assegnato a una sequenza fissata di turni, che inizierà tenendo conto del tipo di turno svolto l'ultimo giorno del mese precedente alla pianificazione.
Le ore destinate a riunioni e corsi di formazione invece sono tenute in considerazione nella programmazione soltanto se se ne conoscono il giorno e l'ora di svolgimento prima di procedere all'assegnazione dei turni.
In generale, ciò che non è noto al momento della pianificazione viene ignorato e verrà inserito in seguito, apportando opportune modifiche durante il mese.

I criteri per confrontare soluzioni diverse al fine di scegliere la migliore si basano sulla minimizzazione di vari aspetti (qui di seguito elencati secondo priorità decrescente):

- il numero di tipi di orario richiesti dai moduli rimasti scoperti;
- le ore complessive di straordinario, cioè lo sforamento in eccesso del budget orario degli operatori (settimanale, mensile e annuale);
- il non raggiungimento del numero di ore previste da contratto, cioè lo sforamento in difetto del budget orario degli operatori (settimanale, mensile e annuale);
- la penalità derivante dalla distribuzione non uniforme dei riposi settimanali e mensili;
- la penalità derivante dall'assegnazione di turni a operatori con qualifica superiore a quella richiesta;
- la penalità derivante dall'assegnazione di turni sgraditi al personale;
- la penalità derivante dall'assegnazione di turni che coprono richieste di un certo modulo a operatori di un altro modulo;
- il numero di operatori in eccesso rispetto a quelli richiesti dai moduli per ogni tipo di orario in ogni giorno della programmazione.

Per quanto riguarda il secondo e il terzo punto, il tentativo è quello di bilanciare il carico di lavoro (settimanale, mensile e annuale) dei dipendenti. É possibile rilassare il vincolo settimanale di 37 ore, aggiungendo però un costo nella funzione obiettivo, e lo stesso si può fare per quello mensile, considerando un costo

maggiore. Tali coefficienti di penalità saranno più o meno alti in base alla gravità delle violazioni associate. Al fine di calcolare e penalizzare lo sforamento annuale l'idea sarebbe di tenere conto ogni mese del risultato del mese precedente e di applicare un peso ogni mese più alto allo scostamento dal numero di ore previsto.

La funzione obiettivo è pertanto definita come una somma di vari termini lineari; ad ognuno di essi sarà associato un coefficiente (peso) il cui valore indicherà la penalizzazione assegnata al termine considerato.
Il valore assunto dalla funzione obiettivo in corrispondenza di una certa soluzione ne indicherà la bontà (o meno). In essa sono infatti riassunte le richieste e gli obiettivi che ci si prefigge di esaudire tramite la programmazione. Prese due soluzioni distinte del problema, la migliore sarà pertanto quella che restituisce il valore minore della funzione obiettivo precedentemente introdotta.

Capitolo 2

Modelli di programmazione

L'identificazione della struttura del problema è fondamentale per la costruzione di un modello matematico ed è quindi necessario un approccio sistematico. Come primo passo verso l'obiettivo finale di risolvere questioni di pianificazione e schedulazione è necessario trasformare la descrizione del problema in un modello matematico.

Il problema solitamente non è descritto in maniera rigosorsa, bensì a parole, sotto forma di promemoria, relazione o documento di specifica. Il modello matematico consiste invece in un'astrazione del problema reale e deve essere formulato in maniera precisa e corretta.
Le decisioni pianificate e le soluzioni suggerite dal modello devono rappresentare la realtà con il desiderato livello di dettaglio, sia in termini di ammissibilità (attraverso i vincoli) che di ottimalità (attraverso la funzione obiettivo). Inoltre deve essere descritto usando oggetti standard (prodotti, macchine, risorse, persone, ...) e vincoli generici (conservazione del flusso di produzione, disponibilità delle risorse, restrizioni sindacali, ...).

La fase di modellizzazione identifica:

- gli oggetti che devono essere manipolati (prodotti, risorse, persone, periodi di tempo, ...);
- i dati associati agli oggetti (domanda di un prodotto in un certo lasso di tempo, disponibilità delle risorse, il monte ore giornaliero del personale di un'impresa, ...);
- le decisioni da prendere riguardo agli oggetti (rappresentate da variabili decisionali) al fine di definire una soluzione del problema;
- i vincoli da soddisfare tramite le decisioni al fine di definire soluzioni ammissibili per il problema;

- la funzione obiettivo che fornisce un metodo per valutare e comparare le soluzioni ammissibili.

Seguirà poi una fase di ottimizzazione per la ricerca della soluzione ottima del problema, ossia la migliore tra tutte quelle ritenute accettabili.

In particolare, ci dedicheremo alla formulazione del problema descritto nel Capitolo 1 in termini di programmazione lineare intera. Con questo tipo di modellizzazione saremo in grado di ottenere la soluzione ottima mediante algoritmi esatti.

2.1 Modelli di programmazione lineare intera per problemi di schedulazione

2.1.1 Programmazione lineare intera

Molti dei problemi di ottimizzazione combinatoria sono risolvibili polinomialmente e possono essere formulati come problemi di programmazione lineare. Un problema di programmazione lineare in forma standard può essere presentato come:

$$\min \quad \sum_{j=1}^{n} c_j x_j \tag{2.1}$$

$$\text{tale che} \quad \sum_{j=1}^{n} a_{i,j} x_j \leq b_i \quad (i = 1, ..., m) \tag{2.2}$$

$$x_j \geq 0 \quad (j = 1, ..., n) \tag{2.3}$$

dove c_j, $a_{i,j}$ e b_i sono numeri reali dati.

I numeri reali $x_1, ..., x_n$ che soddisfano le equazioni (2.2) e (2.3) sono chiamati soluzione ammissibile; tale assegnazione è soggetta a vincoli più o meno forti da rispettare: i vincoli *hard* non possono essere violati, mentre i vincoli *soft* possono essere violati, a costo di peggiorare la qualità della soluzione in termini di penalità assegnate in funzione obiettivo.
Lo scopo è quello di trovare una soluzione ottima, cioè una soluzione ammissibile che minimizza la funzione obiettivo $\sum_{j=1}^{n} c_j x_j$ e soddisfa tutti vincoli imposti.
Oltre al caso in cui il programma lineare ammette una soluzione ottima, possono verificarsi altre due situazioni:

- il problema non ammette soluzione ammissibile, ossia non esistono valori per le variabili x_j che soddisfano tutte le disuguaglianze in (2.2) e (2.3);

- il problema non è limitato, ossia $\forall y \in \mathbb{R}$ esiste una soluzione ammissibile $(x_1, ..., x_n)$ tale che $\sum_{j=1}^{n} c_j x_j < y$.

In un problema di programmazione lineare, oltre a quelli del tipo (2.2), possiamo anche avere vincoli della forma

$$\sum_{j=1}^{n} a_{i,j} x_j \geq b_i \quad (i = 1, ..., m) \tag{2.4}$$

oppure

$$\sum_{j=1}^{n} a_{i,j} x_j = b_i \quad (i = 1, ..., m). \tag{2.5}$$

Ad ogni modo, la (2.4) equivale a

$$\sum_{j=1}^{n} -a_{i,j} x_j \leq -b_i \quad (i = 1, ..., m)$$

e la (2.5) si può esprimere come combinazione di

$$\sum_{j=1}^{n} a_{i,j} x_j \leq b_i \quad (i = 1, ..., m)$$

e

$$\sum_{j=1}^{n} -a_{i,j} x_j \leq -b_i \quad (i = 1, ..., m).$$

In molte applicazioni non avremo restrizioni sulla non negatività delle variabili x_j e scriveremo semplicemente $x_j \in \mathbb{R}$, considerando $x_j = x_j^+ - x_j^-$ con x_j^+, $x_j^- \geq 0$.
Inoltre, per risolvere un problema della forma

$$\max \quad \sum_{j=1}^{n} c_j x_j \qquad \text{t.c. (2.2) e (2.3)}$$

possiamo risolvere il problema

$$-\min \quad \sum_{j=1}^{n} -c_j x_j \qquad \text{t.c. (2.2) e (2.3).}$$

Dunque, in tutti i casi il problema può essere riformulato come problema di programmazione lineare in forma standard.

L'approccio tipico per la loro risoluzione è l'algoritmo del simplesso: si tratta di una procedura iterativa che trova la soluzione ottima o riconosce l'inammissibilità o l'illimitatezza del problema in un numero finito di passi. Sebbene il numero di iterazioni possa essere esponenziale rispetto alla dimensione del problema, il metodo del simplesso è molto efficiente in pratica.

Se tutte le variabili x_j dell'istanza devono essere intere il problema si chiama di programmazione lineare intera. Un programma lineare intero binario è invece un problema di programmazione lineare intera dove le x_j sono variabili decisionali binarie, ossia $x_j \in \{0, 1\}$. Abbiamo poi modelli di programmazione lineare mista dove i vincoli e la funzione obiettivo sono lineari e le variabili decisionali sono sia reali che intere.
Al contrario dei corrispettivi continui, i problemi di programmazione lineare interi sono *NP-hard* e quindi molto più complessi da risolvere computazionalmente.

2.1.2 Rassegna bibliografica

Nell'ambito dei problemi di schedulazione, uno dei temi più profondamente trattati in letteratura negli ultimi decenni è la pianificazione dei turni della forza lavoro, in particolare delle infermiere negli ospedali e strutture simili [8, 16, 17]. Problemi di questo tipo includono tipicamente decisioni come determinare il numero di infermiere da impiegare, assegnare le infermiere ad un insieme di turni in modo da rispettare i vincoli o assegnare i pazienti alle infermiere all'inizio del turno.

Lavieri e Puterman (2009) [25] hanno proposto un modello di programmazione lineare intera per risolvere un problema di pianificazione della forza lavoro di un ospedale. Le decisioni da prendere nel loro modello includono il numero di studenti da ammettere ai differenti programmi di formazione infermieristica, il numero di infermiere e manager da assumere e il numero di infermiere da promuovere a posizioni manageriali. Devono essere prese decisioni per ogni anno dell'orizzonte di programmazione, che ha una durata di 20 anni.
I vincoli del modello rappresentano varie richieste, tra cui: il numero degli studenti non deve superare il limite imposto dal programma di formazione, il numero di infermiere deve essere in grado di soddisfare i bisogni della popolazione, il numero di studenti ammesso ai diversi tipi di programmi deve essere bilanciato e il numero di anni di esperienza deve essere preso fortemente in considerazione nel momento in cui si decide di promuovere delle infermiere.
L'obiettivo del modello è quello di determinare la quantità di dipendenti necessari ogni anno al fine di minimizzare il costo totale dei corsi di formazione (per

educare gli studenti e promuovere le infermiere), di assunzioni e salari annuali, senza abbassare la qualità del servizio. Si propone inoltre di seguire dinamicamente i cambiamenti del sistema tenendo conto dell'invecchiamento di studenti e infermiere e dell'esperienza maturata.
Tale modello si differenzia da quello proposto in questo lavoro di tesi, principalmente perché si allontana dal nostro obiettivo, che è invece quello di fornire un piano mensile lavorativo dei dipendenti a partire da un insieme di lavoratori fissato in partenza e su cui, almeno in fase di programmazione, non sono previste modifiche.

Purnomo e Bard (2007) [29] hanno costruito un calendario ciclico bisettimanale per le infermiere come soluzione a un problema di programmazione lineare mista dove lo scopo era quello di minimizzare la somma pesata del numero di turni rimasti scoperti e la violazione dei vincoli *soft* che rappresentano le preferenze delle infermiere. I vincoli *hard* del modello descrivono un insieme di richieste legali e burocratiche legate al numero massimo consentito di ore di lavoro giornaliero e settimanale e al minimo periodo di pausa da garantire tra due turni.
Gli autori hanno sviluppato un algoritmo *branch-and-price* per risolvere istanze del problema di grandi dimensioni e l'hanno perfezionato usando due diverse regole di *branching* (*subproblem variable branching* e *master problem variable branching*) e un'euristica di arrotondamento (*effective rounding heuristic*).
Dai loro risultati numerici si può osservare che l'algoritmo proposto può risolvere istanze di problemi di grandi dimensioni (fino a 200 infermiere) in un tempo ragionevole. I risultati indicano inoltre che il *subproblem variable branching* è più adatto a problemi di piccole e medie dimensioni, mentre il *master problem variable branching* funziona meglio per problemi con dimensioni maggiori.
Questo modello è molto vicino a quello proposto in questa tesi poichè si evidenziano forti similarità riguardo ai vincoli e alla funzione obiettivo, in cui i turni rimasti scoperti sono fortemente penalizzati e in cui si tiene conto delle preferenze espresse dagli operatori. Si riscontrano però differenze nell'intervallo di programmazione, che per noi è mensile mentre nel lavoro di Purnomo e Bard è bisettimanale, e nel numero di operatori considerati, che nel nostro caso è molto inferiore a 200. Quest'ultimo dato però non fa che semplificare il problema riducendone la dimensione.

Punnakitikashem e al. (2008) [28] hanno studiato il problema della schedulazione degli orari di lavoro delle infermiere che tiene conto dell'assegnamento paziente-infermiera all'inizio del turno. Considerano l'incertezza riguardo all'ammontare delle cure richieste da ogni paziente e formulano il problema tramite programmazione lineare mista in due fasi: nella prima si procede con

l'assegnazione di un'infermiera a ogni paziente; nella seconda si procede a minimizzare l'eccesso di carico di lavoro per le infermiere a partire dalla schedulazione fatta.
Prima dell'inizio del turno ogni paziente deve essere assegnato opportunamente a un'infermiera, tenendo conto in fase di pianificazione di eventuali assegnamenti non ammissibili. Si assume inoltre che non avvengano riassegnamenti durante il turno. Ad ogni modo le pratiche di assegnamento vengono dinamicamente aggiornate a causa dei nuovi pazienti in arrivo e di quelli dimessi. A questo proposito, modellando la necessità di cura dei pazienti, è utile tenere in considerazione la probabilità di nuovi arrivi o dimissioni. Come risultato di questo approccio l'insieme dei pazienti considerati include anche i pazienti potenziali oltre a quelli reali.
Ci sono due tipi di cure fornite dalle infermiere: cure dirette, che riguardano il tempo speso con i pazienti, e cure indirette, che riguardano il tempo speso in altre mansioni. All'aumentare del carico di lavoro delle infermiere, i loro pazienti ricevono meno cure dirette. Dunque l'eccessivo carico ha un impatto negativo sulla qualità del servizio.
Il principale vincolo da rispettare è l'assegnazione di ogni paziente a un'infermiera, mentre la funzione obiettivo del modello è data dal costo totale derivante dagli assegnamenti fatti. Per come è strutturato il problema e la funzione obiettivo, minimizzare il costo totale equivale a minimizzare l'eccesso di carico di lavoro delle infermiere.
Nelle due fasi di risoluzione del problema si considera uno scenario casuale ω, che determina le possibili dimissioni e arrivi di nuovi pazienti.
Punnakitikashem e al. si servono dell'algoritmo *L-shaped* per risolvere la seconda fase del problema e ne accrescono la risolvibilità tramite una procedura polinomiale in tempo, ricavando limiti inferiori e superiori al numero di pazienti che possono essere assegnati a una stessa infermiera.
Le istanze testate dagli autori non sono risolvibili in un intervallo di tempo ragionevole, dunque gli autori si propongono di usare l'algoritmo per ottenere una buona soluzione in un tempo limite di 30 minuti, rinunciando a trovarne una ottima. Comparano poi l'eccesso di carico di lavoro previsto, ottenuto dalla soluzione dell'algoritmo *L-shaped*, alle soluzioni ottenute da approcci alternativi, quali la risoluzione di un *mean value problem*, l'uso di euristiche per bilanciare i carichi basandosi sulla previsione delle cure totali richieste, e assegnamenti e scenari randomizzati. I risultati numerici ottenuti mostrano come l'algoritmo *L-shaped* riduca maggiormente il sovraccarico di lavoro rispetto agli altri metodi.
Il nostro problema si discosta notevolmente da quello appena presentato, poiché

presso la struttura "Il Gignoro" non avvengono assegnamenti diretti tra utenti e operatori; l'unico vincolo di associazione presente è quello tra operatori e turni, che però porterà ad un assegnamento indiretto tra gli operatori e i pazienti che hanno generato le domande di copertura dei tipi di orario.

Caprara e al. (2003) [9] affrontano il problema dello *staff scheduling* decomponendolo in due fasi: per prima cosa è necessaria la definizione dell'alternarsi dei giorni lavorativi e di riposo (tale sequenza è chiamata *pattern*) di ogni impiegato; in seguito si procede con la definizione dell'assegnamento dei turni ad ogni impiegato in ogni giorno lavorativo.
Il primo passo è formulato come *covering problem* per cui vengono presentati vari modelli di programmazione lineare intera e algoritmi enumerativi esatti basati su tali modelli. L'esperienza pratica mostra come il miglior approccio sia basato sul modello in cui le variabili sono associate a *patterns* ammissibili e generate sia tramite programmazione dinamica sia risolvendo un altro problema di programmazione lineare intera.
Il secondo passo è formulato come *feasibility problem* e risolto euristicamente attraverso una serie di problemi di trasporto. Nonostante in generale questa procedura possa non trovare una soluzione (sebbene esista), gli autori presentano condizioni sufficienti che garantiscono il successo di questo approccio.
Propongono inoltre un algoritmo iterativo euristico da seguire nel caso in cui il secondo passo non abbia prodotto una soluzione ammissibile.
L'obiettivo principale è la minimizzazione del numero di impiegati necessari per coprire tutti gli assegnamenti giornalieri nell'arco dell'orizzonte di programmazione.
Si decompone la ricerca della soluzione in cinque passi:

1. definizione dei tipi di orari previsti;
2. definizione del numero minimo di impiegati necessari per coprire tali turni;
3. definizione del numero dei giorni di riposo previsti per ogni dipendente;
4. definizione dell'alternanza tra periodi di lavoro e periodi di riposo di ogni impiegato (*pattern*), senza specificare quale turno gli è stato assegnato nei giorni lavorativi;
5. definizione dell'assegnamento giornaliero dei turni nei periodi lavorativi di ogni dipendente.

Sia dunque m il numero di impiegati, consideriamo allora assegnamenti a lungo termine che coprano m settimane. L'assegnamento per ogni impiegato si ripete ciclicamente ogni m settimane e in ogni settimana l'impiegato $i+1$ ha

gli stessi assegnamenti dell'impiegato i nella settimana precedente.
In alcuni caso gli impiegati hanno caratteristiche diverse e quindi devono essere associati a turni diversi; tuttavia nel lavoro di Caprara e al. si assume che tutti gli impiegati abbiano lo stesso tipo di caratteristiche.

Come abbiamo detto l'approccio proposto trova la soluzione del problema in due fasi: nella prima (corrispondente ai passi 2, 3 e 4) si determina il minimo numero di impiegati e i *patterns* associati ad ogni impiegato, in modo da garantire che almeno il numero minimo richiesto di impiegati lavori ogni giorno. Questa fase è affrontata formulando il problema tramite programmazione lineare intera e cercando una soluzione ottima tramite il metodo *branch-and-bound*, possibilmente combinato con il *column generation*.
Nella seconda fase (corrispondente al passo 5) si associa un turno a ogni impiegato per ogni giorno lavorativo, verificando l'ammissibilità dell'assegnamento e bilanciando il lavoro totale tra gli impiegati; si tratta di un problema combinatorio formulato come *transportation problem* per ogni giorno nell'orizzonte di pianificazione.
Se non viene trovata una soluzione ammissibile al primo tentativo, si applicano iterativamente le fasi 1 e 2 (cambiando adeguatamente il carico di lavoro degli impiegati) finché non se ne trova una.
La decomposizione del problema, sebbene non necessariamente conduca a una soluzione ottima, è motivata dal fatto che la parte difficile da modellare e risolvere è la prima, mentre assegnare turni agli impiegati una volta decisi i *patterns* risulta relativamente semplice. D'altro canto una formulazione tramite programmazione lineare intera per l'intero problema sarebbe risultata di dimensioni considerevolmente più grandi di quelle della formulazione necessaria nella fase 1.
Quest'ultimo problema e il modello fornito non si discostano molto da quelli presentati in questo lavoro e suggerisce dunque un approccio risolutivo in due fasi. Una differenza si coglie però nell'assunzione semplificatrice dell'uniformità tra impiegati, che non si verifica invece nel nostro caso e di cui bisogna tener conto in maniera considerevole in fase di input per l'assegnazione dei turni ammissibili agli operatori.

Brucker e al. (2011) in [5] affrontano approfonditamente il tema della schedulazione del personale, esponendo una rassegna di vari problemi e modelli incontrati in letteratura, sia generali sia rivolti a casi specifici.
Il primo riferimento è all'opera di De Causmaecker et al. [12], che classifica i problemi di *staff scheduling* in base alla tipologia del personale coinvolto; ciascun problema conduce a un modello differente.
La classificazione si divide in quattro tipi di pianificazione:

- pianificazione centrata sulla permanenza: si tratta del caso di ospedali e centrali di polizia, dove la quantità di personale necessario è definita a priori;
- pianificazione centrata sulla fluttuazione: si tratta del caso di call center, ristoranti e centri di distribuzione, dove la quantità di personale varia in base alla fluttuazione della richiesta del servizio;
- pianificazione centrata sulla mobilità: è il caso delle compagnie di trasporti, in cui i compiti del personale coinvolgono gli spostamenti;
- pianificazione centrata su un progetto: è il caso di compagnie che suddividono il lavoro in progetti a cui assegnano un certo numero di impiegati, come ad esempio progetti edili o di sviluppo software.

Nel caso della struttura "Il Gignoro" si tratta di un problema di pianificazione centrata sulla permanenza. Vediamo dunque come viene descritto il problema da Brucker e al. [5] e qual è il modello che viene proposto per questa tipologia di pianificazione.
Si considera nello specifico il problema della turnazione delle infermiere, nel caso in cui viene assegnato un solo turno per ogni giorno dei cinque dell'orizzonte di programmazione [6].
Ci possono essere diversi tipi di infermiere e generalmente l'insieme dei tipi di orario di lavoro possibili è abbastanza piccolo. La domanda di copertura degli orari di servizio avviene tramite la richiesta di un certo numero di infermiere di un certo tipo per l'orario di un certo giorno, esattamente come risulta nel problema presentato in questo lavoro di tesi.
L'assegnamento deve soddisfare vincoli *hard* e *soft*. Possibili vincoli *hard* sono:

1. la domanda di copertura deve essere soddisfatta;
2. ogni giorno ogni infermiera ha un solo turno assegnato;
3. il numero massimo di turni di notte è di 3 ogni 5 settimane;
4. assegnare a ogni infermiera almeno 2 weekend liberi ogni 5 settimane.

Possibili vincoli *soft* sono:

1. evitare certi tipi di turni in successione;
2. assegnare il weekend libero oppure il weekend completo (che prevede turni lavorativi sia il sabato che la domenica);

L'obiettivo è trovare un assegnamento di turni che soddisfa i vincoli *hard* e che minimizza la somma delle penalità derivanti dalla violazione dei vincoli *soft*. Un obiettivo ulteriore potrebbe essere minimizzare il numero di infermiere impiegate.
Riadattando la durata dell'orizzonte di programmazione e la definizione di alcuni vincoli, tale descrizione si avvicina perfettamente al nostro problema.
Tutti i possibili assegnamenti di turni agli operatori per ogni giorno dell'orizzonte di programmazione possono essere esplorati tramite una *tabu search*, fino ad eleggere quello ottimo. Per istanze di piccole dimensioni, il problema può essere risolto facendo uso di solutori ILP; per problemi di dimensioni maggiori è consigliata la risoluzione tramite metodi euristici che combinano la ricerca locale e le tecniche di flusso su reti.

In [8] troviamo una rassegna di approcci modellistici e risolutivi di vario tipo: di meta-euristica, euristica, intelligenza artificiale e programmazione matematica. Tali approcci multi-disciplinari sono stati molto incoraggiati dalla *First International Nurse Rostering Competition 2010* [21].
Nella letteratura più recente riguardante la turnazione delle infermiere si può riscontrare inoltre una forte tendenza all'uso di una schedulazione ciclica dei turni (spesso sulle 5 settimane) [6, 17]. Nel nostro modello tale ciclicità si può evidenziare nella schedulazione dei nottanti.

Blöchliger I. (2004) in [2] propone un tutorial molto utile per la modellizzazione di un problema di *staff scheduling*; in particolare viene trattato il caso del personale di un ospedale. Vengono forniti tutti gli strumenti per la raccolta e la definizione dei dati e per la formulazione di vincoli e funzione obiettivo, ma soprattutto ciò che è interessante è l'attenzione rivolta all'identificazione di parti del problema che possono essere trattate in maniera indipendente. Ad esempio, la schedulazione del personale che si occupa delle pulizie non interferisce con la schedulazione degli impiegati d'ufficio; tuttavia bisogna tener conto del fatto che non è possibile pulire una sala operatoria se è in corso un intervento. Queste riflessioni possono sembrare banali, eppure risultano essere di grande rilevanza poichè sfruttare l'indipendenza di alcuni fattori può ridurre largamente la dimensione del problema. É dunque necessario individuare gruppi, strutture o fattori indipendenti all'interno del problema (ad esempio tra il personale o tra unità operative dell'ospedale), sebbene questi non lo sembrino a prima vista.
Questa decomposizione non aiuta a modellare il problema, ma può avere un impatto significativo a livello computazionale e sulla qualità della soluzione.
Purtroppo nel caso del nostro modello né i moduli né i dipendenti possono essere considerati come entità totalmente indipendenti. L'unica accortezza che possiamo avere può essere quella di considerare come fissati a priori i turni dei

nottanti, che seguono la loro sequenza ciclica indipendentemente dalla schedulazione dei restanti operatori; questo ci permette di restringere il numero di operatori a cui assegnare i turni.
Blöchliger conclude proponendo alcune tecniche risolutive per problemi di questo tipo, suggerendo due strade: si può tentare di modellare il problema in modo che si adatti perfettamente a una formulazione conosciuta e usare strumenti standard per risolverlo, oppure si possono applicare algoritmi euristici.
Vincoli logici e tecniche deduttive rappresentano una terza possibilità per approcciare i problemi di *staff scheduling* [1, 10].

Horio e Suzuki (2010) [22] mostrano come risolvere problemi di *manpower scheduling* tramite un'estensione del RCPSP (*Resource-Constrained Project Scheduling Problem*).
Nei problemi di schedulazione compaiono molti tipi diversi di vincoli, ma non tutti riguardano quantità limitate di risorse, come invece generalmente accade nei problemi *resource-constrained*. Infatti, se da un lato abbiamo un numero massimo di impiegati disponibili, abbiamo anche vincoli come il numero minimo di impiegati necessari o la pausa da garantire tra un periodo lavorativo e il successivo. In generale non è possibile trasformare questi vincoli per adattarli al RCPSP, ma gli autori propongono una nuova formulazione tramite un'estensione del RCPSP, chiamata RCPSP/τ, e l'utilizzo di risorse fittizie. Inoltre, grazie allo sviluppo del solutore SEES (*Scheduling Expert Engine System*) possono risolvere il problema della turnazione in pochi minuti. Per la buona riuscita dell'algoritmo è necessario fissare a priori il numero di impiegati coinvolti nella schedulazione e il totale dei giorni lavorativi di ogni impiegato all'interno dell'orizzonte temporale considerato per la programmazione.
In particolare in [22] viene considerato un problema di schedulazione delle infermiere su tre tipi di orario.

Un problema simile a quello da me trattato è stato affrontato anche nel lavoro di tesi *"Integrazione di un algoritmo per l'ottimizzazione dei turni del personale in un sistema informativo aziendale"* [30], dove è stato proposto un modello PLI per la sua formulazione. In tale elaborato non vengono però discussi risultati computazionali e, da quanto descritto, il data set utilizzato si riferisce a istanze di dimensione inferiore a quelle testate in questo lavoro.

In conclusione il nostro problema risulta essere ampiamente trattato in letteratura e non sembra presentare sfaccettature particolarmente peculiari. Si tratterà quindi di identificare gli aspetti similari ai molteplici problemi proposti e sfruttare al meglio i metodi riportati, sia a livello di modellizzazione che di risoluzione.

2.2 Modello matematico per la schedulazione degli orari di lavoro del personale della struttura "Il Gignoro"

2.2.1 Descrizione dei dati

La descrizione dettagliata delle caratteristiche dei dati del problema risulterà fondamentale al fine di fornire un modello matematico corretto che porti a scelte algoritmiche e soluzioni soddisfacenti. Il momento della raccolta dei dati è tra i più importanti nella procedura di modellizzazione e richiede informazioni quanto più esaustive possibili.

Nel seguito ignoreremo il concetto di modulo, poiché risulta ininfluente ai fini della modellizzazione.
I dati forniti in input sono:

- **orizzonte di pianificazione:** considereremo l'intervallo D coperto dalla programmazione mensile; sarà identificato da d giorni, $D = \{1, ..., d\}$, con $d \in \{28, 30, 31\}$ (oppure $d \in \{29, 30, 31\}$ se l'anno è bisestile).

- **operatori:** denoteremo con P l'insieme degli operatori, $P = \{1, ..., p\}$.

- **turni:** per identificare i turni possibili è necessario combinare adeguatamente i tipi di orario, considereremo quindi l'insieme $Z = \{1, ..., z\}$ dei tipi di orario. Ciascun tipo di orario è caratterizzato da un intervallo temporale, da una qualifica richiesta e dal modulo (o da un'alternativa di moduli) a cui è associato; tali informazioni sono ininfluenti ai fini modellistici e se ne tiene conto solo in fase di creazione dei dati in input. Può accadere infatti che due tipi di orario presentino la stessa finestra temporale ma due qualifiche diverse oppure che siano richiesti da due moduli diversi; tali tipi di orario risulteranno quindi distinti all'interno dell'insieme Z.
  Sia T l'insieme dei turni derivanti dalle varie combinazioni, $T = \{0, ..., t\}$, e sia h_s il numero di ore previste per il turno $s \in T$, che contribuiranno a formare il budget orario dell'operatore a cui viene assegnato quel turno.
  Supponiamo per comodità che il turno con indice 0 rappresenti le ferie e quello con indice 1 rappresenti il giorno di riposo.

- **turni ammissibili per un operatore:** sia G l'insieme delle coppie (x, s) con $x \in P$ e $s \in T$. Ogni coppia in G indica che l'operatore x ha la qualifica necessaria per svolgere il turno s. A ciascuna coppia è associato un peso $g_{x,s} \in \mathbb{R}$ che terrà conto di tre fattori: una penalità per gli assegnamenti di operatori con una certa qualifica a turni per cui è richiesta una

qualifica inferiore, la preferenza eventualmente espressa dall'operatore per quel turno e una penalità per gli assegnamenti di operatori di un certo modulo (non "Jolly") a turni s che coprono orari di un altro modulo.
Indicheremo con $T(x) \subseteq T$ l'insieme dei turni ammissibili per l'operatore x. Se un turno risulta essere ammissibile per un operatore allora tale assegnamento è possibile per ogni giorno dell'orizzonte di programmazione.

- **domande di copertura:** sia $r_{k,u} \in \mathbb{N}$ il numero di operatori richiesti per il giorno $k \in D$ per il tipo di orario $u \in Z$. $r_{k,u} = 0$ se e solo se non ci sono richieste di quel tipo.

- **assegnamenti programmati:** sia B l'insieme delle triple (x, s, k) con $x \in P, s \in T(x), k \in D$, che indicano che l'operatore x deve essere assegnato al turno s nel giorno k.

- **matrice di incidenza turni-orari:** denoteremo con A la matrice di incidenza tra tipi di orario richiesti dai moduli e turni ammissibili, dove

$$a_{s,u} = \begin{cases} 1 & \text{se il turno } s \text{ copre il tipo di orario } u \\ 0 & \text{altrimenti} \end{cases},$$

con $s \in T$ e $u \in Z$.
Questa matrice ci permette di identificare con chiarezza la situazione rispetto alla domanda di copertura da parte dei moduli e alla rispettiva capacità di risposta da parte degli operatori.
In questo modo, per ogni modulo e per ogni giorno, il problema si ridurrà a quantificare il numero di turni (e quindi operatori) associati a una certa qualifica che serve per coprire i tipi di orari richiesti dal modulo in quel giorno.

- **scostamento dal budget orario:** sia $y \in \{1, ..., 12\}$ il mese considerato per la programmazione. Denoteremo con $v_{x,f} \in \mathbb{R}$ lo scostamento in termini di ore del budget orario previsto per il mese $f \in \{1, ..., y-1\}$ dalle ore effettivamente svolte dall'operatore $x \in P$ nel mese in esame. Lo scostamento può essere in eccesso o in difetto.
In particolare, il budget orario previsto mensilmente ammonta a $\frac{37}{6} \times n$ ore, dove $\frac{37}{6}$ è il numero medio di ore lavorative giornaliere e n è il numero di giorni lavorativi del mese considerato.
I valori $v_{x,f}$, calcolati e memorizzati in seguito alla stesura degli orari dei mesi precedenti a quello considerato, ci serviranno al fine di penalizzare

in funzione obiettivo lo scostamento accumulato; $\forall x \in P$ considereremo la costante $v_x = \sum_{f=1}^{y-1} v_{x,f}$, ossia la somma degli scostamenti raggiunti dall'operatore x nei mesi precedenti a quello considerato.

- **parametri di penalità in funzione obiettivo:** siano $\eta_1, ..., \eta_{11} \in \mathbb{R}$ i coefficienti di penalità che compariranno in funzione obiettivo al fine di assegnare un peso ad ogni violazione dei criteri che contribuiscono a definirla. Si considerino ordinati in modo decrescente, $\eta_1 > ... > \eta_{11}$, in modo da sottolineare l'ordine di priorità delle richieste. É assegnato un peso maggiore ai criteri più stringenti.
  In realtà i pesi η_i non devono necessariamente essere tutti distinti, ad esempio nel caso di termini con pari priorità.

Ai fini della modellizzazione, assumeremo che l'intervallo di programmazione $D = \{1, ..., d\}$ possa essere partizionato in quattro insiemi settimanali, $S_1, ..., S_4$. Considereremo quindi gli insiemi $S_1 = \{1, ..., \lfloor d/4 \rfloor\}$, $S_2 = \{\lfloor d/4 \rfloor + 1, ..., \lfloor d/2 \rfloor\}$, $S_3 = \{\lfloor d/2 \rfloor + 1, ..., \lfloor 3d/4 \rfloor\}$, $S_4 = \{\lfloor 3d/4 \rfloor + 1, ..., d\}$.
In accordo a tale assunzione modellistica, in sede di programmazione dovremo assegnare ad ogni operatore almeno due turni di tipo 1 (turno di riposo) nel periodo $S_1 \cup S_2$, che denoteremo D_1, e altrettanti nel periodo $S_3 \cup S_4$, che denoteremo con D_2. D_1 e D_2 indicano rispettivamente la prima e la seconda metà del mese.
Sarebbe preferibile avere un turno di tipo 1 in ogni periodo S_j $(j = 1, ..., 4)$, in modo da garantire un turno di riposo alla settimana.

Il fine ultimo della programmazione sarà assegnare esattamente un turno $s \in T(x)$ ad ogni operatore $x \in P$ per ogni giorno $k \in D$ dell'orizzonte di pianificazione (compresi ferie, riposi, ecc).

2.2.2 Formulazione dei vincoli

Nel corso della descrizione dei vincoli del problema terremo conto di tutti gli insiemi e delle variabili sopra definiti; ci sarà inoltre indispensabile un insieme di variabili decisionali binarie, che indicheremo con $\alpha_{x,s,k}$, $\forall x \in P, s \in T(x), k \in D$:

$$\alpha_{x,s,k} = \begin{cases} 1 & \text{se l'operatore } x \text{ è assegnato al turno } s \text{ nel giorno } k \\ 0 & \text{altrimenti} \end{cases} \tag{2.6}$$

Saranno i valori ottimi assegnati a tali variabili ad individuare la programmazione ottima mensile degli operatori.

In ogni giorno dell'orizzonte di programmazione deve essere assegnato a ogni operatore esattamente un turno. Terremo conto soltanto dei turni che risultano ammissibili per l'operatore:

$$\sum_{s \in T(x)} \alpha_{x,s,k} = 1 \qquad \forall x \in P, \quad k \in D \tag{2.7}$$

Dovremo tenere conto degli assegnamenti fissati precedentemente alla programmazione; sarà il caso delle ferie, delle malattie o riposi programmati, dei turni in quinta dei nottanti, delle riunioni e dei corsi di formazione:

$$\alpha_{x,s,k} = 1 \qquad \forall (x,s,k) \in B \tag{2.8}$$

Come detto in precedenza, vogliamo garantire a ogni operatore una distribuzione uniforme dei riposi, cioè turni di tipo 1, nell'arco dell'orizzonte temporale D; vorremmo assegnarne almeno due in ognuna delle due metà del mese. Considereremo quindi la variabile non negativa $\gamma_{x,h}$, che registra il numero di violazioni a questo vincolo: $\gamma_{x,h} = 0$ se e solo se vengono assegnati almeno due riposi nel periodo D_h, $h \in \{1,2\}$, all'operatore x; altrimenti conta il numero di riposi non assegnati.

$$\sum_{k \in D_h} \alpha_{x,1,k} + \gamma_{x,h} \geq 2 \qquad \forall x \in P, \quad h \in \{1,2\} \tag{2.9}$$

$$\gamma_{x,h} \geq 0 \qquad \forall x \in P, \quad h \in \{1,2\} \tag{2.10}$$

In particolare vorremo prevedere almeno un giorno di riposo alla settimana. Considereremo quindi la variabile non negativa $\delta_{x,j}$, che registra il numero di violazioni a questa richiesta: $\delta_{x,j} = 0$ se e solo se è stato assegnato almeno un riposo nella settimana S_j, $j \in \{1,...,4\}$, all'operatore x.

$$\sum_{k \in S_j} \alpha_{x,1,k} + \delta_{x,j} \geq 1 \qquad \forall x \in P, \quad j \in \{1,...,4\} \tag{2.11}$$

$$\delta_{x,j} \geq 0 \qquad \forall x \in P, \quad j \in \{1,...,4\} \tag{2.12}$$

Per ogni giorno dell'intervallo temporale coperto dalla pianificazione dovremo provvedere a soddisfare la copertura delle richieste dei moduli, che avvengono per numero di operatori necessari in un certo tipo di orario. Dovremo quindi assegnare agli operatori turni che coprano tali orari.
Denoteremo con la variabile non negativa $\theta^{+}_{k,u}$ il coefficiente che registra quante richieste sono rimaste scoperte per il tipo di orario u nel giorno k e con la vari-

abile non negativa $\theta^-_{k,u}$ il coefficiente che registra il sovrannumero di operatori rispetto alla richiesta fatta per il tipo di orario u nel giorno k:

$$\sum_{x \in P} \sum_{\substack{s \in T(x): a_{s,u}=1, \\ a_{s,u} \in A}} \alpha_{x,s,k} + \theta^+_{k,u} - \theta^-_{k,u} = r_{k,u}$$

$$\forall k \in D, \quad u \in Z \tag{2.13}$$

$$\theta^+_{k,u} \geq 0, \quad \theta^-_{k,u} \geq 0 \qquad \forall k \in D, \quad u \in Z \tag{2.14}$$

$\theta^+_{k,u}$ e $\theta^-_{k,u}$ saranno uguali a zero se e solo se le richieste di copertura del tipo di orario u nel giorno k sono state soddisfatte esattamente. Queste variabili di scarto ci aiutano a identificare le situazioni problematiche in cui il servizio non è garantito. L'eccesso di operatori utilizzati per coprire un certo tipo di orario sarà penalizzato in maniera sostanzialmente meno grave rispetto alla mancanza di copertura.

Secondo quanto stabilito da contratto dovremo prevedere 37 ore di lavoro settimanali per ogni operatore. Denoteremo con $\varepsilon^+_{x,j}$ la variabile che registra lo scostamento in eccesso delle 37 ore previste dalle reali ore assegnate all'operatore x nella settimana S_j del mese considerato in programmazione; con $\varepsilon^-_{x,j}$ quella che registra lo scostamento in difetto.

$$\sum_{s \in T(x)} \sum_{k \in S_j} h_s \cdot \alpha_{x,s,k} + \varepsilon^+_{x,j} - \varepsilon^-_{x,j} = 37 \qquad \forall x \in P, \quad j \in \{1, ..., 4\} \tag{2.15}$$

$$\varepsilon^+_{x,j} \geq 0, \quad \varepsilon^-_{x,j} \geq 0 \qquad \forall x \in P, \quad j \in \{1, ..., 4\} \tag{2.16}$$

Dovremmo inoltre prevedere $\frac{37}{6} \times n$ ore di lavoro mensili per ogni operatore, dove n corrisponde al numero di giorni lavorativi del mese. Denoteremo con λ^+_x la variabile che registra lo scostamento in eccesso delle ore mensili previste dalle reali ore assegnate all'operatore x nel mese considerato in programmazione, corrispondente all'intervallo temporale D. Denoteremo con con λ^-_x la variabile che registra lo scostamento in difetto.

$$\sum_{s \in T(x)} \sum_{k \in D} h_s \cdot \alpha_{x,s,k} + \lambda^+_x - \lambda^-_x = \frac{37}{6} \times n \qquad \forall x \in P \tag{2.17}$$

$$\lambda^+_x \geq 0, \quad \lambda^-_x \geq 0 \qquad \forall x \in P \tag{2.18}$$

Infine dovremo tener conto dello scostamento accumulato mese dopo mese. Indicheremo con β^+_x lo scostamento totale in eccesso e con β^-_x lo scostamento totale in difetto accumulati dall'operatore x nei mesi, considerando anche quello

in programmazione:

$$\beta_x^+ - \beta_x^- = v_x + \lambda_x^+ - \lambda_x^- \qquad \forall x \in P \tag{2.19}$$

$$\beta_x^+ \geq 0, \quad \beta_x^- \geq 0 \qquad \forall x \in P \tag{2.20}$$

Nell'affrontare questa modellizzazione abbiamo supposto che il problema non fosse decomponibile in sottoproblemi più semplici e indipendenti tra loro. La soluzione ottima sarebbe stata fornita dalla ricombinazione delle soluzioni ottime dei sottoproblemi.
Come già detto non è infatti possibile portare avanti una schedulazione per moduli, a causa della presenza del modulo "Jolly" e delle possibili assegnazioni di operatori a turni che coprono orari richiesti da un modulo diverso dal proprio; né si può decomporre il problema per livelli di qualifica, poiché abbiamo visto che in caso di necessità operatori con una qualifica di un certo grado possono essere assegnati anche a orari a cui è associata una qualifica di livello inferiore. Una strada percorribile potrebbe essere la decomposizione del problema per tipi di orario legati a certe qualifiche; sarebbe possibile sotto la condizione che esistano uno o più gruppi di operatori che possono svolgere esclusivamente un certo numero di tipi di orario e non ne possono svolgere altri, questo a causa della loro qualifica e della mansione associata agli orari. In questo modo tali gruppi risulterebbero indipendenti ai fini dell'assegnazione dei turni.

2.2.3 Formulazione della funzione obiettivo

Dato un problema di *scheduling*, la funzione obiettivo complessiva da minimizzare è la somma dei pesi delle violazioni dei vincoli soft. Scegliamo una minimizzazione totale delle violazioni dei vincoli come unica funzione obiettivo, strutturandola come somma di diversi termini lineari.

La funzione obiettivo per il nostro modello risulterà essere la minimizzazione della somma di tutti i termini descritti in seguito (elencati in ordine decrescente di priorità):

- il principale obiettivo della programmazione sarà quello di rispondere in maniera adeguata e soddisfacente a tutte le domande di copertura fatte dai moduli per ogni giorno del mese. Sarà quindi necessario minimizzare il numero di richieste non coperte dalla programmazione:

$$f_1 = \sum_{k \in D} \sum_{u \in Z} \theta_{k,u}^+$$

- in previsione di minimizzare a fine anno lo sforamento (in eccesso o in difetto) del budget orario di ogni operatore, penalizzeremo gli scostamenti accumulati da ciascuno mese dopo mese:

$$f_2 = \sum_{x \in P} \beta_x^+$$

$$f_3 = \sum_{x \in P} \beta_x^-$$

- al fine di minimizzare lo sforamento mensile (in eccesso o in difetto) del budget orario di ogni operatore, penalizzeremo lo scostamento registrato nel mese considerato in programmazione:

$$f_4 = \sum_{x \in P} \lambda_x^+$$

$$f_5 = \sum_{x \in P} \lambda_x^-$$

- al fine di minimizzare lo sforamento settimanale (in eccesso o in difetto) del budget orario di ogni operatore, penalizzeremo lo scostamento registrato ogni settimana dalle 37 ore previste da contratto:

$$f_6 = \sum_{x \in P} \sum_{j \in \{1,\dots,4\}} \varepsilon_{x,j}^+$$

$$f_7 = \sum_{x \in P} \sum_{j \in \{1,\dots,4\}} \varepsilon_{x,j}^-$$

- vorremo penalizzare inoltre la mancata assegnazione ad ogni operatore di almeno due riposi in ciascuna metà del mese:

$$f_8 = \sum_{x \in P} \sum_{h \in \{1,2\}} \gamma_{x,h}$$

- penalizzeremo con un peso minore la mancata assegnazione ad ogni operatore di un riposo alla settimana:

$$f_9 = \sum_{x \in P} \sum_{j \in \{1,\dots,4\}} \delta_{x,j}$$

- volendo tener conto in maniera opportuna delle qualifiche degli operatori, delle loro preferenze e delle associazioni operatore-modulo, consideriamo

il coefficiente $g_{x,s}$ associato ad ogni coppia $(x, s) \in G$. Vogliamo minimizzare gli assegnamenti che risultano meno graditi o che affidano ad un operatore un turno che richiede una qualifica inferiore alla mansione per cui è preposto o che copre la richiesta di un modulo che non è il proprio; cercheremo quindi di avere $\alpha_{x,s,k} = 1$ quando il valore di $g_{x,s}$ è più basso:

$$f_{10} = \sum_{x \in P} \sum_{s \in T(x)} g_{x,s} \sum_{k \in D} \alpha_{x,s,k}$$

- vogliamo infine penalizzare il sovrannumero di operatori assegnati a soddisfare la domanda di copertura per il tipo di orario u nel giorno k:

$$f_{11} = \sum_{k \in D} \sum_{u \in Z} \theta^{-}_{k,u}$$

Una volta descritte le funzioni f_i, è ora necessario scalarle tramite opportuni pesi che ne modellino l'ordine di priorità nella minimizzazione, rafforzandoli o rilassandoli anche in base allo scenario di schedulazione che dobbiamo affrontare. A tal proposito considereremo i pesi $\eta_1, ..., \eta_{11}$ definiti in precedenza, con η_i associato a f_i, $i \in \{1, ..., 11\}$.

In conclusione, la funzione obiettivo del nostro modello risulterà:

$$\min \sum_{i=1}^{11} \eta_i \cdot f_i$$

Gli η_i sono fattori di scala ordinati in maniera decrescente, $\eta_1 > ... > \eta_{11}$, che modellano l'idea secondo cui ad esempio è sempre preferibile far diminuire di un'unità f_1 piuttosto che far passare una qualsiasi f_i, $i \in \{2, ..., 11\}$, dal suo valore massimo al suo valore minimo.

Capitolo 3

Implementazione del modello e risultati

Dopo aver analizzato il problema riguardante la struttura "Il Gignoro" e aver formulato il modello matematico volto a risolverlo, è stata effettuata una verifica dei concetti sopra esposti tramite esperimenti computazionali svolti su dati reali forniti dalla struttura; sono stati quindi sviluppati strumenti software che hanno permesso di ottenere risultati utili al fine di valutare l'efficacia e l'efficienza del metodo proposto.

3.1 Strumenti implementativi

L'implementazione dell'algoritmo è stata effettutata tramite il linguaggio di programmazione C++, sfruttando librerie sviluppate da COIN-OR, l'interfaccia OSI, tre diversi solutori per problemi di Programmazione Lineare Intera (Cbc, GLPK e CPLEX) e un linguaggio di modellazione algebrico, FlopC++.

3.1.1 COIN-OR

Il progetto COIN-OR [1] (*COmputational INfrastructure for Operations Research*) è volto ad incoraggiare lo sviluppo di software open-source per la Ricerca Operativa.
Negli articoli di ricerca vengono molto spesso forniti risultati numerici a sostegno delle tesi esposte; tuttavia, i dettagli implementativi, i modelli e i dati utilizzati tipicamente non vengono pubblicati, rendendo di fatto complessa la riproduzione o il confronto dei risultati computazionali. Il successo negli anni dell'idea di libera distribuzione ha spinto un gruppo di ricercatori dell'IBM a concepire un

[1] http://www.coin-or.org

progetto di sviluppo e distribuzione di software open-source in cui fosse possibile pubblicare implementazioni, modelli e dati in modo analogo alla pubblicazione di un articolo scientifico. COIN-OR è un'iniziativa nata proprio con lo scopo di costruire una comunità open-source che da una parte velocizzi lo sviluppo di modelli, algoritmi e ricerche computazionali d'avanguardia e dall'altra fornisca un punto di discussione e confronto sul software stesso.

Oggi COIN-OR comprende progetti di vario genere: strumenti di ottimizzazione lineare continua (Clp, DyLP), nonlineare (IPOPT), stocastica (SMI), discreta (Cbc, SYMPHONY); tool per la rappresentazioni di grafi o reti (Cgc); librerie di modellazione algebrica (FlopC++) e un numero di interfacce di alto livello utilizzabili per accedere a classi di risolutori (OSI, CoinMP).
É possibile effettuare il download di ogni distribuzione in più versioni dal rispettivo sito internet, ognuna adattata a diverse esigenze. Inoltre ogni pacchetto è sostituibile ed aggiornabile ogniqualvolta vengano apportati dei miglioramenti, visto il continuo sviluppo con licenza open-source.

3.1.2 OSI

L'interfaccia OSI [2] (*Open Solver Interface*) è una collezione di classi C++ distribuita gratuitamente sotto Common Public License 1.0, che fornisce gli strumenti per interagire con solutori di problemi di programmazione lineare intera e mista. OSI supporta numerosi solutori, sia commerciali che liberamente distribuiti. Questo assicura una sorta di astrazione rispetto al solutore scelto, permettendo di isolare l'applicazione del particolare solutore. La scelta del pacchetto da utilizzare può essere effettuata a *run-time*, rendendo possibili dei confronti fra software in modo rapido e diretto. I solutori attualmente supportati sono: COIN-OR LP solver (OsiClp), COIN-OR Branch and Cut solver (OsiCbc), CPLEX (OsiCpx), DyLP (OsiDylp), FortMP (OsiFmp), GLPK (OsiGlpk), Mosek (OsiMsk), SYMPHONY (OsiSym), The Volume Algorithm (OsiVol) e XPRESS-MP (OsiXpr).

Il nucleo di OSI è una classe astratta, OsiSolverInterface, in cui sono dichiarati i principali metodi che permettono di agire su un generico risolutore. Da questa sono derivate, per ogni *solver*, delle specifiche classi che contengono le vere e proprie implementazioni. Ad esempio OsiCbc è la classe derivata per il solutore Cbc, mentre OsiCpx è la classe derivata per CPLEX.

Tra le funzionalità offerte dall'interfaccia vi sono metodi per:

- caricare un problema da file;

[2] http://www.coin-or.org/projects/Osi.xml

- risolvere il problema;
- ottenere informazioni sul processo di risoluzione;
- costruire, estrarre o modificare il modello (vincoli, funzione obiettivo, numero e tipo di variabili);
- salvare le soluzioni.

Lo scopo di OSI è quello di fornire una valida e affidabile alternativa ai software a pagamento per la risoluzione di problemi del tipo LP e MILP, senza porre virtualmente limiti di variabili.

3.1.3 Cbc, GLPK e CPLEX

Cbc [3] (*COIN-OR branch-and-cut*) è un software di calcolo liberamente distribuito per problemi di programmazione lineare intera mista scritto in C++ che fa parte dei progetti COIN-OR. Può essere utilizzato come libreria o eseguito in modo autonomo da terminale e si affida ad altri progetti COIN-OR per funzionalità aggiuntive (OSI, Clp, Cgl, CoinUtils, ...). Il solutore Cbc è testato per problemi con oltre 1.000.000 di variabili ed è pertanto accessibile all'utente un grande potenziale di calcolo open-source.
Questo software ha valso ai suoi sviluppatori, Andrew Mason e Ian Dunning dell'Università di Auckland, il primo premio della *COIN-OR INFORMS Cup* del 2011, patrocinata da IBM.

GLPK [4] (*GNU Linear Programming Kit*) è una libreria software scritta in ANSI C e liberamente distribuita, utilizzabile per risolvere sia problemi di programmazione lineare continui (LP) che misti interi (MILP). Include il metodo del simplesso primale e duale e un metodo a punto interno per la soluzione di problemi lineari; per la soluzione di problemi interi e misti interi viene utilizzato invece il metodo del *branch-and-bound*. Il pacchetto GLPK propone inoltre un solutore *"stand-alone"* per LP/MIP.

IBM ILOG CPLEX Optimization Studio [5], informalmente abbreviato con CPLEX, è un software commerciale per l'ottimizzazione molto noto, è sviluppato da ILOG ed è in grado di risolvere problemi di programmazione lineare anche di notevoli dimensioni, di programmazione lineare intera e quadratica convessa. CPLEX prende il nome dal metodo del simplesso implementato in linguaggio C, anche se ad oggi comprende anche algoritmi addizionali nel campo della programmazione matematica ed offre interfacce verso altri ambienti e linguaggi,

[3] http://www.coin-or.org/projects/Cbc
[4] http://www.gnu.org/software/glpk/glpk.html
[5] http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/index.html

come ad esempio il C++. Esiste inoltre un eseguibile interattivo *"stand-alone"* utile per il *debugging*.
Originariamente sviluppato da Robert E. Bixby, è stato commercializzato a partire dal 1988 dalla CPLEX Optimization Inc., acquisita da ILOG nel 1997, a sua volta acquisita dall IBM nel 2009. CPLEX è attualmente mantenuto attivo e sviluppato in IBM.
Nel 2004, lo sviluppo di CPLEX è stato insignito del premio *INFORMS Impact Prize*.

3.1.4 FlopC++

FlopC++ [6] (*Formulation of Linear Optimization Problems in C++*) è un linguaggio open-source di modellazione algebrico implementato come libreria di classi C++.
Usando FlopC++ i modelli di ottimizzazione lineare possono essere definiti in un semplice stile dichiarativo all'interno di un programma scritto in C++. Viene conservata la tradizionale forza di un linguaggio di modellazione algebrico e allo stesso tempo viene molto facilitato l'incorporamento di modelli di ottimizzazione lineari in applicazioni software.
Il download dal sito della distribuzione FlopC++ include automaticamente anche i progetti OSI, Cbc e altri, indispensabili al suo utilizzo.

3.2 Implementazione del modello matematico

Al fine di effettuare test computazionali sul modello formulato è stato necessario implementare un codice che:

1. legge i dati dell'istanza da un file in formato opportuno;
2. costruisce il modello e lo passa ad un solutore;
3. chiama il solutore;
4. prende la soluzione individuata dal solutore e la traduce in un formato di output opportuno.

Tale codice risulta funzionare come una *black box* in cui si inseriscono i dati e da cui escono i risultati.

Per la lettura in input si è deciso di utilizzare un semplice file di testo (.txt) in cui vengono elencate le dimensioni e i dati del problema. Servirà inoltre

[6] http://www.coin-or.org/projects/FlopC++.xml

un file testuale da cui leggere i coefficienti di penalità da assegnare in funzione obiettivo.

La costruzione del modello all'interno dell'algoritmo può avvenire in due modi: costruendo la matrice dei coefficienti del sistema lineare riga per riga e colonna per colonna, usando i metodi opportuni di OsiSolverInterface [7], oppure utilizzando un linguaggio di modellazione, come ad esempio FlopC++ in C++. L'interfaccia OSI fornisce infatti varie funzioni che permettono la definizione del modello di un problema. Esistono dei metodi che consentono di manipolare direttamente la matrice del sistema ed i vincoli sulle sue righe e colonne; tale costruzione però prevede l'inserimento dei coefficienti uno ad uno e non risulta molto conveniente, soprattutto per istanze di grandi dimensioni, come nel nostro caso. Pertanto all'interno dell'algoritmo si è deciso di costruire il modello usando gli opportuni costrutti di FlopC++ (vedi Appendice A).

Una volta costruito, il modello viene passato ad un solutore che procede alla ricerca della soluzione ottima del problema; per i nostri test abbiamo utilizzato i solver Cbc e CPLEX tramite le classi derivate di Osi, mentre per il solutore GLPK è stata sfruttata la sua versione eseguibile fornendo in input il modello del problema in formato .lp.

Una volta ottenute le soluzioni il codice fornisce in output un file di testo che conterrà gli assegnamenti operatore-turno per ogni giorno dell'orizzonte di pianificazione considerato.

Le versioni delle librerie sfruttate sono: OSI 0.99, FlopC++ 1.1, Cbc 2.8, GLPK 4.53 e CPLEX 12.5.

3.3 Risultati implementativi

I test sono stati effettuati dando in input dati reali estratti dalla programmazione effettiva della struttura "Il Gignoro" per il mese di marzo dell'anno in corso, inizialmente su una versione semplificata dell'istanza e in seguito su una versione completa.

Poiché non tutte le istanze testate sono risultate risolvibili in un intervallo di tempo compatibile con le esigenze di programmazione della struttura sanitaria, si è deciso di usare l'algoritmo implementato per ottenere soluzione ammissibile, sperabilmente di buona qualità, entro tempi limite opportunamente stabiliti, rinunciando quindi all'individuazione di una soluzione che sia certamente ottima per il modello matematico proposto.

[7] http://coral.ie.lehigh.edu/ coin/COIN/Osi/Doc/html/classOsiSolverInterface.html

Illustriamo adesso alcuni dei risultati ottenuti utilizzando il solutore CPLEX con tre diversi tempi limite (5, 30 e 60 minuti) al variare dei quattro diversi settaggi scelti per i coefficienti di penalità (Set 1, ..., Set 4).

Per maggiori dettagli sui set di coefficienti utilizzati vedere l'appendice B.

Nelle tabelle i tempi sono espressi in termini di secondi; compare inoltre il valore ottimo della funzione obiettivo e, al variare dei tempi di esecuzione, il valore assunto dalla funzione obiettivo in corrispondenza della soluzione trovata. Il gap a terminazione fornisce un'indicazione della qualità della soluzione riportando il valore percentuale della distanza della soluzione trovata dall'ottimalità.

Riportiamo inoltre indicatori di qualità della soluzione tramite lo studio dei valori assunti dalle variabili di scarto.
Richiamando le definizioni date nel Capitolo 2, indichiamo con $f_1 = \sum_{k \in D} \sum_{u \in Z} \theta^+_{k,u}$ il numero di richieste di copertura non soddisfatte nell'arco dell'orizzonte di programmazione; con $f_4 = \sum_{x \in P} \lambda^+_x$ riportiamo il numero mensile complessivo di ore di straordinario, mentre con $f_5 = \sum_{x \in P} \lambda^-_x$ il numero totale di ore mancanti per il raggiungimento della soglia mensile prevista da contratto.
Ricordiamo le funzioni $f_6 = \sum_{x \in P} \sum_{j \in \{1,...,4\}} \varepsilon^+_{x,j}$ e $f_7 = \sum_{x \in P} \sum_{j \in \{1,...,4\}} \varepsilon^-_{x,j}$ rappresentanti lo scostamento complessivo (in eccesso e in difetto rispettivamente) dalle 37 ore previste per ogni settimana delle quattro dell'intervallo di pianificazione; indichiamo con $f_6\ (i)$ e $f_7\ (i)$ gli scostamenti registrati durante l'i-esima settimana.
Il valore di $f_8 = \sum_{x \in P} \sum_{h \in \{1,2\}} \gamma_{x,h}$ rappresenta il numero di operatori a cui vengono negati i due riposi ogni due settimane, mentre il valore di $f_9 = \sum_{x \in P} \sum_{j \in \{1,...,4\}} \delta_{x,j}$ indica il numero di riposi settimanali non assegnati.
Infine con $f_{11} = \sum_{k \in D} \sum_{u \in Z} \theta^-_{k,u}$ riportiamo il numero di operatori in sovrannumero rispetto alle richieste di copertura espresse per il periodo di programmazione.

Omettiamo gli indicatori per $f_2 = \sum_{x \in P} \beta^+_x$ e $f_3 = \sum_{x \in P} \beta^-_x$ poiché assumono rispettivamente lo stesso valore di f_4 e f_5. Questo poiché per definizione per ogni operatore x vale $\beta^+_x - \beta^-_x = v_x + \lambda^+_x - \lambda^-_x$ e nei file di dati in input forniti da DVF risulta $v_x = 0 \quad \forall x$, vale a dire che nessun operatore ha accumulato scostamenti in difetto o in eccesso nei primi due mesi di programmazione dell'anno corrente.

Riportiamo prima di tutto le tabelle relative ai test effettuati sulla versione semplificata dei dati in input.

Set 1		Valore ottimo	191 156.7530
Time limit	300	1 800	3 600
Valore funzione obiettivo	458 882	410 616	410 464
Gap	58.34%	53.45%	53.43%
f_1	0	0	0
f_4	0.5	0	0
f_5	22.6667	19.6667	19.6667
f_6 (1)	218.833	218.333	218.333
f_6 (2)	210.5	207.5	210
f_6 (3)	360.5	353.5	357.5
f_6 (4)	182.5	170.5	169.5
f_7 (1)	9	9	9
f_7 (2)	15	8.5	9
f_7 (3)	5	0	0
f_7 (4)	19.5	6	3
f_8	0	0	0
f_9	0	0	0
f_{11}	13	11	11

Set 2		Valore ottimo	1.13522e+08
Time limit	300	1 800	3 600
Valore funzione obiettivo	2.31083e+11	1.61392e+11	1.61392e+11
Gap	99.95%	99.93%	99.93%
f_1	0	0	0
f_4	15.33333	3.333333	3.333333
f_5	21.5	23	23
f_6 (1)	232.833	229.833	229.833
f_6 (2)	235	233.5	233.5
f_6 (3)	364.5	356.5	356.5
f_6 (4)	181	181	181
f_7 (1)	27	27	27
f_7 (2)	17.5	17.5	17.5
f_7 (3)	7.5	7.5	7.5
f_7 (4)	21.5	22.5	22.5
f_8	0	0	0
f_9	0	0	0
f_{11}	7	13	13

Set 3		Valore ottimo	2.27060e+07
Time limit	300	1 800	3 600
Valore funzione obiettivo	3.69907e+10	3.69907e+10	3.63341e+10
Gap	99.94%	99.94%	99.94%
f_1	0	0	0
f_4	10	10	9.5
f_5	19.6667	19.6667	19.6667
f_6 (1)	214.833	214.833	214.833
f_6 (2)	228.5	228.5	228
f_6 (3)	358.5	358.5	358.5
f_6 (4)	182	182	182
f_7 (1)	10.5	10.5	10.5
f_7 (2)	17.5	17.5	11.5
f_7 (3)	3.5	3.5	3.5
f_7 (4)	16	16	16
f_8	0	0	0
f_9	0	0	0
f_{11}	8	8	8

Set 4		Valore ottimo	2 272 354.7946
Time limit	300	1 800	3 600
Valore funzione obiettivo	3.53334e+07	3.18494e+07	1.37174e+07
Gap	93.57%	92.87%	83.43%
f_1	0	0	0
f_4	5.5	2	1.5
f_5	21.1667	22.1667	5.6667
f_6 (1)	219.333	209.833	215.833
f_6 (2)	223.5	220	222.5
f_6 (3)	354.5	353	362.5
f_6 (4)	181.5	172	184.5
f_7 (1)	10	8.5	10.5
f_7 (2)	8	7.5	7
f_7 (3)	7	7	4.5
f_7 (4)	23.5	6	18.5
f_8	0	0	0
f_9	0	0	0
f_{11}	6	7	6

I test effettuati con il solutore open-source Cbc non riescono a trovare una soluzione intera del problema entro i tempi limite stabiliti. Sul primo set di coefficienti Cbc necessita di almeno 10 800 secondi prima di generare una soluzione ammissibile intera.

Altri test sono stati effettuati con il solutore open-source GLPK e anche in questo caso non è stato possibile trovare una soluzione ammissibile entro i tempi prefissati. E' stato verificato che per il primo set di coefficienti sul tempo necessario a Cbc per produrre il primo risultato (3 ore), GLPK non fornisce alcuna soluzione ammissibile intera.

Questo confronto sottolinea la differenza di prestazioni che può sussistere tra un solutore commerciale e solutori liberamente distribuiti.

I risultati dei test effettuati con il solutore CPLEX sono stati tradotti in un formato comprensibile ai pianificatori di DVF, in modo che potessero facilmente leggerli ed esprimere un giudizio in merito. I commenti sono stati positivi e incoraggianti e sollevano speranze in merito al reale impiego della procedura di ottimizzazione.

Riportiamo inoltre le tabelle inerenti ai test effettuati con la versione completa del file di dati. Alcune soluzioni risultano essere ottime e sono indicate dal valore 0,01% del gap. Dunque il modello settato su certi parametri di penalità (Set 1 e Set 4) risolve all'ottimo il problema reale.
Non forniamo i valori ottenuti dall'elaborazione su 60 minuti in quanto non evidenziano differenze rispetto all'esecuzione su 30 minuti.

Set 1	Valore ottimo	1.58896e+07
Time limit	300	1 800
Valore funzione obiettivo	1.58937e+07	1.58911e+07
Gap	0.03%	0.01%
f_1	2	2
f_4	0	0
f_5	1420.83	1420.83
f_6 (1)	5	1.5
f_6 (2)	1.5	1.5
f_6 (3)	57.167	52.167
f_6 (4)	0	0
f_7 (1)	144.667	141.167
f_7 (2)	141	141
f_7 (3)	9.667	4.667
f_7 (4)	199.167	199.167
f_8	0	0
f_9	0	0
f_{11}	521	519

Set 2	Valore ottimo	9.84284e+12
Time limit	300	1 800
Valore funzione obiettivo	3.84830e+13	9.87925e+12
Gap	74.42%	0.37%
f_1	48	2
f_4	20	0
f_5	1447.33	1426.83
f_6 (1)	32.5	19.5
f_6 (2)	92.167	64.5
f_6 (3)	108	142.667
f_6 (4)	47.5	23.667
f_7 (1)	205.167	159. 167
f_7 (2)	208.167	204
f_7 (3)	49	101.167
f_7 (4)	255.167	222.833
f_8	0	0
f_9	0	0
f_{11}	764	681

Set 3	Valore ottimo	1.96857e+12
Time limit	300	1 800
Valore funzione obiettivo	9.80804e+12	1.97100e+12
Gap	79.93%	0.12%
f_1	65	2
f_4	31.833	0
f_5	1447.83	1422.83
f_6 (1)	55.5	33.333
f_6 (2)	59.167	51.833
f_6 (3)	118.167	71.5
f_6 (4)	121.833	47.5
f_7 (1)	191.167	173
f_7 (2)	204.833	193.333
f_7 (3)	122.667	24
f_7 (4)	262	246.667
f_8	0	0
f_9	0	0
f_{11}	781	665

Set 4	Valore ottimo	1.96974e+09
Time limit	300	1 800
Valore funzione obiettivo	1.96993e+09	1.96993e+09
Gap	0.01%	0.01%
f_1	2	2
f_4	0	0
f_5	1420.83	1420.83
f_6 (1)	7	7
f_6 (2)	7.167	7.167
f_6 (3)	74.667	74.667
f_6 (4)	4	4
f_7 (1)	146.667	146.667
f_7 (2)	146.667	146.667
f_7 (3)	27.167	27.167
f_7 (4)	203.167	203.167
f_8	0	0
f_9	0	0
f_{11}	728	728

L'esecuzione dell'istanza completa con il solutore Cbc sul primo set di coefficienti non conduce a una soluzione ammissibile intera del problema entro i tempi limite stabiliti, né sul tempo limite di 3 ore.

Un'analoga situazione si riscontra utilizzando il solver GLPK.

I risultati sono stati ottenuti lavorando su una macchina con processore quad-core Xeon E5504 a 2GHz con 16Gb di RAM.

Appendice A

Estratti del codice: formato dei dati e costruzione del modello

In seguito vengono presentati alcuni estratti del codice utilizzato per i test implementativi. Il linguaggio di programmazione utilizzato è il C++.

Sono di particolare rilevanza le parti riguardanti la costruzione del modello tramite i costrutti peculiari del linguaggio di modellazione algebrica FlopC++ e l'interazione tra l'interfaccia OSI e il solutore.

Innanzitutto è indispensabile includere le librerie per il linguaggio

```
#include "flopc.hpp"
```

e per il solutore scelto (Cbc o CPLEX)

```
#if (WHICH_OSI)
 #include "OsiCbcSolverInterface.hpp"
#else
 #include "OsiCpxSolverInterface.hpp"
 #include "cplex.h"
#endif
```

Dopo aver letto il file di input e il file contenente i valori da assegnare ai coefficienti di penalità, e aver salvato tali dati in opportuni array e matrici, si può procedere alla parte saliente dell'algoritmo, vale a dire la creazione del modello matematico precedentemente formulato.
É necessario innanzitutto inizializzare un puntatore al solutore designato

```
#if (WHICH_OSI)
  OsiCbcSolverInterface *si = new OsiCbcSolverInterface;
#else
  OsiCpxSolverInterface *si = new OsiCpxSolverInterface;
#endif
```

dopodiché è possibile inizializzare il modello e passarlo al solutore

```
MP_model dvf(si);
```

L'utente può scegliere il tempo limite di esecuzione del programma, sia per il solutore Cbc

```
si->setMaximumSeconds(maxtime);
```

che per CPLEX

```
CPXENVptr env = si->getEnvironmentPtr();
status1 = CPXsetdblparam(env, CPX_PARAM_TILIM, maxtime);
```

Inoltre per CPLEX è possibile settare altri parametri interessanti, quali la rottura dell'eventuale simmetria dei dati del problema, la possibilità di aumentare la frequenza del processo euristico e l'aggressività di alcuni tipi di tagli:

```
status2 = CPXsetintparam(env, CPX_PARAM_SYMMETRY, 3);
status3 = CPXsetintparam(env, CPX_PARAM_HEURFREQ, 10);
status4 = CPXsetintparam(env, CPX_PARAM_FLOWCOVERS, 2);
status5 = CPXsetintparam(env, CPX_PARAM_MIRCUTS, 2);
```

Dopo aver inizializzato il modello, si procede quindi alla definizione degli insiemi, dei dati e degli indici necessari per la formulazione dei vincoli e della funzione obiettivo:

```
// INSIEMI

MP_set Operatore(n),      // indice i
       Orario(m),         // indice j
       Turno(s),          // indice w
       Giorno(t),         // indice k
       PreAss(l),         // indice z
       Settimana(4),      // indice b
       Mese(2),           // indice c
       Coefficiente(11);

// DATI

MP_data Costo(Operatore, Turno);
for(int i = 0; i < Operatore.size(); i++)
 for(int w = 0; w < Turno.size(); w++)
  Costo(i, w) = C[i][w];
delete[] C;

MP_data Richiesta(Orario, Giorno);
```

```
for(int j = 0; j < Orario.size(); j++)
 for(int k = 0; k < Giorno.size(); k++)
  Richiesta(j, k) = R[j][k];
delete[] R;

MP_data Budget(Operatore);
Budget.value(&V[0]);
delete[] V;

MP_data Ore(Turno);
Ore.value(&H[0]);
delete[] H;

MP_data Incidenza(Orario, Turno);
for(int j = 0; j < Orario.size(); j++)
  for(int w = 0; w < Turno.size(); w++)
    Incidenza(j, w) = A[j][w];
delete[] A;

MP_data Penalita(Coefficiente);
Penalita.value(&Q[0]);
delete[] Q;

// INDICI

MP_index i, w, k, j, z, b, c;
```

Ci serviranno inoltre alcuni sottoinsiemi degli insiemi sopra definiti:

```
// SOTTOINSIEMI

// triple ammissibili per gli assegnamenti

MP_subset<3> TurnoAmm(Operatore, Turno, Giorno);
forall(Operatore(i) * Turno(w) *
  Giorno.such_that(!(Costo(i, w) < 0)),
    TurnoAmm.insert(i, w, Giorno));

// preassegnamenti

MP_subset<3> Tripla(Operatore, Turno, Giorno);
for (int z = 0; z < l; z++)
  Tripla.insert(T[z][0], T[z][1], T[z][2]);
delete[] T;
```

```
// giorni della c-esima meta' del mese (c=1,2)

MP_subset<1> PMM(Giorno);
for (int d = 0; d < (t-1)/2 + 1; d++)
 PMM.insert(d);

MP_subset<1> SMM(Giorno);
for (int d = (t-1)/2 + 1; d < t; d++)
 SMM.insert(d);

// giorni della b-esima settimana del mese (b=1,..,4)

MP_subset<1> PS(Giorno);
for (int d = 0; d < (t-1)/4 + 1; d++)
 PS.insert(d);

MP_subset<1> SS(Giorno);
for (int d = (t-1)/4 + 1; d < (t-1)/2 + 1; d++)
 SS.insert(d);

MP_subset<1> TS(Giorno);
for (int d = (t-1)/2 + 1; d < 3*(t-1)/4 + 1; d++)
 TS.insert(d);

MP_subset<1> QS(Giorno);
for (int d = 3*(t-1)/4 + 1; d < t; d++)
 QS.insert(d);
```

Inizializziamo quindi la variabile decisionale binaria `x` (indicizzata solo sulle triple dei turni ammissibili)

```
MP_binary_variable x(TurnoAmm);
```

e le variabili di scarto

```
MP_variable theta_p(Giorno, Orario),
            theta_m(Giorno, Orario),
            beta_p(Operatore),
            beta_m(Operatore),
            lambda_p(Operatore),
            lambda_m(Operatore),
            eps_p(Operatore, Settimana),
            eps_m(Operatore, Settimana),
            gamma(Operatore, Mese),
            delta(Operatore, Settimana);
```

Si procede dunque alla formulazione dei vincoli:

```
MP_constraint Assegn(Operatore, Giorno),
              Pre_Assegn(Tripla),
              Riposo_Mese1(Operatore),
              Riposo_Mese2(Operatore),
              Riposo_Sett1(Operatore),
              Riposo_Sett2(Operatore),
              Riposo_Sett3(Operatore),
              Riposo_Sett4(Operatore),
              Copertura(Giorno, Orario),
              Scost_Sett1(Operatore),
              Scost_Sett2(Operatore),
              Scost_Sett3(Operatore),
              Scost_Sett4(Operatore),
              Scost_Mese(Operatore),
              Scost_Tot(Operatore);

Assegn(i, k) =
  sum(TurnoAmm(i, w, k), x(TurnoAmm)) == 1;

Pre_Assegn(Tripla(i, w, k)) =
   x(TurnoAmm(i, w, k)) == 1;

Riposo_Mese1(i) =
  sum(PMM(k) * TurnoAmm(i, w, k).such_that(w == 0),
  x(TurnoAmm)) + gamma(i, 0) >= 2;

Riposo_Mese2(i) =
  sum(SMM(k) * TurnoAmm(i, w, k).such_that(w == 0),
  x(TurnoAmm)) + gamma(i, 1) >= 2;

Riposo_Sett1(i) =
  sum(PS(k) * TurnoAmm(i, w, k).such_that(w == 0),
  x(TurnoAmm)) + delta(i, 0) >= 1;

Riposo_Sett2(i) =
  sum(SS(k) * TurnoAmm(i, w, k).such_that(w == 0),
  x(TurnoAmm)) + delta(i, 1) >= 1;

Riposo_Sett3(i) =
  sum(TS(k) * TurnoAmm(i, w, k).such_that(w == 0),
  x(TurnoAmm)) + delta(i, 2) >= 1;

Riposo_Sett4(i) =
  sum(QS(k) * TurnoAmm(i, w, k).such_that(w == 0),
```

```
    x(TurnoAmm)) + delta(i, 3) >= 1;

  Copertura(k, j) =
    sum(TurnoAmm(i, w, k),
      Incidenza(j, w) * x(TurnoAmm))
    + theta_p(k, j) - theta_m(k, j) == Richiesta (j, k);

  Scost_Sett1(i) =
    sum(PS(k) * TurnoAmm(i, w, k),
      Ore(w) * x(TurnoAmm)) +
    eps_p(i, 0) - eps_m(i, 0) == 37;

  Scost_Sett2(i) =
    sum(SS(k) * TurnoAmm(i, w, k),
      Ore(w) * x(TurnoAmm)) +
    eps_p(i, 1) - eps_m(i, 1) == 37;

  Scost_Sett3(i) =
    sum(TS(k) * TurnoAmm(i, w, k),
      Ore(w) * x(TurnoAmm)) +
    eps_p(i, 2) - eps_m(i, 2) == 37;

  Scost_Sett4(i) =
    sum(QS(k) * TurnoAmm(i, w, k),
      Ore(w) * x(TurnoAmm)) +
    eps_p(i, 3) - eps_m(i, 3) == 37;

  Scost_Mese(i) =
    sum(TurnoAmm(i, w, k),
      Ore(w) * x(TurnoAmm)) +
    lambda_p(i) - lambda_m(i) == (37/6) * g;

  Scost_Tot(i) =
    beta_p(i) - beta_m(i) - lambda_p(i)
    + lambda_m(i) == Budget(i);
```

Infine, una volta aggiunti i vincoli al modello (tramite il metodo `.add()`), si procede alla formulazione della funzione obiettivo

```
  MP_expression OBJ;

  OBJ =
  Penalita(0) * sum(Giorno(k) * Orario(j),
                    theta_p(k, j)) +
  Penalita(1) * sum(Operatore(i), beta_p(i)) +
```

```
Penalita(2) * sum(Operatore(i), beta_m(i)) +
Penalita(3) * sum(Operatore(i), lambda_p(i)) +
Penalita(4) * sum(Operatore(i), lambda_m(i)) +
Penalita(5) * sum(Operatore(i) * Settimana(b),
                  eps_p(i, b)) +
Penalita(6) * sum(Operatore(i) * Settimana(b),
                  eps_m(i, b)) +
Penalita(7) * sum(Operatore(i) * Mese(c),
                  gamma(i, c)) +
Penalita(8) * sum(Operatore(i) * Settimana(b),
                  delta(i, b)) +
Penalita(9) * sum(TurnoAmm(i, w, k),
                  Costo(i, w) * x(TurnoAmm)) +
Penalita(10) * sum(Giorno(k) * Orario(j),
                   theta_m(k, j));
```

e alla chiamata del solutore per la minimizzazione

```
dvf.minimize(OBJ);
```

Sarà possibile ottenere i valori delle variabili utilizzate tramite il metodo

```
const double *sol = si->getColSolution();
```

Il vettore `sol` conterrà gli assegnamenti fatti per la variabile `x` e per le variabili di scarto.

Appendice B

Settaggio dei coefficienti di penalità da assegnare in funzione obiettivo

Per l'esecuzione dei test implementativi è necessario immettere in input i coefficienti di penalità da assegnare ai membri della funzione obiettivo. Ci siamo serviti di un file di testo in cui sono salvati i valori dei coefficienti.

Oltre a utilizzare un set di coefficienti di penalità ottenuti per via empirica, basandosi su osservazioni legate alle caratteristiche delle diverse funzioni che contribuiscono a definire la funzione obiettivo (Set 1), sono stati sperimentati tre ulteriori set di parametri (Set 2, 3 e 4), generati mediante procedure piu' rigorose.

É stato prima di tutto necessario suddividere i termini della funzione in classi gerarchiche secondo priorità, poiché generalmente da un gradino all'altro della scala gerarchica i pesi si distanziano di almeno un ordine di grandezza. Dopodiché è stato stimato il massimo miglioramento che ogni membro della funzione obiettivo può raggiungere, tramite la differenza tra il massimo e il minimo della funzione stessa. A questo punto i coefficienti sono stati calcolati applicando la proprietà secondo cui il miglioramento di una unità della funzione obiettivo di ordine superiore è sempre preferibile a un qualsiasi miglioramento della funzione gerarchicamente inferiore.

In questo modo abbiamo ottenuto coefficienti molto grandi, fino a un ordine di grandezza di 1e+13. Per i test non sono stati utilizzati tali coefficienti bensì settaggi in cui i pesi trovati sono stati diminuiti tramite fattori di scala in $(0, 1]$.

Riportiamo nella seguente tabella i valori assegnati ai coefficienti di penalità η_i, $i = 1, ..., 11$, descritti nel Capitolo 2.

Coefficiente	Set 1	Set 2	Set 3	Set 4
η_1	100 000	616 265 000 000	123 253 000 000	123 253 000
η_2	15 000	6 500 000 000	1 300 000 000	1 300 000
η_3	10 000	6 000 000 000	1 200 000 000	1 200 000
η_4	1 500	65 000 000	13 000 000	13 000
η_5	1 000	60 000 000	12 000 000	12 000
η_6	200	120 000	24 000	2 400
η_7	100	110 000	22 000	2 200
η_8	20	6 000	1 200	120
η_9	10	5 000	1 000	100
η_{10}	1	1	1	1
η_{11}	1	0.5	0.5	0.5

Bibliografia

[1] Abdennadher S., Schlenker H., *Nurse scheduling using constraint logic programming.* AAAI/IAAI, 838-843, 1999

[2] Blöchliger I., *Modeling staff scheduling problems. A tutorial.* European Journal of Operational Research, 158: 533-542, 2004

[3] Brucker P., *Scheduling Algorithms.* Springer , 2007

[4] Brucker P., Knust S., *Complex Scheduling.* Springer, 2012

[5] Brucker P., Qu R., Burke E., *Personnel Scheduling: Models and Complexity.* European Journal of Operational Research 210(3): 467-473, 2011

[6] Brucker P., Qu R., Burke E., Post G., *A decomposition, construction and post-processing approach for a specific nurse rostering problem.* Proceedings of the 2nd Multi-disciplinary Conference on Scheduling: Theory and Applications (MISTA 2005), 18-21 July:397–406, 2005

[7] Batun S., Begen M.A., *Optimization in Healthcare Delivery Modeling: Methods and Applications.* (Capitolo 4, [13])

[8] Burke E.K., De Causmaecker P., Vanden Berghe G., Van Landeghem H., *The state of the art of nurse rostering.* Journal of Scheduling, 7(6): 441–499, 2004

[9] Caprara A., Monaci M., Toth P., *Models and algorithms for a staff scheduling problem* (pp 445-476 *Mathematical Programming*, Springer, September 2003, Volume 98, Issue 1-3)

[10] Caseau Y., Guillo P., Levenez E., *A deductive and object-oriented approach to a complex scheduling problem.* JIIS, 4:149-166, 1995

[11] Davis L., *Handbook of genetic algorithms.* Van Nostrand Reinhold, New York, 1991

[12] De Causmaecker P., Demeester P., Vanden Berghe G., Verbeke B., *Analysis of real-world personnel scheduling problems.* Proceedings of the 5th International Conference on the Practice and Theory of Automated Timetabling, 18th August-20th August 2004, Pittsburgh, PA USA: 183–198, 2005

[13] Denton B.T., *Handbook of Healthcare Operations Management. Methods and Applications.* Springer, 2013

[14] Dowsland K.A., *Nurse scheduling with tabu search and strategic oscillation.* European Journal of Operational Research, 106:393-407, 1998

[15] Emmons H., Vairaktarakis G., *Flow Shop Scheduling. Theoretical Results, Algorithms, and Applications.* Springer, 2013

[16] Ernst A.T., Jiang H., Krishnamoorthy M., Owens B., Sier D., *An Annotated Bibliography of Personnel Scheduling and Rostering.* Annals of OR, 127: 21–144, 2004

[17] Ernst A.T., Jiang H., Krishnamoorthy M., Sier D., *Staff scheduling and rostering: A review of applications, methods and models.* European Journal of Operational Research, 153(1): 3-27, 2004

[18] Gawiejnowicz S., *Time-Dependent Scheduling.* Springer, 2008

[19] Glover F., *Tabu search - part i.* ORSA Journal of Computing, 1:190-205, 1989

[20] Gröbner M., Wilke P., *Optimizing employee schedules by a hybrid genetic algorithm.* Springer Lecture Notes in Computer Science, 2037:463-472, 2001

[21] Haspeslagh S., De Causmaecker P., Stolevik M., Schaerf A., *First International Nurse Rostering Competition 2010.* Proceedings of the 8th International Conference for the Practice and Theory of Automated Timetabling, Northern Ireland, 10th-13th August, 2010, 498–501, 2010

[22] Horio M., Suzuki A., *A Solution Method for Manpower Scheduling Problems by RCPSP/τ* The 9th International Symposium on Operations Research and Its Applications (ISORA'10), China, August 19–23, p.249-261,2010

[23] Jaumard B., Semet F., Vovor T., *A generalized linear programming model for nurse scheduling.* European Journal of Operational Research, 107:1-18, 1998

[24] Kirkpatrick S., Gelatt D.J., Vecchi M.P., *Optimization by simulated annealing.* Science, 220:671-680, 1983

[25] Lavieri M.S., Puterman M.L. *Optimizing nursing human resource planning in British Columbia.* Health Care Manag Sci 12(2):119–128, 2009

[26] Mönch L., Fowler J.W., Mason S.J., *Production Planning and Control for Semiconductor Wafer Fabrication Facilities. Modeling, Analysis, and Systems.* Springer, 2013

[27] Pochet Y., Wolsey L.A, *Production Planning by Mixed Integer Programming.* Springer, 2006

[28] Punnakitikashem P., Rosenberger J.M., Behan D.B., *Stochastic programming for nurse assignment.* Comput Optim Appl 40(3):321–349, 2008

[29] Purnomo H.W., Bard J.F., *Cyclic preference scheduling for nurses using branch and price.* Nav Res Logist 54(2):200–220, 2007

[30] Spini D., *Integrazione di un algoritmo per l'ottimizzazione dei turni del personale in un sistema informativo aziendale.* Tesi di Laurea in Informatica, Università degli Studi di Milano, A.A. 2004/2005

[31] Vanhoucke M., *Project Management with Dynamic Scheduling. Baseline Scheduling, Risk Analysis and Project Control.* Springer, 2012