UNIVERSITÀ DEGLI STUDI DI MILANO

Facoltà di Scienze Matematiche, Fisiche e Naturali

Dipartimento di Tecnologie dell'Informazione

(Corso di Laurea in Informatica)

Integrazione di un algoritmo per l'ottimizzazione dei turni del personale in un sistema informativo aziendale

Relatore :
Prof. Roberto Cordone

Correlatore :
Matteo Salani

Tesi di Laurea di :
Spini Daniel
Matr. 617806

Anno Accademico 2004/2005

Ai miei genitori.

Ringraziamenti

Un ringraziamento sentito al prof. Roberto Cordone e a Matteo, i quali mi hanno dato un enorme aiuto per la realizzazione di questa tesi, e a Michele per l'aiuto fornito sulla parte del database.

Voglio ringraziare per il sostegno che ho avuto in questi anni, la mia famiglia, tutti i miei amici che con me, dopo le scuole superiori, hanno potuto vivere questa esperienza.

Ringraziamenti vanno anche al "Presidentissimo" dott. Emanuele con il quale ho superato lo scoglio dell'ultimo esame, a Paolo per l'aiuto dato per la parte di programmazione e agli Andrea di Lodi.

Un pensiero sentito va a Serena con la quale ho condiviso momenti speciali durante questi ultimi anni e che mi auguro di condividerne altri.

Indice

Introduzione

L’organizzazione della turnazione del lavoro dipendente in ambito sanitario è un problema decisionale complesso. Il suo obiettivo primario è garantire l’erogazione del servizio ogni giorno della settimana, 24 ore su 24, assegnando agli operatori disponibili i turni richiesti dal servizio, nel rispetto dei vincoli sanciti dalle leggi nazionali e dai contratti collettivi nazionali di lavoro.

La presenza di numerosi altri vincoli di tipo sociale e professionale è spesso ignorata a causa della complessità del processo decisionale. Tale complessità, oltre a produrre un ricorso eccessivo agli straordinari, può anche far sì che il decisore non si avveda della violazione di leggi e contratti, esponendo l’ente sanitario a sanzioni civili o penali.

Lo scopo di queste tesi è la realizzazione di un sistema di supporto alle decisioni (DSS) che può essere utilizzato da un decisore esperto per costruire differenti soluzioni rispettose dei vincoli di legge, per valutarle in modo quantitativo e per scegliere la più soddisfacente.

Avendo il problema di assegnazione dei turni molti obiettivi, il DSS proposto associa a ciascun obiettivo un peso, basato sulle preferenze del decisore e da lui modificabile a piacere, in modo da orientare la generazione delle soluzioni verso il compromesso più desiderabile.

Il DSS comprende una rigorosa modellazione matematica del problema, un solutore di modelli di programmazione matematica intera, una base di dati in grado di gestire e visualizzare sia i dati della struttura sia le soluzioni alternative da proporre al decisore e un’interfaccia di gestione. Il sistema si integra inoltre con i normali strumenti di produttività d’ufficio per la presentazione delle turnazioni calcolate e per una efficace reportistica.

La tesi è organizzata nel modo seguente:

Il capitolo 1 presenta le normative del lavoro che coinvolgono l'assegnazione dei turni negli istituti sanitari e il modello matematico utilizzato per descrivere gli obiettivi e i vincoli del problema.

Il capitolo 2 descrive le componenti del DSS realizzato nel lavoro di tesi.

Il capitolo 3 fornisce una descrizione della base di dati, da quante e quali tabelle è composta, e come queste sono in relazione l'una con le altre. Nel capitolo è mostrato l'uso delle *stored procedure* per inserire, modificare e cancellare i dati nel database.

Il capitolo 4 descrive il solutore *open source glpsol*, utilizzato per l'ottimizzazione del modello matematico in base ai dati forniti dal sistema.

Il capitolo 5 descrive l'interfaccia che gestisce il DSS; vengono presentati i componenti utilizzati per la connessione alla base di dati, il componente per l'esecuzione delle *stored procedure*, quello per l'esportazione dei dati in fogli di lavoro Excel, e le tecniche di interrogazione delle tabelle del database attraverso opportune query SQL; inoltre si espone come è possibile eseguire il solutore dall'interfaccia creata, visto che GLPK non ne dispone di una sua.

Il capitolo 6 presenta una trattazione per la determinazione dei pesi dei criteri decisionali della funzione obiettivo trattati nel capitolo 1.

1. Descrizione del problema

Questa tesi affronta il problema di realizzare un sistema informativo di supporto alle decisioni per la turnazione degli operatori in strutture che operano in ambito sociale.

Questi operatori sono regolati da un contratto lavorativo con particolari clausole, le quali indicano il massimo delle ore lavorative settimanali, i turni di riposo dopo un certo numero di ore di servizio effettuate, l'assegnazione dei giorni di ferie e delle malattie, e altre condizioni riguardanti le retribuzioni attinenti a ore di straordinario, o quelle per il lavoro nei giorni festivi. Il software che si intende realizzare deve tenere conto di queste restrizioni, e quindi deve fornire una turnazione esaustiva applicabile, che non violi i vincoli di legge, per i quali sono previste sanzioni civili e penali.

In Italia il rapporto di lavoro è regolamentato da un ampio corpus legislativo, questo decreto, il Decreto Legislativo 66/2003 integrato successivamente dal Decreto Legislativo 213/2004 definisce i diritti dei lavoratori e i doveri dei datori di lavoro.

Alla legislazione si affiancano i Contratti Collettivi Nazionali del Lavoro o CCNL. Questi contratti, definiti di comune accordo dalle parti sociali per i lavoratori, dai rappresentati di categoria, per i datori di lavoro, e dal governo, hanno il compito di adattare le norme nazionali ai singoli ambiti produttivi.

Rispetto alle norme di legge ai contratti collettivi è consentito, con alcune restrizioni, di modificare alcuni parametri, individuare delle eccezioni o aggiungere delle limitazioni ancora più stringenti di quelle definite dal legislatore. Spetta ai contratti collettivi specificare i parametri di remunerazione dei lavoratori. Alcuni aspetti particolari dei contratti collettivi, quali ad esempio la retribuzione, possono essere ridefiniti in sede locale, sia essa regionale o provinciale. Ogni Contratto Collettivo deve poi essere

concordato, a livello individuale, da ogni singola azienda rispetto ai propri dipendenti. Questa ultima trattativa, detta anche contratto individuale, tende a specificare i punti lasciati aperti dal CCNL e ad introdurre altri benefici.
Il Contratto Collettivo Nazionale di riferimento per questo lavoro è il contratto UNEBA, il quale regola il rapporto di lavoro del personale dipendente per gli istituti operanti nel campo del sociale, per attività educative, di assistenza e di beneficenza, nonché di culto o religione dipendenti dall'autorità ecclesiastica. Il contratto può essere esteso anche al personale dipendente di altre istituzioni che dichiarino di accettarne completamente la disciplina nel contratto individuale di lavoro.

1.1 Norme che influenzano l'organizzazione del lavoro

1.1.1 Orario di lavoro

L'orario lavorativo determina i limiti quantitativi delle prestazioni richieste al lavoratore, la distribuzione dei giorni lavorativi nella settimana e l'inizio e la fine di ogni giorno lavorativo. Per legge l'orario lavorativo è fissato a 40 ore settimanali. I singoli contratti collettivi di categoria possono diminuire questo limite. Il lavoratore ha diritto ad un riposo completo di 24 ore nell'arco della settimana.
L'orario settimanale può essere distribuito su cinque o sei giorni. In una settimana da cinque giorni il sabato è il giorno di riposo mentre la domenica viene considerata una giornata festiva. Per le settimane da sei giorni il riposo coincide con la domenica. L'orario lavorativo ed il numero di giornate lavorative in una settimana determinano la durata di una giornata lavorativa. La definizione precisa della distribuzione dell'orario, unitamente agli orari di

inizio e fine lavoro, spetta ai singoli datori di lavoro. In ogni caso al lavoratore deve essere garantito un riposo di almeno 11 ore consecutive tra una giornata lavorativa e la successiva.

Straordinari e lavoro supplementare

L'orario lavorativo determina un limite che, in caso di necessità, può essere superato.

Si indica come lavoro supplementare il numero di ore che eccedono l'orario base, ma non quello di legge. Tutte le ore che superano il limite legale sono definite ore straordinarie. Generalmente le ore straordinarie sono pagate con una maggiorazione. Il numero massimo di ore straordinarie che un lavoratore può effettuare è pari a 250 ore all'anno. In ogni caso, di norma, non è possibile effettuare più di 48 ore in una settimana. In casi eccezionali è possibile superare questo limite, previa comunicazione alla Direzione Provinciale del Lavoro.

Lavoratori a tempo parziale

Nel caso dei lavoratori a tempo parziale, o part time, l'orario lavorativo viene ridotto secondo un accordo tra il lavoratore ed il datore di lavoro. La distribuzione delle ore e dei giorni lavorativi può essere differente rispetto a quella relativa all'orario lavorativo standard. Gli straordinari sono pagati con una maggiorazione ulteriore fino al raggiungimento dell'orario lavorativo standard, oltre questo valore sono pagati come per un lavoratore a tempo completo. Nel caso che il CCNL introduca il lavoro supplementare, è compito del Contratto Collettivo definire quanta parte del lavoro eccedente l'orario base debba essere considerato come lavoro supplementare. Per tutti gli altri aspetti un lavoratore part time è equiparato ad uno a tempo completo.

Lavoro notturno

Generalmente viene definito turno notturno qualsiasi turno che abbia almeno un'ora nel periodo che va dalle ore 24 alle ore 6. Ogni contratto collettivo può allargare questo periodo. Non possono essere adibiti al lavoro notturno i minorenni, le donne in stato interessante e le madri con figli di età inferiore ai 2 anni. Nessun lavoratore può effettuare più di 8 ore di lavoro notturno nell'arco di 24 ore.

1.1.2 Ferie, permessi e malattie

Ferie

Ogni lavoratore ha diritto ad un periodo di ferie di almeno 4 settimane. Di queste, almeno due devono essere consumate durante l'anno, mentre le restanti devono essere consumate nei 18 mesi successivi. Le ferie sono concordate tra il lavoratore ed il datore di lavoro. Può quindi esistere un limite alle ferie che il dipendente può godere in modo continuativo, e le ferie possono essere richieste dal dipendente o determinate, a seconda delle necessità, dal datore di lavoro. Non è possibile ottenere un'indennità sostitutiva per le ferie non godute, a meno di una risoluzione del rapporto.

Riduzione d'orario

Al dipendente viene garantito un monte ore permessi pari a 24 ore. Questo monte ore matura nella misura di 2 ore ogni mese lavorato.

Permessi retribuiti

I contratti collettivi possono introdurre una limitata riduzione dell'orario su base annua. Queste ore di riduzione vanno a formare un monte permessi

retribuiti, dal quale il lavoratore può attingere sia in misura oraria che giornaliera.

Il monte ore matura col progredire dell'anno lavorativo. Ad esempio, se il CCNL prevede 24 ore di riduzione, ogni mese lavorato il monte permessi viene incrementato di 2 ore (24 ore / 12 mesi = 2 ore / mese). Alla fine dell'anno i permessi non goduti danno luogo ad un'indennità sostitutiva.

Malattie

In caso di necessità il servizio sanitario nazionale può assegnare al dipendente alcune giornate di riposo. In questo caso il dipendente è giustificato nel non presentarsi al lavoro, ma non può prolungare l'assenza oltre il periodo certificato da un medico. Se il dipendente è in ferie, le giornate di malattia non sono conteggiate nel numero delle ferie consumate. Per periodi inferiori a tre giorni non è richiesto il certificato medico.

1.2 Contratto Collettivo UNEBA

Orario di lavoro

Il CCNL UNEBA stabilisce l'orario di lavoro in 38 ore settimanali. La pianificazione degli orari viene definita tenendo conto delle necessità organizzative dell'istituto. L'orario di lavoro è distribuito, nell'arco della settimana, in modo da garantire un giorno di riposo, normalmente coincidente con la domenica. Al lavoratore, al quale è richiesto di effettuare una prestazione lavorativa durante il riposo settimanale, spetta un riposo compensativo.

Straordinari

Le ore straordinarie sono misurate, ogni mese, rispetto al totale delle ore lavorate nella media dei sei mesi precedenti, scontando le ore già pagate come straordinarie, durante quei mesi. Questo comporta la possibilità di gestione di una banca ore nella quale far confluire gli straordinari. Le ore presenti in banca possono essere utilizzate per compensare ore eventualmente non lavorate. Questa banca ore può essere gestita anche in modo opposto, facendovi confluire le ore non lavorate, le quali copriranno successive ore straordinarie. Le ore straordinarie possono quindi essere compensate da periodi d'astensione dal lavoro. Nel caso le ore non siano compensate, queste devono essere pagate con una maggiorazione. Un lavoratore può effettuare al massimo 120 ore di lavoro straordinario nell'arco di un anno.

Lavoratore part time

Un lavoratore part time può effettuare un numero di ore di lavoro supplementare pari al 15% del proprio orario di lavoro. Ad esempio un lavoratore part time assunto a 19 ore può effettuare 2 ore e 51 minuti di lavoro supplementare alla settimana. Queste ore sono pagate normalmente. Le ore che eccedono questo valore sono considerate come lavoro straordinario.

Lavoro notturno e festivo

Viene considerato lavoro notturno tutto il lavoro svolto dalle ore 22 alle ore 6. A tutti i lavoratori che effettuano lavoro notturno, o festivo, spetta una maggiorazione del compenso.

Festività

Sono considerate festività le domeniche e tutte quelle date definite come festa nazionale, oppure festa religiosa, quali il 2 giugno (festa della repubblica) e il 25 dicembre (Santo Natale). Nel caso che la festività cada nel giorno di riposo, al dipendente spetta un compenso pari al normale importo di una giornata lavorativa, normalmente un ventiseiesimo della paga mensile.

Ferie

Al lavoratore spettano 33 giorni di ferie calcolati in un settimana di sei giorni, per un totale di cinque settimane e mezzo effettive. Il dipendente non può richiedere più di tre settimane di ferie consecutive. La Direzione dell'Istituto deve determinare, possibilmente concordandole con il dipendente, due settimane di ferie nel periodo che va dal 1 giugno al 30 settembre. Le restanti ferie sono a discrezione del dipendente, previo accordo con la Direzione dell'Istituto.

1.3 Vincoli organizzativi

Spesso ogni istituto è una realtà a sé, caratterizzata da particolari consuetudini e necessità. A seconda di queste, i singoli istituti si possono dotare di una propria organizzazione del lavoro, la quale non può però contrastare con le normative vigenti. In particolare, il DSS proposto in questa tesi considera i vincoli organizzativi della struttura "casa di riposo di Ghedi" considerata nel precedente lavoro di tesi [MM04]. Questa struttura è organizzata in nuclei, organizza il proprio lavoro tramite sequenze di tre turni lavorativi ed un

riposo (denominate 3+1) e utilizza un pattern ciclico per assegnare i turni alla maggior parte degli operatori.
Nel seguito descriviamo in dettaglio questi aspetti.

1.3.1 I Nuclei

La casa di riposo è strutturata in reparti o nuclei. In particolare sono presenti quattro reparti, identificati da un colore: nucleo Verde, nucleo Giallo, nucleo Blu e nucleo Rosa. Alcuni operatori operano principalmente in un nucleo solo. Questo permette la conoscenza sia tra gli operatori che tra gli operatori e gli ospiti di un nucleo.

I Turni

La tabella 1.1 riporta i vari turni, con le loro caratteristiche. Generalmente per ogni nucleo sono definiti i seguenti turni: Notte, Mattina, Pomeriggio, Pomeriggio2 e Sera. Per il nucleo verde è definito un ulteriore turno (3,5V). A questi turni se ne affiancano altri che non sono associati ad un nucleo. Generalmente ogni turno deve essere effettuato da un solo operatore per nucleo, ma per alcuni turni la richiesta può essere maggiore (ad esempio, il turno della Mattina prevede 4 operatori per nucleo).
I turni 3,5b e 3,5m possono essere raggruppati e sostituiti da un terzo turno b/m nelle giornate, note a priori, in cui mancano entrambi gli operatori dedicati a questi turni.

Turno	Orario	Verde	Giallo	Blu	Rosa	Richiesta
Notte	0:00 – 7:00	1	1	1	1	4
Mattina	6:00 – 13:00	4	4	4	4	16
Pomeriggio	13:00 – 20:00	1	1	1	1	4
Pomeriggio2	14:00 – 21:00	1	1	1	1	4
Sera	17:00 – 24:00	1	1	1	1	4
3,5V	7:00 – 10:30	1				1
3,5b	6:00 – 9:30					1
3,5m	9:00 – 12:30					1
mb	7:00 – 14:00					1
b/m	6:00 – 13:00			Saltuario		

Tab.1.1: esempio di turni.

1.3.2 Turnazione 3+1

L'istituto, in particolar modo per gli operatori a tempo pieno, definisce una turnazione che prevede una successione di tre giornate consecutive, con il medesimo turno, ed una giornata di riposo. Questa turnazione viene chiamata del 3+1.

Tramite il 3+1 è possibile suddividere gli operatori in due gruppi. Il primo, al quale appartiene chi aderisce al 3+1, ha il compito di garantire la copertura dei turni dei vari reparti. I lavoratori del secondo gruppo, spesso a tempo parziale, tappano i buchi alla copertura generati dalle assenze per ferie e malattie degli operatori sia del primo gruppo, sia degli altri nuclei.

La turnazione 3+1 ha due vantaggi: permette di valutare facilmente il numero di operatori necessari e di forzare gli operatori a consumare, durante l'anno, parte delle ferie e dei permessi. In questo modo è possibile evitare di pagare a fine anno, i permessi non goduti, e di rispettare le norme che prevedono che un certo numero di giorni di ferie siano effettuate entro la fine dell'anno.

Dimensionamento del personale di un nucleo

Ogni nucleo richiede la copertura di 8 turni ogni giorno (4 Mattine, 1 Pomeriggio, 1 Pomeriggio2, 1 Sera e 1 Notte). In una turnazione del 3+1 è disponibile ogni giorno, il 75% degli operatori. Sono quindi necessari 11 operatori a tempo pieno: 8 infatti, è il 75% di 10,67 che, arrotondato all'intero superiore, dà 11. Ferie e permessi coprono in media il 9% dell'anno, per cui serve un altro operatore; 10,67 è il 91% di 11,72 che arrotondato da 12. Con 12 operatori è possibile coprire tutti i turni del nucleo e se il tasso di malattia si mantiene sotto il 2,8%, non è necessario nessun altro operatore, infatti la differenza tra i 12 operatori effettivi e gli 11,72 necessari è di 0,33 operatori che è circa il 2,8% del totale.

Recupero dei riposi come ferie e permessi

In media un mese è costituito da 26 giornate lavorative. Un anno composto da 365 giorni è suddivisibile mediamente in 12 mesi di 30,42 giorni ciascuno. Ricondotto alla settimana da 6 giorni, genera un mese composto da 26,07 giornate di lavoro effettivo. Lavorando in media, secondo l'orario di lavoro da 38 ore, 6 ore e 33 minuti al giorno si ottiene che il carico medio delle ore lavorate, nel mese, deve essere pari a 165 ore, per essere precisi 165,13 ore. La turnazione del 3+1 garantisce 22,82 giornate lavorative per un totale, essendo il turno medio di 7 ore, di 161 ore lavorative. Le 4 ore mancanti possono essere recuperate come ferie e/o permessi. In questo modo si consumano i permessi, i quali andrebbero retribuiti al termine dell'anno, e le ferie che devono essere completamente godute nell'arco di 30 mesi dall'inizio dell'anno.

1.3.3 Pattern ciclici di turni

La casa di riposo di Ghedi, come molte strutture ospedaliere, utilizza una sequenza ciclica di turni per assegnare i compiti di alcuni operatori. Questa sequenza, se rispettata, presenta due vantaggi:

1. **Conoscenza del turno:** l'operatore memorizza con estrema facilità il suo orario e può, quindi, regolare i suoi impegni esterni più semplicemente. Questo permette all'operatore, e alla sua famiglia, di godere maggiormente del tempo libero a disposizione.

2. **Conoscenza tra gli operatori:** se il ciclo viene sempre rispettato, alla lunga, gli operatori che effettueranno i medesimi turni, anche tra nuclei diversi, tendono ad essere sempre gli stessi. In questo modo è possibile creare dei gruppi di lavoro affiatati tra di loro, i quali possono sostenersi a vicenda in caso di necessità.

La sequenza impiegata a Ghedi è caratterizzata da un periodo di 24 giorni, diviso in sei gruppi, composti da tre turni identici ed un riposo. Il primo operatore effettua la sequenza partendo dal primo giorno. Il secondo operatore inizia la sequenza tre giorni più tardi mentre il terzo inizia sei giorni più tardi, e così via. Il fatto che la sequenza abbia un periodo di 24 giorni e che ogni operatore sia sfasato di 3 giorni rispetto al precedente, fa sì che 8 operatori coprono in modo uniforme la sequenza completa, eseguendo tutte le tipologie di turno.
Ogni giorno sei operatori coprono Notte, Sera, Pomeriggio, Pomeriggio2 e due Mattine e gli altri due operatori sono in riposo. Quindi la sequenza è sottodimensionata: sono necessari altri operatori per coprire i due turni mattutini non assegnati.

A questo pattern se ne può quindi affiancare un altro che permette di gestire i rimanenti 4 operatori, in modo da ricondurci ai 12 operatori minimi necessari come descritto in sezione 1.3.2. Il secondo pattern definisce i turni lavorativi di quei operatori appartenenti alla categoria dei part time, i quali serviranno ad occupare i turni lasciati liberi dagli altri lavoratori aderenti alla turnazione del 3+1 che sono in ferie, malattia o in riposo, e c'è bisogno di sostituirli. La tabella 1.2 definisce la turnazione del 3+1 degli otto operatori assunti con contratto a tempo pieno sui 24 giorni lavorativi previsti, inoltre sono presenti gli altri quattro operatori part time che lavorano nei giorni in cui i lavoratori titolari sono in riposo.
Va sottolineato il fatto che l'introduzione di due pattern disomogenei, presenta il problema di creare una distribuzione non equa dei turni e il sotto utilizzo di alcuni operatori.

1.3.4 Numero minimo di riposi

L'organizzazione della casa di riposo prevede che ogni mese, siano garantiti almeno 5 riposi ad ogni operatore. Questo vincolo è una clausola del contratto singolare, la quale migliora una condizione definita da quello collettivo. Questo infatti prevede un riposo ogni sei giorni lavorati.

Op.	Giorni											
	1	2	3	4	5	6	7	8	9	10	11	12
1	N	N	N		S	S	S		P2	P2	P2	
2	M	M		N	N	N		S	S	S		P2
3	P		M	M	M		N	N	N		S	S
4		P	P	P		M	M	M		N	N	N
5	M	M	M		P	P	P		M	M	M	
6	P2	P2		M	M	M		P	P	P		M
7	S		P2	P2	P2		M	M	M		P	P
8		S	S	S		P2	P2	P2		M	M	M
9	M	M			M	M			M	M		
10		M	M			M	M			M	M	
11			M	M			M	M			M	M
12	M			M	M			M	M			M

Op.	Giorni											
	13	14	15	16	17	18	19	20	21	22	23	24
1	M	M	M		P	P	P		M	M	M	
2	P2	P2		M	M	M		P	P	P		M
3	S		P2	P2	P2		M	M	M		P	P
4		S	S	S		P2	P2	P2		M	M	M
5	N	N	N		S	S	S		P2	P2	P2	
6	M	M		N	N	N		S	S	S		P2
7	P		M	M	M		N	N	N		S	S
8		P	P	P		M	M	M		N	N	N
9	M	M			M	M			M	M		
10		M	M			M	M			M	M	
11			M	M			M	M			M	M
12	M			M	M			M	M			M

Tab 1.2: sequenza dei turni applicata a tutti gli operatori.
Lo spazio bianco implica un riposo

1.4 Nurse Rostering Problem

In letteratura il problema di definire i turni di servizio per gli operatori di una casa di riposo, e più in generale di una struttura ospedaliera, è identificato come *Nurse Rostering Problem (NRP)* o *Nurse Scheduling Problem (NSP)*. Il NRP è un argomento molto interessante nel campo, più generale, dello staff scheduling and rostering. Ne è un esempio l'elevato numero di lavori presentati alle varie edizioni della conferenza internazionale *Practice And Theory of Automated Timetabling (PATAT)*. L'interesse per questo problema consiste nel fatto che, sebbene sia regolamentato in modo differente a seconda della nazione, è alla fine riconducibile a poche famiglie di vincoli che possono presentarsi o meno nei singoli casi.

Ad esempio, [CLLR03] individua sedici famiglie di vincoli che caratterizzano il problema. Il caso di Ghedi ne presenta ben quattordici, dimostrandosi quindi piuttosto complesso nella struttura, se non nelle dimensioni, che sono abbastanza ridotte. Un'altra particolarità del NRP consiste nell'omogeneità di molti approcci tradizionali, con carta e penna: essi sono pressoché identici, sebbene concepiti per strutture e nazioni differenti.

Ad esempio la casa di riposo di Ghedi fa ricorso a sequenze cicliche di turni, tecnica risolutiva presente in molti casi citati in letteratura (si veda ad esempio [BCB01] e [Pla01]). Queste sequenze cicliche sono progettate per coprire tutti i turni richiesti utilizzando al meglio gli operatori disponibili. In caso di carico fisso ci si limita ad assegnare ad ogni operatore un preciso pattern. Nel caso si debbano coprire delle assenze, per ferie o malattia, si possono utilizzare gli operatori ad hoc.

Un altro motivo che rende il NRP interessante consiste nel fatto d'essere di difficile soluzione. Tipicamente, infatti, è un problema con vincoli molto stretti, tanto che spesso non può essere soddisfatto, quando d'altra parte è necessario determinare una soluzione ad ogni costo. Spesso è necessario

definire i carichi di lavoro per un alto numero di operatori in un periodo che va da un paio di giorni ad uno o più mesi, con un notevole impatto sociale. Il tutto deve avvenire nel minor tempo possibile, visto che bisogna anche poter gestire le emergenze. Ad esempio quando, a causa di un imprevisto, una soluzione prevista diventa irrealizzabile, è necessario determinarne una nuova nel giro di pochi minuti per tamponare la situazione.
In ogni caso, come evidenziato in letteratura, la soluzione automatica si è sempre dimostrata migliore rispetto a quella manuale sia in termini di tempo che di correttezza.

1.4.1 Rassegna bibliografica

L' NRP è stato affrontato facendo uso di diverse tecniche:

- approcci di tipo modellistico [JSV98] alcuni dei quali sfruttano solutori di programmazione lineare [MK92];

- metodi di programmazione matematica [MWG76];

- algoritmi euristici, che ricalcano in parte le procedure manuali ([Hun95] e [JK92]), tra i quali:
  - metodi greedy [SW77];
  - tabù search [GL97];
  - algoritmi genetici [AD01];

- la *Constraint Programming (CP)*, cioè la modellazione del problema attraverso variabili con un insieme di definizione finito e di vincoli (*constraint*) che vietano alcune combinazioni di valori.

Alcuni approcci sono frutto di una combinazione di tecniche [AW04].
In molti lavori si evidenzia come, per riuscire ad ottenere una soluzione, sia necessario rilassare uno o più vincoli [CB02]; in questi casi riveste un ruolo importante la scelta del vincolo da rilassare e, nelle applicazioni pratiche, spesso si lascia che sia il decisore, o schedulatore, a decidere a quali condizioni rinunciare.
È normale definire NRP come un problema di ottimizzazione [JSV98] (spesso multi obiettivo) oppure come un problema di soddisfazione dei vincoli [AH01]. Questi due approcci non sono tra loro esclusivi, e possono essere combinati per cercare una soluzione del problema.
Per una bibliografia più dettagliata si può far riferimento a [EJK+04, §3.7] ed a [CLLR03].

1.4.2 Il NRP come problema di ottimizzazione

Definendo il NRP come un problema di ottimizzazione si cerca di minimizzare, o massimizzare, una funzione obiettivo, rispettando tutti i vincoli della formulazione. Si tratta, storicamente, del primo approccio al problema, nel quale si cerca di risolvere il modello tramite le tecniche proprie della programmazione matematica, come la programmazione lineare intera, eventualmente multi obiettivo (Goal Programming), appoggiandosi a strutture quali grafi o reti.
Sebbene l'approccio matematico porti, in teoria, alla migliore soluzione possibile, in molti casi non risulta applicabile in pratica. Infatti l'alto numero di vincoli, spesso difficilmente descrivibili, la presenza di più funzioni obiettivo, la necessità di ottenere una soluzione in tempi brevi e la possibilità di avere istanze che non ammettono soluzione riducono il campo d'applicazione della programmazione matematica. Risulta invece più produttivo utilizzare metodi euristici o meta-euristici.

In questo caso è più semplice, e sufficiente per scopi pratici, gestire il NRP come un problema di soddisfazione dei vincoli o *Constraint Satisfaction Problem (CSP)*. Ultimamente, questo approccio ha avuto una larga diffusione, grazie anche alla recente introduzione della programmazione per vincoli o *Constraint Programming (CP)*, seguita da una varietà di linguaggi dedicati, i quali rendono molto semplice descrivere anche un'insieme complesso di vincoli.
Alla CP sono state affiancate, in ogni caso, anche altre tecniche euristiche o meta-euristiche quali tabu search, algoritmi genetici e simulated annealing.

1.5 Un modello matematico

In questa sezione definiamo un modello matematico che caratterizza le soluzioni ammissibili del NRP. per la specifica applicazione alla casa di riposo di Ghedi, e definisce le funzioni obiettivo che ne valutano la qualità.

1.5.1 Rappresentazione della soluzione

Le soluzioni del NRP vengono spesso rappresentate da una tabella che riporta, per ogni coppia operatore-giorno, il compito assegnato (vedi tabella 1.3). Questa descrizione viene chiamata *operatore-giorno* o *nurse-day*.

Op.	**G1**	**G2**	**G3**	**G4**	**G5**	**G6**	**G7**
1	T1	T1	T1	R	T2	T2	T2
2	R	T2	T2	T2	R	T1	T1
3	T2	R	R	T1	T1	R	R

Tab. 1.3: Schema operatore-giorno per un esempio di assegnazione dei turni

Un'altra descrizione comune consiste in un insieme di griglie che riportano, giorno per giorno, quale operatore effettua un determinato turno. Questa visione viene chiamata *operatore-compito* o *nurse-task*.
Esiste un terzo modo per descrivere le soluzioni, che consiste nel rappresentare gli assegnamenti in una tabella turno-giorno. Sebbene risulti essere pratico per il responsabile del servizio, il quale può sapere direttamente che operatore sta eseguendo un determinato servizio, risulta essere poco utilizzato poiché:

1. è di difficile consultazione per gli operatori;

2. non è adatto per i turni che non hanno un numero fisso di richieste.

Il modello descritto nel seguito come pure la base dati descritta nel cap. 3 consente tutte e tre le descrizioni. La descrizione che adottiamo nell'interfaccia grafica (vedi cap. 5) è quella operatore-giorno.

1.5.2 I dati del problema

Orizzonte temporale

È dato un insieme L di giorni (*orizzonte temporale*) che corrisponde a m mesi consecutivi. Si definisce il sottoinsieme $M_k \subseteq L$ come l'insieme dei giorni che costituiscono il mese $k \in \{1..m\}$. Inoltre l'orizzonte temporale è anche suddiviso in s settimane consecutive: il sottoinsieme $S_t \subseteq L$ è l'insieme dei giorni che appartengono alla settimana $t \in \{1..s\}$. Otteniamo quindi che:

$$L = \bigcup_{k=1}^{m} M_k = \bigcup_{t=1}^{s} S_t$$

Orario contrattuale, ferie e malattie

È dato un insieme *N* di operatori. Per ogni operatore $i \in N$ si definisce $h_i \geq 0$ come il numero medio di ore che egli deve effettuare in una settimana in base al proprio contratto di lavoro e $H_{ki} \geq 0$ come il numero di ore che deve effettuare durante il mese *k*.

Le ore in eccesso rispetto al valore H_{ki} sono accumulate, mese per mese, come straordinari. Per ogni operatore è noto il numero di ore di straordinario f_i che ha accumulato dall'inizio dell'anno fino al giorno che precede l'inizio dell'orizzonte temporale. L'operatore può anche richiedere una o più giornate di ferie che formano l'insieme $L_i^F \subseteq L$. Similmente può ottenere, dal servizio sanitario, una o più giornate di malattia che definiscono l'insieme $L_i^I \subseteq L$. Per taluni operatori il contratto prevede, nel limite del possibile, un riposo ogni tre giorni lavorativi consecutivi. Questo tipo di turnazione ciclica, detta del 3+1, definisce l'insieme degli operatori $N_3 \subseteq N$.

Turni e carico di lavoro

È dato l'insieme *T* di turni. I turni possono essere classificati come lavorativi $(W \subseteq T)$, di riposo $(R \subseteq T)$, di ferie $(F \subseteq T)$ o di malattia $(I \subseteq T)$. I turni lavorativi sono quelli nei quali l'operatore si presenta alla casa di riposo per svolgere un compito che gli è stato assegnato. In questo insieme di turni si identificano quelli che vengono definiti come turni notturni $(Wo \subseteq W)$, cioè quei turni che hanno almeno un'ora di lavoro compresa nel periodo tra le ore 24 e le 6.

I turni di riposo, malattia e ferie sono turni nei quali l'operatore non si presenta alla casa di riposo e quindi non può svolgere nessun compito. Le quattro categorie dei turni sono tra di loro disgiunte e sono una partizione dell'insieme *T* dei turni. Si ha quindi che:

$$T \equiv W \cup F \cup I \cup R$$

$$W \cap F \equiv W \cap I \equiv W \cap R \equiv F \cap I \equiv F \cap R \equiv I \cap R \equiv \phi$$

Tutti i turni, a parte quelli di riposo, concorrono al calcolo del monte ore lavorate dall'operatore. In questo caso si parla anche di giorni lavorativi. I turni lavorativi hanno una durata fissa d_j, espressa in ore, mentre per i turni di malattia e riposo la durata v_i dipende dal tipo di contratto d'assunzione dell'operatore.

Ad esempio, per un operatore inquadrato con un contratto da 38 ore per sei giorni alla settimana, si ha che v_i = 6h 20'. Tutti i turni lavorativi hanno un orario d'inizio b_j e di fine e_j per i quali valgono le relazioni $0 \leq b_j < e_j \leq 24$ e $e_j - b_j \geq d_j$. La durata può risultare inferiore al periodo che intercorre tra l'inizio e la fine del turno nel caso, più generale, che sia possibile effettuare una pausa.

Va sottolineato che non tutti i turni possono essere svolti da un operatore. Infatti, può accadere che un operatore non sia abilitato o si trovi in condizioni tali da non poter eseguire un certo turno, oppure che il turno non sia di sua competenza. In questo caso si definisce l'insieme $W_{il} \subseteq T$ come l'insieme dei turni che l'operatore i può svolgere il giorno l. Infine, si definisce come carico di lavoro $c_{jl} \geq 0$ il numero di operatori che devono svolgere il turno $j \in W$ nel giorno $l \in L$.

Reparti

La casa di riposo è suddivisa in un insieme D di reparti. È possibile associare ai singoli reparti i vari turni lavorativi, indicando con $T_r \subseteq W$ l'insieme dei turni che fanno riferimento ad un particolare reparto $r \in D$. Alcuni operatori sono associati ad un reparto: $N_r \subseteq N$ è l'insieme degli operatori che afferiscono al reparto r.

Operatore con turno preferenziale

In alcuni casi è possibile assegnare ad un operatore un turno lavorativo preferenziale. Questa scelta offre la possibilità di non sovrapporre impegni personali di un operatore con gli orari di lavoro che deve svolgere, oppure solo la preferenza di lavorare in un particolare turno piuttosto che in un altro per un determinato giorno. La gestione di questo meccanismo è descritta nella sezione 1.6.5.

Operatori Jolly

In casi eccezionali è possibile utilizzare degli operatori di riserva, o jolly, al fine di riuscire a completare il carico di lavoro. Si definisce quindi l'insieme $N^J \subseteq N$ degli operatori jolly.

Operatori Notturni

Si definisce *No* come insieme degli operatori che possono svolgere un turno di lavoro notturno $(No \subseteq N)$.

Frontiera

Le decisioni che devono essere prese per l'orizzonte temporale L sono influenzate dalle assegnazioni dei turni nel periodo immediatamente precedente. In particolare, le decisioni precedenti influiscono solo sui primi giorni del orizzonte temporale. Definiremo come *frontiera* questo periodo iniziale che subisce l'influenza delle decisioni prese nel passato. In particolare è necessario conoscere il numero $a_i \in \{0..6\}$ di giorni consecutivi lavorati prima del primo giorno del periodo L, l'ultimo turno $y_i \in T$ effettuato

dall'operatore i e, il numero di ore g_i già lavorate nella prima settimana, se questa non inizia di lunedì.

1.5.3 Variabili decisionali

Il nostro scopo consiste nel definire per ogni operatore $i \in N$ che turno $j \in T$ svolgerà il giorno $l \in L$. La soluzione si presta ad essere modellata tramite le variabili decisionali x_{ijl} binarie. Con $x_{ijl} = 1$ indichiamo che l'operatore *i* deve effettuare il turno *j* nel giorno *l*, in caso contrario è $x_{ijl} = 0$. Altre variabili ausiliarie verranno introdotte nel seguito per descrivere specifici elementi del problema.

1.6 Funzioni obiettivo

Le soluzioni ammissibili e non ammissibili si distinguono definendo un insieme di vincoli. I vincoli possono essere suddivisi in due classi: vincoli rigidi o *hard* e vincoli flessibili o *soft* [CLLR03]. I primi sono vincoli che non possono essere mai violati. I secondi sono invece vincoli che possono essere violati a patto di pagare una penalità.

Se imponessimo il rispetto rigoroso di tutti i vincoli, molte istanze non avrebbero una soluzione ammissibile. Poiché stiamo modellando un caso reale la risposta "non esistono soluzioni" non è accettabile. Si rende quindi necessario rilassare alcuni vincoli, in particolar modo quelli flessibili, ed introdurre opportune funzioni obiettivo per scegliere la soluzione che viola il meno possibile l'insieme dei vincoli rilassati.

Le funzioni obiettivo, prese in considerazione nel seguito, sono:

1. minimizzare il costo della soluzione;
2. minimizzare la violazione di un vincolo flessibile;

Ci troviamo quindi, di fronte ad un modello di programmazione a molti obiettivi.

1.6.1 Utilizzo degli operatori jolly

L'utilizzo degli operatori jolly dovrebbe essere un evento raro, tipico di condizioni critiche. Se divenisse la norma, significherebbe che la struttura è sotto organico. L'obiettivo consiste nel minimizzare l'utilizzo degli operatori jolly, o più precisamente il numero di ore da loro effettuato.

$$z_1 = \sum_{i \in NJ} \sum_{l \in L} \sum_{j \in W} d_j x_{ijl} \qquad \text{(O.1)}$$

1.6.2 Scostamento orario contrattuale

Uno dei principali motivi per cui un'istanza non ammette una soluzione ammissibile, consiste nell'impossibilità di coprire tutti i turni richiesti, facendo effettuare all'operatore solamente le ore definite contrattualmente. L'introduzione di ore straordinarie permette, al contrario, di gestire anche gli eventuali carichi aggiuntivi temporanei senza richiedere l'introduzione di altri operatori.

Definiamo la quantità $o_{ki} \geq 0$ come il numero di ore straordinarie lavorate dall'operatore *i* il mese *k*.

Lo straordinario è regolamentato e penalizzato, principalmente per tutelare il dipendente dallo sfruttamento. A questo si aggiunge il fatto che lo

straordinario deve essere corrisposto con un compenso maggiorato. Questo dà luogo a una seconda funzione obiettivo:

$$z_2 = \sum_{i \in N} \sum_{k=1}^{m} o_{ki} \qquad \text{(O.2)}$$

1.6.3 Ore non lavorate

Ogni operatore deve effettuare un numero minimo di ore lavorative a settimana. Il numero effettivo di ore lavorate di ogni operatore non dovrebbe essere inferiore al valore definito nel contratto, quindi definiamo la variabile u_{ti} e la funzione obiettivo come numero totale di ore non lavorate dall'operatore in quella settimana.

$$z_3 = \sum_{i \in N} \sum_{t=1}^{s} u_{ti} \qquad \text{(O.3)}$$

1.6.4 Violazione della turnazione 3+1

Se si ammette di poter negare il riposo ogni tre giorni lavorativi agli operatori dell'insieme N_3 è necessario penalizzare la violazione di questa turnazione.
Il valore di $p_{il} = 1$ indica che per l'operatore i non è previsto un riposo nel periodo di quattro giorni che va dal giorno l al giorno $l + 3$ mentre $p_{il} = 0$ indica che si sta rispettando la turnazione 3+1. Una nuova funzione obiettivo consiste, quindi, nel minimizzare il numero di attivazioni della variabile p_{il}:

$$z_4 = \sum_{i \in N3} \sum_{l=1}^{|L|-3} p_{il} \qquad \text{(O.4)}$$

La violazione della turnazione 3+1 potrebbe essere distribuita in modo equo minimizzando la violazione massima π, e introducendo il seguente vincolo:

$$\sum_{l=1}^{|L|-3} p_{il} \leq \pi, \quad i \in N3 \qquad \text{(O.4')}$$

1.6.5 Scostamento Pattern

Alcuni operatori hanno un turno preferenziale oppure è stato loro assegnato un certo turno in una schedulazione predefinita che si vorrebbe rispettare. Tuttavia, se necessario, questi assegnamenti possono essere modificati. Per penalizzare le modifiche si cerca di minimizzare lo scostamento dell'ora di inizio e di fine del turno che devono eseguire, dall'ora di inizio e di fine del turno a cui avrebbero dovuto essere assegnati. Ne deriva una quinta funzione.

$$z_5 = \sum_{i \in N} \sum_{j \in W} \sum_{l \in L} \psi_{ijl} \, x_{ijl} \qquad \text{(O.5)}$$

dove

$$\psi_{ijl} = \left|\omega_{il} - b_j\right| - \left|\Omega_{il} - e_j\right|$$

e ω_{il} e Ω_{il} sono l'ora di inizio e fine del turno preferenziale per l'operatore *i* nel giorno *l*.

1.6.6 Assegnazione di un operatore al di fuori del reparto

Alcuni operatori sono assegnati in modo permanente ad un reparto. Questi operatori possono operare al di fuori del proprio reparto ma questo assegnamento va penalizzato. Per conseguenza dobbiamo introdurre una nuova famiglia di variabili $q_{rl} \geq 0$ ad indicare il numero di operatori che, nella giornata *l*, svolgono un compito al di fuori del proprio reparto *r*. Si introduce la sesta funzione obiettivo

$$z_6 = \sum_{l \in L} \sum_{r \in D} q_{rl} \qquad \text{(O.6)}$$

la quale consiste minimizzare, giorno per giorno, il numero di operatori che svolgono un turno al di fuori del proprio reparto.

1.7 Vincoli

Descriviamo ora i vincoli che individuano le soluzioni ammissibili. Tali vincoli derivano in parte dal tipo di servizio che la struttura di Ghedi offre, in parte da normative e in parte dal contratto UNEBA.
Alcuni vincoli, infine servono a determinare il valore delle funzioni obiettivo precedentemente elencate.

1.7.1 Vincoli di servizio

Questi vincoli sono introdotti dalla natura del servizio offerto dalla struttura e sono vincoli *Hard*. Il primo vincolo definisce la necessità di effettuare un certo numero di turni in una determinata giornata.

$$\sum_{i \in N} x_{ijl} = c_{jl}, \quad j \in W, l \in L \tag{V.1}$$

Il secondo vincolo limita il numero di turni che un operatore può svolgere in una giornata.

$$\sum_{j \in T} x_{ijl} = 1, \quad i \in N / N^J, l \in L \tag{V.2}$$

$$\sum_{j \in W} x_{ijl} \leq 1, \quad i \in N^J, l \in L \tag{V.2'}$$

Gli operatori normali devono effettuare un turno di servizio ogni giorno (eventualmente riposi, ferie o malattie). Gli operatori jolly possono svolgere al più un turno di servizio al giorno (lavorativo, ovviamente), ma possono anche non essere impiegati.

1.7.2 Vincoli determinati dalla normativa del lavoro

Tutti i vincoli introdotti dalla normativa che regola il lavoro sono per definizione rigidi, in quanto definiti da leggi.
A un operatore non devono essere assegnati dei turni che non può svolgere.

$$x_{ijl} = 0, \quad j \notin W_{il}, i \in N, l \in L \tag{V.3}$$

La normativa italiana prevede che al lavoratore sia garantito almeno un riposo di 24 ore ogni sei giorni lavorati.

$$\sum_{t=0}^{6} \sum_{j \in W \cup I \cup F} x_{ij(l+t)} \leq 6, \quad i \in N, l \in 2..|L| - 6 \tag{V.4}$$

Per tenere conto del periodo precedente l'orizzonte temporale, è necessario modificare il vincolo nei giorni della frontiera, definendo

$$\sum_{t=0}^{6-a_i} \sum_{j \in W \cup I \cup F} x_{ij(t+1)} \leq 6 - a_i, \quad i \in N \tag{V.4'}$$

Dove a_i rappresenta il giorno della frontiera.
Tra due turni di lavoro successivi, al lavoratore deve essere garantito un riposo di almeno undici ore.

$$x_{ij_1 - l - 1} + x_{ij_2 l} \leq 1, \quad i \in N, l \in L, j_1, j_2 \in W : 24 - e_{j_1} + b_{j_2} < 11 \tag{V.5}$$

Anche in questo caso il vincolo va modificato per i giorni appartenenti alla frontiera.

$$x_{ij1} = 0, \quad i \in N, j \in W : y_1 \in W \wedge 24 - e_{yi} + b_j < 11 \tag{V.5'}$$

Dove e_{yi} indica l'ora di fine del turno lavorativo effettuato dall'operatore.

Va sottolineato il fatto che, per il vincolo (V.4), le giornate di malattia o ferie sono considerate lavorative, mentre non lo sono ai fini del vincolo (V.5). Bisogna pure garantire che l'operatore si possa godere le ferie concertate con il datore di lavoro, o come ferie o come giorno di riposo.

$$\sum_{j \in F \cup R} x_{ijl} = 1, \quad l \in L_i^F, i \in N \tag{V.6}$$

Similmente, l'operatore non deve essere chiamato in servizio durante un congedo per malattia.

$$\sum_{j \in I \cup R} x_{ijl} = 1 \quad se\ l \in L_i^I; \quad i \in N, l \in L \tag{V.7}$$

$$\sum_{j \in I} x_{ijl} = 0 \quad se\ l \notin L_i^I; \quad i \in N, l \in L \tag{V.8}$$

I vincoli (V.6), (V.7) e (V.8) differiscono, sebbene siano concettualmente simili gli insiemi L_i^F e L_i^I, per il fatto che, mentre è possibile assegnare dei giorni di ferie extra al lavoratore, lo stesso non vale per le malattie. Il fatto che si considerino anche i giorni di riposo come ferie o malattie, serve a non violare il vincolo (V.4).

$$\sum_{l \in S_t} \left(\sum_{j \in W} d_j x_{ijl} + \sum_{j \in I \cup F} v_i x_{ijl} \right) \leq 48, \quad i \in N, t \in \{2..s\} \tag{V.9}$$

Se il primo giorno del nostro orizzonte temporale non corrisponde con il primo giorno della settimana, il vincolo diventa

$$\sum_{l \in S_1} \left(\sum_{j \in W} d_j x_{ijl} + \sum_{j \in I \cup F} v_i x_{ijl} \right) + g_i \leq 48, \quad i \in N \qquad \text{(V.9')}$$

1.7.3 Vincoli introdotti dal CCNL UNEBA

Un vincolo rigido definito dal contratto UNEBA indica il numero minimo di riposi che deve godere un operatore nell'arco di un mese.

$$\sum_{l \in M_k} \sum_{j \in R} x_{ijl} \geq 5, \quad k \in \{1..m\}, i \in N \qquad \text{(V.10)}$$

Un secondo vincolo rigido definito dal contratto UNEBA indica il numero massimo di ore di straordinari che un operatore può effettuare in un anno. Anche la normativa fissa un tetto massimo al numero di ore di straordinari: per il contratto UNEBA il limite è pari a 120 ore contro le 250 ore definite dalla normativa.

$$f_i + \sum_{k=1} o_{ki} \leq 120, \quad i \in N \qquad \text{(V.11)}$$

1.7.4 Vincolo distribuzione dei turni notturni

L'introduzione di questo vincolo serve per distribuire i turni definiti notturni, tra tutti gli operatori abilitati a svolgere il lavoro notturno.

$$\sum_{j \in Wo} \sum_{l \in L} xijl \leq M * (1+\delta), \quad i \in No \qquad \text{(V.12)}$$

Dove il parametro M rappresenta il valore medio dei turni notturni per operatore e viene definito come

$$M = \left(\sum_{t \in To} \sum_{l \in L} ctl \right) / |No|$$

mentre δ assume un valore reale compreso tra 0 e 1, e indica la percentuale massima di turni notturni che un operatore può svolgere in più rispetto alla media M .

1.8 Violazione vincoli operativi ed organizzativi

I vincoli di questo gruppo rappresentano gli accordi tra gli operatori e la cooperativa.

Non sono imposizioni definite da un contratto nazionale, né derivano da accordi commerciali tra la cooperativa e la casa di riposo. Sono tipicamente vincoli flessibili per cui definiscono alcune funzioni obiettivo.

Il numero di ore di lavoro in un mese (V.13) e in una settimana (V.14) è limitato per ogni lavoratore.

Quello relativo al mese non può superare H_{ki}, a meno che vi siano straordinari.

$$\sum_{l \in Mk} \left(\sum_{j \in W} d_j x_{ijl} + \sum_{j \in I \cup F} v_i x_{ijl} \right) \leq H_{ki} + o_{ki}, \quad i \in N, k \in \{1..m\}$$

(V.13)

Quello relativo alle settimane deve superare h_i, a meno che vi siano ore non lavorate.

$$\sum_{l \in St} \left(\sum_{j \in W} d_j x_{ijl} + \sum_{j \in I \cup F} v_i x_{ijl} \right) + u_{ti} \geq h_i, \quad i \in N, t \in \{2..s\}$$

(V.14)

Se il primo giorno del nostro orizzonte temporale non corrisponde con il primo giorno della settimana.

$$\sum_{l \in S1} \left(\sum_{j \in W} d_j x_{ijl} + \sum_{j \in I \cup F} v_i x_{ijl} \right) + g_i + u_{i1} \geq h_i, \quad i \in N \qquad \text{(V.14')}$$

Il vincolo (V.15) definisce la turnazione 3+1. Questo vincolo si applica solamente agli operatori dell'insieme N_3.

$$\sum_{t=0}^{3} \sum_{j \notin R} x_{ij(l+t)} \leq 3 + p_{il}, \quad i \in N3, l \in 2..|L| - 3 \qquad \text{(V.15)}$$

Pure questo vincolo deve essere riformulato nei pressi della frontiera.

$$\sum_{t=0}^{3-a_i} \sum_{j \notin R} x_{ij(t+1)} \leq 3 - a_i + p_{i1}, \quad i \in N3 \qquad \text{(V.15')}$$

L’ultimo vincolo determina la permanenza di un operatore nel reparto assegnatogli.

$$\sum_{i \in Nr} \sum_{j \in Tr} x_{ijl} + q_{rl} \geq \min\left\{ \sum_{j \in Tr} c_{jl}, \left| N_r \right| \right\}, \quad r \in D, l \in L \qquad \text{(V.16)}$$

Se il numero di operatori richiesti dai turni associati a un reparto è inferiore a quello degli operatori associati al reparto, gli operatori in eccesso fanno turni fuori dal reparto. In caso contrario, tutti gli operatori associati al reparto vi prestano servizio, e i turni mancanti sono coperti da operatori esterni, grazie al vincolo V.1.

2. Architettura di supporto alle decisioni

In questo capitolo viene descritta l'architettura di supporto alle decisioni realizzata.
Un DSS viene utilizzato da un decisore esperto come strumento per calcolare e scegliere tra diverse soluzioni alternative.
L'interfaccia gestisce l'interazione tra le altre componenti del sistema quali il database, il sistema di formalizzazione delle preferenze ed il risolutore. Con l'ausilio di finestre, dall'interfaccia è possibile visualizzare e modificare dati nel database, inserire i pesi da attribuire ai criteri decisionali, creare i dati utili per il risolutore, lanciare l'esecuzione del risolutore stesso, elaborare e visualizzare la soluzione proposta, e produrre adeguati report su schedulazioni memorizzate, o sul lavoro svolto dagli operatori.

2.1 Architettura del sistema di supporto alle decisioni

L'architettura del sistema di supporto alle decisioni che è stato implementato, si compone di quattro parti principali (vedi fig. 2.1):

- il *database* (DB), utilizzato per conservare le informazioni sugli operatori e sui turni presenti nella struttura ospedaliera, nonché tutte le informazioni relative alle diverse soluzioni calcolate;
- il *sistema di formalizzazione delle preferenze*, permette al decisore di esprimere preferenze relative tra i criteri decisionali e di formalizzare tali preferenze come pesi nella funzione obiettivo;

- il *risolutore*, un algoritmo di ottimizzazione che definisce un assegnamento ottimale rispetto alle preferenze dell'utente di turni agli operatori per l'intervallo definito dall'utente;
- l'*interfaccia*, realizzata con lo scopo di fornire una gestione semplice e rapida delle informazioni contenute nella base dati, e di trasmettere le informazioni necessarie al risolutore.

L'interfaccia ha il compito di collegare le informazioni della base di dati, l'utente e l'algoritmo di risoluzione.

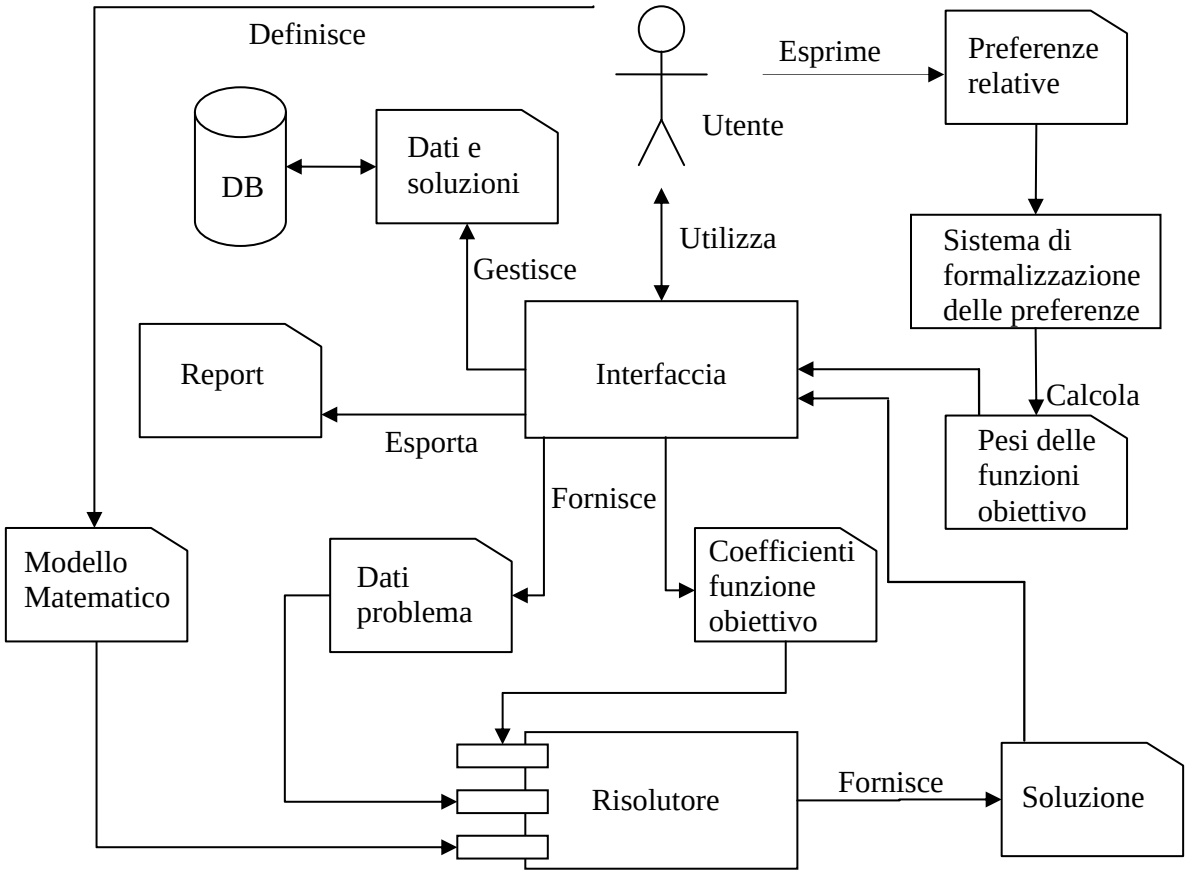

Fig. 2.1 Architettura di supporto alle decisioni.

Le quattro componenti principali dell'architettura in fig. 2.1, il database, il sistema di formalizzazione delle preferenze, il risolutore e l'interfaccia, sono

collegate in modo da facilitare le operazioni dell'utente per l'utilizzo dell'intero sistema.

Tramite finestre specifiche l'utente può gestire le informazioni contenute nel database e creare in modo automatico i dati necessari al risolutore, in un file opportunamente formattato, per calcolare la soluzione ottima.

Per la gestione di dati si intende l'inserimento di nuove tuple, la modifica di dati esistenti o la cancellazione di informazioni contenute in certe tabelle.

L'interfaccia è composta da diverse finestre ognuna delle quali gestisce particolari informazioni contenute nelle tabelle della base di dati.

Queste operazioni vengono svolte da *stored procedure* che sono eseguite dall'interfaccia; ogni procedura effettua una specifica operazione su una distinta tabella.

Per creare il file da passare al risolutore, l'utente deve specificare il periodo sul quale vuole generare una schedulazione, mentre gli altri dati riguardanti gli operatori e i turni sono ricavati tramite interrogazioni della base di dati.

L'attivazione del risolutore avviene in modo automatico; al termine dell'elaborazione del problema proposto, il risolutore, fornisce la soluzione in un file di dati che viene memorizzato nelle tabelle apposite della base di dati mediante opportune procedure di lettura.

In seguito il decisore può visualizzare la soluzione proposta nell'apposita finestra dell'interfaccia per la gestione delle schedulazioni. L'interfaccia propone la schedulazione calcolata in un tabella operatore-giorno e visualizza i valori delle singole funzioni obiettivo $z_1 \ldots z_6$. Il decisore può, in un secondo tempo, esportare nel formato di un "foglio di lavoro excel" la schedulazione che preferisce.

Il decisore, inoltre, può forzare l'assegnamento di alcuni turni in alcuni giorni a determinati operatori, rispettando invece le altre assegnazioni proposte dal solutore con l'ausilio dell'interfaccia.

2.1.2 Il Database

La base di dati è formata da 14 tabelle, due delle quali sono le principali, mentre tutte le altre forniranno un supporto alla memorizzazione dei dati in esse, in modo da eliminare tutte le informazioni duplicate utilizzando il metodo di *normalizzazione* delle informazioni.
Le due tabelle principali riguardano, rispettivamente gli operatori e i turni lavorativi.
Per la gestione dell'inserimento, della modifica e della cancellazione dei dati nelle tabelle del database, sono presenti delle *stored procedure*, le quali accelerano il collegamento con la base di dati, e facilitano la modularità e il riutilizzo del codice a livello dell'interfaccia.

2.1.3 Sistema di formalizzazione delle preferenze

All'utente viene data la possibilità di impostare i pesi da assegnare alle funzioni obiettivo per ogni schedulazione. Inizialmente sono presenti dei valori calcolati con il metodo del *confronto a coppie* descritto nel cap.6, in ogni modo si possono cambiare a seconda delle proprie esigenze. Come viene fatto notare in seguito, l'assegnamento dei pesi in modo razionale è difficile se vengono inseriti in maniera casuale.
L'inserimento dei valori viene fatto tramite componenti nella finestra dell'interfaccia nella quale si crea il file di dati da passare al solutore.

2.1.4 Solutore

L'algoritmo che costruisce un assegnamento di turni ottimale, è un algoritmo generico di Programmazione Lineare Intera (PLI). È un algoritmo *open*

source e fornisce una soluzione ottima del problema, richiedendo in ingresso due file. Il primo file rappresenta il modello matematico generico cioè l'elenco delle variabili di decisione dell'obbiettivo da ottimizzare dei vincoli da rispettare, mentre il secondo file, strettamente correlato al primo fornisce i dati specifici degli operatori, dei turni e del periodo di tempo al quale si riferisce il caso in esame (l'istanza del problema).

Il file di ritorno con la possibile soluzione è di difficile lettura per gli utenti che dovranno utilizzare il sistema. Quindi opportune procedure vengono lanciate in automatico dall'interfaccia dopo che il risolutore ha finito l'elaborazione di un problema, per scandire le informazioni proposte e memorizzarle nel database. In tal modo, dall'interfaccia si potrà avere in seguito una visualizzazione semplificata della schedulazione temporale degli assegnamenti dei turni agli operatori.

2.1.5 Interfaccia

L'interfaccia per la gestione della base di dati è stata implementata nell'ambiente di sviluppo Delphi, il quale utilizza come linguaggio di programmazione l'Object Pascal.

Questo ambiente grafico ha lo scopo di far interagire in modo rapido e intuitivo l'utente con la base di dati ed il risolutore. L'utente ha una visione semplificata delle informazioni contenute nelle tabelle della base di dati, in quanto le relazioni fra dati di tabelle diverse vengono create col valore della chiave primaria (o col gruppo di valori che formano la chiave primaria), che rappresenta l'attributo ID, mentre l'utente ha accesso per correlare dati diversi, solo al valore del campo "codice" che si riferisce ai dati di una particolare tupla definita da quello specifico valore ID.

Ad esempio quando si assegna un operatore ad un reparto, nella struttura del database viene creata una relazione tra i dati dell'operatore e quelli del

reparto, memorizzando nel campo “reparto” della tupla dell’operatore il valore dell’attributo ID del reparto. Al livello dell’interfaccia, invece, si crea l’accoppiamento operatore-reparto attraverso la finestra degli operatori: selezionato l’operatore, nell’oggetto che visualizza i codici dei reparti, è possibile inserire il codice del reparto desiderato.

Questo meccanismo è ripetuto per tutti i tipi di assegnamento gestiti dall’interfaccia, la quale visualizza sempre il campo “codice” ma crea le dovute relazioni di dati, risalendo in automatico al valore del campo ID corrispondente.

2.1.6 Sistema di selezione dell’alternativa preferita

Il risolutore fornisce la soluzione ottima del problema, in base ai pesi forniti dall’utente.

La soluzione proposta soddisfa i vincoli definiti nel modello matematico, e il suo costo cambia in base ai pesi definiti a priori delle sei funzioni obiettivo che descrivono altrettanti criteri di preferenza (vedi cap. 1).

Modificando i pesi degli obiettivi, il risolutore può fornire soluzioni diverse, in modo tale da privilegiare un obiettivo piuttosto che un altro.

La soluzione che viene generata non è un assegnamento definitivo, ma solo un aiuto per il decisore. Questi potrà accettare la soluzione rendendola attiva, oppure impostare alcuni assegnamenti e mantenerne altri generando così altre soluzioni.

3. Database

Le informazioni inerenti il personale che lavora nella struttura, e la definizione dei loro turni di servizio, sono inserite in un database realizzato con SQL SERVER 2000. Il database è composto da diverse tabelle fra loro legate. Prima di definire la struttura delle tabelle e delle loro relazioni, dobbiamo avere una visione chiara di che cosa sia un database, e a che cosa serva utilizzare un database.

Per ogni tabella presente nel database, inoltre, c'è una *stored procedure* per l'inserimento di nuovi dati, una per la modifica di dati esistenti, e un'altra per la cancellazione dei dati, la quale può essere fisica, oppure solo un aggiornamento del valore di un campo per evitarne la visualizzazione di queste informazioni dall'interfaccia, ma che permette di tenerle memorizzate nella base di dati per la lettura dello "storico". Nelle sezioni successive viene descritta la struttura generale della base di dati, di ogni tabella, e il codice relativo alle differenti operazioni che una *stored procedure* esegue sul database.

3.1 Database

Un database rappresenta un contenitore in cui archiviare i dati. Un database non presenta informazioni fruibili dall'utente in maniera diretta. Piuttosto, l'utente esegue un'applicazione che accede ai dati del database e li presenta in un formato comprensibile. Esempio pratico è l'interfaccia utilizzata per gestire l'inserimento o la cancellazione delle informazioni nella struttura di memorizzazione.

I sistemi di database sono più potenti dei classici file di dati in quanto prevedono un'organizzazione più capillare dei dati. In un database ben progettato non sono presenti porzioni di dati duplicate che l'utente o l'applicazione dovrebbe aggiornare contemporaneamente per mantenerne la coerenza. Le porzioni di dati sono raggruppate in una singola struttura o *record* ed è possibile definire relazioni tra *record*. Quando si lavora con i file di dati, un'applicazione deve essere codificata in modo da essere compatibile con la struttura specifica di ciascun file di dati. Al contrario, un database contiene un catalogo che viene utilizzato dalle applicazioni per stabilire in che modo sono organizzati i dati. La maggior parte delle applicazioni per database utilizza il catalogo per rappresentare in maniera dinamica agli utenti i dati contenuti in database diversi, senza essere legata ad un formato di dati.

3.1.1 Database relazionali

Sebbene esistano diverse tecniche di organizzazione dei dati in un database, i database relazionali rappresentano decisamente uno dei sistemi più efficaci. Un sistema di database relazionale utilizza la teoria matematica degli insiemi per organizzare i dati in maniera efficace. In un database relazionale i dati sono raccolti in tabelle, definite *relazioni*. Una tabella rappresenta classi di oggetti importanti per un'organizzazione. Nel nostro caso, il database conterrà una tabella per gli operatori, una per i turni e una per i reparti. Ciascuna tabella è costituita da colonne e righe, definite rispettivamente *attributi* e *tuple*. Ciascuna colonna rappresenta un attributo della classe di oggetti rappresentata dalla tabella. La tabella degli operatori presenta, pertanto, colonne relative ad attributi quali il codice identificativo della persona, il nome della persona, il codice relativo al reparto di appartenenza e l'orario di lavoro effettuato.

Ciascuna riga rappresenta un'istanza della classe dell'oggetto rappresentata dalla tabella; ad esempio una riga della tabella degli operatori rappresenta l'operatore con identificativo 7. Esistono diversi modi per organizzare i dati in tabelle. La teoria dei database relazionali definisce un processo chiamato *normalizzazione* che assicura che il set di tabelle specificato organizzi i dati in maniera efficace.

3.2 Componenti di un database SQL Server

Un database SQL Server è composto da tabelle che memorizzano insiemi specifici di dati strutturati. Le tabelle dispongono di diversi tipi di controlli (vincoli, regole, trigger, default, e tipi di dati utente personalizzati) che assicurano la validità dei dati, e degli indici che consentono una ricerca rapida delle righe (come gli indici dei libri).
Un database SQL Server è in grado di memorizzare procedure che utilizzano il codice di programmazione Transact-SQL per eseguire operazioni con i dati, come la memorizzazione di viste che forniscono un accesso personalizzato ai dati di una tabella.

3.2.1 Normalizzazione di un progetto di database

Il progetto di un database comprende il processo di normalizzazione, che implica l'utilizzo di metodi formali per dividere i dati in più tabelle correlate. La caratteristica di un database normalizzato è di consistere in un numero elevato di tabelle ristrette (con poche colonne); invece, un database denormalizzato consiste in poche tabelle ampie (con molte più colonne). Spesso, una normalizzazione adeguata comporta un miglioramento delle

prestazioni, in quanto i dati sono riorganizzati in modo tale che ogni tabella contenga informazioni dello stesso tipo. Questo facilita la ricerca di dati particolari, dato che evita di scorrere un gran numero di colonne di un'unica tabella, oppure facilita la creazione di nuove tabelle per inserire nuove informazioni nella base di dati.

Le regole di normalizzazione identificano attributi che devono essere presenti in un database ben progettato. Queste regole possono rivelarsi complicate e la loro descrizione richiederebbe troppo spazio. Tuttavia, le seguenti regole fondamentali consentono di ottenere un valido progetto di database: ogni tabella

1. deve disporre di un identificatore;
2. deve memorizzare dati per un solo tipo di identità;
3. non deve contenere valori o colonne ripetute.

Identificatore di tabella

Una delle regole importanti nel progetto di un database è che ciascuna tabella disponga di un identificatore di riga univoco, il quale è una colonna o un insieme di colonne utilizzate per distinguere ciascun record da tutti gli altri presenti nella tabella. È importante che ciascuna tabella disponga di una colonna ID e che nessuna coppia di record condivida lo stesso valore ID. La colonna che funge da identificatore di riga univoco per una tabella è detto *chiave primaria* della tabella.

Memorizzazione di dati in tabella

Il tentativo di memorizzare un numero elevato di informazioni in una tabella può impedire una gestione efficace e affidabile dei dati nella tabella.

Nel database realizzato, c'è una tabella che descrive i turni effettuati (fig. 3.1); questa tabella ha un attributo che specifica il numero di operatori che devono lavorare in quello specifico turno. Un'altra tabella correlata a questa tabella, riporta la richiesta aggiuntiva di operatori per quel particolare turno in un determinato periodo. Se si fosse deciso invece di aggiungere a ciascuna riga nella tabella principale dei turni l'informazione sulla richiesta di operatori extra per un turno, il numero di attributi presenti sarebbe aumentato, e parecchie tuple sarebbero state ripetute quasi identicamente, poiché un turno può avere una richiesta di operatori maggiore del solito in un dato periodo di tempo, e una richiesta inferiore in un altro. Questo produrrebbe tuple con le stesse informazioni riguardanti codice, nome, reparto di appartenenza, ora di inizio e di fine del turno.

Fig. 3.1 Tabella "turni" e tabella "rich_extra"

3.3 Descrizione Tabelle

Il database realizzato per la gestione delle informazioni consiste in due tabelle principali; altre tabelle minori sono in relazione con le prime in modo tale da normalizzare tutte le informazioni presenti e ridurre il numero di tuple ridondanti.

Lo schema in fig. 3.2 è la struttura della base di dati con tutte le sue tabelle e le relazioni fra loro.

Le tabelle principali sono O*peratori* e T*urni*. Altre tabelle relazionate a queste riportano informazioni sulla gestione del personale (*Ferie, Malattie, Lavoro, Costi, Orari, Turni_ass* e *Turni_pref*), altre ancora descrivono le schedulazioni previste (*Schedulazioni* e *Assegnazioni*) ed i turni effettivamente svolti dagli operatori (*Cartellini*). Infine la tabella *Reparti* che si lega a quella dei *Turni* (la quale ha correlata una tabella descrittiva delle richieste extra di operatori per certi periodi di un particolare turno) e a quella degli operatori.

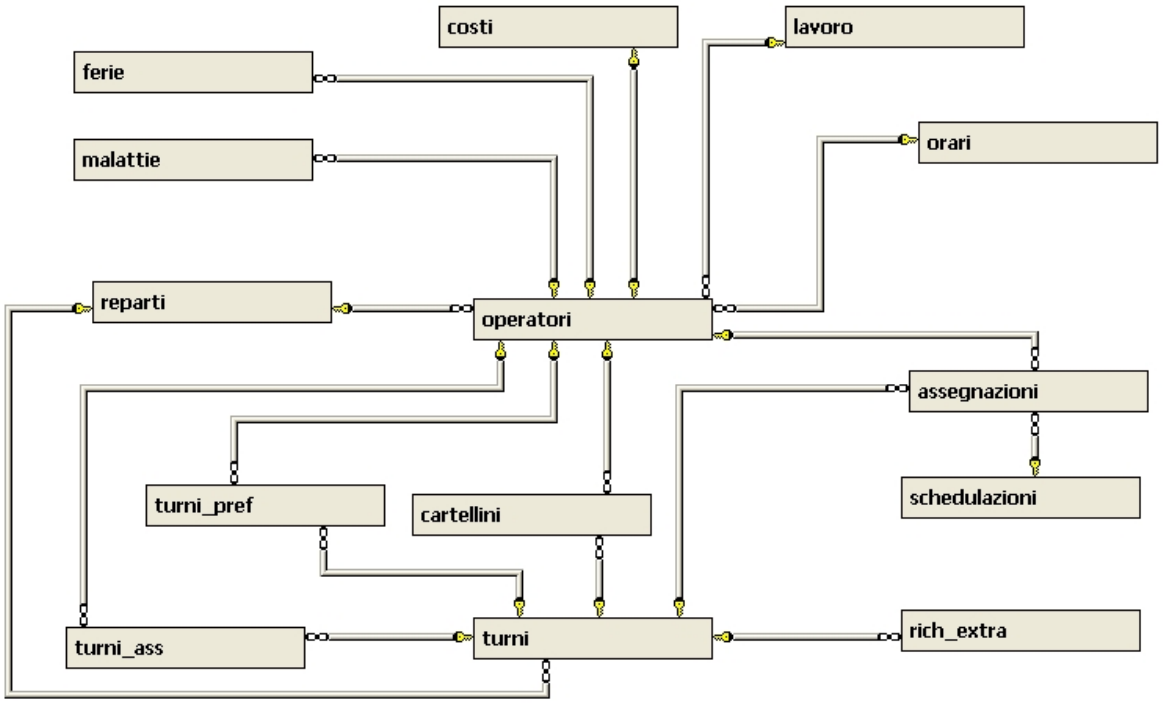

Fig. 3.2 Database “Turni”

3.3.1 Reparti e Schedulazioni

La tabelle *Reparti* e S*chedulazioni* presentate in fig. 3.3, sono tabelle semplici e non dipendono da nessun altra tabella.

Fig. 3.3 Tabelle “reparti” e “schedulazioni”

Reparti

La tabella *Reparti* è utilizzata per classificare gli operatori e i turni a seconda del reparto d'appartenenza. Di fatto ogni tupla della tabella modella un reparto. Per ogni reparto è definito, oltre ad una chiave primaria (PK) definita da un valore progressivo, un codice univoco e due descrizioni, una lunga ed una breve. La descrizione lunga è il nome vero e proprio, mentre quella breve può essere utilizzata, ogni qual volta sia necessario ridurre lo spazio occupato.

Schedulazioni

Schedulazioni contiene l'intestazione di tutte le schedulazioni effettuate dall'applicazione. Per ogni schedulazione è registrato un codice univoco ed una descrizione, il periodo di riferimento (inizio e fine), il costo della schedulazione stessa, cioè il valore totale della funzione obiettivo calcolata dal risolutore e il valore di ciascuno degli obiettivi $z_1 \ldots z_6$ definiti nel capitolo 1.

3.3.2 Turni e Rich_extra

In figura 3.1 è riportato lo schema della tabella *Turni*. Ogni tupla è caratterizzata da una chiave, da un codice univoco e da una descrizione del turno. L'appartenenza ad un reparto particolare è identificata da una chiave esterna (*foreign key*, FK). Per ogni turno sono indicati l'ora d'inizio e di fine del turno e l'effettiva durata di questo. In ultimo esiste la possibilità di indicare il numero di persone che devono effettuare il turno ogni giorno.
Nella figura è presente anche la tabella *Rich_extra* la quale riporta informazioni inerenti a variazioni del carico di lavoro richiesto per un determinato turno durante un certo periodo di tempo.

3.3.3 Operatori, Lavoro, Costi, Turni_pref e Turni_ass

Queste tabelle riportano dati gestionali riguardanti gli operatori, e sono collegate alla tabella *Operatori* con relazione uno-a-molti. Una relazione uno-a-molti è il tipo più comune di relazione. Una riga nella tabella *Operatori* può avere più righe corrispondenti nelle altre tabelle, ma ciascuna riga di queste può avere solo una riga corrispondente nella tabella *Operatori*. Una relazione uno-a-molti viene creata se solo una delle colonne correlate è una chiave primaria o dispone di un vincolo univoco.

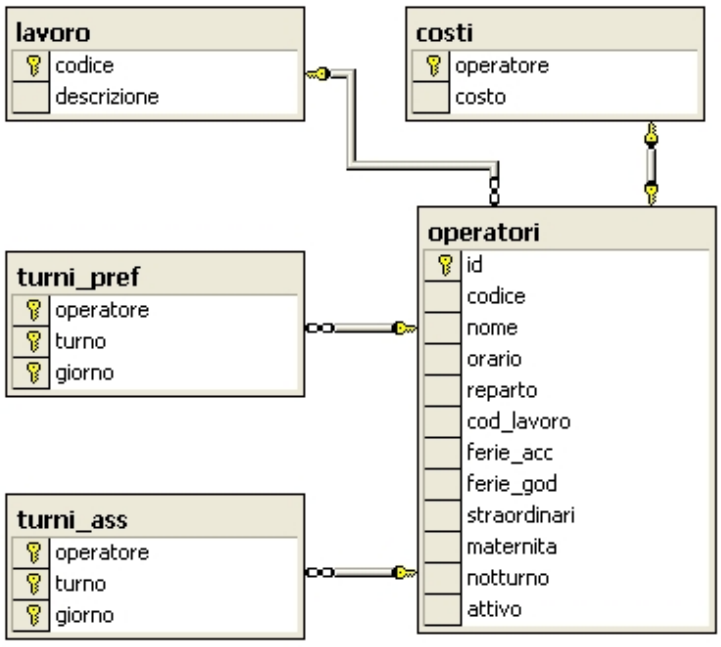

Fig. 3.4 Tabelle “Operatori”, “Lavoro” , “Costi” , “Turni_pref” e “Turni_ass”

Operatori

Per ogni operatore sono definiti una chiave primaria, un codice univoco ed il nome dell'operatore. A queste informazioni sono aggiunti i riferimenti all'orario con il quale l'operatore è assunto, al reparto al quale afferisce e la tipologia di lavoro che svolge. Sono anche registrati il numero di giorni di ferie accumulate dall'inizio dell'anno ad oggi e di quelli già goduti. In ultimo, sono indicate le informazioni in merito al fatto che l'operatore sia in maternità o che possa effettuare lavoro notturno.

Lavoro

Questa tabella riporta il codice e la descrizione di una particolare tipologia di lavoro che un operatore deve svolgere:

1. l'operatore può svolgere qualsiasi tipo di lavoro;
2. l'operatore è assunto come lavoratore "3+1" cioè ogni 3 giorni di lavoro consecutivi, il quarto giorno deve essere obbligatoriamente di riposo;
3. l'operatore è classificato nella tipologia "Jolly".

La chiave primaria di questa tabella è l'attributo codice.

Costi

Ogni operatore ha un suo costo, in base all'esperienza di lavoro oppure al tipo di contratto di assunzione.

Turni_pref e Turni_ass

Queste due tabelle, che sono strutturalmente identiche, riportano il turno che un operatore preferirebbe fare in uno specifico giorno (*Turni_pref*) e il turno che viene assegnato ad un operatore in un determinato giorno (*Turni_ass*). Quest'ultima tabella, dunque, contiene decisioni già prese e non modificabili dal DSS.

3.3.4 Ferie e Malattie

Le tabelle in figura 3.5 sono identiche nella struttura; descrivono un periodo, di ferie o di malattia di un operatore, definito da una data di inizio e una di fine.

La chiave primaria di queste tabelle è l'insieme dei tre attributi, dato che solo così si ottiene l'identificazione univoca di ciascuna informazione. Uno stesso operatore infatti può nell'arco di un anno effettuare diversi periodi di ferie (o di malattia) delimitati da date d'inizio e di fine diverse. Dall'altra parte, nello stesso periodo possono essere in ferie o malattia operatori diversi.

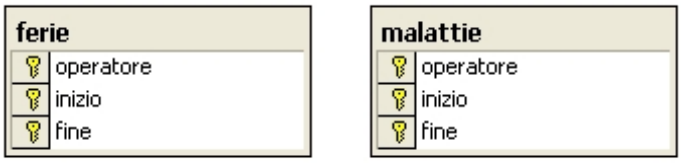

Fig. 3.5 Tabelle "Ferie" e "Malattie"

3.3.5 Orari

La tabella *Orari*, definisce l'orario contrattuale di assunzione di ogni operatore specificando le ore di lavoro giornaliere per ciascun giorno della settimana.

C'è una chiave primaria di identificazione di ogni singola tupla (ID), un codice di identificazione dell'orario, una descrizione, il numero delle ore settimanali e di quelle giornaliere. Questi valori vengono ricavati con una semplice formula matematica dove il numero di ore settimanali è ricavato dalla somma delle ore di ogni giorno di una settimana, mentre il valore dell'ora giornaliera è il rapporto tra il valore delle ore settimanali ed il numero di giorni di lavoro.

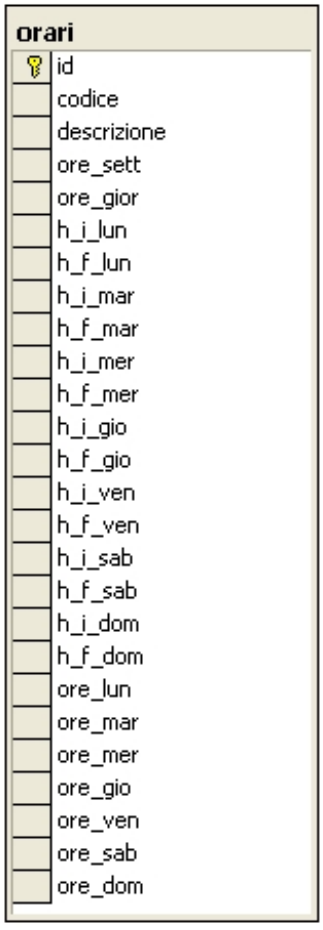

Fig. 3.6 Tabella "Orari"

3.3.6 Cartellini e Assegnazioni

In figura 3.7 sono riportati gli schemi delle tabelle *Cartellini*, che rappresenta i turni effettivamente svolti da un operatore, e *Assegnazioni* che riporta i turni assegnati da una schedulazione.
La tabella *Cartellini* presenta come chiave primaria la coppia di attributi data-operatore, mentre la tabella *Assegnazioni* ha come chiave primaria schedulazione-data-operatore, infatti ci possono essere più schedulazioni presenti per uno stesso periodo di tempo.

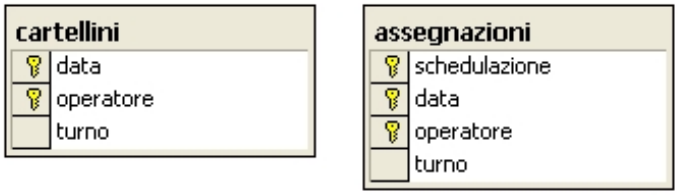

Fig. 3.7 Tabelle “Cartellini” e “Assegnazioni”

Cartellini

Ogni tupla rappresenta una tripla giorno-operatore-turno. Per ogni tripla è indicato il turno effettuato da un operatore in un giorno.

Assegnazioni

La tabella *Assegnazioni* è pressoché identica a *Cartellini*. L’unica differenza consiste nel fatto che ogni assegnazione è associata anche alla schedulazione dalla quale è stata generata.

3.4 Controlli di coerenza

In tutte le tabelle sono presenti dei controlli di coerenza che vietano i valori inamissibili per determinati attributi di specifiche tabelle. I principali sono:

- *tabella Operatori*: il valore degli attributi “maternità” e “notturno” è mutuamente esclusivo, in quanto (vedi cap. 1), un operatore che è in stato di gravidanza non può svolgere lavoro notturno.

- *Tabella Turni*: la durata di un turno lavorativo deve essere superiore a 0; il valore 0 è utilizzato per definire la durata dei turni di “riposo”, “ferie” e “malattia”, per poterli distinguere da tutti gli altri turni lavorativi.

- *Tabelle Ferie e Malattie*: la data di inizio di un periodo di ferie o malattia, deve precedere o uguagliare la data di fine.

- *Tabella Rich_Extra*: anche per questa tabella la data di fine deve essere maggiore o al più uguale alla data di inizio.

3.5 Osservazioni finali

Nelle tabelle Operatori, Turni e Reparti è presente un attributo “attivo” di tipo bit (0 o 1) il quale indica la cancellazione virtuale dalla base di dati delle informazioni relative ad una tupla di queste tabelle; la cancellazione fisica delle informazioni, avrebbe causato problemi su come gestire lo storico dei dati, così facendo le informazioni sono sempre presenti nel database, ma con

l’attributo “attivo” posto a 0 non verranno più visualizzate dall’interfaccia, simulando così l’eliminazione di queste informazioni.

3.6 Stored Procedure

Le *stored procedure* sono un’insieme precompilato di istruzioni *transact-SQL* memorizzate con un nome ed elaborate come unità. Esse forniscono vantaggi nelle prestazioni, in particolar modo accelerano l’inserimento, la modifica e la cancellazione di nuove tuple nel database, in quanto sono memorizzate nella base di dati, e quindi a stretto contatto con l’intera struttura di memorizzazione. Inoltre garantiscono la modularità e il riutilizzo del codice.
Come avviene negli altri linguaggi di programmazione, le *stored procedure* accettano parametri di input e possono restituire parametri di output.

3.6.1 Passaggio di parametri

L’esecuzione delle *stored procedure* utilizzate per inserire, modificare e cancellare tuple nel database, richiede il passaggio di parametri di input.
La seguente *stored procedure* inserisce in modo corretto una nuova tupla nella tabella degli operatori solo se i parametri passati in ingresso alla procedura rispecchiano l’ordine e il tipo di ogni attributo della tabella in cui si deve effettuare l’inserimento; se l’ordine non è lo stesso, oppure un dato è di diverso tipo da come è definito nella tabella, non viene effettuato nessun inserimento di nuovi dati nella tabella.
La dichiarazione della *stored procedure* comprende il suo nome, e la lista dei parametri d’ingresso.

```
CREATE PROCEDURE operatori_insert
(@codice varchar(10), @nome varchar(80), @orario int,
@reparto int, @cod_lavoro int, @ferie_acc
decimal(4,1),
@ferie_god decimal(4,1),@straordinari decimal(5,2),
@maternita bit, @notturno bit, @id int OUTPUT) AS
```

Quando un parametro è definito di input e di output, è preceduto dal simbolo "at" (@), seguito dal nome del parametro e dall'indicazione del tipo di dato.

```
SET @id=(SELECT id FROM operatori WHERE
codice=@codice)
```

Il valore del campo ID non viene passato esplicitamente dall'esterno, ma generato in automatico; questo attributo identifica il valore della *primary key* della tabella presa in considerazione (tabella Operatori). In nessuna tabella della base di dati ci possono essere tuple differenti con valori uguali per l'attributo della chiave primaria. Quindi, per evitare un possibile errore a chi effettua il passaggio dei parametri alla *stored procedure*, il valore del parametro ID viene definito come valore massimo dell'attributo ID presente nella tabella in cui si effettua l'inserimento aumentato di un'unità.

3.6.2 Procedure di Inserimento

L'inserimento di nuovi dati in una qualsiasi tabella del database tramite *stored procedure* avviene tramite il comando *INSERT INTO <nome tabella>*.
Ad esempio, per inserire una nuova tupla nella tabella degli operatori, si scrive:

```
INSERT INTO operatori
(codice, nome, orario, reparto, cod_lavoro, ferie_acc,
ferie_god, straordinari, maternita, notturno, attivo)
```

Dopo il comando di specifica di inserimento, seguono gli attributi della tabella, nello stesso ordine con il quale sono stati dichiarati al momento della creazione della tabella stessa.

```
VALUES
(@codice, @nome, @orario, @reparto, @cod_lavoro,
@ferie_acc, @ferie_god, @straordinari, @maternita,
@notturno, 1)
```

Il comando *VALUES* seguito dell'elenco dei parametri di ingresso, permette di associare questi ultimi agli attributi definiti in precedenza. In questo modo si identificano i campi della tabella in cui si inserirà la nuova tupla di dati.

3.6.3 Procedure di modifica

La modifica di dati esistenti in una determinata tabella della base di dati, avviene con l'utilizzo del comando *UPDATE <nome tabella> SET*.

```
UPDATE operatori SET
codice=@codice, nome=@nome,
orario=@orario, reparto=@reparto,
cod_lavoro=@cod_lavoro,
ferie_acc=@ferie_acc,
ferie_god=@ferie_god,
straordinari=@straordinari,
maternita=@maternita,
notturno=@notturno, attivo = 1
```

```
WHERE id = @id
```

Dopo la parola chiave *SET*, viene definito l'assegnamento dei parametri in ingresso, ai rispettivi attributi della tabella precisati dall'elenco della dichiarazione della procedura, e inseriti nella tabella del database nella tupla identificata dal valore dell'attributo ID passato come parametro di ingresso.

3.6.4 Procedure di Cancellazione

Per cancellare fisicamente dei dati da un database, si utilizza la parola di comando *DELETE <nome tabella> WHERE id = @id*, naturalmente passando come parametro di ingresso il valore dell'attributo ID che identifica la tupla nella tabella da eliminare. Tuttavia si è deciso di non eliminare fisicamente i dati riguardanti gli operatori, i turni, e i reparti, ma di renderli solamente "inattivi", cambiando il valore di un attributo della tupla in modo che l'interfaccia non visualizzi più dall'interfaccia quei dati.

```
UPDATE operatori SET
attivo = '0'
WHERE id = @id
```

Ad esempio, l'istruzione seguente modifica l'attributo *"attivo"* nella tabella operatori assegnandogli, il valore associato alla non attività di un operatore. Per le altre tabelle, quelle di supporto alle due tabelle principali, i dati che non servono vengono cancellati in modo definitivo dalla base di dati, tramite il comando *DELETE*.

4. Il Solutore

In questo capitolo si descrive l'utilizzo di un solutore di Programmazione Lineare Intera per la risoluzione del problema tramite un approccio prettamente modellistico. Si tratta del solutore *open source* GLPK (GNU Linear Programming Kit). Un solutore lineare può essere utilizzato sia per determinare la schedulazione ottimale, sia per verificare l'ammissibilità di un'istanza. Per ricavare una soluzione, si risolve un problema di Programmazione Lineare Intera (PLI) con la tecnica *branch & bound*, che divide il problema in un insieme di sottoproblemi strutturati ad albero. La strategia di visita di tale albero adottata nel nostro caso è la DFS (Depth First Search) che consente di ricavare il più presto una soluzione ammissibile del problema presentato. Non si deve infatti dimenticare che il tempo di elaborazione è limitato. Purtroppo, l'impiego di questo solutore è limitato ad istanze di piccola dimensione dalla complessità del problema, e dal poco tempo a disposizione. Su istanze più grosse il solutore fornisce soluzioni, ma non ne garantisce l'ottimalità.

4.1 Strumenti general purpose per la PLI

Una volta che un problema è stato modellato come un problema di PLI, per determinarne una soluzione si può implementare un algoritmo risolutivo ex-novo oppure fare affidamento su strumenti preesistenti. Questi strumenti, definiti *general purpose*, perchè non sono legati ad un unico dominio applicativo o problema, permettono teoricamente di risolvere qualsiasi problema per il quale sia stato costruito un modello matematico di PLI. In

pratica, però, la soluzione può richiedere un tempo che, sebbene finito, è assolutamente non praticabile.
Tra gli strumenti a disposizione in questo genere di approccio si ricordano:

- i solutori
- i generatori algebrici di modelli
- i linguaggi di modellazione

Il solutore è lo strumento base per affrontare il problema permettendo di determinare la soluzione. Spesso il solutore richiede di formulare il problema in una modalità molto complicata. In questo caso l'utilizzo di un generatore algebrico di modelli, insieme ad un linguaggio di programmazione ad alto livello detto di modellazione, permette di formulare in modo semplice un problema che possa essere interpretato e risolto dal solutore.

4.1.1 I solutori di PLI

Un solutore di Programmazione Lineare (PL), o solutore lineare, è un applicativo che riceve in ingresso un modello matematico ed i dati che caratterizzano un'istanza del problema e restituisce una soluzione ottima. Tipicamente un solutore lineare determina la soluzione con il metodo del simplesso e le sue varianti. Un solutore lineare può, in genere, anche risolvere problemi di Programmazione Lineare Intera, servendosi di algoritmi di *branch & bound* predefiniti, eventualmente con l'aggiunta di piani di taglio, e di euristiche d'arrotondamento per generare i *bound* richiesti dal metodo.
Molto spesso il solutore lineare rende disponibile un insieme di librerie in linguaggi di programmazione ad alto livello, che permettono di richiamare le

funzioni del solutore da programmi scritti ad hoc. È quindi possibile utilizzare metodi di risoluzione più raffinati quali, ad esempio, le tecniche di *branch & bound*, i piani di taglio e i metodi di *branch & price*. In circolazione si trovano vari pacchetti software come, per citare uno tra i più potenti e diffusi, CPLEX della ILog [ILO].
Un solutore può quindi essere utilizzato in due modi differenti:

1. come libreria esterna che opera su strutture dati condivise o su una descrizione del problema su file di testo;

2. come programma stand alone che opera su una descrizione del problema su file di testo;

Esistono diversi formati attraverso i quali descrivere un problema in un file di testo. Nel caso più semplice questi formati prevedono una formulazione estesa del modello, cioè richiedono di definire uno per uno i vincoli e le variabili che lo compongono. Questo è poco pratico per istanze di grande dimensione. Tuttavia molti risolutori consentono anche di impiegare un linguaggio di modellazione per la descrizione del problema e dei dati. Successivamente un generatore algebrico di modelli ha il compito di tradurre il modello, descritto mediante il linguaggio di modellazione, in uno interpretabile dal solutore.

4.1.2 I linguaggi di modellazione

I linguaggi di modellazione sono veri e propri linguaggi ad alto livello. Sono quindi molto semplici da utilizzare, avvicinandosi al formalismo matematico, se non al linguaggio naturale.

L'utilizzo di una grammatica libera dal contesto, la definizione di variabili ed alias e la possibilità di aggiungere commenti, rendono i modelli descritti tramite un linguaggio di modellazione, facilmente interpretabili da parte di persone non esperte di programmazione, purché esperte di modellazione.

L'utente deve definire, tramite il linguaggio, il modello ed i dati, i quali possono anche risiedere in un database. Il vantaggio fondamentale dell'utilizzo di un linguaggio di modellazione è che il passaggio ad un'altra istanza dello stesso problema, richiede solamente la modifica dei dati, mentre il modello non cambia. D'altra parte, una diversa formulazione dello stesso problema richiede, in genere, la modifica di una o più famiglie di vincoli, spesso senza cambiare i dati. Il solutore rimane, in ogni caso, invariato.

Il solutore *glpsol* utilizza il linguaggio di modellazione GNU Math Prog [GNU04a] e un generatore algebrico interno. Il modello ed i dati passati a *glpsol* sono, di fatto, contenuti in due semplici file di testo. Il primo descrive il modello matematico, definendo dati, variabili, funzioni obiettivo e vincoli. Il secondo riporta i valori numerici dei dati che compaiono nel primo.

Queste caratteristiche di un linguaggio di programmazione matematica, permettono di apportare aggiustamenti successivi al modello in un ottica *what if*, per poi verificare in modo semplice come si modifica la soluzione, o meglio il processo che porta ad ottenere la soluzione.

4.2 GNU Linear Programming Kit (GLPK)

Per la nostra applicazione abbiamo deciso di utilizzare il pacchetto GLPK. La scelta di GLPK è dettata principalmente da un fattore economico: il fatto di essere *open source* riduce a zero il costo della licenza d'utilizzo. GLPK ha il vantaggio di utilizzare il linguaggio di modellazione GNU Math Prog, che permette di non definire il modello in maniera estesa ma di separare i dati dal modello vero e proprio.
GLPK può essere utilizzato come libreria esterna o programma stand alone.
Nel primo caso le chiamate alle librerie permettono con semplici sostituzioni il passaggio da una all'altra libreria. Nel secondo GLPK può utilizzare gli stessi formati usati da altri solutori commerciali e può tradurre un modello scritto nel linguaggio di modellazione Math Prog nei formati standard estesi MPS e CPLEX LP o in un file di testo[GNU04b]. Nella nostra applicazione, GLPK viene usato come programma *stand-alone* a cui non viene passata un'istanza per esteso, ma un modello definito in un primo file, e i dati corrispondenti in un secondo file correlato al primo. Entrambi i file sono scritti nel linguaggio Math Prog (vedi Appendice A).

4.2.1 Il linguaggio di modellazione GNU Math Prog

GNU Math Prog, o solamente Math Prog, è un sottoinsieme del linguaggio di programmazione matematica AMPL[AMP]. Math Prog permette di descrivere il modello in un modo molto intuitivo e naturale. Questo accelera la fase di scelta della formulazione per il problema, dato che consente di provarne molte in breve tempo.
Gli elementi base di un modello in Math Prog sono:

Insiemi : rappresentano il dominio dei valori sia dei parametri che delle variabili; possono essere numerici o costituiti da simboli rappresentati da stringhe di testo;

Parametri : sono i dati, eventualmente organizzati in vettori mono o pluridimensionali, i quali possono essere caricati anche da altri file;

Variabili : descrivono la soluzione, ed è il compito del solutore fissarne il valore al termine dell'esecuzione;

Vincoli : definiscono il problema, distinguendo tra soluzioni ammissibili e non ammissibili;

Funzione obiettivo : valuta le soluzioni generate stabilendo la migliore.

A questi elementi si aggiunge un nutrito insieme di funzioni predefinite, la possibilità di associare regole di validazione dei dati e, come ci si attende da un linguaggio ad alto livello, di inserire commenti in qualsiasi punto del testo. La grammatica è libera dal contesto o *context free*. Ogni istruzione è terminata da un punto e virgola: è quindi possibile spezzare un'istruzione complessa su più righe, permettendo anche di aggiungere spazi, tabulazioni o righe vuote, al fine di facilitare la lettura del modello stesso. Math Prog permette di definire dei valori di default per i dati, di utilizzare uno speciale formato tabulare per assegnare anche più di un parametro contemporaneamente e di utilizzare dei caratteri jolly per assegnare valori solo a una parte di un insieme o di un vettore. In questo modo risulta essere molto semplice l'inserimento, anche manuale, di un alto numero di dati.

4.2.2 Il solutore glpsol

Il solutore lineare presente nel pacchetto GLPK si chiama *glpsol*. È un'applicazione richiamabile solo da linea di comando, per la quale non esiste un'interfaccia grafica. Il suo utilizzo base richiede l'indicazione di uno o più file contenenti il modello ed i dati.
Durante il processo *glpsol* invia allo standard output un insieme di informazioni, tra queste i valori della soluzione LP e della migliore soluzione intera trovata via via. Al termine del processo, *glpsol* riporta il tempo impiegato e la dimensione della memoria utilizzata. È possibile intervenire, in ogni caso, sul comportamento di *glpsol* tramite alcuni parametri. Discutiamo nel seguito i più rilevanti per l'applicazione.

Limitarsi alla valutazione del rilassamento lineare. Il sapere a priori se un problema non è ammissibile ci permette di introdurre rilassamenti al modello, richiedere all'operatore di diminuire l'inammissibilità o di applicare qualche tecnica per ridurne la complessità. Inizialmente *glpsol* risolve il problema come un problema di Programmazione Lineare (PL). Se il modello è effettivamente di PL, il processo termina con la soluzione. Se invece si tratta di un modello di Programmazione Lineare Intera (PLI), si è di fatto risolto il rilassamento continuo del problema. In questo modo *glpsol* può verificare l'eventuale insoddisfacibilità ed eventualmente, determinare un *bound primale* del problema. Successivamente *glpsol*, basandosi sulla soluzione rilassata, determina una soluzione ottima del problema di PLI.

Esplorazione dell'albero di valutazione. Se lo scopo consiste nel cercare una soluzione ammissibile conviene esplorare l'albero di decisione in profondità (*depth first search*), in modo da trovare il prima possibile una soluzione ammissibile, anche se non particolarmente buona. Purtroppo anche

se il problema è ammissibile, questa tecnica non garantisce di trovare una soluzione intera nel tempo prefissato. Questo caso si presenta per istanze molto complesse e grandi, sia in termine di numero di vincoli che in termine di numero di variabili.

Limitazione del tempo di valutazione. In assenza di indicazioni, *glpsol* prosegue a risolvere il modello finché non trova la soluzione ottima o ne dimostra l'inammissibilità. Per un modello complesso questo può portare ad esecuzioni che durano ore, se non giorni. Questi tempi lunghi contrastano con l'esigenza di tempi di calcolo ridotti e di una forte interazione con l'operatore. Può quindi essere necessario limitare il tempo macchina dedicato alla soluzione del modello. Al termine del tempo prefissato possiamo quindi trovarci di fronte a tre casi:

- *glpsol* trova la soluzione ottima, o dimostra che non esiste soluzione ammissibile, eventualmente terminando prima del limite fissato;

- *glpsol* trova, nel tempo limite fissato, una soluzione ammissibile ed una stima della ottimalità di questa;

- *glpsol* non trova nessuna soluzione nel tempo limite fissato, il che non vuol dire che l'istanza sia inammissibile per il problema MIP.

Nel primo caso, a meno che l'operatore non voglia apportare delle modifiche, la procedura ha termine. Nei rimanenti casi si può modificare l'istanza rilassando qualche vincolo oppure guidando *glpsol* tramite il fissaggio di qualche variabile. Questo può portare a soluzioni non ottime, ma semplifica il problema permettendo di trovare una soluzione in termini più rapidi. Un'altra

possibilità consiste nel concedere più tempo al solutore. In tutti i casi è richiesta una nuova ricerca della soluzione.

4.3 Come viene generata una soluzione

Quando si lancia l'esecuzione del risolutore, si può specificare un tempo limite di esecuzione entro il quale verrà proposta la soluzione trovata, oppure omettere questo parametro, e lasciare al risolutore tutto il tempo necessario per arrivare alla soluzione ottima del problema. Naturalmente è anche possibile che non si riesca a trovare una soluzione, sia perché il tempo a disposizione è troppo poco (nel primo caso), sia perché può avvenire che il problema proprio non abbia soluzione.
Quando si assegna un tempo limite al solutore, entro questo periodo l'elaborazione fornisce una soluzione, la quale potrebbe non essere ottima. L'utente può memorizzare la schedulazione, e poi decidere se accettarla come soluzione del problema proposto, oppure tentare di elaborare nuovamente il problema allungando il tempo di calcolo del solutore, oppure tralasciando di indicare un tempo limite.
Nel caso in cui l'utente decida di eseguire *glpsol* senza definire un tempo limite di elaborazione, questo troverà la soluzione ottima del problema, ma il tempo di calcolo potrebbe essere inaccettabile per chi utilizza il sistema.

4.3.1 Caratteristica del file d'uscita

Il file della soluzione, che il risolutore, al termine dell'esecuzione, fornisce all'utente, è in formato testo, e comprende numerose informazioni, in buona parte ridondanti.

Le informazioni che più interessano al nostro caso, sono solo quelle che rappresentano le assegnazioni dei turni agli operatori, e quelle che indicano il valore di ogni funzione obiettivo.
Le variabili che indicano l'assegnamento dei turni corrispondono nel file della soluzione a stringhe del seguente formato:

x[<codice operatore>, <codice turno>, <giorno>]

Per ogni variabile, il file contiene una riga che contiene la stringa sopra descritta, un asterisco e tre numeri di cui il primo indica se il turno specificato, nel giorno indicato dal suo valore ordinale tra l'insieme dei giorni della schedulazione, viene assegnato all'operatore specificato (1) oppure no (0).
L'esempio successivo può chiarire meglio quando c'è l'assegnamento di un turno ad un operatore in uno specifico giorno:

```
x[1,mattina,1]      *      0      0      1
x[1,pomeriggio,1]   *      1      0      1
```

Queste due righe indicano il non assegnamento dell'operatore identificato col codice "1" al turno "mattina" nel primo giorno della schedulazione (prima riga), e l'assegnamento del turno "pomeriggio" all'operatore "1" nel giorno primo della schedulazione (seconda riga).
Se il valore è:

- 0: non viene creato l'assegnamento identificato dalla variabile;
- 1: viene creato l'assegnamento identificato dalla variabile.

Per estrarre dal file solo le variabili dell'assegnamento, ed i valori delle funzioni obiettivo, sono necessarie delle procedure eseguite in automatico dall'interfaccia che leggono e memorizzano il file prodotto dal risolutore (vedi cap.5).

4.4 Interfacciamento col sistema

Il solutore *glpsol* non è dotato di interfaccia grafica; l'esecuzione avviene richiamandolo da riga di comando. Questo metodo d'esecuzione non è consono ad utenti poco esperti quali quelli cui il sistema è rivolto. Per ovviare a questo inconveniente, l'interfaccia gestisce automaticamente il passaggio dei file di ingresso al risolutore, il suo avvio e il recupero della soluzione proposta, visualizzandola in modo chiaro ed intuitivo.

5. Interfaccia

In questo capitolo, si descriverà come l'interfaccia, posta idealmente al centro, gestisce l'intera architettura del sistema informativo. In particolare verranno presentati i componenti utilizzati per la connessione al database, le procedure per ricavare dal database le informazioni richieste, il meccanismo di comunicazione con il risolutore (il passaggio dei dati e dei criteri di decisione e il recupero della soluzione), la gestione del risultato in modo che esso possa essere inserito correttamente nella base di dati, e la visualizzazione delle soluzioni in maniera chiara e comprensibile anche ad utenti inesperti di programmazione matematica.

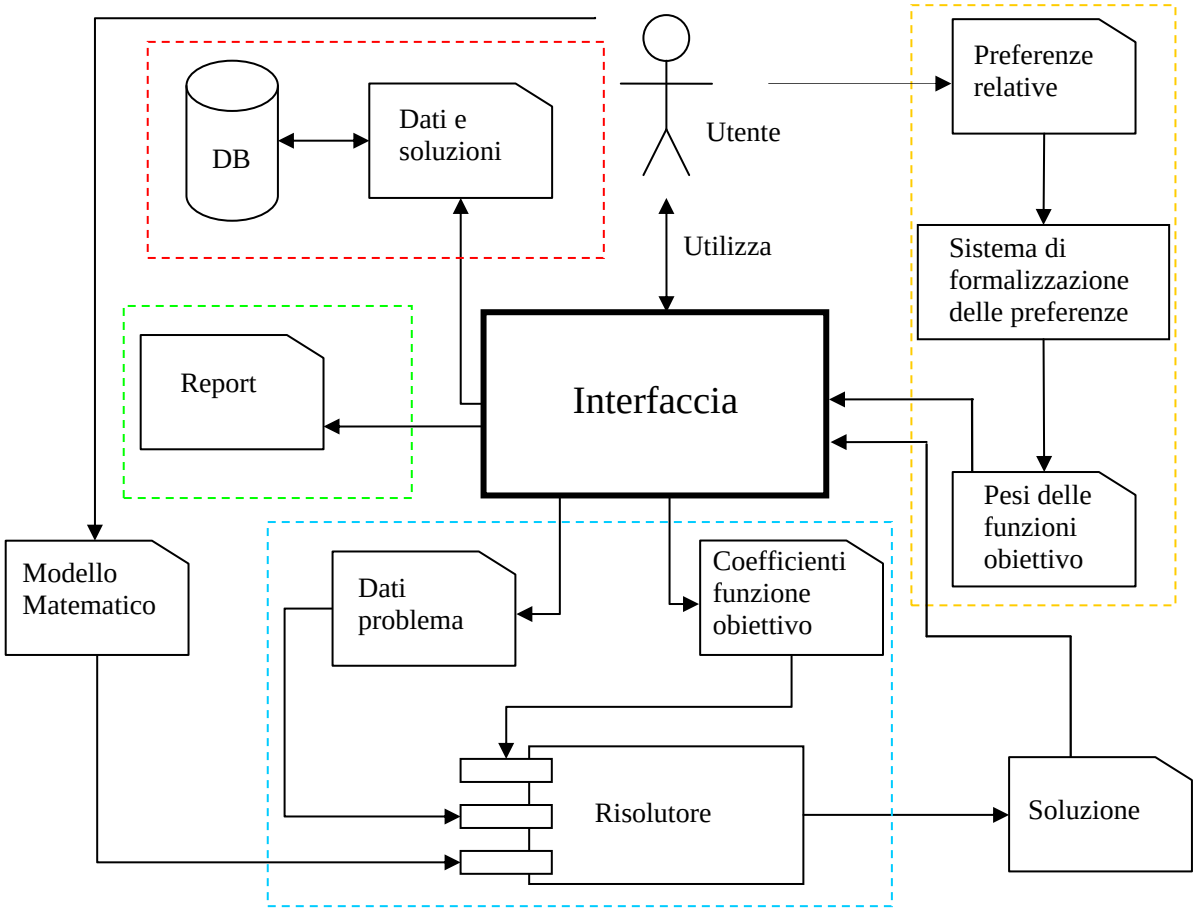

Fig. 5.1 Architettura di supporto alle decisioni.

Dalla fig. 5.1 viene evidenziato come l'interfaccia è il perno del sistema, gestendo le altre componenti indicate nei riquadri tratteggiati: il database (riquadro rosso), il sistema di formalizzazione delle preferenze (riquadro giallo), ed il risolutore (riquadro blu). Inoltre viene evidenziata la parte dei report sulle soluzioni che è possibile esportare da una finestra dell'interfaccia (riquadro verde).

5.1 Ambiente di sviluppo

L'ambiente di sviluppo utilizzato per creare l'interfaccia di gestione dell'intera architettura, è il Delphi. Delphi è stato creato dalla Borland, e si basa sul linguaggio di programmazione Object Pascal (una versione di Pascal orientata agli oggetti). È molto utilizzato per lo sviluppo di applicazioni desktop e applicazioni enterprise che utilizzano database, ma essendo uno strumento di carattere generico lo si può utilizzare per molti tipi di progetti.
Le componenti principali di Delphi sono il linguaggio, la Visual Component Library (VCL/CLX), e una facile e robusta connettività ai database, combinati con un potente Integrated Development Environment (IDE) e altri strumenti di supporto.

5.1.1 Linguaggio Object Pascal

Delphi si basa sull'Object Pascal, che è un linguaggio compilato, fortemente tipizzato, come d'altronde il Pascal, che supporta la progettazione strutturata e orientata agli oggetti. Offre un codice di facile lettura, una compilazione veloce e l'uso di più file *"unit"* per la programmazione modulare. L'Object

Pascal ha funzionalità speciali che supportano la struttura a componenti e l'ambiente Rapid Application Development (RAD) di Delphi.

5.1.2 Vantaggi di Delphi

Per la gestione del sistema informativo, Delphi offre delle funzionalità rapide e semplici per il collegamento al database costruito con SQL SERVER 2000 e garantisce un facile recupero delle informazioni tramite l'esecuzione di query SQL. Queste vengono passate come stringhe di caratteri a procedure e funzioni, il cui codice è recuperabile in rete da pacchetti distribuiti in modo *open source*. Inoltre è possibile reperire componenti per produrre report in formato EXCEL.

5.1.3 La gestione delle tabelle

L'interfaccia deve gestire tutto il database. In particolare, deve permettere all'utente di visualizzare informazioni, di modificarle, cancellarle o inserirne di nuove.
La schermata principale dell'interfaccia elenca quattro categorie di informazioni che è possibile gestire e che corrispondono ad altrettante tabelle principali:

1. tabella degli "Operatori";
2. tabella degli "Orari" contrattuali;
3. tabella dei "Turni";
4. tabella dei "Reparti".

In ogni finestra corrispondente, è anche possibile gestire i dati di tabelle minori che sono in relazione con la tabella principale; come spiegato nel cap.3, l'utente gestisce le informazioni tramite la visualizzazione del codice corrispondente, mentre a livello del database le relazioni tra tabelle saranno create per mezzo del valore dell'attributo ID.

5.2 Interfacciamento con il Database

Le operazioni per la gestione del database vengono effettuate tramite l'esecuzione di *stored procedure* apposite (Cap. 3), oppure tramite query SQL in grado di interrogare il database per reperire informazioni di ogni tipo. Per poter interrogare la base di dati, o per aggiornarla in qualche modo, bisogna prima di tutto creare un collegamento tra l'interfaccia ed il database stesso. Delphi mette a disposizione dei componenti che svolgono queste operazioni di connessione. I componenti utilizzati per la connessione al database sono tre:

- ADOConnection, stabilisce una connessione ad una fonte dati usando un OLE DB Provider.

- ADODataSet, consente di recuperare un insieme di record da una connessione TADOConnection collegata dalla proprietà Connection.

- DataSource, è un componente database non visuale con compiti di connessione tra un DataSet ed i componenti associati ai dati configurati in una scheda e che consentono la visualizzazione, lo spostamento e modifica dei dati associati al DataSet.

A questi tre componenti va aggiunto un quarto che esegue delle stored procedure su tabelle del database con il quale si è creata la connessione;

- ADOStoredProc, permette di eseguire una *stored procedure* che è presente nel database.

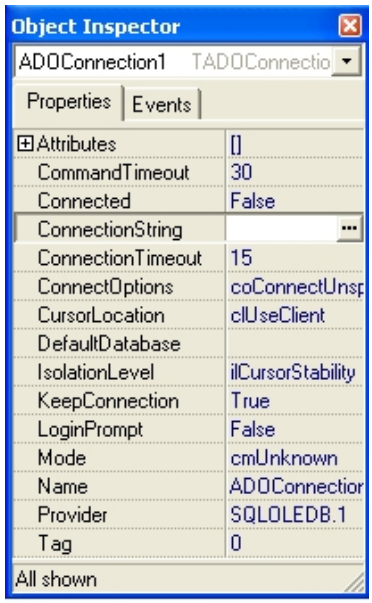

Fig. 5.1 ADOConnection

ADOConnection

Questo componente permette di stabilire la connessione col database; questa avviene inserendo una riga di comando, con i giusti riferimenti al nome del database ed al nome del server dove è memorizzata la base di dati, in una proprietà del componente (fig. 5.1).
La proprietà che gestisce la connessione col database è *ConnectionString*, e la stringa da inserire per creare la connessione con un database è:

Provider=SQLOLEDB.1;Integrated Security=SSPI;Persist Security Info=False;
Initial Catalog=<*nome database*>;Data Source=<*nome server*>

La proprietà *"Provider"* definisce la tipologia di provider a cui si sta creando una connessione, cioè un database OLE SQL.
Il nome della base di dati a cui ci si vuole collegare è definita con la stringa "Initial Catalog=<*nome database*>". Il nome del server dove è memorizzata la base di dati definita nella stringa precedente è "Data Source=<*nome server*>".

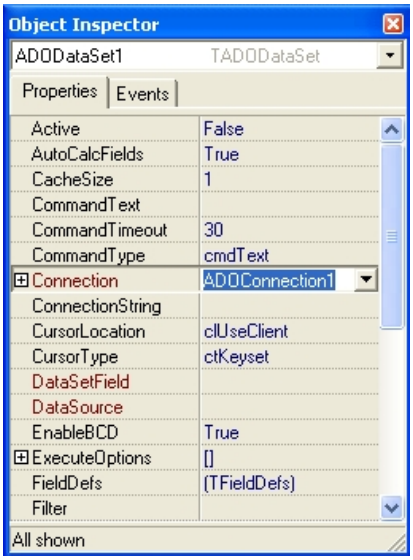

Fig. 5.2 ADODataSet

ADODataSet

Il componente adibito al recupero effettivo dei dati di un database, è ADODataSet.
Questo componente deve essere collegato all'elemento che crea la connessione, e questa associazione avviene impostando la sua proprietà *"Connection"* col nome di quel componente ADOConnection (vedi fig. 5.2).
Fatto ciò, si passa alla proprietà *CommandText* una stringa che identifica la query SQL da eseguire sul database.
Infine, per attivare il componente, si assegna alla proprietà *"Active"* il valore *true*. Tutte queste operazioni non devono essere statiche, ovvero eseguite una sola volta, ma ogni volta che si vuole interrogare il database. Quindi si deve

avere la possibilità di passare la query relativa e di recuperare il risultato in diverse occasioni. Per fare ciò bisogna:

1. disattivare il componente ADODataSet, ponendo Active:=false;

2. assegnare la query alla proprietà *CommandText*;

3. attivare il componente, ponendo Active:=true;

Il passaggio della stringa che rappresenta la query da eseguire viene descritto nella sezione 5.2.3 insieme con altre procedure per il controllo.

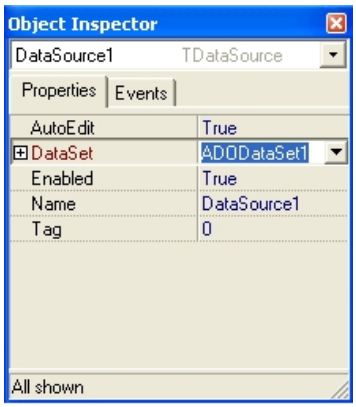

Fig. 5.3 DataSource

DataSource

Questo componente si collega all'ADODataSet per la gestione dei dati che vengono ricavati da interrogazioni al database. Per collegare i due componenti, bisogna impostare nella proprietà *"DataSet"* il nome del componente ADODataSet.

Ogni controllo correlato ai dati deve essere associato ad un componente DataSource, affinché i dati contenuti in quest'ultimo possano essere

visualizzati e conseguentemente manipolati. Allo stesso modo tutti gli ADODataSet debbono essere associati ad un componente DataSource affinché i relativi dati possano essere visualizzati e manipolati nei controlli associati alle informazioni e posti in una scheda.

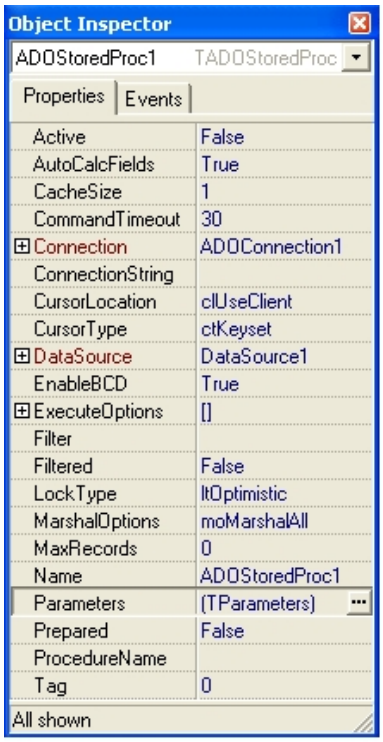

Fig. 5.4 ADOStoredProc

ADOStoredProc

Le *stored procedure*, sono memorizzate nel database (vedi cap. 3). Quindi, per eseguirne una, dopo aver creato una connessione con la base di dati, bisogna indicare il nome della *stored procedure* che si vuole eseguire, creando i vari parametri di ingresso come sono definiti nell'intestazione della procedura. Il componente adibito a questa operazione è ADOStoredProc.

Questo componente deve essere collegato, come i precedenti, impostando la proprietà *"Connection"* col nome del componente ADOConnection corrispondente. Le *stored procedure* utilizzate inseriscono, modificano e

cancellano delle informazioni nelle tabelle della base di dati; queste hanno dei parametri in ingresso che devono essere passati in modo automatico dall'interfaccia.
Per passare dei parametri ad una procedura, e per lanciarne l'esecuzione, si deve:

1. disattivare il componente ADODataSet, ponendo Active := false;

2. cancellare della proprietà *ProcedureName* il nome della *stored procedure* eseguita in precedenza, ponendo ProcedureName := ''; questo serve a impedire che il nome della procedura eseguita in precedenza,e quello della procedura che si vuole eseguire vengano concatenati, provocando un errore, perché la stringa risultante non corrisponde a nessuna memorizzata nel database.

3. assegnare alla proprietà *ProcedureName* il nome della procedura che si vuole eseguire.

4. Creare e passare il set dei parametri di ingresso alla procedura, Parameters.CreateParameters(<*nome parametro*>, <*tipo parametro*>, <*tipologia del parametro*>, <*dimensione in bit del parametro*>, <*nome variabile*>); con la quale si indica il nome del parametro, (stringa di caratteri), il tipo di dato del parametro (intero, carattere…), la tipologia del parametro, ovvero se si tratta di un parametro di ingresso (pdInput) di uscita (pdOutput), la dimensione del parametro in bit, e per ultimo il nome della variabile che contiene il valore del parametro da passare alla procedura. Il nome deve essere identico al nome del parametro dichiarato nella *stored procedure*. Se i parametri da passare alla *stored procedure* sono più di uno, essi dovranno essere creati in successione nello stesso ordine con cui sono stati dichiarati nella *stored procedure*.

Inoltre non ci devono essere comandi intermedi tra la creazione di due parametri della stessa procedura.

5. Disattivare il componente ADOStoredProc, ponendo Active := false;

6. eseguire la procedura col nome definito e con i parametri passati in modo corretto, chiamando la funzione ExecProc;

5.2.1 Esecuzione di query SQL

L'esecuzione di query SQL che interrogano il database, produce risultati che vengono gestiti dal componente ADODataSet. L'esecuzione di una query avviene utilizzando una procedura che, ricevuta in ingresso una stringa di caratteri, ne valuta l'effettiva correttezza, e se essa non produce errori la assegna alla proprietà *"CommandText"* di questo componente. Una volta passata la query SQL, viene attivato il componente per eseguire l'interrogazione sul database.

5.3 Interfacciamento col sistema di formalizzazione delle preferenze

Le funzioni obiettivo descritte nel cap.1 hanno associato un peso, che descrive la loro influenza nella scelta della soluzione. Un modo per definire i pesi è il metodo del confronto a coppie descritto nel cap. 6.
Questi pesi indicano l'importanza di un obiettivo rispetto ad un altro. Nella finestra dove viene creato il file di dati da passare al solutore, ci sono dei

componenti che visualizzano i valori per ogni criterio decisionale; l'utente può decidere se ricercare una soluzione per una schedulazione con quei pesi assegnati alle funzioni obiettivo, oppure decidere di modificarli secondo una logica ben precisa.

5.4 Interfacciamento col risolutore GLPK

L'interfaccia deve anche creare il file di ingresso al solutore *glpsol* (cap.4), lanciarne l'esecuzione e, una volta che il solutore ha finito l'elaborazione, recuperare dal file della soluzione le informazioni che devono essere tenute in considerazione, e memorizzarle nel database.
La memorizzazione della soluzione proposta da *glpsol* nelle rispettive tabelle del database avviene tramite *stored procedure*.

5.4.1 Creazione del file di dati per il risolutore

La unit *system* di Delphi, contiene procedure di sistema le quali permettono la creazione, la scrittura e la lettura di dati nei file di testo.
Il file di dati, che verrà passato come parametro di ingresso al solutore *glpsol*, viene creato da una finestra dell'interfaccia che contiene tutte le informazioni necessarie all'esecuzione del problema. Il file deve essere formattato, come descritto nella sezione 5.4.2. I dati necessari sono estratti dal database; la scrittura del file avviene in automatico con l'esecuzione di una sequenza di operazioni associate ad un evento. Il file del modello viene creato una volta per tutte, e posto nella stessa directory del file di dati.

5.4.2 File di dati

Il file di dati è strettamente correlato al file del modello matematico; i nomi dei parametri, delle variabili e delle funzioni obiettivo devono corrispondere a come sono scritte nel modello matematico.
Per la creazione di questo file dall'interfaccia, è sufficiente immettere le date di inizio e di fine del periodo di schedulazione: esso verrà scritto in automatico con l'ausilio di funzioni di sistema (le stesse usate per elaborare il file di uscita del risolutore). Sostanzialmente si tratta di scrivere riga per riga nel file stringhe di caratteri, le quali rappresentano i valori dei dati del problema. I dati necessari vengono ricavati interrogando le opportune tabelle nel database dall'esecuzione di query SQL.

5.4.3 Esecuzione del risolutore glpsol

Una volta che si è creato il file di dati, il suo nome viene passato come parametro di ingresso al risolutore insieme al file del modello matematico al risolutore. Come detto nel cap.4, il solutore *glpsol* per essere eseguito deve essere lanciato da riga di comando.
Per eseguire un'applicazione da riga di comando, in Delphi, è sufficiente una chiamata alla funzione API "WinExec" la cui sintassi è:

```
WinExec(CmdLine: PChar; CmdShow: Word)
```

Il primo parametro è il nome dell'eseguibile, il secondo la modalità di esecuzione del programma (ridotto a icona, in finestra massimizzata, etc.). Se nel parametro *CmdLine* non è incluso il percorso completo dell'applicazione, la funzione ricercherà il file eseguibile prima nella directory corrente, poi nella directory di Windows, nella directory "system" di Windows, e infine nella directory che ospita l'eseguibile della applicazione in esecuzione.

La stringa per eseguire il solutore glpsol passandogli come parametri i due file è:

glpsol --model <nome file modello> --data <nome file data> --output <nome file output>

Al termine dell'elaborazione il solutore fornirà il file di uscita col nome definito nella riga di comando; questo file deve essere elaborato.

5.4.4 Elaborazione del risultato

Le informazioni del file prodotto dal solutore devono essere lette per memorizzare nella base di dati solo quelle che effettivamente indicano il turno svolto da un certo operatore in un certo giorno, ignorando le altre.
Il file di uscita prodotto viene passato come parametro di ingresso ad una prima procedura, la quale legge tutte le righe e memorizza i dati utili in un primo file (il formato delle righe che interessano nel nostro caso è descritto nel cap. 4).
Questo viene passato a sua volta ad una seconda procedura che scandisce il valore delle variabili e le allinea ognuna su una riga differente, e le memorizza in un secondo file. Una terza procedura riceve questo file, e salva in un terzo file le righe corrispondenti all'assegnazione di un turno in un determinato giorno ad uno specifico operatore, cioè le variabili di valore 1 (vedi cap. 4).
Una quarta procedura riscrive in modo ordinato le informazioni che poi dovranno essere memorizzate nella base di dati.
Una quinta procedura, scandisce il file della soluzione generata, e salva in sei variabili il valore delle sei funzioni obiettivo calcolato dal risolutore.

La memorizzazione delle informazioni nelle tabelle del database avviene tramite l'esecuzione di *stored procedure*. Il passaggio dei valori dei parametri da inserire nelle tabelle viene svolto da una procedura, che legge riga per riga l'ultimo file elaborato, e separa il valore dell'operatore, quello del turno e quello del giorno.

5.5 Possibilità di modifica di una schedulazione

L'utente può definire i pesi dei criteri decisionali in base alle sue esigenze, avendo in questo modo la possibilità di scelta tra diversi scenari di risoluzione del problema. All'utente viene data anche l'opportunità di modificare una schedulazione esistente. Il verificarsi di situazioni di emergenza del personale in un certo periodo di tempo già elaborato in precedenza, impone all'utente la rielaborazione della schedulazione attiva per riassegnare i turni lavorativi agli operatori attivi per quel periodo di tempo. Nel farlo si devono tenere in considerazione gli assegnamenti fatti dall'elaborazione precedente, e cambiare quelli compresi tra le due nuove date.

La rielaborazione di una schedulazione esistente può essere calcolata sullo stesso periodo di quella precedente, oppure allungando il periodo di assegnamento.

Identifichiamo con *alpha* la data di inizio del nuovo periodo di elaborazione, e con *beta* la data di fine di questo periodo. Ovviamente *alpha* deve essere compresa tra la data di inizio della schedulazione esistente, e la data di fine, mentre la data di fine del nuovo periodo di elaborazione (*beta*) deve seguire *alpha*, ma può sia precedere sia seguire la fine della schedulazione precedente.

Si definiscono in generale tre periodi:

1. fra l’inizio della schedulazione di partenza e *alpha* - 1

2. fra *alpha* e *beta*.

3. fra il giorno successivo a *beta* e la data di fine della schedulazione esistente (periodo inesistente quando beta segue la data di fine schedulazione precedente).

Nei periodi che delimitano parti della schedulazione esistente, le assegnazioni relative al passato devono essere mantenute tali, mentre si andranno a modificare solo quelle comprese nel nuovo periodo di elaborazione.
Le interrogazioni da effettuare sulla base di dati per ricavare le informazioni inerenti ai turni assegnabili da scrivere in modo corretto nel file di dati, devono essere fatte sulla tabella “Turni_Ass” con il valore della data compresa tra *alpha* e *beta* per il periodo nuovo di valutazione, mentre per quanto riguarda le assegnazioni per il periodo della schedulazione esistente, invece, le query SQL devono essere svolte sulla tabella “Assegnazioni”, riportando le tuple che hanno la data compresa l’inizio e la fine di questo periodo.
Per riportare le informazioni sull’ora di inizio e di fine dei turni preferenziali di un operatore, si devono eseguire due query SQL; per il periodo che identifica la schedulazione esistente, il valore da assegnare sarà ‘25’, per segnalare che sono già stati fissati dei turni a degli operatori nei giorni indicati, con un orario di inizio e di fine ben precisi, mentre per il periodo compreso tra *alpha* e *beta*, relativo alla nuova schedulazione, si deve interrogare la tabella “Assegnazioni”.
Nel caso in cui si vuole partire dalla schedulazione esistente, e allungare il periodo di assegnazione dei turni, il valore di *beta* è maggiore della fine della schedulazione esistente. Le query SQL da eseguire sono le stesse di prima per

quanto riguarda le informazioni sui turni assegnabili a certi operatori; per quanto concerne la definizione dell'orario di inizio e di fine dei turni preferenziali nel periodo compreso tra *alpha* e *beta*, cioè quello che identifica il nuovo spazio di tempo della schedulazione da elaborare, si dovrà eseguire una query SQL sulla tabella "Assegnazioni", mentre per il periodo della schedulazione esistente compreso tra la data di inizio e *alpha* – 1 il valore da riportare sarà sempre di '25' per indicare nuovamente l'assegnazione fissata dei turni nella schedulazione precedente.
Per il periodo compreso tra la data di fine della schedulazione esistente e *beta*, le informazioni nel file di dati devono essere scritte come quando si vuole elaborare per la prima volta un nuovo periodo di schedulazione.

5.6 Esportare una schedulazione in Excel

Una volta memorizzate le informazioni di una schedulazione nella tabella apposita della base di dati, si dà all'utente la possibilità di visualizzarla in modo ordinato in una finestra; l'elemento che lo consente è una *StringGrid*, un componente base di Delphi, che può visualizzare dati in una tabella.
Sulle righe della tabella, ci sono i codici di tutti gli operatori, mentre sulle colonne ci sono le date di ogni giorno della schedulazione; ogni cella visualizza il codice del turno che l'operatore associato alla riga deve svolgere nel giorno associato alla colonna.
L'utente può esportare la visualizzazione della schedulazione nella *StringGrid* in un foglio di lavoro di Microsoft Excel; questa operazione viene svolta dal componente OLEExcel, che è un componente scaricato dal web [TP] (uno dei vantaggi di Delphi è che si possono trovare pacchetti *open source* per aumentarne le funzionalità), il quale dato in ingresso un componente *StringGrid*, esporta la tabella nelle celle di un foglio di lavoro Excel. Il codice

da eseguire per assegnare la tabella (*StringGrid*) al nome del componente OLEExcel è il seguente:

```
<nome componente OLEExcel>.CreateExcelInstance;
<nome componente OLEExcel>.StringGridToExcel(<nome StringGrid>);
<nome componente OLEExcel>.Visible := True;
```

La prima riga di codice crea un'istanza di Excel, nella seconda, invece, viene assegnato il nome del componente *StringGrid* al componente OLEExcel, mentre l'ultima riga rende visibile l'istanza di Excel nel foglio di lavoro.
L'esportazione in Excel serve all'utente anche per scegliere la soluzione da rendere attiva.

5.7 Reportistica

Una sezione importante nell'interfaccia è dedicata alla reportistica di alcune informazioni relative al lavoro degli operatori.
Utilizzando il componente per esportare delle informazioni contenute in una *StringGrid* in un foglio di lavoro Excel, si ha la possibilità di consultare dei dati relativi a specifici lavoratori. Queste informazioni, ottenute eseguendo query SQL su tabelle del database, costituiscono un metodo per presentare dati in modo fomattato e organizzato.
Ci sono tipi di reportistica, riferiti rispettivamente:

1. ad una schedulazione memorizzata nel database;

2. al lavoro effettivamente svolto dagli operatori in un determinato intervallo di tempo.

In entrambi i casi il report fornisce le informazioni relative a un operatore scelto dall'utente, il numero di riposi, di ferie, di malattie e delle ore di straordinario effettuate in dato periodo di lavoro.
Ciascuna di queste informazioni è visualizzata in uno specifico componente dell'interfaccia; allo stesso tempo questi dati vengono memorizzati in una *StringGrid* invisibile, la quale verrà passata al componente per l'esportazione di tabelle in fogli di lavoro Excel.

6. Selezione dell'alternativa preferita

In questo capitolo, viene definito il modo in cui si cambiano le sei funzioni obiettivo elencate nel cap. 1 in una sola, utilizzata dal risolutore. Si spiega inoltre come il solutore possa fornire soluzioni diverse, ricevendo in ingresso lo stesso file di dati, ma diversi pesi per le funzioni obiettivo. Un utente esperto può cambiare a piacere il valore dei criteri decisionali. Tuttavia è possibile suggerire all'utente un valore iniziale calcolato in base ai principi dell'Analisi Gerarchica [AHP80] secondo gli indici di importanza relativa definiti dalla semantica di Saaty, e con il metodo del *confronto a coppie*. Nelle sezioni successive si elencano le operazioni eseguite per assegnare ad ogni singolo criterio decisionale un valore sensato rispetto a quello degli altri criteri.

6.1 La funzione obiettivo

I solutori lineari operano con una sola funzione obiettivo. Anche gli algoritmi euristici tendono ad ottimizzare una sola funzione obiettivo, anche se possono tenerne presenti varie. Questi comportamenti determinano l'opportunità di raggruppare in qualche modo tutte le funzioni obiettivo descritte nel cap.1.
Introducendo opportuni pesi per esprimere l'importanza relativa delle funzioni obiettivo, possiamo riunirle in un unica funzione che è la loro combinazione lineare:

$$\min Obj = \tilde{w}_1 z_1 + \tilde{w}_2 z_2 + \tilde{w}_3 z_3 + \tilde{w}_4 z_4 + \tilde{w}_5 z_5 + \tilde{w}_6 z_6$$

dove $\tilde{w}_1 \ldots \tilde{w}_6$ indicano rispettivamente i pesi delle singole funzioni obiettivo calcolati come descritto nelle sezioni successive.

6.2 Definizione dei pesi

Nel file dati che si passa come parametro di ingresso al solutore *glpsol*, vengono definiti dei pesi per i criteri decisionali sui quali si deve fornire una soluzione ammissibile del problema. Questi criteri definiscono alcune grandezze che esprimono la qualità della schedulazione proposta all'utente. In base al valore che il solutore assegna al termine dell'elaborazione alle variabili associate ai parametri, l'utente deciderà se la soluzione suggerita può essere accettata, oppure se procedere con la rielaborazione del problema in base a valori dei pesi differenti.
I criteri decisionali che si decide di valutare per la risoluzione del problema che si vuole proporre sono:

- Ore Jolly; l'operatore definito come "jolly" ha un contratto di assunzione diverso dagli altri: funge da aiuto nelle situazioni di emergenza e gli vengono assegnati i turni lavorativi che sono scoperti perché gli operatori fissi sono in ferie, o in malattia e che non si possono coprire con straordinari. Il costo del lavoro di questi operatori, è maggiore di altri, in quanto non sono assunti in pianta stabile nell'istituto.

- Ore Straordinarie; le ore in eccesso che un operatore può lavorare rispetto alle ore di lavoro che prevede il suo contratto di assunzione per il periodo di una settimana, hanno un costo remunerativo maggiore.

- Ore Non Lavorate; sono le ore lavorative che un operatore non effettua in base al valore totale delle ore che il suo contratto di assunzione prevede per il periodo di una settimana. Costituiscono uno spreco di manutenzione, visto che vengono comunque pagate.

- Violazione "3+1"; gli operatori classificati come lavoratori "3+1"; per contratto dovrebbero effettuare tre giorni consecutivi di lavoro, ed un turno di riposo nel quarto giorno. La violazione di questa turnazione fissa per questa categoria di lavoratori va minimizzata.

- Violazione Pattern; lo scostamento dell'orario di inizio e di fine del turno lavorativo al quale viene assegnato un operatore, dall'orario di inizio e di fine del suo turno preferenziale va minimizzato.

- Violazione Reparto; l'assegnamento di un operatore al di fuori del proprio reparto di appartenenza va minimizzato.

Per prima cosa, tutte le funzioni obiettivo vengono riportate alla stessa unità di misura, in modo da confrontare grandezze analoghe.
Le funzioni considerate sono o durate (misurate in ore) o numeri puri (numero di violazioni di un vincolo). In questo caso, si moltiplica il valore della funzione obiettivo per la durata del turno che costituisce una soluzione (il turno eseguito fuori reparto o in violazione al riposo del quarto giorno). Assumiamo quindi che z_i sia sempre espresso in ore (i = 1,..,6).
L'utente che sottopone il problema al solutore *glpsol*, assegna un valore di importanza a questi pesi, in base a quella che ritiene essere la loro importanza relativa.

Indice di importanza relativa	Definizione
$\frac{w_1}{w_2} = 1$	L'obiettivo 1 e l'obiettivo 2 hanno eguale peso
$\frac{w_1}{w_2} = 3$	L'obiettivo 1 ha peso debolmente maggiore dell'obiettivo 2
$\frac{w_1}{w_2} = 5$	L'obiettivo 1 ha notevole importanza dell'obiettivo 2
$\frac{w_1}{w_2} = 7$	L'obiettivo 1 ha dimostrata rilevanza dell'obiettivo 2
$\frac{w_1}{w_2} = 9$	L'obiettivo 1 ha assoluta rilevanza dominante dell'obiettivo 2
$\frac{w_1}{w_2} = 2,4,6,8$	Valori intermedi tra due giudizi adiacenti (valori di compromesso)

Tab. 6.1 Scala semantica di Saaty

6.2.1 Come si calcolano i pesi

La ridefinizione dei pesi in modo adeguato non può avvenire dando sei valori casuali: ogni criterio decisionale è più o meno importante di un altro. I valori dei pesi non sono importanti: ciò che conta sono i loro rapporti. Quindi si è scelto di normalizzare i valori tra 0 e 1, ovvero di dividerli per la loro somma.
Il procedimento per assegnare un valore razionale ad ognuno di questi pesi viene effettuato col metodo dell'Analisi Gerarchica, chi si basa su *confronti a coppie*.
Le fasi per il calcolo dei valori dei criteri decisionali definiti nella sezione 6.2, sono le seguenti:

1. il decisore deve confrontare i criteri a coppie, e definire il grado di importanza relativa dei due criteri. I valori di importanza vengono definiti in base a valori della scala semantica di Saaty (tab. 6.1).

2. Dai confronti a coppie deriva una matrice quadrata il cui elemento a_{ij}, evidenzia il peso dell'obiettivo *i* rispetto all'obiettivo *j*. Nel nostro caso,

p1	=	Criterio Ore Jolly
p2	=	Criterio Ore Straordinarie
p3	=	Criterio Ore non lavorate
p4	=	Criterio Violazione "3+1"
p5	=	Criterio Violazione pattern
p6	=	Criterio Violazione reparto

e la matrice ottenuta consultando esperti nel campo è risultata essere quella in tab.6.2. Si notano alcune proprietà evidenti di questa matrice quadrata; i valori sulla diagonale sono tutti 1, e la prima riga è il reciproco della prima colonna.

Questa matrice però non ha la proprietà di coerenza fra colonne, cioè le colonne non sono proporzionali l'una rispetto all'altra, come si può intuire meglio dall'esempio:

il valore dell'elemento $a_{ij} \times a_{jk} \neq a_{ik}$

$a_{ij} = 5; \quad a_{jk} = 5; \quad a_{ik} = 7$

$5 * 5 = 25 \neq 7$

	p1	p2	p3	p4	p5	p6
p1	1	5	7	7	9	9
p2	1/5	1	5	5	7	7
p3	1/7	1/5	1	1	3	3
p4	1/7	1/5	1	1	3	3
p5	1/9	1/7	1/3	1/3	1	3
p6	1/9	1/7	1/3	1/3	1/3	1

Tab. 6.2 Matrice quadrata

3. Quindi se la matrice fosse coerente avrebbe n-1 autovalori nulli. Se è vicina alla coerenza, ha un autovalore grosso in modulo e n-1 piccoli. Si calcolano gli autovalori della matrice, e si considera quello il cui valore in modulo si discosta in modo evidente da tutti gli altri. L'idea è di costruire la matrice coerente più simile a quella data.

4. Dall'autovalore considerato si calcola l'autovettore. I singoli valori ottenuti vengono normalizzati a 1, in questo modo otteniamo un vettore di valori da associare ai pesi delle funzioni obiettivo.

Il risultato della matrice di tabella 6.2 è il vettore definito in tabella 6.3.

Criterio	Valore calcolato
Ore Jolly	$\tilde{w}_1 = 0.5321$
Ore Straordinarie	$\tilde{w}_2 = 0.2466$
Ore Non Lavorate	$\tilde{w}_3 = 0.0752$
Violazione "3+1"	$\tilde{w}_4 = 0.0752$
Violazione Pattern	$\tilde{w}_5 = 0.0420$
Violazione Reparto	$\tilde{w}_6 = 0.0288$

Tab. 6.3 Valori criteri

La tabella 6.3 mostra i pesi ottenuti per i sei criteri decisionali desiderati. Le ore lavorative di un operatore jolly influiscono in modo determinante sul

calcolo totale del costo di una schedulazione; ciò implica che si deve cercare di minimizzare il loro utilizzo nell’assegnamento dei turni. Anche le ore straordinarie hanno un costo maggiore delle ore non lavorate, quindi da questi valori si sottolinea il fatto che si preferisce far rientrare in una schedulazione un operatore in più il quale non lavora per tutto il numero delle ore lavorative definite nel suo contratto di assunzione, piuttosto che far fare delle ore di straordinario ad un altro operatore.

All’utente però viene data la possibilità di modificare i pesi, definendo a proprio piacere quale criterio possa essere il più importante, e quale il meno importante della lista.

6.3 Soluzioni diverse per pesi diversi

Elaborando lo stesso problema con gli stessi dati, ma con pesi diversi per i criteri differenti, si possono ottenere schedulazioni differenti, in quanto cambia la definizione delle funzioni obiettivo e quindi la soluzione ottima. Se per esempio, nella prima schedulazione il criterio delle ore degli operatori jolly è quello col peso maggiore, si cercherà di non utilizzare lavoratori appartenenti a questa categoria; al contrario, se nella seconda schedulazione le ore jolly hanno un peso che si avvicina a quello degli altri criteri, l’utilizzo di operatori jolly può essere ammesso, ed il costo totale di questa schedulazione, per il decisore può essere accettabile.

7. Conclusioni

Il lavoro svolto riguarda la costruzione di un sistema informativo per la gestione dei turni lavorativi della casa di riposo di Ghedi, ma l'intenzione è quella di poter applicare il sistema anche ad altre strutture analoghe. Il problema fondamentale affrontato è quello di integrare la gestione delle informazioni nella struttura (contenute in un database), con la possibilità di ottimizzare i turni di lavoro svolti dagli operatori (attraverso un risolutore di PLI) e con la possibilità di generare soluzioni che privilegino aspetti diversi della gestione (la presenza di ore straordinarie, piuttosto che la violazione di vincoli contrattuali o sociali) e scegliere la più soddisfacente. Il tutto attraverso in un'interfaccia semplice e di facile utilizzo.

Innanzitutto si è creato un database adeguato per la memorizzazione delle informazioni necessarie, e che risulti avere il minor numero possibile, se non la totale mancanza di dati ridondanti.

Si è poi corretto e completato un modello matematico preesistente [MM04] in cui vengono definiti nel linguaggio AMPL gli obiettivi da raggiungere ed i vincoli da rispettare secondo le norme lavorative previste dalla legge italiana, e secondo alcune esigenze lavorative particolari della struttura di Ghedi, ma tipiche della maggior parte delle strutture analoghe.

Si è poi data la possibilità di assegnare dei pesi alle singole funzioni obiettivo, per attribuire un grado di importanza ad un obiettivo rispetto ad un altro. La scelta razionale di questi pesi non è semplice; inizialmente sono stati calcolati dei valori col metodo del *confronto a coppie*, per dare all'utente un punto di partenza opportuno.

Un'adeguata interfaccia gestisce il tutto. Tramite l'esecuzione di query SQL e *stored procedure* si gestiscono i dati nel database, si visualizzano specifiche informazioni, e soprattutto si può modificarle, cancellarle ed inserirne di

nuove. Una specifica finestra permette di definire l'intervallo temporale per cui si richiede una nuova schedulazione e quindi controllare le informazioni che verranno passate al solutore *open source* GLPK. Questo fornisce l'assegnazione ottimale dei turni per il periodo richiesto agli operatori, in base al modello matematico ed al file di dati creato.

L'interfaccia gestisce in modo automatico il passaggio dei file al solutore, e il recupero, una volta finita l'elaborazione, della soluzione proposta. La sezione che interessa più di tutte al decisore dei turni, è quella che dà la possibilità di modificare una schedulazione esistente. L'utente può modificare le assegnazioni dei turni su un periodo di tempo che contiene già assegnamenti dell'elaborazione precedente, in modo da porre rimedio a emergenze che si sono verificate dopo la prima risoluzione per quel periodo.

L'intera architettura realizzata è impostata su regole lavorative proprie della casa di riposo di Ghedi, e l'interfaccia è stata creata per la visualizzazione e la gestione del database creato per questo istituto. I concetti fondamentali, definiti nel modello matematico, e le tecniche di connessione e gestione delle informazioni di una base di dati, sono però sufficientemente elastiche da permettere l'utilizzo di questo sistema anche per realtà lavorative diverse che presentano problemi diversi.

Appendice A

```
param cardOPERATORI;
param cardTURNI;
param cardGIORNI;
param cardREPARTI;
param cardOP_NOTTURNI;
param m;
param s;

#Operatori, turni ed orizzonte temporale
set OPERATORI;
set Jolly within OPERATORI;
set TURNI;
set GIORNI := 1..cardGIORNI;
set MESI := 1..m;
set GIORNI_MESE{MESI} within GIORNI;
set settimane := 1..s;
set GIORNI_SETTIMANA{settimane} within GIORNI;
set OP_NOTTURNI within OPERATORI;

#Orario contrattuale, ferie e malattie
param ORE_MIN{OPERATORI} >= 0;
param ORE_MAX{OPERATORI, MESI} >= 0;
param ORE_STRAORD{OPERATORI} >= 0;
set Ferie{OPERATORI} within GIORNI;
set Malattie{OPERATORI} within GIORNI;
set OPERATORI_TREPIUUNO within OPERATORI;

#Turni e carico di lavoro
set T_LAVORATIVI within TURNI;
set T_RIPOSO within TURNI;
set T_FERIE within TURNI;
set T_MALATTIA within TURNI;
set T_NOTTURNI within T_LAVORATIVI;
param durata_lavorativi{T_LAVORATIVI} >= 0, <=8;
param durata_malattia_riposo{OPERATORI} >= 0, <=8;
param ora_inizio{TURNI} >= 0, <= 24;
param ora_fine{TURNI} >= 0, <= 24;
set TURNI_ASSEGNABILI{OPERATORI, GIORNI} within TURNI default TURNI;
param carico{T_LAVORATIVI, GIORNI} >= 0;
param max_eccesso_turni_notturni;
param media_notturni :=
(sum{t in T_NOTTURNI, l in GIORNI} carico[t,l])/cardOP_NOTTURNI;


#Reparti
set REPARTI;
```

```
set TURNI_REPARTO{REPARTI} within T_LAVORATIVI;
set OPERATORI_REPARTO{REPARTI} within OPERATORI;
param caricoReparto{r in REPARTI, l in GIORNI} :=
sum{j in TURNI_REPARTO[r]} carico[j,l];

param Dimensione_Reparto{r in REPARTI} :=
sum{i in OPERATORI_REPARTO[r]} 1;

#Variabili e parametri
var x{OPERATORI, TURNI, GIORNI} binary;
param giorni_consecutivi_lavorati{OPERATORI} integer >= 0, <= 6;
set ultimo_turno_effettuato{OPERATORI} within TURNI;
param ore_lavorate_prima_settimana{OPERATORI} >= 0 default 0;
var straordinarie{OPERATORI, MESI} >= 0;
var sottoordinarie{OPERATORI, settimane} >= 0;
var violazione_trepiuuno{OPERATORI_TREPIUUNO, 1..cardGIORNI-3} binary;
var violazione_reparto{REPARTI, GIORNI} integer >= 0;

param pesoOreJolly default 0.5321;
param pesoOreStraordinarie default 0.2466;
param pesoOreNonLavorate default 0.0752;
param pesoViolazioneTrepiuUno default 0.0752;
param pesoScostamentoPattern default 0.0420;
param pesoScostamentoReparto default 0.0288;
param delta default 100;

var TotaleOreStraordinarie;
var TotaleOreNonLavorate;
var TotaleViolazioneTrepiuUno;
var TotaleScostamentoPattern;
var TotaleScostamentoReparto;
var TotaleOreJolly;

#pattern
param oraInizioPreferenziale{OPERATORI, GIORNI} default 0, >= 0, <= 25;
param oraFinePreferenziale{OPERATORI, GIORNI} default 0, >= 0, <= 25;

param ScostamentoPattern { i in OPERATORI, j in T_LAVORATIVI, l in GIORNI}
:= (if (oraInizioPreferenziale[i,l] <> 25 and oraFinePreferenziale[i,l] <> 25)
    then abs(oraInizioPreferenziale[i,l] - ora_inizio[ j ]) +
         abs(oraFinePreferenziale[i,l] - ora_fine[j])
    else 0);

minimize costo: (pesoOreJolly * TotaleOreJolly) +
                (pesoOreStraordinarie * TotaleOreStraordinarie) +
                (pesoOreNonLavorate * TotaleOreNonLavorate) +
                (pesoViolazioneTrepiuUno * TotaleViolazioneTrepiuUno) +
                (pesoScostamentoPattern * TotaleScostamentoPattern) +
                (pesoScostamentoReparto * TotaleScostamentoReparto);
```

```
#obiettivo O.1
s.t. obiettivo_TotaleOreJolly:
TotaleOreJolly = sum{i in Jolly, t in T_LAVORATIVI, l in GIORNI} (x[i,t,l] *
durata_lavorativi[t]);

#obiettivo O.2
s.t. obiettivo_OreStraordinarie:
TotaleOreStraordinarie = sum{i in OPERATORI, k in MESI} straordinarie[i,k];

#obiettivo O.3
s.t. obiettivo_OreNonLavorate:
TotaleOreNonLavorate = sum{i in OPERATORI, t in settimane} sottoordinarie[i,t];

#obiettivo O.4
s.t. obiettivo_ViolazioneTrepiuUno:
TotaleViolazioneTrepiuUno = sum{i in OPERATORI_TREPIUUNO, l in
1..cardGIORNI-3} (violazione_trepiuuno[i,l]);

#obiettivo O.5
s.t. obiettivo_ScostamentoPattern:
TotaleScostamentoPattern = sum{i in OPERATORI, j in T_LAVORATIVI, l in
GIORNI} (x[i,j,l] * ScostamentoPattern[i,j,l]);

#obiettivo O.6
s.t. obiettivo_ScostamentoReparto:
TotaleScostamentoReparto = sum{r in REPARTI, l in GIORNI}
(violazione_reparto[r,l]);

#vincolo V.1
s.t. soddisfa_carico{j in T_LAVORATIVI, l in GIORNI}:
sum{i in OPERATORI} x[i,j,l] = carico[j,l];

#vincolo V.2
s.t. turni{i in OPERATORI, l in GIORNI}: sum{j in TURNI} x[i,j,l] = 1;

#vincolo V.3
s.t. assegnabile{i in OPERATORI, l in GIORNI, j in T_LAVORATIVI: j not in
TURNI_ASSEGNABILI[i,l]}: x[i,j,l] = 0;

#vincolo V.3bis
s.t. non_assegnabile{i in OPERATORI diff OP_NOTTURNI, l in GIORNI, j in
T_NOTTURNI}: x[i,j,l] = 0;

#vincolo V.4
s.t. riposo{i in OPERATORI, l in {2..cardGIORNI-6}}:
sum{t in {0..6}, j in TURNI: j not in T_RIPOSO} x[i,j,l+t] <= 6;

#vincolo V.4'
s.t. riposo_f{i in OPERATORI}:
sum{t in {0..6-giorni_consecutivi_lavorati[i]}, j in TURNI: j not in T_RIPOSO}
x[i,j,t+1] <= 6 - giorni_consecutivi_lavorati[i];
```

```
#vincolo V.5
s.t. distanza{i in OPERATORI, l in {1..cardGIORNI-1}, j1 in T_LAVORATIVI, j2 in
T_LAVORATIVI: 24 - ora_fine[j1] + ora_inizio[j2] < 11}: x[i,j1,l] + x[i,j2,l+1] <= 1;

#vincolo V.5'
s.t. distanza_f{i in OPERATORI, l in GIORNI, j in T_LAVORATIVI, k in
ultimo_turno_effettuato[i] : 24 - ora_fine[k] + ora_inizio[j] < 11}: x[i,j,l] = 0;

#vincolo V.6
s.t. ferie{i in OPERATORI, l in GIORNI: l in Ferie[i]}:
sum{j in T_FERIE union T_RIPOSO} x[i,j,l] = 1;

#vincolo V.7
s.t. malattie{i in OPERATORI, l in GIORNI: l in Malattie[i]}:
sum{j in T_MALATTIA union T_RIPOSO} x[i,j,l] = 1;

#vincolo V.8
s.t. nmalattie{i in OPERATORI, l in GIORNI, j in T_MALATTIA: l not in Malattie[i]}:
x[i,j,l] = 0;

#vincolo V.9
s.t. lim_max_settimana{i in OPERATORI, t in {2..s}}:
sum{l in GIORNI_SETTIMANA[t]} ((sum{j in T_LAVORATIVI} durata_lavorativi[j] *
x[i,j,l]) + (sum{j in T_FERIE union T_MALATTIA} durata_malattia_riposo[i] * x[i,j,l]))
<= 48;

#vincolo V.9'
s.t. lim_max_prima_settimana_f{i in OPERATORI}:
sum{l in GIORNI_SETTIMANA[1]} ((sum{j in T_LAVORATIVI} durata_lavorativi[j] *
x[i,j,l]) + (sum{j in T_FERIE union T_MALATTIA} durata_malattia_riposo[i] * x[i,j,l]))
+ ore_lavorate_prima_settimana[i] <= 48;

#vincolo V.10
s.t. riposi{i in OPERATORI, k in MESI}:
sum{l in GIORNI_MESE[k], j in T_RIPOSO} x[i,j,l] >= 5;

#vincolo V.12
s.t. distribuisci_turni_notturni{i in OP_NOTTURNI}:
sum{t in T_NOTTURNI, l in GIORNI} x[i,t,l] <= media_notturni * (1 +
max_eccesso_turni_notturni);

#vincolo V.13
s.t. lim_mese{i in OPERATORI, k in MESI}:
sum{l in GIORNI_MESE[k]} ((sum{j in T_LAVORATIVI} durata_lavorativi[j] * x[i,j,l])
+ (sum{j in T_FERIE union T_MALATTIA} durata_malattia_riposo[i] * x[i,j,l])) <=
ORE_MAX[i,k] + straordinarie[i,k];

#vincolo V.14
s.t. lim_min_settimana{i in OPERATORI, t in {2..s}}:
```

```
sum{l in GIORNI_SETTIMANA[t]} ((sum{j in T_LAVORATIVI} durata_lavorativi[j] *
x[i,j,l]) + (sum{j in T_FERIE union T_MALATTIA} durata_malattia_riposo[i] * x[i,j,l]))
>= ORE_MIN[i] - sottoordinarie[i,t];

#vincolo V.14'
s.t. lim_min_prima_settimana{i in OPERATORI}:
sum{l in GIORNI_SETTIMANA[1]} ((sum{j in T_LAVORATIVI} durata_lavorativi[j] *
x[i,j,l]) + (sum{j in T_FERIE union T_MALATTIA} durata_malattia_riposo[i] * x[i,j,l]))
+ ore_lavorate_prima_settimana[i] >= ORE_MIN[i] - sottoordinarie[i,1];

#vincolo V.15
s.t. seq3p1{i in OPERATORI_TREPIUUNO, l in {2..cardGIORNI-3}}:
sum{t in {0..3}, j in TURNI: j not in T_RIPOSO} x[i,j,l+t] <= 3 +
violazione_trepiuuno[i,l];

#vincolo V.15'
s.t. seq3p_f{i in OPERATORI_TREPIUUNO}:
sum{t in {0..3-giorni_consecutivi_lavorati[i]}, j in TURNI: j not in T_RIPOSO}
x[i,j,t+1] <= 3 - giorni_consecutivi_lavorati[i]  + violazione_trepiuuno[i,1];

#vincolo V.16
s.t. rep_pref{r in REPARTI, l in GIORNI}:
sum{i in OPERATORI_REPARTO[r], j in TURNI_REPARTO[r]} x[i,j,l] >=
min(Dimensione_Reparto[r], caricoReparto[r,l]) - violazione_reparto[r,l];

s.t. terna{i in OPERATORI, l in {1..cardGIORNI-2}, j1 in T_LAVORATIVI, j2 in
T_LAVORATIVI, j3 in T_LAVORATIVI: 24 - ora_fine[j1] + ora_inizio[j3] + (24 -
durata_lavorativi[j2]) < 22 and 24 - ora_fine[j1] +   ora_inizio[j2] < 11} :
x[i,j1,l] + x[i,j3,l+2] <= 2 * (1 - x[i,j2,l+1]);

end;
```

```
#########################################################
/*                  data inizio periodo     01/11/05                */
/*                  data  fine  periodo     30/11/05                */
#########################################################

/* Definizione valori dei Pesi: */

param pesoOreJolly := 0.5321;
param pesoOreStraordinarie:= 0.2466;
param pesoOreNonLavorate := 0.0752;
param pesoViolazioneTrepiuUno := 0.0752;
param pesoScostamentoPattern := 0.042;
param pesoScostamentoReparto := 0.0288;


/* set OPERATORI :
*  codice       -    nome
*  1            -    oper1
*  2            -    oper2
*  3            -    oper3
*  4            -    oper4
*  5            -    oper5
*  6            -    oper6
*  7            -    oper7
*/

set OPERATORI :=  1 2 3 4 5 6 7;

set OP_NOTTURNI := 6;

set Jolly := 7;

/* n°operatori */
param cardOPERATORI := 7;

/* n°operatori notturni */
param cardOP_NOTTURNI := 1;

/* set TURNI :
*  codice       -    nome
*  ferie        -    ferie
*  malattia     -    malattia
*  riposo       -    riposo
*  turno1       -    mattina
*  turno3       -    sera
*  turno4       -    notte
*  turno5       -    pomeriggio
*/

set TURNI :=  ferie malattia riposo turno1 turno3 turno4 turno5;
```

```
set T_NOTTURNI := turno3;

/* definizione del valore in percentuale della distribuzione dei turni notturni */
param max_eccesso_turni_notturni := 0.5;

/* n°turni */
param cardTURNI := 7;

set REPARTI :=  reparto1 reparto2;

/* n°reparti */
param cardREPARTI := 2;

/* n°giorni nel mese */
param cardGIORNI := 30;

/* n°mese */
param m := 1;

/* n°settimane nel mese */
param s := 5;

/* definizione giorni del mese */
set GIORNI_MESE[1] :=  1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21
22 23 24 25 26 27 28 29 30;

/* definizione dei giorni delle settimane nel mese */
set GIORNI_SETTIMANA[1] :=  1 2 3 4 5 6;
set GIORNI_SETTIMANA[2] :=  7 8 9 10 11 12 13;
set GIORNI_SETTIMANA[3] :=  14 15 16 17 18 19 20;
set GIORNI_SETTIMANA[4] :=  21 22 23 24 25 26 27;
set GIORNI_SETTIMANA[5] :=  28 29 30;

/* definizione ore min lavorate dagli operatori */
param ORE_MIN :=  [1] 42 [2] 42 [3] 42 [4] 42 [5] 42 [6] 42 [7] 42;

/* definizione ore max lavorate dagli operatori */
param ORE_MAX :=  [1,1] 182 [2,1] 182 [3,1] 182 [4,1] 182 [5,1] 182 [6,1] 182
[7,1] 182;

/* definizione ore malattia/riposo degli operatori */
param durata_malattia_riposo :=  [1] 7 [2] 7 [3] 7 [4] 7 [5] 7 [6] 7 [7] 7;

/* definizione ferie degli operatori */
set Ferie[1] := 15 16 17 18 19 20 ;
set Ferie[2];
set Ferie[3];
set Ferie[4];
set Ferie[5];
set Ferie[6];
set Ferie[7];
```

```
/* definizione malattie degli operatori */
set Malattie[1;
set Malattie[2];
set Malattie[3];
set Malattie[4];
set Malattie[5] := 4 5 6 7 8 ;
set Malattie[6];
set Malattie[7];

/* definizione degli operatori che lavorano in un reparto */
set OPERATORI_REPARTO[reparto1] := 1 2 3;
set OPERATORI_REPARTO[reparto2] := 4 5 6 7;

/* definizione degli operatori che svolgono il "3 + 1" */
set OPERATORI_TREPIUUNO := 1 2 3 4 5 6;

/* turni lavorativi */
set T_LAVORATIVI := turno1 turno3 turno4 turno5 ;

/* turni di riposo */
set T_RIPOSO := riposo    ;

/* turni di ferie */
set T_FERIE := ferie    ;

/* turni di malattia */
set T_MALATTIA := malattia    ;

/* durata turni lavorativi in ore */
param durata_lavorativi default 7 := [turno1] 7 [turno3] 7 [turno4] 7 [turno5] 7 ;

/* ora inizio turno */
param ora_inizio := [ferie] 00 [malattia] 00 [riposo] 00 [turno1] 06 [turno3] 17
[turno4] 00 [turno5] 12 ;

/* ora fine turno */
param ora_fine := [ferie] 00 [malattia] 00 [riposo] 00 [turno1] 13 [turno3] 00 [turno4]
07 [turno5] 19 ;

/* definizione dei turni di un reparto */
set TURNI_REPARTO[reparto1] := turno1 turno5;
set TURNI_REPARTO[reparto2] := turno3 turno4;

/* turni assegnabili ad operatori in determinati giorni */
#set TURNI_ASSEGNABILI default TURNI;

/* giorni lavorativi consecutivi degli operatori */
param giorni_consecutivi_lavorati default 0;

/* ultimo turno lavorativo effettuato dall'operatore */
```

```
set ultimo_turno_effettuato[1] := riposo;
set ultimo_turno_effettuato[2] := riposo;
set ultimo_turno_effettuato[3] := riposo;
set ultimo_turno_effettuato[4] := riposo;
set ultimo_turno_effettuato[5] := riposo;
set ultimo_turno_effettuato[6] := riposo;
set ultimo_turno_effettuato[7] := riposo;


/* ora inizio del turno preferenziale per ogni giorno di ogni operatore */
param oraInizioPreferenziale :  1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
21 22 23 24 25 26 27 28 29 30:=
1 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25
25 25 25 25 25
2 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25
25 25 25 25 25
3 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25
25 25 25 25 25
4 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25
25 25 25 25 25
5 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25
25 25 25 25 25
6 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25
25 25 25 25 25
7 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25
25 25 25 25 25
;

/* ora fine del turno preferenziale per ogni giorno di ogni operatore */
param oraFinePreferenziale :  1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
21 22 23 24 25 26 27 28 29 30:=
1 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25
25 25 25 25 25
2 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25
25 25 25 25 25
3 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25
25 25 25 25 25
4 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25
25 25 25 25 25
5 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25
25 25 25 25 25
6 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25
25 25 25 25 25
7 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25
25 25 25 25 25
;

/* se viene effettuato quel turno in quel giorno */
param carico := [turno1, 1] 2 [turno1, 2] 2 [turno1, 3] 2 [turno1, 4] 2 [turno1, 5] 2
[turno1, 6] 2 [turno1, 7] 2 [turno1, 8] 2 [turno1, 9] 2 [turno1, 10] 2 [turno1, 11] 2
[turno1, 12] 2 [turno1, 13] 2 [turno1, 14] 2 [turno1, 15] 2 [turno1, 16] 2 [turno1, 17]
```

2 [turno1, 18] 2 [turno1, 19] 2 [turno1, 20] 2 [turno1, 21] 2 [turno1, 22] 2 [turno1, 23] 2 [turno1, 24] 2 [turno1, 25] 2 [turno1, 26] 2 [turno1, 27] 2 [turno1, 28] 2 [turno1, 29] 2 [turno1, 30] 2 [turno3, 1] 1 [turno3, 2] 1 [turno3, 3] 1 [turno3, 4] 1 [turno3, 5] 1 [turno3, 6] 1 [turno3, 7] 1 [turno3, 8] 1 [turno3, 9] 1 [turno3, 10] 1 [turno3, 11] 1 [turno3, 12] 1 [turno3, 13] 1 [turno3, 14] 1 [turno3, 15] 1 [turno3, 16] 1 [turno3, 17] 1 [turno3, 18] 1 [turno3, 19] 1 [turno3, 20] 1 [turno3, 21] 1 [turno3, 22] 1 [turno3, 23] 1 [turno3, 24] 1 [turno3, 25] 1 [turno3, 26] 1 [turno3, 27] 1 [turno3, 28] 1 [turno3, 29] 1 [turno3, 30] 1 [turno4, 1] 1 [turno4, 2] 1 [turno4, 3] 1 [turno4, 4] 1 [turno4, 5] 1 [turno4, 6] 1 [turno4, 7] 1 [turno4, 8] 1 [turno4, 9] 1 [turno4, 10] 1 [turno4, 11] 1 [turno4, 12] 1 [turno4, 13] 1 [turno4, 14] 1 [turno4, 15] 1 [turno4, 16] 1 [turno4, 17] 1 [turno4, 18] 1 [turno4, 19] 1 [turno4, 20] 1 [turno4, 21] 1 [turno4, 22] 1 [turno4, 23] 1 [turno4, 24] 1 [turno4, 25] 1 [turno4, 26] 1 [turno4, 27] 1 [turno4, 28] 1 [turno4, 29] 1 [turno4, 30] 1 [turno5, 1] 1 [turno5, 2] 1 [turno5, 3] 1 [turno5, 4] 1 [turno5, 5] 1 [turno5, 6] 1 [turno5, 7] 1 [turno5, 8] 1 [turno5, 9] 1 [turno5, 10] 1 [turno5, 11] 1 [turno5, 12] 1 [turno5, 13] 1 [turno5, 14] 1 [turno5, 15] 1 [turno5, 16] 1 [turno5, 17] 1 [turno5, 18] 1 [turno5, 19] 1 [turno5, 20] 1 [turno5, 21] 1 [turno5, 22] 1 [turno5, 23] 1 [turno5, 24] 1 [turno5, 25] 1 [turno5, 26] 1 [turno5, 27] 1 [turno5, 28] 1 [turno5, 29] 1 [turno5, 30] 1;

Bibliografia

[AD01] Aickelin, U. e K. Dowsland: Exploiting problem structure in a genetic algorithm approach to a nurse rostering problem. Journal of Scheduling, 2001.

[AH01] Hofe, H. Meyer auf'm: Constraint Programming and Large Scale Discrete Optimization, volume 57 della serie DIMACS Series in Discrete Mathematics and Teorethical Computer Science, capitolo Nurse rostering as constraint satisfaction with fuzzy constraints and inferred control strategies, pagine 67–99. E. C. Freuder and R. J.Wallace, 2001.

[AHP80] T. L. Saaty. The Analytic Hierarchy Process. McGraw Hill Company, 1980.

[AMP] AMPL: AMPL R A Modeling Language for Mathematical Programming. http://www.ampl.com/.

[AW04] Aickelin, Uwe e Paul White: Building Better Nurse Scheduling Algorithms. Annals of Operations Research, 128:159–177, 2004.

[BCB01] Burke, E., P. D. Causmaecker e G. V. Berghe: A memetic approach to the nurse rostering problem. Applied Intelligence, 15:199–214, 2001.

[CB02] Causmaecker, Patrick De e Greet Vanden Berghe: Relaxation of Coverage Constraints in Hospital Personnel Rostering. Nel

PATAT 2002 4th international conference on the Practice and Theory of Automated Timetabling, Gebr. Desmetstraat 1, 9000 Gent, Belgium, 2002.

[CLLR03] Cheang, B., H. Li, A. Lim e B. Rodrigues: Nurse rostering problem – a bibliographic survey. European Journal of Operational Research,
151:447–460, 2003.

[EJK+04] Ernst, A.T., H. Jiang, M. Krishnamoorthy, B. Owens e D. Sier: An Annotated Bibliography of Personnel Scheduling and Rostering. Annals of Operations Research, 127:21–144, 2004.

[GL97] Glover, F. e M. Laguna: Tabu Search. Kluwer Academic Publishers, 1997.

[GNU04a] GNU Linear Programming Kit: Modeling Language GNU MathProg, Draft edizione, Jan 2004. Version 4.4.

[GNU04b] GNU Linear Programming Kit: Reference Manual, Draft edizione, Jan 2004. Version 4.4.

[Hun95] Hung, R.: Hospital Nurse scheduling. Journal of Nursing Administration, 7/8(25):21–23, 1995.

[ILO] ILOG: ILOG CPLEX: High-performance software for mathematical programming and optimization.
http://www.ilog.com/products/cplex/.

[JK92] Jelinek, R. C. e J. A. Kavois: Nurse staffing and scheduling: past solutions and future directions. Journal of the Society for Health System, (3), 1992.

[JSV98] Jaumard, B., F. Semet e T. Vovor: A generalized linear programming model for nurse scheduling. European Journal of Operational Research, pagine
1–18, 1998.

[MK92] Millar, H. e M. Kiragu: Cyclic and non-cyclic scheduling of 12h shift nurses by network programming. European Journal of Operational Research, 3(104):582–592, 1992.

[MM04] Michele Milesi : Progetto di un software per ottimizzare i turni degli operatori nelle case di riposo, Dic 2004.

[MWG76] Miller, H. E., P. William e J. R. Gustave: Nurse scheduling using mathematical programming. Operations Research, 5(24):857–870, 1976.

[Pla01] Plane: Soft constraints in the Nurse Rostering Problem, Full description of all the constraint types in use in Plane System, Feb 2001.
http://www.cs.nott.ac.uk/_gvb/constraints.ps.

[SW77] Smith, L. D. e A. Wiggins: A computer-based nurse scheduling system. Computers and Operations Research, (4):195–212, 1977.

[TP] Download componenti Delphi: http://www.torry.net