ALMA MATER STUDIORUM-UNIVERSITÀ DI BOLOGNA

SCUOLA DI SCIENZE

Corso di Laurea Magistrale in Informatica

UN SOFTWARE PER LA GESTIONE DELL'ORARIO DELLE LEZIONI

Relatore:

Professore

MORENO MARZOLLA

Presentata da:

VINCENZO GAMBALE

Sessione I

Anno Accademico 2014-2015

Indice

Elenco delle Figure


Stampato
su carta
riciclata 100%

Introduzione

My favorite things in life don't cost any money. It's really clear that the most precious resource we all have is time.

Steve Jobs

Lo scopo di questa tesi è lo sviluppo di un'interfaccia Web per un software di ottimizzazione, il cui scopo è la definizione dell'orario delle lezioni universitarie. Presenteremo di seguito il modello risolutivo al problema del timetabling proposto da Daskalaki et alii[5], e lo adatteremo alle esigenze della facoltà di Informatica dell'Università di Bologna. In effetti, la necessità di costituire un software atto alla risoluzione del problema dell'orario universitario rappresenta il tema principale trattato in questa tesi, che si ripropone, oltretutto, la costruzione di un'interfaccia grafica accattivante e intuitiva, che possa agevolmente essere utilizzata da docenti e personale di segreteria.

Il problema del Timetabling riguarda l'assegnamento di un insieme di attività, nel tempo e nello spazio, soggette a vincoli, necessari affinché vengano raggiunti determinati obiettivi. In generale, i problemi di timetabling sono noti per essere NP-ardui, cioè non esiste nessun algoritmo polinomiale in grado di risolvere il problema. Sono due i tipi principali di timetabling: il course timetabling, relativo alle istituzioni educative quali scuole medie-superiori e lo university timetabling, che si occupa della programmazione di un intero semestre o di un intero anno. La risoluzione del problema del timetabling impone il soddisfacimento di molti vincoli, a partire dai quali sarà possibile raffinare i risultati finali del software. Un tipico problema degli orari di un'università può coinvolgere migliaia di corsi, migliaia di studenti, centinaia di docenti, centinaia di aule e di altre risorse. La presenza di questa molteplicità di vincoli, preferenze e attori in gioco rendono il problema di difficile risoluzione, soprattutto in tempi ragionevoli.

Esistono molte metodologie per risolvere il problema del timetabling come: la programmazione lineare, metodi di euristiche e meta-euristiche, la constraint programming. In questa tesi esamineremo un approccio basato sulla programmazione lineare intera costruito partendo dal modello proposto da Daskalki et alii. Inoltre, verrà esaminato il problema dell'ora di pranzo, che consiste nel mantenere all'interno dell'orario di lezione un'ora libera per consentire ai docenti e agli studenti di pranzare. Come anticipato, lo scopo di questa tesi è adattare il modello sopracitato alle problematiche presenti nella Scuola di Scienze dell'Università di Bologna, per poi sviluppare un'interfaccia grafica a supporto di tale modello. Nel perseguire questo obiettivo, si è scelto di suddividere il nostro studio come segue. Nel *Primo Capitolo* descriveremo il problema del timetabling delineandone le caratteristiche principali. In seguito analizzeremo il timetabling accademico definendone i vincoli hard e soft, infine descriveremo alcuni approcci utili alla risoluzione di questo tipo di problema. Nel *Secondo Capitolo* faremo una breve introduzione sulla complessità computazionale e indagheremo la NP-completezza, illustreremo poi le tecniche di programmazione lineare intera. Nel *Terzo Capitolo* passeremo alla descrizione della struttura dei corsi offerti dalla scuola di Scienze per quanto riguarda gli indirizzi di Informatica e Informatica per il Managment. Vedremo poi il modello di risoluzione adottato in questa ricerca e analizzeremo il problema della pausa pranzo, proponendo alcune soluzioni. Nel *Quarto Capitolo* esamineremo la progettazione della GUI e la sua realizzazione con linguaggi web-oriented.

Il modello di programmazione lineare, basato su una versione sviluppata da M. Marzolla è presente in Appendice A. L'implementazione della GUI è formata da circa 2300 linee di codice in PHP, HTML e CSS ed è accessibile dal CD-ROM presente in copertina.

Capitolo 1

Il Problema dell'orario

In questo capitolo presenteremo il problema dell'orario. Partiremo con un'introduzione alla terminologia usata nel corso del capitolo, per poi arrivare a una prima definizione del problema. Mostreremo quindi le possibili versioni in cui il problema dell'orario può presentarsi, ne definiremo la struttura formale e presenteremo alcune delle metodologie proposte in letteratura per risolverlo. Concluderemo il capitolo con la descrizione del problema dell'orario relativo al corso di laurea in informatica della scuola di Scienze dell'Università di Bologna.

1.1 Un po' di terminologia

Prima di addentrarci nella descrizione del problema dell'orario, introduciamo alcuni concetti basilari che costituiranno le fondamenta dello sviluppo di questo elaborato, procediamo quindi con una panoramica generale della terminologia protagonista di questo studio.

Definiremo con il termine "evento" un'attività programmata, quale un corso o una sessione in laboratorio; con il termine "timeslot" indicheremo invece un intervallo di tempo in cui può essere schedulato un evento; con "risorsa" indicheremo qualsiasi oggetto necessario alla programmazione di un evento, come ad esempio le aule; infine definiamo ogni utente come un individuo che prende parte attiva all'interno dell'orario.

Definiamo ora il problema dell'orario, chiamato anche problema del timetabling, come l'assegnazione di una sequenza di eventi, soggetti a determinati vincoli, a diversi timeslot[1]. Anche se apparentemente sembrano sinonimi dello stesso problema, il timetabling non è da confondere con i problemi di schedulazione e sequenziamento. Una distinzione tra questi tipi di problemi viene fornita da A. Wren[20]:

Sequenziamento: usato per decidere l'ordine con il quale un dato numero di attività debba essere svolto. Consiste semplicemente nell'ordinare un insieme di attività,

[1] Luca Di Gaspero e Andrea Schaerf, *Il problema della generazione automatica dell'orario delle lezioni: teoria e pratica,* Paper, Università di Udine

precisando quale attività segue o precede un'altra. Un esempio di questo tipo di problema sono gli aerei in volo in attesa di autorizzazione all'atterraggio.

Timetabling: è l'allocazione di risorse soggette a vincoli, in slot collocati nello spazio-tempo, in modo da soddisfare il più possibile determinati obbiettivi. Un orario mostra il momento in cui avviene un determinato avvenimento. Ad esempio un orario dei bus mostra quali viaggi devono avvenire su determinate linee, ma non mostra quale autobus o quale conducente è stato assegnato a quello slot. Questo tipo di problema viene affrontato dallo scheduling.

Schedulazione: è l'allocazione soggetta a vincoli delle risorse ad oggetti che fanno parte dello spazio-tempo in modo da minimizzare il costo totale variando le impostazioni delle risorse utilizzate.

Dopo aver introdotto la terminologia utilizzata in questo capitolo e aver definito il problema dell'orario, nel prossimo paragrafo esamineremo le varie categorie in cui si divide il timetabling accademico.

1.2 Classificazioni del Timetabling Accademico

Il timetabling accademico è caratterizzato da diverse risorse e da vari utenti. Le risorse principali sono le aule e i corsi, mentre gli utenti sono gli studenti e i professori. Il problema del timetabling accademico comprende molte sottocategorie, elencate in letteratura attraverso metodologie diverse. Vediamo, ad esempio, una delle categorizzazioni più semplici, quella adottata da Andrea Schaef [14], che racchiude le sottocategorie del problema del timetabling in tre classi:

School Timetabling: corrispondente alla programmazione settimanale delle lezioni, progettato per risolvere il problema dell'orario nelle scuole dell'obbligo (elementari, medie e liceo). L'obiettivo di questo tipo di timetabling è quello di evitare che gli orari o le classi assegnate alle maestre possano sovrapporsi.

Course Timetabling : rappresenta il programma settimanale di tutte le lezioni di un insieme di corsi universitari, progettato in modo tale da minimizzare le sovrapposizioni tra corsi che hanno studenti in comune.

Exam Timetabling: programma della sessione d'esame, utilizzato affinché non ci siano esami dello stesso anno accademico nello stesso slot temporale.

Una classificazione più articolata è quella proposta da However, Aldy Gunawan and et. al. [9] e presentata in Figura 1.1

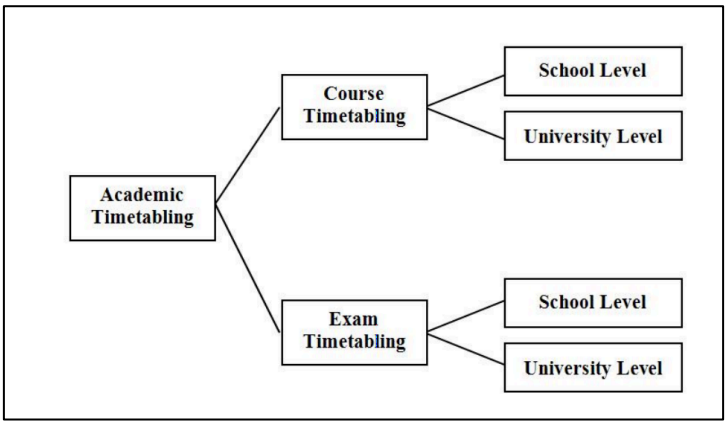

Figura 1.1 Classificazione del timetabling secondo Hower,Aldy Gunawan fonte:www.uleth.ca/dspace/bitstream/handle/10133/633/zibran,%20minhaz.pdf?sequence=1 p. 23

Come si può notare, in Figura 1.1 si fa distinzione tra "course timetabling" e "exam timetabling", come già evidenziato nella categorizzazione di Schaef. A differenza di quest'ultima, però, viene applicata un'ulteriore suddivisione tra il timetabling scolastico e quello universitario, che rappresentano due ambienti completamente diversi. Un "course timetabling" a livello scolastico specifica quando ad ogni classe viene assegnata una particolare lezione, tenuta da un determinato insegnante in una data aula. A livello universitario la situazione è molto più complicata. La sostanziale differenza sta nel fatto che spesso i corsi universitari hanno studenti in comune tra loro, mentre le classi dello "school timetabling" sono composte da insiemi disgiunti di studenti. Di conseguenza, a livello universitario, i corsi con studenti in comune non devono sovrapporsi. In questa tesi ci concentreremo unicamente sul "course timetabling" a livello universitario.

1.3 Timetabling Universitario

Realizzare un orario che sia in grado di soddisfare i regolamenti e i bisogni di un'istituzione accademica e, allo stesso tempo, tenga conto delle necessità di studenti e professori, è un arduo compito, soprattutto per coloro che devono realizzarlo, se pensiamo a tutte le richieste che ci sono in gioco. In molte istituzioni accademiche questo compito viene attribuito allo staff amministrativo. Spesso si cerca di risolvere il problema dell'orario prendendo in considerazione gli orari degli anni accademici precedenti, per crearne di nuovi, tentando di applicare il minor numero di cambiamenti possibili, ma questa non è sempre la soluzione migliore. Con l'ausilio di strumenti software possiamo riuscire ad automatizzare il processo di creazione dell'orario. Possiamo definire il problema del timetabling universitario come il processo che consiste nell'assegnare i corsi universitari a specifici timeslot, che coprono i cinque giorni lavorativi della settimana, ad aule adatte alle esigenze degli studenti iscritti e alle necessità di ciascun corso. L'obiettivo delle istituzioni accademiche è la costruzione di orari che siano effettivi, cioè realizzabili e utilizzabili dall'istituzione, e soddisfacenti, cioè capaci di soddisfare la maggior parte dei bisogni degli utenti. Un corso proposto da un determinato dipartimento può essere composto sia da ore di lezione in aula, che chiameremo lezioni frontali, che da ore di lezione in laboratorio. Le lezioni frontali vengono tenute da un professore, ma occasionalmente possono essere tenute da più professori che decidono di dividersi le ore in aula. Le lezioni in laboratorio sono spesso parte integrante delle lezioni frontali ma a differenza di esse vengono eseguite all'interno di aule particolari, come i laboratori. Ogni corso universitario può essere a scelta o obbligatorio. I corsi obbligatori sono quelli che il dipartimento ritiene che siano alla base del percorso formativo degli studenti. Durante la laurea triennale, la maggior parte dei corsi sono obbligatori e concepiti per essere seguiti da studenti dello stesso anno. Al contrario i corsi a scelta rappresentano la parte minore del percorso formativo, e possono essere seguiti da studenti di anni diversi; allo stesso modo dei corsi obbligatori essi non devono sovrapporsi con gli altri corsi a scelta e con i corsi obbligatori stessi. Dopo aver definito le possibili tipologie di corso, passiamo alla definizione delle risorse. Ci riferiremo a due principali tipi di risorse: le risorse umane, studenti e docenti, e le risorse materiali, ovvero le aule. I professori hanno la possibilità di decidere, in base ai propri impegni accademici e personali, di indicare gli slot di

tempo migliori in cui tenere le lezioni. La disponibilità delle aule è soggetta alle disposizioni dell'Università. Si tenga presente che non tutte le scuole hanno un numero adeguato di aule che soddisfi il fabbisogno delle risorse umane; per ovviare al problema alcune scuole utilizzano le aule di altre scuole. Ogni aula ha una sua capienza e disponibilità, ad esempio un'aula può ospitare cento persone mentre altre hanno una capienza massima di venti; ci sono poi aule libere solo in due giorni alla settimana, mentre altre hanno uno spettro di disponibilità oraria più vasto. Spetta agli addetti all'orario l'assegnazione dei corsi alle aule, in modo tale da soddisfare le necessità di studenti e professori. La costruzione di un orario settimanale, come già detto, si rivela spesso come un lavoro noioso che richiede molte ore/uomo. Nella creazione di un orario bisogna tener conto delle richieste degli utenti che vengono chiamate anche vincoli. Molti studiosi approcciandosi a questo problema suddividono i vincoli in due categorie principali:

Hard : rappresentano i vincoli che devono essere necessariamente soddisfatti all'interno dell'orario. Sono vincoli che non possono essere violati e sono i più importanti

Soft : rappresentano vincoli che se vengono soddisfatti migliorano la soluzione ma non devono essere soddisfatti obbligatoriamente.

1.3.1 Hard Constraint

I vincoli hard nel problema dell'orario universitario si basano sulle seguenti regole :

1. ***Non sono consentite collisioni***: una collisione all'interno dell'orario avviene quando sono programmati, nel medesimo timeslot, due o più eventi che coinvolgano lo stesso professore o lo stesso anno di corso o la stessa aula. Una collisione è possibile anche quando alla stessa aula vengono assegnati due corsi appartenenti allo stesso anno di corso.

2. ***Un orario deve essere completo***: definiamo un orario completo quando tutti i corsi previsti per ogni gruppo di studenti sono presenti all'interno di esso, con il giusto numero di ore di lezione e con la giusta suddivisione in blocchi(il

concetto di blocco verrà definito in seguito). Un orario è completo anche quando a ogni professore sono state assegnate tutte le ore di lezione.

3. ***Un orario deve essere in grado di accogliere le richieste di un professore*** : una volta assegnatogli un corso il professore deve essere in grado di inserire il numero di ore di lezione che vuole svolgere e come vuole suddividere le ore di lezione all'interno della settimana. Ad esempio, un professore che ha nove ore di lezione settimanali deve poter scegliere di dividere tali ore in tre blocchi da tre ore.

Un esempio di hard constraint è : per ogni periodo dovrebbero essere disponibili risorse sufficienti (aule, personale, etc.) per tutte le lezioni in programma per tale periodo.

1.3.2 SoftConstraint

A differenza dei vincoli hard i vincoli soft devono rispettare le seguenti regole:

1. ***Preferenze per l'insegnamento*** : le preferenze di un professore rispetto alle lezioni in timeslot specifici devono essere rispettate, se possibile. Ogni persona del corpo docente può esprimere una preferenza in merito al periodo preferito per svolgere i suoi corsi. Ad esempio, deve poter scegliere se vuole svolgere il corso nelle prime ore del mattino o magari nelle ultime ore del pomeriggio.

2. ***Orari compatti***: gli orari degli studenti dovrebbero essere quanto più compatti possibile, e devono essere in grado di consentire delle pause. Gli studenti sentiranno meno il peso delle lezioni se i corsi più articolati vengono distribuiti nelle prime ore del mattino. Gli altri corsi possono quindi riempire il vuoto in un dato giorno in modo da ridurre al minimo gli slot vuoti tra le lezioni. Tuttavia, nella maggior parte dei casi, gli studenti preferiscono uno slot vuoto intorno all'ora di pranzo, in modo da poter riposare prima di andare avanti con le loro lezioni.

3. ***Ridurre al minimo il cambio di aula :*** questo punto è particolarmente valido per gli studenti dei primi anni. Questi ultimi, di solito in numero maggiore rispetto agli studenti degli anni successivi, frequentano le lezioni nelle aule più grandi ed è la cosa migliore rimanere nello stesso posto per un paio di lezioni prima di fare uno spostamento in un laboratorio o in una classe più piccola.

Un esempio di soft costraint è : un professore può decidere di insegnare un determinato corso, in una determinata aula, in un determinato giorno e in un determinata ora.

1.4 Risolvere il problema del timetabling

Esistono in letteratura diverse metodologie per risolvere il problema del timetabling. Alcuni studiosi propongono soluzioni basate su algoritmi euristici greedy, meta-euristiche, come la hill climbing e la tabu search. Un'altra tendenza è quella di utilizzare la programmazione lineare intera e le tecniche di programmazione a vincoli. Infine, alcuni studiosi hanno proposto approcci misti, combinando elementi d'intelligenza artificiale con elementi presenti nelle due categorie sopracitate. Possiamo dividere gli approcci conosciuti in quattro grandi categorie : approcci basati sui metodi che utilizzano meta-euristiche, metodi che utilizzano la ricerca operativa, approcci basati sulla constraint programming e metodologie fondate sull'intelligenza artificiale.

1.4.1. Approcci basati sulle meta-euristiche.

A questa categoria appartengono due grandi sotto-categorie, che si distinguono in base alle soluzioni di partenza usate per la risoluzione del problema. La prima sottocategoria si basa su una popolazione iniziale delle soluzioni, cioè inizialmente viene selezionato un insieme di soluzioni da cui partire; la seconda è basata su di una singola soluzione.

- ***Popolazione iniziale delle soluzioni :***

In questo tipo di approccio abbiamo un'insieme iniziale di soluzioni chiamato popolazione. Ad ogni iterazione viene selezionata la soluzione migliore della

popolazione, secondo un'euristica che assegna un punteggio ad ogni elemento della popolazione. A questa classe appartengono:

Gli algoritmi evolutivi e genetici : gli algoritmi evolutivi (EA) si basano su un modello di computazione ispirato al meccanismo evolutivo naturale. Gli EA operano su una popolazione di possibili soluzioni e sono strutturati in tre fasi: selezione, rigenerazione e sostituzione. Per esempio, assegniamo ad ogni individuo appartenente alla popolazione iniziale una stringa numerica che lo caratterizza. Nella fase di selezione, gli individui con un punteggio di fitness [2]alto vengono scelti per essere i genitori della nuova generazione. Nella fase di rigenerazione invece ai genitori selezionati nella prima fase viene applicata prima una funzione di crossover, che mescola i "geni" dei genitori in modo tale da distribuire il patrimonio genetico, infine un'operazione di mutazione, che perturba una parte della stringa risultante, come possiamo vedere in Figura 1.2. Nella fase di sostituzione, la popolazione originale (iniziale) viene sostituita con i figli appena generati.

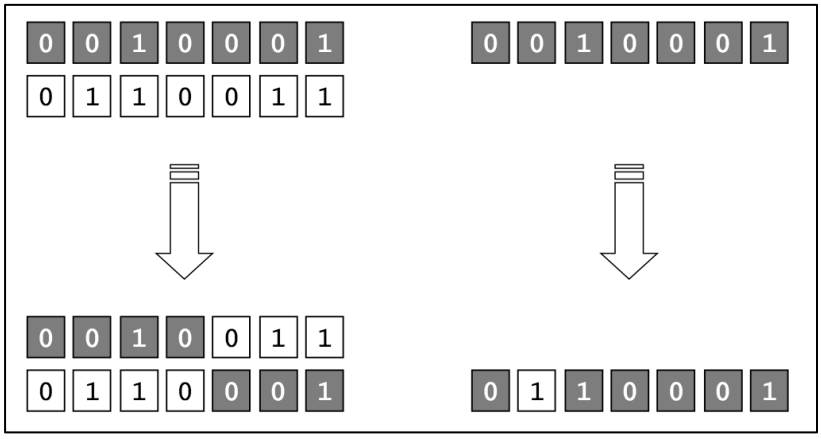

Figura 1.2 Esempio si crossover e mutazione fonte: http://slideplayer.it/slide/610522/

Gli algoritmi genetici, rappresentano un sottoinsieme degli algoritmi evolutivi. Si basano sui seguenti passi: 1) generazione di una popolazione iniziale; 2) assegnazione di un punteggio alla popolazione, attraverso una funzione di valutazione; 3) selezione dei genitori che prenderanno parte all'applicazione del crossover, cui segue l'effettiva attuazione del crossover; 4) applicazione di una mutazione; 5) valutazione finale di figli e genitori, per individuare chi parteciperà alla generazione di una nuova progenie.

[2] Funzione che restituisce valori maggiori per soluzioni migliori

Algoritmi Memetici: possiamo considerare un meme come un'unità contenente informazioni, che viene generata quando le credenze delle persone cambiano. La differenza tra meme e gene sta nel fatto che il gene viene trasmesso e non può essere rifiutato, mentre il meme viene scelto solo se può migliorare la soluzione. Con questo tipo di algoritmi lo spazio delle possibili soluzioni si riduce, grazie agli spazi di ottimizzazione sub locali, che rappresentano uno dei migliori vantaggi di questo procedimento. Gli algoritmi memetici sono una combinazione di algoritmi genetici e ricerca hill climbing (ricerca locale)[11]. Sadaf e Shengxiang, nel loro articolo "A Memetic Algorithm for the University Course Timetabling Problem" integrano algoritmi di ricerca locale e genetici. Questa tecnica utilizza due metodi di ricerca locale per migliorare la ricerca in un algoritmo genetico. Una delle ricerche locali viene eseguita su un singolo slot temporale, un'altra ricerca viene effettuata su fasce orarie di varie dimensioni, che sono insiemi di slot temporali[13].

- ***Soluzione Singola :***

A differenza degli algoritmi sopracitati, che utilizzano una popolazione di soluzioni, questo tipo di algoritmi seleziona una sola soluzione per analizzare il problema. Questa soluzione iniziale viene manipolata e raffinata nei vari passaggi successivi, fino ad ottenere una soluzione ottima. Questi algoritmi terminano quando non c'è più modo di migliorare la soluzione. A questo gruppo appartengono:

Tabu Search : è una procedura meta-euristica che migliora l'efficienza di un classico algoritmo di ricerca locale memorizzando informazioni che riguardano il processo di ricerca, in modo da evitare situazioni di stallo in corrispondenza del raggiungimento di un minimo locale. Ad ogni iterazione vengono rimosse alcune soluzioni non ottime, chiamate tabu, che non verranno prese in considerazione nelle iterazioni successive. Per avere memoria del processo di ricerca, si considera un insieme di informazioni dette attributi, che consentono di evitare di rivedere soluzioni già esaminate, che vengono memorizzate in una o più liste tabu. Ad ogni iterazione, a queste liste vengono aggiunti nuovi attributi, come possiamo vedere in Figura 1.3. Queste liste sono di dimensioni limitate e FIFO. L'algoritmo è composto da tre passi. Nel primo passo si trova una soluzione S che faccia partire l'algoritmo, in seguito all'iterazione k-esima si definisce

un intorno N(S,k) della soluzione corrente, infine si individua una soluzione S' migliore di S sulla base di un estimatore e si sostituisce S con S'. In ultima istanza, bisogna verificare se è stato soddisfatto il criterio di arresto, che può essere il numero di iterazioni, in caso contrario è necessario incrementare k e ritornare alla definizione dell'iterazione.

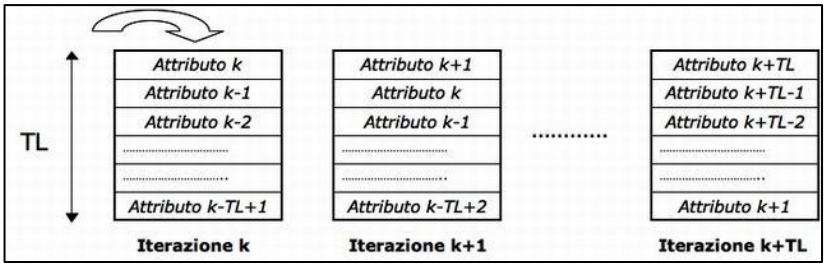

Figura 1.3 Riempimento delle liste tabu ad ogni iterazione fonte: http://www.federica.unina.it/ingegneria/ricerca-operativa-ing-2/tabu-search/

La soluzione finale sarà la migliore trovata durante tutto il processo d'iterazione. Un esempio di applicazione al problema dell'orario è l'utilizzo della Tabu search per assegnare gli studenti a gruppi di corsi, al fine di generare calendari efficienti e bilanciare il numero di studenti che si sono iscritti a un determinato corso. Il processo si sviluppa in tre fasi: nella prima fase, viene generato un insieme di soluzioni per ogni studente; nella seconda fase, vengono combinate una serie di soluzioni per poi applicare la tabu search e per ottenere soluzioni ottime, senza considerare la soluzione peggiore per ogni studente. Durante la terza fase, vengono assegnate le aule e migliorata l'allocazione, naturalmente senza cambiare l'assegnamento iniziale dei corsi alle fasce orarie[2].

***Ricerca locale*:** la ricerca locale è stata introdotta per trovare una soluzione che massimizzi un parametro, cercando tra un insieme di soluzioni. Questi algoritmi si spostano da una soluzione a un'altra all'interno di uno spazio di ricerca (spazio delle soluzioni prese in esame) applicando piccoli cambiamenti, finché viene trovata la soluzione desiderata. L'algoritmo di ricerca locale parte da una soluzione di base e si muove verso soluzioni vicine, questo è possibile solo quando sono definite le relazioni con le soluzioni vicine. La scelta del vicino viene effettuata selezionando il vicino che massimizza la soluzione localmente. Un esempio di questo tipo di algoritmi è l'hill climbing. Questo algoritmo deve il suo nome al fatto che, nel panorama dello spazio degli stati, segue sempre le salite più ripide. Si tratta di un semplice ciclo, che quindi si

muove continuamente verso l'alto (ovvero nella direzione dei valori crescenti) e termina quando raggiunge un picco che non ha vicini di valore più alto, come possiamo vedere in Figura 1.4

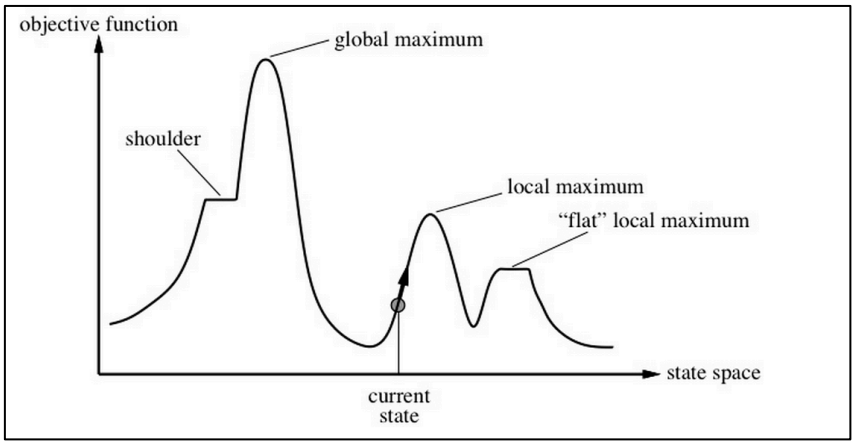

Figura 1.4 Panorama dello spazio degli stati monodimensionale, in cui l'altezza corrisponde alla funzione obiettivo fonte: https://www.studyblue.com/notes/note/n/ai-chp-41/deck/3859418

Simulated Anealling **:** questo metodo è una ricerca locale ispirata al fenomeno fisico della ricottura dei metalli. Questo approccio ci consente di non rimanere intrappolati nei massimi locali, ottimizzando l'uso del concetto di ricerca locale. Il ciclo base di questo algoritmo è molto simile a quello dell'hill climbing descritto in precedenza, ma invece di scegliere sempre la mossa migliore, l'algoritmo sceglie una mossa casuale e si comporta di conseguenza; ci troviamo quindi in una dimensione in cui due sono i casi possibili: se la mossa migliora la situazione, allora viene sempre accettata, in caso contrario l'algoritmo accetta la mossa con una probabilità minore di 1. Questa probabilità decresce esponenzialmente, sia con la cattiva qualità della mossa, misurata dal peggioramento ΔE della valutazione, sia con lo scorrere del tempo. Le mosse cattive quindi saranno accettate più facilmente all'inizio e diventeranno poi sempre meno probabili. Un'applicazione di questo algoritmo all'orario può essere rappresentato combinando Kempe chain[3] con un approccio Simulated Anealling. In questo caso, il problema viene riformulato, rilassando i vincoli hard e trasformandoli in vincoli soft. Il problema viene rilassato in due fasi: nella prima viene usata un'euristica, basata su

[3] Sia G un grafo planare i cui vertici sono stati adeguatamente colorati e supponiamo che v appartenga ai vertici di G ed è colorato con C-1. Una catena di Kempe C-1,C-2 che contiene v è la massima componente connessa di G che : contiene v ed e ed inoltre contiene solo i vertici colorati con C-1 e C-2.

grafi, affinché venga trovata una soluzione al problema. Nella seconda viene utilizzato il simulated annealing, così che vengano minimizzate le violazioni dei vincoli soft[18].

1.4.2 Approcci basati sulla ricerca operativa

In questa categoria ricadono approcci basati sulla teoria della colorazione dei grafi, metodi di programmazione lineare, e programmazione lineare intera, e metodi che utilizzano la programmazione a vincoli.

Colorazione dei grafi **:** un primo approccio al problema dell'orario è stato introdotto da Gotlieb, che propose una formulazione del problema utilizzando tre insiemi distinti: aule, timeslot e corsi. Gallese e Powell, hanno modellato per primi il problema del timetabling riconducendolo al problema della colorazione dei grafi, ma questo approccio si è rivelato inutile nel caso di lezioni pre-assegnate. De Werra , propone l'utilizzo di un grafo non orientato, come possiamo vedere in Figura 1.5. In questo caso l'obiettivo è quello di colorare tutti i vertici di un grafo con un determinato numero di colori, in modo tale che i nodi adiacenti abbiano colori diversi. L'orario risultante da questa operazione non deve presentare conflitti e dovrebbe essere colorato con il minor numero di colori possibile. Nel problema dell'orario basato sulla teoria della colorazione dei grafi, gli eventi vengono rappresentati come nodi, gli archi come vincoli (preferibilmente vincoli hard), e le fasce orarie vengono rappresentate con i colori, come mostrato in Figura 1.5. L'obiettivo è quello di trovare il minor numero di colori, in modo da poter pianificare gli eventi in un solo giorno. Se due nodi vicini hanno lo stesso colore allora vuol dire che c'è un conflitto a livello di orario.

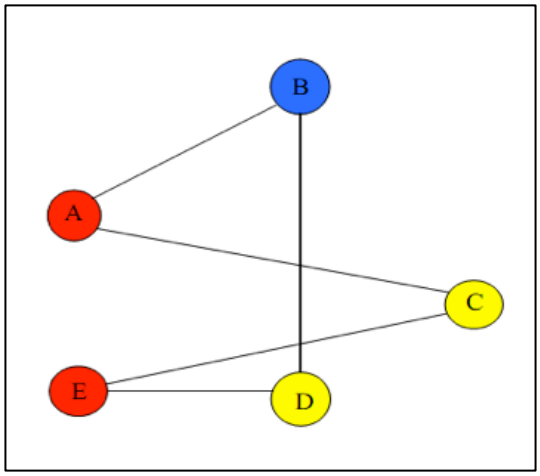

Figura 1.5 Esempio di colorazione di un grafo

***Programmazione lineare & programmazione lineare intera*:** Feiring [6], definisce la programmazione lineare (LP) come un sottoinsieme della programmazione matematica, in cui scopo è quello di assegnare in modo efficiente le risorse alle attività, al fine di massimizzare l'obiettivo e minimizzare i costi. Nella programmazione lineare intera, definita da Bunday, tutte o alcune delle variabili vengono definite come valori interi non negativi. Questi tipi di approcci sono interamente basati sulla matematica e non risulta sempre naturale trasformare problemi in termini di programmazione lineare e programmazione lineare intera. Questo tipo di formulazione è adatta al problema dell'orario e ci consente di rappresentarlo utilizzando espressioni matematiche grazie alle quali è possibile raggiungere una soluzione ottima. Innanzitutto, bisogna convertire le risorse, i vincoli e l'obiettivo posto, in linguaggio matematico. Potremmo pensare di utilizzare una variabile che rappresenta lo scheduling di un corso in una data aula, in un determinato giorno in un determinato time slot. Possiamo utilizzare delle equazioni/disequazioni per rappresentare i vincoli del nostro problema. Una volta formalizzato in termini matematici, è possibile trovare una soluzione al problema dell'orario identificando un valore per tutte le variabili sopracitate, così che tale valore massimizzi la funzione obiettivo, ovviamente senza violarne i vincoli.

1.4.3 Approcci basati sulla constraint programming

Questo tipo di approcci modellano il problema dell'orario come un problema di soddisfacimento di vincoli. La struttura di questi problemi, chiamati CSP, è la seguente: Un CSP viene rappresentato con una tripla <X,D,C> in cui X rappresenta le variabili di decisione, cioè le variabili che vogliamo esaminare, D è l'insieme dei domini $D_1,D_2,\ldots D_n$ relativi alle variabili X e C rappresenta l'insieme dei vincoli che sono relazioni sulle variabili che ne limitano il dominio. Una soluzione ad un CSP è un assegnamento di valori alle variabili che soddisfa tutti i vincoli contemporaneamente. Nel nostro caso le variabili rappresentano gli eventi e i valori rappresentano le risorse. Le variabili devono essere assegnate ai valori adatti rispondenti a una serie di vincoli. Nel caso non fosse possibile trovare una soluzione, si utilizza il backtraking[4].

[4] E una tecnica per trovare una soluzione a problemi di soddisfacimento di vincoli. Nel caso non si riesca a trovare una soluzione bisogna tornare indietro alla variabile precedente e provare ad usare un valore diverso

1.4.4 Approcci basati sull'intelligenza artificiale.

In questa classe troviamo approcci combinatori, aprocci fuzzy, approcci basati sulle hyper euristic e metodi basati sulla conoscenza e sull'intelligenza artificiale. Vediamone alcune applicazioni:

Approcci combinatori: negli ultimi tempi gli approcci combinatori vengono utilizzati per risolvere problemi di ottimizzazione NP-Completi, poiché risultano molto efficienti e raggiungono ottime prestazioni nella risoluzione di questo tipo di problemi. In questo tipo di approccio vengono applicati insieme più algoritmi e tecniche combinatorie, in modo tale da sfruttare gli aspetti positivi di più algoritmi. Dato che ogni algoritmo o tecnica presenta un punto debole, una combinazione appropriata riduce al minimo le difficoltà degli algoritmi, generando soluzioni più efficienti rispetto al singolo algoritmo. Un tipico approccio combinatorio è articolato in tre fasi: nella prima fase individua una possibile soluzione iniziale, utilizzando una tecnica di constraint programming; nella seconda fase, per migliorare la soluzione, utilizza il simulated anealling; infine, nella terza fase, per trovare la soluzione ottima, viene utilizzato l'hill climbing.

Abdullah e Hamdan [1], hanno presentato un particolare tipo di approccio combinatorio, per risolvere il problema dell'orario. Benché differente da quello appena descritto, anche quest'ultimo si compone di tre fasi: nella prima fase la soluzione iniziale viene generata utilizzando una funzione euristica; nella seconda fase viene eseguita una tecnica di ottimizzazione, che utilizza un algoritmo di ottimizzazione iterativa casuale; nella terza fase viene utilizzato il simulated annealing, come criterio di accettazione della soluzione.

Logica fuzzy: con la logica fuzzy a ciascuna proposizione si può attribuire un grado di verità compreso tra 0 e 1. Possiamo considerare la logica fuzzy come una logica continua, ispirata all'argomentazione approssimativa dell'essere umano [3]. Golabpour[8] presenta una soluzione fuzzy basata su un approccio memetico per risolvere il problema dell' orario, in cui una soluzione al problema viene confrontata con la soluzione data da un algoritmo genetico e la soluzione data da una algoritmo memetico. Il risultato dovrà soddisfare i vincoli presenti in entrambi i problemi. La

novità di questo tipo di approccio è che la logica fuzzy viene utilizzata come base della ricerca locale in un algoritmo memetico.

1.5 In breve

In questo capitolo abbiamo esaminato il problema dell'orario, introducendo la terminologia che sarà alla base del nostro studio. Abbiamo esaminato le differenze che intercorrono tra il timetabling e altri problemi ad esso simili, quali ad esempio lo scheduling e il sequencing. Sono state elencate le classificazioni del problema del timetabling accademico, dando maggiore rilevanza al timetabling universitario, per l'organizzazione dei corsi. Infine abbiamo elencato gli approcci alla soluzione del problema presenti in letteratura. In questa tesi la risoluzione del problema sarà affrontata utilizzando la programmazione lineare intera, poiché il modello che è alla base dell'interfaccia è basato su proprio su questo tipo di programmazione.
Nel prossimo capitolo introdurremo alcune nozioni teoriche relative alla ricerca operativa e alla programmazione lineare, utili a comprendere la formalizzazione, in termini matematici, del problema dell'orario.

Capitolo 2

Nozioni Preliminari

In questo capitolo introdurremo alcuni aspetti teorici utili alla comprensione dell'intero lavoro di tesi. Inizieremo introducendo, la complessità computazionale e l'NP-Completezza, per poi passare alla definizione della ricerca operativa con la sua storia e le sue classificazioni. Vedremo una particolare branca della ricerca operativa, la programmazione lineare (PL) che si occupa di risolvere problemi di ottimizzazione. In seguito, esamineremo alcuni degli algoritmi adatti alla risoluzione di problemi di PL, come il simplesso e i metodi di branch and bound. Infine indagheremo alcuni tools utili alla risoluzione di questo tipo di problemi.

2.1 Complessità computazionale

Spesso, in informatica, c'è bisogno di confrontare algoritmi diversi per capire quanto velocemente possano essere eseguiti e di quanta memoria abbiano bisogno. Esistono due approcci principali per questo tipo di analisi: il primo consiste nell'eseguire una implementazione degli algoritmi su un elaboratore misurando la velocità di esecuzione e la memoria occupata, raccogliendo dei dati per poi confrontarli con dei benchmark esistenti. Il secondo, consiste nell'effettuare uno studio di un algoritmo ricorrendo alla teoria dell'analisi asintotica, un metodo per descrivere e analizzare il comportamento di un sistema ai suoi limiti. Questo approccio risulta migliore poiché riusciamo a dividere la componente riguardante i linguaggi di programmazione utilizzati per implementare l'algoritmo, dalla componente puramente computazionale. Utilizziamo come esempio un programma che calcola la somma di una sequenza scritta in pseudo-codice:

```
{sum(seq) :
      sum = 0
      for i = 0 to length(seq)-1 do
            sum = sum + seq[i]
      return sum}
```

Possiamo valutare questo algoritmo con un approccio definito in due fasi: nella prima esaminiamo l'input calcolandone la dimensione, in questo caso il numero di elementi della sequenza indicati con n. Nella seconda fase esaminiamo l'implementazione,

potremmo contare il numero di righe di codice eseguite, oppure il numero di assegnamenti effettuati, il numero di addizioni, o l'accesso agli array etc. Alla fine delle due fasi avremo una stima del tempo di esecuzione dell' algoritmo al variare della grandezza dell'input che chiameremo T(n). In questo caso:

$$T(n) \cong 2n + 2$$

Non tutti gli algoritmi, però, sono così semplici; inoltre la ricerca di un parametro n, che caratterizzi il numero di passi eseguiti, non è sempre lineare. Di solito misuriamo il caso pessimo e il caso medio: il caso pessimo prevede i dati più sfavorevoli per l'algoritmo; mentre il caso medio è il caso più utile da analizzare, benché la sua analisi sia piuttosto complicata. Il caso medio può fornire un reale indicatore della complessità dell'algoritmo, ma esso non è necessariamente rappresentativo del costo "reale", in quanto nella realtà potrebbero presentarsi istanze del problema che generano il caso peggiore.

Spesso gli algoritmi rendono difficile un'analisi esatta ed è quindi utile ricorrere a delle approssimazioni. Un'approssimazione adeguata è $O(n)$. Formalmente diremo che:

Definizione 2.1(O(n)) $O(f(n))$ *è l'insieme di tutte le funzioni g(n) tali che esistano due costanti positive c ed m per cui* $g(n) \leq cf(n) \; \forall \; n \geq m$

Possiamo affermare che, con n tendente all'infinito, una funzione costo lineare è preferibile a una funzione costo quadratica.

Diremo che un algoritmo è risolvibile in tempo polinomiale quando ha una complessità $O(f(n))$, dove *f(n)* è una funzione polinomiale, con input di lunghezza n. Questi algoritmi vengono considerati trattabili o "facili" ed appartengono alla classe P[17]. Esiste un'altra classe di algoritmi chiamata NP, che comprende tutti i problemi che sono "verificabili" in tempo polinomiale. Consideriamo come esempio il problema del Ciclo Hamiltoniano[5]. Se consideriamo un grafo G e supponiamo di aver trovato già una soluzione al problema, per verificare che essa sia un ciclo hamiltoniano non dobbiamo far altro che esaminare tutti gli archi per verificare se appartengono alla

[5] Nella teoria dei grafi un cammino è detto hamiltoniano se tocca tutti i vertici una ed una sola volta. Si parla di ciclo hamiltoniano quando in un cammino hamiltoniano esiste un arco che collega il primo con l'ultimo nodo.

soluzione quest'operazione è risolvibile in O(n), quindi il problema appartiene alla classe NP. Dati due problemi A e B, si dice che A si riduce a B se esiste un algoritmo polinomiale che associa ad ogni istanza di A un'istanza di B, in modo tale che la soluzione dell'istanza di B fornisca una soluzione della corrispondente istanza di A. Un problema A appartenente alla classe NP, si dice NP-completo se B si riduce ad A per ogni B∈NP. In questo senso i problemi NP-completi sono i problemi più difficili della classe NP.

2.2 Ricerca Operativa

Morse e Kimball, nel loro manuale *Metodi di Ricerca Operativa,* definiscono la "ricerca operativa" come: "Applicazione del metodo scientifico da parte di gruppi interdisciplinari a sistemi complessi e organizzati per fornire al personale dirigente soluzioni utilizzabili nei processi decisionali" [10]. Possiamo inquadrare, dunque, la ricerca operativa come una branca della matematica applicata che si occupa della risoluzione di problemi decisionali complessi attraverso la creazione di modelli matematici e metodi quantitativi avanzati. La prima apparizione della ricerca operativa risale alla prima metà del novecento, ad opera di Frederick Taylor, un ingegnere americano. Taylor la utilizzò per il miglioramento dei processi produttivi rivoluzionando l'aspetto manageriale delle fabbriche. Il metodo di Taylor si basa sull'ottimizzazione dei tempi nei processi produttivi utilizzando strumenti analitici. La Gran Bretagna negli anni trenta utilizzò la ricerca operativa per gestire l'enorme mole di dati inviati dai radar. Era indispensabile ottimizzare la distribuzione delle apparecchiature sul territorio e riuscire ad inviare via radio la soluzione ad opportune località. Il problema non è la correttezza dei dati inviati, bensì come utilizzarli per trarne vantaggio[16]. Per questo motivo venne fondato l' Operational Research Team, incaricato di studiare scientificamente i problemi operativi, cercando di trovare soluzioni organizzative. Questo team multidisciplinare riuscì ad ottenere ottimi risultati sia nell'analisi dei dati dei radar che nella gestione della flotta aerea Inglese. Nel corso della seconda guerra mondiale, nel Regno Unito, nel Canada e negli USA furono impegnati settecento scienziati per risolvere problematiche relative alla ricerca operativa. Alla fine della guerra la ricerca operativa ampliò il suo campo d'azione, passando da quello prettamente militare a quello civile. Questo periodo fu segnato da tre eventi fondamentali : J.L Von Neumann propose dei modelli concettuali per lo studio

di processi di crescita economica; seguì poi la nascita dei primi modelli per le applicazioni logistiche grazie a P.M.S. Blackett e T.C Koopmans; infine, avvenne lo sviluppo della programmazione lineare, con il metodo di risoluzione del Simplesso proposto da G.B. Dantzig[4] . Negli anni sessanta, la ricerca operativa venne applicata anche in ambito civile, grazie alla diffusione dei primi sistemi di archiviazione automatici nelle grandi aziende. Fu proprio in questi anni che Giuseppe Pompilj, matematico italiano, portò gli studi di ricerca operativa in Italia e nel 1961 fondò l'AIRO (Associazione italiana di Ricerca Operativa). Negli ultimi anni del ventesimo secolo la rilevanza applicativa della ricerca operativa è riconosciuta e apprezzata soprattutto in ambito industriale. Attualmente la ricerca operativa viene utilizzata nei trasporti, nella finanza, nella logistica, nella gestione delle risorse umane, nella gestione dei servizi sanitari, nelle telecomunicazioni, in Data Managing e in bio-informatica.

2.2.1 Classificazione

Una delle più importanti classificazioni della ricerca operativa la divide in tre sotto-branche:

- **Processi Stocastici**: atti alla realizzazione dei modelli probabilistici per determinare il comportamento di alcuni sistemi. Questa branca è alla base dell'Ottimizzazione Stocastica e dell'Ingegneria Finanziaria.
- **Simulazione**: si occupa di risolvere per ottimalità prettamente problemi "NP Ardui". Questa tecnica consiste di due fasi: nella prima, il problema viene formalizzato con un modello matematico; nella seconda, vengono determinati i parametri "buoni" mediante metodi statistici o attraverso la teoria dei giochi.
- **Ottimizzazione**: con questa tecnica viene formalizzato il problema con l'ausilio di un modello matematico, individuando per esso la soluzione ottima o sub-ottima

In questa tesi ci occuperemo soprattutto di problemi di ottimizzazione. Questa classe di problemi può essere risolta utilizzando la programmazione lineare.

2.3 La programmazione lineare

La programmazione lineare è una branca della ricerca operativa che ci consente di formulare in termini matematici un problema di ottimizzazione, usando un insieme di

vincoli e una funzione obiettivo. La funzione obiettivo deve essere una funzione lineare sulle incognite (variabili), mentre l'insieme dei vincoli è composto da equazioni e disequazioni lineari. La programmazione lineare si distingue in programmazione lineare intera e binaria. Nella programmazione lineare intera le variabili sono vincolate ad assumere valori interi, mentre nella programmazione lineare binaria, detta anche programmazione 0-1, le variabili sono vincolate ad assumere valori binari. Ad ogni modo, il punto di partenza della programmazione lineare e della programmazione lineare intera, sta nella trasformazione di un problema di ottimizzazione in un problema di programmazione lineare, così da risolverlo con algoritmi noti, come vedremo in seguito. La forma standard di un problema di programmazione lineare è la seguente:

$$minimize\ c^t x$$
$$soggetto\ a\ \ Ax = b, x\ \leq 10$$

L'espressione $c^t x$ rappresenta la funzione obiettivo, mentre $Ax = b, x \leq 10$ descrivono i vincoli del nostro problema. Con il simbolo ($*^t$) indichiamo l'operazione di matrice trasposta, $x \in \mathbb{R}^n$ è il vettore delle variabili di decisione che devono essere individuate, $A \in \mathbb{R}^{mxn}$ è la matrice dei coefficienti noti, mentre $c \in \mathbb{R}^n$ e $b \in \mathbb{R}^m$ rappresentano i vettori dei valori noti. Introduciamo adesso due concetti fondamentali: quello di soluzione ammissibile e quello soluzione ottima. Con il termine soluzione ammissibile indicheremo un assegnamento delle variabili, del vettore x che soddisfa i vincoli del problema. Useremo, invece, il termine soluzione ottima per indicare una soluzione ammissibile che ottiene il minimo o il massimo valore possibile, a seconda della funzione obiettivo.

2.3.1 Algoritmi per la programmazione lineare

Esistono diversi algoritmi per risolvere la programmazione lineare, come ad esmpio il metodo geometrico, il metodo del simplesso, il metodo dell'ellissoide e le tecniche denominate interior-point.[12]. Vedremo in particolare il metodo geometrico, utile per determinare le soluzioni del problema in caso di poche variabili. e il metodo del simplesso, introdotto da George Dantzig, che ad oggi risulta essere il metodo più utilizzato per la risoluzione di problemi di programmazione lineare.

- **Metodo Geometrico**

Il metodo geometrico per la risoluzione di problemi di programmazione lineare si configura in una procedura geometrica. Prima di addentrarci nella descrizione del metodo geometrico, poniamo come preambolo: sia $X = \{\underline{x} \mid A\underline{x} \geq \underline{b} \; e \; \underline{x} \geq 0\}$ una regione ammissibile, cioè l'insieme di tutte le soluzioni ammissibili del nostro problema, composta da tutti i vettori $\underline{x}$ che soddisfano i vincoli del problema $A\underline{x} \geq \underline{b} \; e \; \underline{x} \geq 0$. Il nostro obiettivo è trovare il massimo (o il minimo a seconda della funzione obiettivo) di z = c$\underline{x}$ tra tutti i punti della regione ammissibile. Definiamo le curve di livello, che rappresentano l'insieme del luogo dei punti che soddisfa l'equazione z = c$\underline{x}$.

Se ci troviamo dinanzi ad un problema di massimizzazione, e quindi siamo alla ricerca di un massimo, le curve di livello devono essere spostate parallelamente nella direzione che massimizza il valore della funzione obiettivo: tale direzione è rappresentata da c, chiamato gradiente. Nel caso il problema sia di minimizzazione, la funzione obiettivo deve essere minimizzata e la direzione è – c. Il punto di ottimo è il punto di massimo della nostra regione limitata, individuato dalle curve di livello ortogonali al gradiente c.

Vediamo un esempio :

$$\min z \;\; con \; z = \; -x_1 - 3x_2$$
$$\begin{cases} x_1 + \; x_2 \; \leq 6 \\ -x_1 + \; 2x_2 \; \leq 8 \\ x_1, x_2 \; \geq 0 \end{cases}$$

La Figura 2.1 mostra la regione ammissibile del nostro problema :

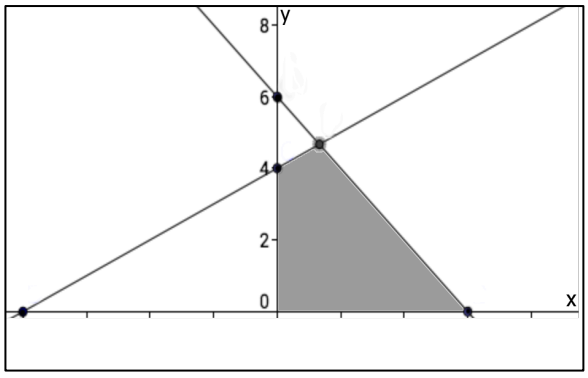

Figura 2.1 Regione ammissibile in grigio del problema studiato

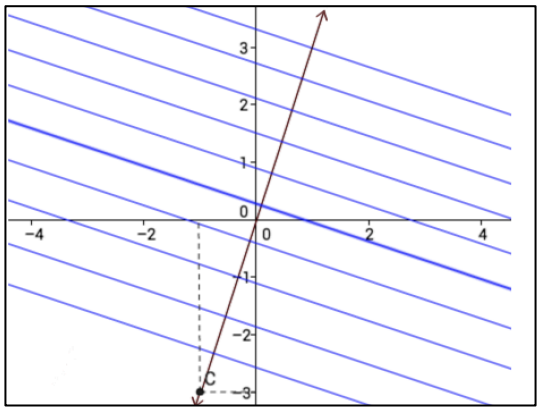

Figura 2.2 Curve di livello per c(-1,-3)

In questo esempio la direzione per una funzione di massimizzazione è c(-1,-3), ma poiché noi vogliamo minimizzare la funzione obiettivo la nostra direzione sarà –c, come possiamo vedere in Figura 2.2. Il metodo geometrico è molto utile alla risoluzione di problemi con due variabili; mentre risulta inapplicabile a problemi con più di tre variabili.

- **Metodo del simplesso**

Il metodo del simplesso è stato ideato nel 1947 da George Dantzig per risolvere problemi di programmazione lineare. Nella programmazione lineare possiamo rappresentare la regione ammissibile con un poliedro[6]. Grazie all'algoritmo del simplesso riusciamo a determinare il tipo di poliedro generato e a trovare una soluzione ottima al problema, che nel caso sia finita, coincide con un vertice del poliedro. L'algoritmo viene utilizzato per problemi di forma standard(cfr.2.3) che abbiano più di tre variabili. Il simplesso, reiterando un numero finito di passaggi consente di raggiungere una soluzione ottima partendo da una soluzione iniziale, nota come soluzione di base ammissibile, perfezionandola ad ogni passaggio. La soluzione di base ammissibile più semplice è quella in cui tutte le variabili di decisione valgono zero e vengono rispettati tutti i vincoli di segno. Il simplesso è basato sull'ingresso di nuove variabili all'interno della soluzione iniziale, una per ogni iterazione. Le nuove variabili sostituiscono le variabili uscenti, in modo tale da migliorare la soluzione iniziale, avvicinandosi ad una soluzione ottima. Ovviamente le nuove variabili che entrano

[6] In geometria solida e in teoria dei grafi, è un solido delimitato da un numero finito di facce piane poligonali

all'interno del sistema devono sempre rispettare i vincoli. Supponiamo di avere il seguente problema :

$$\max z = c_1x_1 + c_2x_2 + \cdots c_nx_n$$
$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1s}x_s + \cdots + a_{1n}x_n \leq b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2s}x_s + \cdots + a_{2n}x_n \leq b_2 \\ a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{is}x_s + \cdots + a_{in}x_n \leq b_i \\ \cdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{ms}x_s + \cdots + a_{mn}x_n \leq b_2 \\ x_i \geq 0 \ (i = 1 \ldots n) \end{cases}$$

Come primo passaggio trasformiamo le disequazioni in equazioni aggiungendo alle variabili di decisione, in questo caso x_1, x_2, x_s , le variabili di scarto $x_{n+1}, x_{n+2}, x_{n+i}$, x_{n+m} non negative, in numero pari ai vincoli.
Così facendo avremo il seguente sistema :

$$\max z = c_1x_1 + c_2x_2 + \cdots c_nx_n + 0x_{n+1} + 0x_{n+2} + 0x_{n+i} + 0\, x_{n+m}$$
$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1s}x_s + \cdots + a_{1n}x_n + x_{n+1} = b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2s}x_s + \cdots + a_{2n}x_n + x_{n+2} = b_2 \\ a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{is}x_s + \cdots + a_{in}x_n + x_{n+i} = b_i \\ \cdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{ms}x_s + \cdots + a_{mn}x_n + x_{n+m} = b_2 \\ x_i \geq 0 \ (i = 1 \ldots m + n) \end{cases}$$

Come detto in precedenza, ponendo le variabili di decisione uguali a zero otteniamo la prima soluzione di base ammissibile, che fornisce il valore z = 0. Costruiamo la tabella :

c_i	x_i	x_1	x_2	...	x_s	...	x_n	x_{n+1}	x_{n+2}	...	x_{n+i}	...	x_{n+m}	b_i
c_{i1}	x_{i1}	a_{11}	a_{12}	…	a_{1s}	…	a_{1n}	1	0	…	0	…	0	b_1
c_{i2}	x_{i2}	a_{21}	a_{22}	…	a_{2s}	…	a_{2n}	0	1	…	0	…	0	b_2
…	…	…	…	…	…	…	…	…	…	…	…	…	…	…
c_{ii}	x_{ii}	a_{i1}	a_{i2}	…	a_{is}	…	a_{im}	0	0	…	1	…	0	b_i
…	…	…	…	…	…	…	…	…	…	…	…	…	…	…
c_{im}	x_{im}	a_{m1}	a_{m2}	…	a_{ms}	…	a_{mn}	0	0	…	0	…	1	b_m
	c_r :	c_1	c_2	…	c_s	…	c_n	0	0	…	0	…	0	
	Δ_r :	Δ_1	Δ_2	…	Δ_s	…	Δ_n	Δ_{n+1}	Δ_{n+2}		Δ_{n+1}	…	Δ_{n+m}	

Con Δ_r indichiamo le valutazioni della funzione obiettivo, questo valore indica se la soluzione trovata è ottima o può essere migliorata. Nel caso di un problema di massimo

la soluzione può essere migliorata quando $\Delta_r \leq 0$, mentre per un problema di minimo quando $\Delta_r \geq 0$. La formula per calcolare Δr è la seguente :

$$\Delta_r = \sum_{r=1}^{n+m} \left(c_r - \sum_{h=1}^{m} a_{hr} * c_{ih} \right)$$

In un problema di massimo tra tutte le valutazioni positive si sceglie quella con valore maggiore, al contrario, nel caso di un problema di minimo sceglieremo la minore, in entrambi i casi tale valore rappresenterà la variabile entrante. Se la variabile entrante è x_s si determina la variabile uscente calcolando i rapporti $\frac{b_i}{a_{is}}$ con $0 \leq i \leq$ m. Se i valori a_{is} sono minori o uguali a 0 il problema tende all'infinito, altrimenti se sono positivi il valore più piccolo del rapporto indica la variabile che deve uscire. Trovato questo valore, a_{is} rappresenta il pivot della trasformazione; se diverso da 1 bisogna renderlo uguale a 1 dividendo tutta la riga per a_{is}. Usando il pivot vengono generati gli zeri nella sua colonna. Fatto ciò viene ricostruita la tabella, scambiando la variabile uscente con la variabile entrante. Infine sono ricalcolate le valutazioni, fino al raggiungimento di una soluzione ottima. Ci sono poi casi in cui non riusciamo a trasformare i vincoli di $\geq$ in uguaglianze e quindi bisogna introdurre delle variabili artificiali. Queste ultime sono della forma x^a e prendono il posto delle variabili di scarto, sia nel sistema che nella funzione obiettivo. Nella funzione, nel caso di un problema di minimo, le variabili artificiali vengono precedute da +M (big M)[15], mentre in un problema di massimo da –M (small M). Bisogna eliminare le variabili artificiali poiché non rispettano i vincoli e, se restano fino alla fine dell'algoritmo del simplesso, allora il problema non è risolvibile.

2.3.2 Metodi di branch and bound

Quando ci troviamo di fronte a problemi di programmazione lineare intera (PLI) molto complessi è utile utilizzare algoritmi di branch and bound, così da preservare l'integralità delle variabili di decisione e risolverli più facilmente. La maggior parte dei risolutori di PLI fa uso di questi metodi cercando una soluzione ottima del problema e risolvendo una sequenza di problemi rilassati che permettono alcuni valori

frazionari(LP), ma risultano più semplici del problema originale. Prendiamo ad esempio un problema del tipo:

$$\max z = 8x_1 + 5x_2$$
$$\begin{cases} x_1 + x_2 \leq 6 \\ 9x_1 + 5x_2 \leq 45 \\ x_1, x_2 \geq 0 \; con \; x_1, x_2 \; interi \end{cases}$$

Il primo vincolo che potremmo eliminare è il vincolo d'integralità $x_1, x_2 \; interi$. Possiamo indicare il sotto-problema LP risultante con il nome SottoProblema1. Proviamo a risolvere questo problema con qualsiasi algoritmo per la programmazione lineare. Se la soluzione ottima è una soluzione intera, allora essa è anche soluzione ottima per il problema di partenza ILP, altrimenti il valore della soluzione ottima trovata è un limite superiore alla soluzione del problema di partenza. In questo caso il problema non ha soluzione intera quindi c'è bisogno di un altro passaggio. Scegliamo in modo arbitrario una variabile di decisione x_1. Esaminando il problema notiamo che in qualsiasi soluzione al problema iniziale $x_1 \leq 3$ e $x_1 \geq 4$. Partendo da questa osservazione possiamo ulteriormente risalassare il problema di partenza con due sotto-problemi: il primo composto dal SottoProblema1 + $x_1 \geq 4$ e il secondo composto dal SottoProblema1 + $x_1 \leq 3$. Procediamo alla risoluzione di questi sotto-problemi.

Dopo aver risolto un singolo sotto-problema:

1. Se la soluzione ottimale ha tutte le variabili di decisione assegnate a valori interi, possiamo confrontare questo valore con il miglior valore trovato finora. Se è peggiore allora scartiamo il sotto-problema, altrimenti memorizziamo questo risultato che potrebbe aiutarci a trovare una soluzione in futuro.
2. Se la soluzione ottimale ha le variabili di decisione che non sono intere allora si procede con un ulteriore diramazione su una di queste variabili.

Quando non ci sono più sotto-problemi da risolvere, il sotto-problema con le variabili di decisione intere avente il miglior valore obiettivo, è la soluzione ottimale al problema ILP.

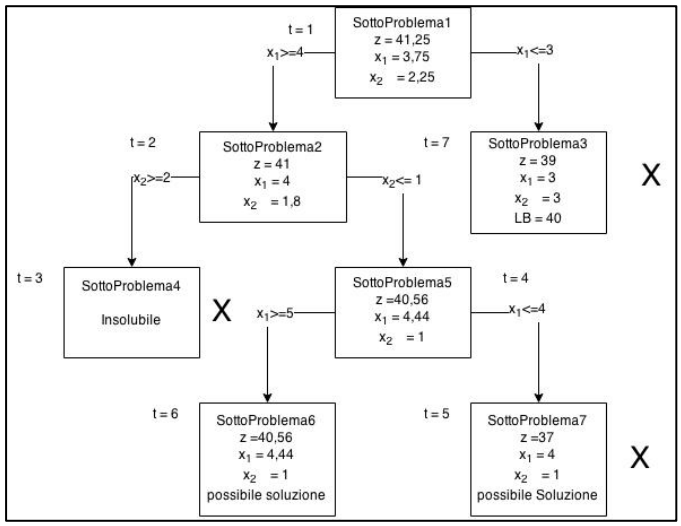

Figura 2.3 Albero delle soluzioni generate

In Figura 2.3 vediamo un albero in cui ogni nodo si riferisce ad un sotto-problema. I nodi sono collegati con archi. I vincoli associati ad ogni nodo dell'albero rappresentano i vincoli associati al problema rilassato, più i vincoli ereditati dagli archi che collegano il Sottoproblema1 al nodo. La croce vicino ad alcuni nodi indica che il nodo è stato scartato. L'etichetta t indica la sequenza temporale con cui vengono esaminati i vari nodi. Come possiamo vedere il Sottoproblema6 rappresenta la soluzione ottima per il problema menzionato sopra. I problemi di programmazione lineare intera spesso sono molto più difficili da risolvere rispetto a quelli di programmazione lineare. È molto difficile predire la difficoltà di un problema di programmazione lineare intera; capita comunque che problemi con un centinaio di variabili siano difficili da risolvere, mentre problemi con un numero di variabili molto più elevato siano di facile risoluzione. Con questo tipo di problemi è molto importante il modello del problema e la scelta del risolutore.

2.4 Strumenti per la programmazione lineare

Esistono in letteratura molti risolutori e pacchetti software creati ad hoc per modellare e risolvere problemi di programmazione lineare. Uno dei più usati è sicuramente AMPL (A Mathematical Programming Language), sviluppato nei laboratori Bell, che fornisce un linguaggio di alto livello per modellare i problemi, e utilizza dei solver come CPLEX, FORT MP, per raggiungere delle soluzioni. Un'alternativa open source ad AMPL è GLPK (GNU Linear Programming Kit) che è in grado di risolvere problemi di

grandi dimensioni. È un insieme di funzioni scritte in ANSI C, organizzate in librerie. Un aspetto molto importante di GLPK sta nel fatto che supporta GNU MathProg modelling language, un sottoinsieme del linguaggio AMPL. GNU MathProg è un linguaggio di modellazione che consente di descrivere modelli matematici di programmazione lineare. La descrizione consiste in un insieme di definizioni di variabili, vincoli e di dati definiti dall' utente[7]. Un ambiente di sviluppo integrato proprietario, adatto alla programmazione lineare, è IBM ILOG CPLEX Optimization studio. Quest'ultimo rappresenta un ottimo tentativo d'unione tra la parte riguardante la modellazione in programmazione matematica e i software di constraint programming. ILOG incorpora un linguaggio di programmazione e ottimizzazione (OPL) con diversi motori e strumenti per la programmazione a vincoli, programmazione lineare e altri problemi di programmazione matematica[19]. ILOG si basa sul linguaggi come AMPL e GAMS.
In questa tesi il linguaggio di modellazione usato è stato GLPK.

2.5 In breve

Usando la programmazione lineare, è possibile modellare e risolvere i problemi di ottimizzazione. Per formulare un problema di programmazione lineare abbiamo bisogno di una funzione obiettivo e di un insieme di vincoli, che non sono altro che uguaglianze o disuguaglianze lineari. La programmazione lineare fa parte della classe dei problemi NP-ardui. Per questo tipo di problemi, a volte ci accontentiamo di una soluzione che soddisfi i vincoli anche se non massimizza o minimizza la funzione obbiettivo. In realtà, anche individuare una soluzione ammissibile è NP-arduo, ma grazie all'ausilio del risolutore qui scelto, cioè GLPK, riusciamo a trovare una soluzione ammissibile al problema dell'orario in tempi ragionevoli. Per risolvere i problemi di ottimizzazione è molto utile il branch and bound, con l'ausilio del metodo del simplesso, che dividendo il problema in sotto-problemi rende più semplice il raggiungimento di una soluzione. In letteratura e in rete esistono diversi programmi, librerie, ed ambienti di sviluppo utili alla formalizzazione e la risoluzione di problemi di ottimizzazione, purtroppo però non tutti sono accessibili.
Dopo quest'introduzione agli strumenti di sviluppo utili per il lavoro di tesi, nel prossimo capitolo tratteremo del problema della gestione dell'orario, in particolar modo

in ambito accademico e verrà descritto un modello adatto alla facoltà d'Informatica dell'Università di Bologna, di cui si può trovare una soluzione soddisfacibile in tempo ragionevole utilizzando GLPK.

Capitolo 3

Il problema del timetabling nel corso di laurea in informatica

In questo capitolo descriviamo l'organizzazione del corso di laurea in informatica della scuola di Scienze presso l'Università di Bologna, con riferimento all'anno accademico 2014/2015. Sarà posta particolare attenzione sull'offerta formativa e sul corpo docente. In seguito presenteremo la formalizzazione matematica del problema dell'orario, basata sull'articolo di S.Daskalaki, T.Birvbas e E.Housos, nell'articolo *An integer programming formulation for a case study in university timetabling* [5], descrivendone gli aspetti peculiari. Infine tenteremo di risolvere il problema dell'ora libera per il pranzo proponendo due soluzioni, che verranno valutate una alla volta, al fine di garantire la risoluzione ottimale.

3.1 Struttura del corso di laurea in informatica

La scuola di Scienze dell'Università di Bologna propone tre corsi di laurea in informatica: due corsi di laurea triennali, rispettivamente denominati "Informatica" e "Informatica per il Management", e un unico corso di laurea magistrale, diviso in tre curricula. Il corpo docente è composto da quarantasette[7] professori, cui sono assegnati sia corsi per le lauree triennali, sia corsi relativi alla laurea magistrale. La laurea triennale in Informatica è composta da 180 crediti formativi universitari, che da ora in poi denoteremo con CFU, di cui: quattro sono attinenti al tirocinio; cinque riguardano la prova finale; dodici crediti sono a scelta dello studente e i restanti crediti sono suddivisi su diciotto esami fondamentali. La laurea triennale in Informatica per il Management, anch'essa prevede 180 CFU di cui: tre per la prova finale; cinque per il tirocinio; dodici a scelta dello studente e il resto suddivisi su ventidue corsi. La magistrale invece, come già esplicato, è divisa in tre curricula: Curriculum A, "Linguaggi e fondamenti"; curriculum B, "Informatica per il management" e curriculum C "Sistemi e reti". Tutti e tre i curricula prevedono 120 CFU, di cui: ventiquattro sono relativi alla prova finale; sei a scelta in ambito informatico; dodici a scelta dello studente; dodici a scelta in

[7] Fonte : http://corsi.unibo.it/informatica-magistrale/Pagine/Docenti.aspx, http://corsi.unibo.it/Laurea/InformaticaManagement/Pagine/Docenti.aspx, http://corsi.unibo.it/Laurea/Informatica/Pagine/Docenti.aspx

ambito matematico-fisico e il resto sono suddivisi su tutti gli esami fondamentali per ogni curricula. L'anno accademico è diviso in due semestri, in cui vengono svolte le lezioni, e tre sessioni d'esame. Le lezioni si svolgono dal lunedì al venerdì e la scuola ha a disposizione nove aule in cui possono essere svolti i corsi. La commissione orario ha il compito di creare l'orario per ogni semestre, in modo da soddisfare i vincoli imposti dai professori e dagli organi accademici.

Nel prossimo paragrafo presenteremo i principali componenti del modello di programmazione lineare.

3.2 Specifiche del modello

In questo paragrafo esploreremo le caratteristiche principali del modello proposto, partendo dai parametri di base, per poi passare ai vincoli e alla funzione obiettivo.

3.2.1 Parametri fondamentali

Il modello è stato costruito su sei parametri fondamentali :

1. **Giorno della settimana**: in cui può essere tenuto un corso o parte di esso. Denoteremo questo insieme con G = {1,2,3,4,5} (1= lunedì, 5= venerdì)

2. **Timeslot**: descrive le unità temporali in cui può essere programmata una lezione. Nel nostro caso, un timeslot è un periodo della durata di un'ora. Solitamente nel corso di laurea in Informatica le lezioni iniziano alle 8:30 e terminano alle 19:30. Indicheremo l'insieme dei timeslot con la lettera T. Ad esempio T={1,2,...,11} per indicare i timeslot dalle 8:30 alle 19:30.

3. **Anno di corso** : rappresenta l'anno di corso (ad es. primo anno magistrale) a cui è rivolto un determinato corso. L'insieme di tutti gli anni di corso lo denoteremo con la lettera L. Potremmo definirlo come L = {anno1,anno2,…anno|L|}.

4. **Professore**: con questo parametro indichiamo il professore che terrà corsi nel corso di un semestre. Nel nostro caso assumiamo che l'assegnamento di un

corso ad un professore avvenga prima della creazione dell'orario da parte degli organi accademici. Sarà il professore a decidere, in base ai crediti del corso, quante ore di lezione svolgere. Un professore può decidere in quante sessioni settimanali dividere il suo corso e la durata di ogni sessione, ad esempio: se un professore ha un corso di sei ore può decidere di dividerlo in due sessioni da tre ore, oppure in tre sessioni da due ore, e cosi via. Denoteremo l'insieme dei professori con la lettera P. Ad esempio P = {prof1,prof2,…,prof|P|}

5. **Corso**: rappresenta un corso che viene offerto a ciascun gruppo di studenti appartenenti ad un determinato anno di corso. Denoteremo questo insieme con la lettera C. Ad esempio C={corso1,corso2,…,corso|C|}, in questo insieme troviamo sia i corsi fondamentali che i corsi a scelta, in modo tale che non potranno mai sovrapporsi.
6. **Aula**: rappresentano le aule messe a disposizione dalla scuola, in cui è possibile svolgere le lezioni. L'insieme di tutte le aule disponibili è rappresentato dalla lettera A . Ad esempio A = {aula1,aula2,…,aula|N|}

L'approccio usato prevede due insiemi di variabili binarie. Il primo lo chiameremo "insieme delle variabili di base" e lo denoteremo con $x_{g,t,l,p,c,a}\ con\ g \in G, l \in L, t \in T, p \in P, c \in C, a \in A$. Una variabile appartenente a questo insieme varrà 1 quando il corso c tenuto dal professore p per l'anno di corso l viene posizionato nel t-esimo slot del g-esimo giorno nell'aula a . Il secondo lo chiameremo "insieme delle variabili ausiliarie" e lo denoteremo con $y_{g,l,h_v,c,a}$ dove $g \in G, l \in L, c \in C, a \in A$,$h_v \in \mathbb{N}$. Le variabili appartenenti a questo insieme avranno valore 1 quando il corso c ha una sessione che richiede h_v timeslot consecutivi ed è assegnato al giorno g per l'anno di corso l nel aula a.

3.2.2 I vincoli del modello

In questa sezione esporremo i vincoli utilizzati nel modello descritto. Per ogni vincolo viene espressa la regola corrispondente e la sua formulazione matematica. I vincoli sono divisi in sei gruppi: i primi cinque gruppi rappresentano i vincoli presenti nell'articolo sopracitato, mentre l'ultimo gruppo comprende il vincolo aggiunto per l'inserimento di un'ora libera, per permettere a studenti e professori di pranzare.

1. **Vincoli di unicità**

Questi vincoli garantiscono che non ci siamo conflitti all'interno dell'orario affinché venga rispettata la regola citata nel capitolo 1.

1)Unicità del professore: per un singolo timeslot ad ogni professore deve essere assegnato al massimo un corso, relativo a un anno di corso e tenuto in una sola aula. Il vincolo viene espresso come :

$$\forall g \in G, \forall t \in T, \forall p \in P \ , \sum_{l \in L_p} \sum_{c \in C_{lp}} \sum_{a \in A_{cl}} x_{g,t,l,p,c,a} \leq 1.$$

Dove L_p rappresenta l'insieme degli anni di corso per cui p ha dei corsi, C_{lp} è l'insieme dei corsi tenuti dal professore p per l'anno di corso l, A_{cl} rappresenta le aule che possono contenere gli studenti dell'anno l che seguono il corso c.

2)Unicità anno di corso: per ogni anno accademico deve essere assegnato al massimo un corso, un docente e un'aula, relativi ad un unico semestre. Questo vincolo assicura che non ci siano conflitti tra i corsi, che siano essi a scelta od obbligatori:

$$\forall l \in L \, , \forall g \in G, \forall t \in T, \sum_{p \in P_{lg}} \sum_{c \in C_{lp}} \sum_{a \in A_{cl}} x_{g,t,l,p,c,a} \leq 1.$$

Dove P_{lg} rappresenta l'insieme dei professori che sono disponibili nel giorni g e hanno almeno un corso durante l'anno di corso l.

3) Unicità aula: per un unico timeslot, ad ogni aula deve essere assegnato al massimo un corso, un professore e un solo anno di corso:

$$\forall a \in A, \forall g \in G \ , \forall t \in T_{gpa}, \sum_{L \in L} \sum_{P \in P_{lg}} \sum_{c \in C} x_{g,t,l,p,c,a} \leq 1.$$

Dove T_{gpa} rappresenta l'insieme dei timeslot del giorno g, in cui il professore p e l'aula a sono disponibili. P_{lg} rappresenta l'insieme dei professori disponibili nel giorno g, che insegnano almeno un corso per l'anno di corso l.

2. Vincoli di completezza

Questi vincoli ci assicurano che venga rispettata la regola di completezza, enunciata nel capitolo 1.

1)Completezza dei corsi: tutti i corsi presenti nel piano didattico di ogni studente devono essere presenti nell'orario e con il giusto numero di ore di lezione

$$\forall l \in L, \sum_{p \epsilon P_l} \sum_{c \epsilon C_{lp}} \sum_{a \epsilon A_{cl}} \sum_{g \epsilon G_{pa}} \sum_{t \epsilon T_{gpa}} x_{g,t,l,p,c,a} = a_l$$

Dove a_l rappresenta il numero totale di sessioni programmato per l-esimo anno di corso. P_l rappresenta l'insieme contenete i professori che hanno almeno un corso per l'anno di corso l. G_{pa} rappresenta l'insieme dei giorni in cui l'aula a e il professore p sono disponibili. T_{gpa} è l'insieme dei timeslot nel giorno g in cui il professore p e l'aula a sono liberi.

2) Completezza dei professori: ad ogni professore devono essere assegnati tutti i timeslot da lui richiesti

$$\forall p \in P, \sum_{l \epsilon L} \sum_{c \epsilon C_{lp}} \sum_{a \epsilon A_{cl}} \sum_{g \epsilon G_{pa}} \sum_{t \epsilon T_{gpa}} x_{g,t,l,p,c,a} = s_p$$

Dove s_p è il numero totale di ore di lezione richieste da un professore p, in accordo con i crediti che caratterizzano il corso.

3. Vincoli di consecutività

Con questi vincoli permettiamo ai professori di suddividere le proprie ore di lezione in sessioni, da distribuire nell'arco della settimana. Nel modello proposto, è il professore che decide come suddividere le sue ore di lezione. Per facilitare il modello viene inserita una nuova variabile $y_{g,p_v,l,h_v,c,a}$ associata ad ogni corso. Come detto in precenza, questa variabile contiene le preferenze del professore, e può essere formalizzata in due passaggi:

1) AssegnamentoOre: ad un corso c che richiede una sessione di h_v ore consecutive, bisogna assegnare h_v ore in un dato giorno:

$$\forall g \in G, \forall l \in L, \forall P \in P_{lg} \forall c \in C_{lp} \forall\, a \in\, A_{cl}$$
$$\sum_{t \in TP_{gpa}} x_{g,t,l,p,c,a} - \sum_{h_v \in H_c} \left(y_{g,l,h_v,c,a} \,*\, h_v\right) = 0.$$

Dove TP_{gpa} è l'insieme dei timeslot per il giorno g in cui il professore p e l'aula a sono liberi, H_c è l'insieme delle lunghezze per ogni sessione decisa dal professore, mentre P_c è l'insieme delle sessioni in cui il professore ha diviso il corso.

2)OreConsecutive: se ad un corso sono state assegnate h_v ore, queste ore devono essere consecutive

$$\forall g \in G, \forall l \in L, \forall p \in P_{lg}, \forall c \in C_{lp}, \forall\, a \in\, A_{cl}, \forall\, t_a \in\, FTP_{gpa}\, \forall\, h_v \in\, H_c \,\wedge\, h_v > 1$$
$$\forall\, t \in \{1, \dots, h_v - 1\}, x_{g,t_a,l,p,c,a} - x_{g,t_a+t,l,p,c,a} \leq 0 (a)$$

$$\forall g \in G, \forall l \in L, \forall p \in P_{lg}, \forall c \in C_{lp}, \forall\, a \in\, A_{cl}, \forall\, t \in\, T_{gpa}\, \forall\, h_v \in\, H_c \,\wedge\, h_v > 1$$
$$\forall\, t_s \in \{2, \dots, h_v\}, - x_{g,t,l,p,c,a} + x_{g,t+1,l,p,c,a} - x_{g,t+t_s,l,p,c,a} \leq 0 (b)$$

FTP_{gpa} rappresenta l'insieme delle ore iniziali di una sessione in un giorno g in cui il professore p e la classe a sono disponibili. Le due disequazioni a e b sono necessarie affinché se un'ora è stata assegnata ad un determinato timeslot, le restanti $h_v - 1$ ore

devono essere consecutive. L'equazione a è necessaria per la prima ora che viene assegnata, mentre l'equazione b serve per quelle a seguire.

4. Vincoli di ripetività

Questi vincoli assicurano che non ci siano sessioni ripetute della stessa materia durante lo stesso giorno.

1) Sessioni: questo primo vincolo assicura che durante la settimana ci siano il giusto numero di sessioni richieste dal professore:

$$\forall l \in L, \forall c \in C, \forall h_v \in H_c, \forall a \in A_{ct}, \sum_{g \in G_{pa}} y_{g,l,h_v,c,a} = b_{c,h_v}$$

Dove b_{c,h_v} è il numero di sessioni richieste da un professore all'interno di una settimana per il corso c.

2) Unicità delle sessioni: le sessioni in cui è diviso un corso devono essere distribuite equamente durante la settimana, cioè nello stesso giorno non devono esserci due sessioni dello stesso corso :

$$\forall l \in L, \forall c \in C, \forall h_v \in H_c, \forall a \in A_{ct}, \forall g \in G, \forall p \in P_{lc} \sum_{h_v \in H_c} y_{g,l,h_v,c,a} \leq 1$$

5. Vincolo di pre-schedule

Questo vincolo serve al docente per specificare l'assoluta necessità di tenere una lezione in un dato timeslot, in uno specifico giorno, in una determinata aula:

$$\forall (g,t,l,p,c,a) \in PRA, x_{g,t,l,p,c,a} = 1$$

PRA rappresenta l'insieme di tutti gli assegnamenti fatti a priori, in cui si specifica che un corso c tenuto dal professore p per l'anno di corso l è pre-assegnato al time slot t del giorno g.

6. Vincoli per la pausa pranzo

Questo insieme contiene i vincoli aggiunti al modello affinché all'interno dell'orario ci sia un'ora libera, in modo da permettere agli studenti, e al corpo docente, di fare una pausa per il pranzo. Per risolvere questo problema sono state create due soluzioni: la prima, che chiameremo "*pranzo come lezione*", tratta l'ora di pranzo come se fosse una lezione di un corso, della durata di un solo timeslot; la seconda, che identificheremo con "*Pranzoslot"*, invece fa in modo che gli slot desiderati siano liberi. È stata scelta la soluzione che avesse un impatto minore sul modello, sia in termini di tempo di calcolo che di memoria utilizzata.

1)Pranzo come lezione: quest'approccio prevede l'inserimento, all'interno del modello, di una lezione chiamata "pranzo", per ogni anno di corso. Ovviamente, per rispettare i vincoli di unicità bisogna inserire per ogni anno di corso una lezione pranzo, identificandola in modo diverso. Nel nostro caso, la lezione prende il nome di *pranzo-annodicorso*. Ad esempio: un pranzo per il primo anno della laurea magistrale verrà indicato con pranzo-1mag. Inoltre, nell'insieme delle aule dev'essere inserita un'aula chiamata "pranzo", che serve per ospitare questo tipo "speciale" di lezione. Per far sì che queste lezioni "pranzo" avvengano nell'orario desiderato bisogna inserire un vincolo per ogni *pranzo-annodicorso*. I vincoli da aggiungere hanno il seguente formato:

$$\forall g \in G, \sum_{a \in A} x_{g,4,l,p,pranzo,a} + x_{g,5,l,p,pranzo,a} = 1$$

Il termine pranzo identifica il corso *pranzo-annodicorso* a cui è relativo il vincolo.

2) Pranzoslot: questo vincolo, senza l'aggiunta di ulteriori corsi e ulteriori aule, fa in modo che all'interno dei timeslot che rappresentano l'ora di pranzo, in un giorno ci sia almeno un'ora di pausa:

$$\forall g \in G, \forall l \in L, \sum_{a \in A} \sum_{c \in C_l} x_{g,4,l,p,c,a} + \sum_{a \in A} \sum_{c \in C_l} x_{g,5,l,p,c,a} \leq 1$$

Questo vincolo impone che nell'aula a non ci può essere contemporaneamente lezione nello slot 4 e nello slot 5, ma solo in uno di essi. È possibile modificare i timeslot in cui si vuole fissare la pausa pranzo.

Per effettuare un confronto tra i due approcci sopradescritti abbiamo effettuato alcuni esperimenti valutando, quando il problema ha soluzione, sia il tempo necessario per risolverlo, che la memoria utilizzata dal risolutore per completare la computazione. Gli esperimenti sono stati eseguiti utilizzando *glpsol*, risolutore open source per la programmazione lineare intera, che fa parte del pacchetto GLPK, nella versione 4.55. Poiché *glpsol* impiega molto tempo per trovare una soluzione ottima al problema dell'orario, si è scelto di effettuare gli esperimenti calcolando la soddisfacibilità del modello che *glpsol* computa molto più velocemente. Sono stati utilizzati i dati relativi ai corsi tenuti durante il primo e il secondo ciclo di lezioni dell'anno accademico 2014/2015, per tutti i corsi di laurea. Abbiamo svolto un totale di dodici esperimenti, relativi ai singoli corsi di laurea. Tenendo presente la suddivisione sopracitata (cfr 3.1) della scuola di Scienze dell'Università di Bologna, abbiamo considerato in primis la sola laurea triennale in Informatica, poi le due triennali in Informatica per il Managment e Informatica; infine abbiamo aggiunto la laurea magistrale. Sono stati considerati tutti i corsi per ogni singolo ciclo di lezioni, come mostrato nella Tabella 1, dove viene rappresentato il numero di corsi per ogni ciclo, suddivisi per classi di laurea. I parametri che riguardano il numero di timeslot e di sessioni per ogni professore sono costanti: il numero di timeslot è stato impostato a 11, il numero massimo di timeslot di cui si compone una sessione è pari a cinque, e altrettante sono le sessioni in cui un professore può dividere il corso. Le preferenze di ogni professore, riguardanti l'orario in cui svolgere le lezioni, vengono generate casualmente per ogni run e per ogni esperimento in numero costante, che è stato impostato a venticinque, in modo che il problema sia soddisfacibile nella maggior parte dei casi.

Tipo di Laurea	**I Ciclo**	**II Ciclo**
Triennale info	15	12
Triennale info + managment	28	21
Completo	42	34

Tabella 1 Parametri utilizzati per analizzare i vincoli del pranzo

Per ogni esperimento sono state eseguiti cinquanta run variando le preferenze di ciascun professore. Dopo aver raccolto i risultati per ogni esperimento è stata fatta una media,

sia per quanto riguarda il tempo di esecuzione espresso in secondi, che per quanto riguarda la memoria utilizzata, espressa in MB.

I risultati degli esperimenti relativi al primo e al secondo ciclo, relativi al tempo di esecuzione, vengono presentati in Figura 3.1 e in Figura 3.2; mentre i risultanti attinenti alla memoria vengono rappresentati in Figura 3.3 e in Figura 3.4

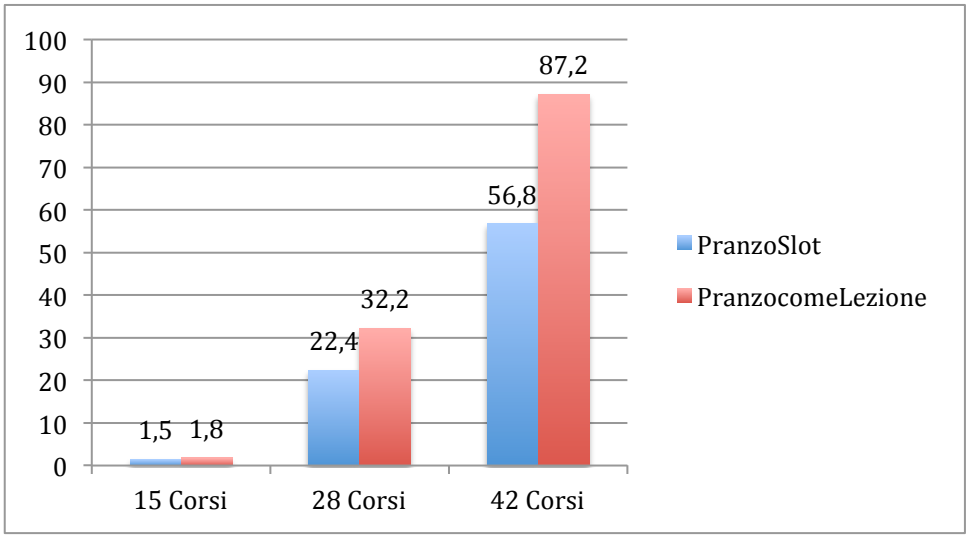

Figura 3.1 Grafico relativo ai risultati riguardanti il tempo di esecuzione utilizzando i dati del primo ciclo

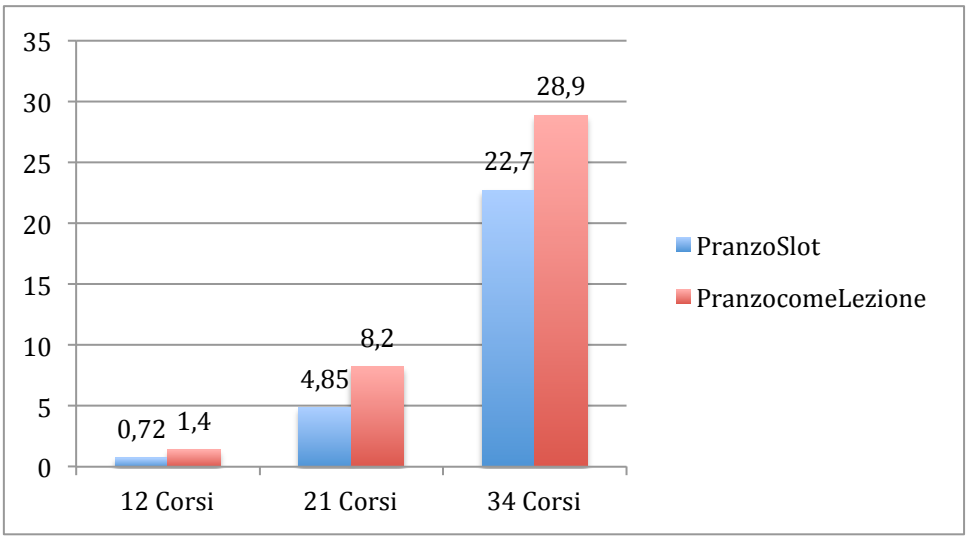

Figura 3.2 Grafico relativo ai risultati riguardanti il tempo di esecuzione utilizzando i dati del secondo ciclo

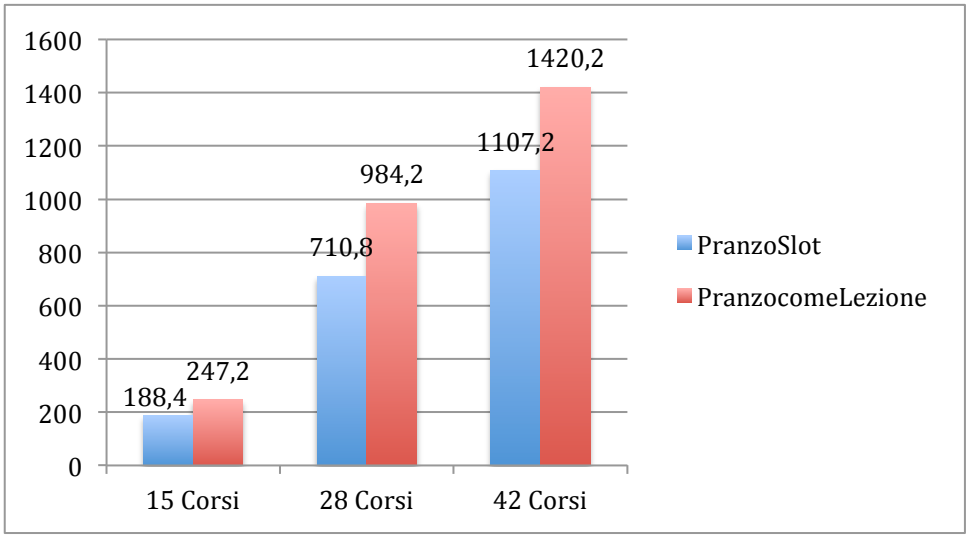

Figura 3.3 Grafico relativo ai risultati riguardanti la memoria utilizzata utilizzando i dati del primo ciclo

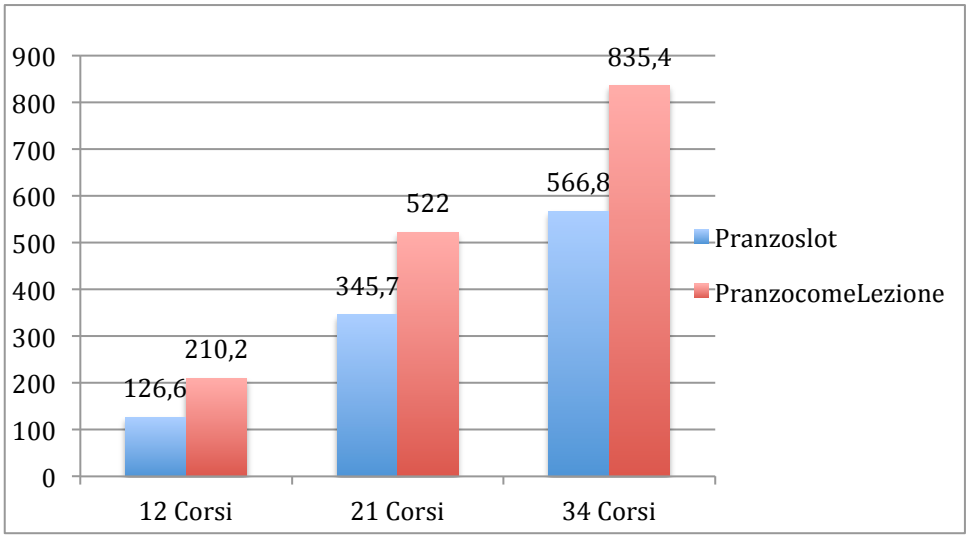

Figura 3.4 Grafico relativo ai risultati riguardanti il tempo di esecuzione utilizzando i dati del secondo ciclo.

Nei grafici riguardanti il tempo d'esecuzione sono stati rappresentati sull'asse delle ascisse il numero di corsi, mentre sull'asse delle ordinate il tempo espresso in secondi. Nei grafici riguardanti l'occupazione della memoria: sull'asse delle ascisse troviamo il numero dei corsi, mentre sull'asse delle ordinate è raffigurato il consumo della memoria espresso in Mb.

Come si può notare dai grafici, all'aumentare del numero dei corsi, aumenta anche il consumo di memoria e il tempo di esecuzione, poiché all'aumentare dei corsi aumenta anche il numero di variabili. Esaminando questi grafici, possiamo dedurre che al variare del numero di corsi, sia in termini di memoria che in termini di tempo d'esecuzione, il modello con il vincolo Pranzoslot risulta essere il più efficiente, poiché utilizza quasi

metà della memoria impegnata da *pranzo come lezione* e risulta molto più veloce nella computazione. Quindi è stato scelto il vincolo *Pranzoslot* per risolvere il problema dell'ora di pranzo.

3.2.3 La funzione obiettivo

Dopo aver delineato tutti i vincoli del nostro problema, affinché non vengano violate le regole principali di un orario, passiamo alla definizione della funzione obiettivo. Dato che alcuni assegnamenti sono preferibili rispetto ad altri, la nostra funzione obiettivo sarà:

$$Maximize \left\{ \sum_{l \in L} \sum_{p \in P} \sum_{c \in C} \sum_{a \in A} \sum_{g \in G} \sum_{t \in t} c_{gtlpca} * x_{gtlpca} \right\}$$

Il nostro obiettivo è massimizzare questa funzione, in cui $c_{gtlpca} * x_{gtlpca}$ rappresenta l'assegnazione di un costo relativo al c al t-esimo periodo del giorno g.

3.3 In breve

In questo capitolo abbiamo esaminato il modello su cui è stata costruita l'interfaccia grafica web. Tale modello contiene sia i vincoli hard che i vincoli soft e si adatta perfettamente alle esigenze della scuola di Informatica. Gli utenti di questo modello sono la segreteria: che provvederà ad impostare i parametri relativi alle aule, i corsi, la capacità delle aule e i parametri base del sistema, come le ore di lezione che possono esserci durante una giornata; e i professori, che potranno selezionare le ore in cui preferiscono fare lezione. Un passo in avanti è stato fatto rispetto al modello di base, è stato aggiunto il vincolo per l'ora di pranzo proponendo due soluzioni al problema e scegliendo la soluzione più "conveniente" in termini di consumo di memoria e velocità computazionale, che è risultata essere la funzione *Pranzoslot*. Il risolutore utilizzato è *glpsol*, che non è in grado di risolvere il problema dell'orario completo in un tempo ragionevole e per questo si è preferito utilizzare la risoluzione SAT del problema, che il

risolutore riesce a portare a termine in tempi molto più ragionevoli. Nel prossimo capitolo introdurremo i concetti di base dell'interfaccia grafica web e mostreremo tutte le sue funzionalità.

Capitolo 4

L'interfaccia

In questo capitolo presenteremo i dettagli dell'implementazione del software per risolvere il problema dell'orario. Il software è basato sul modello descritto nel capitolo precedente e riesce a risolvere il problema della soddisfacibilità e dell'ottimalità dell'orario. Inizialmente presenteremo brevemente l'implementazione del modello in GLPK, in seguito passeremo all'analisi dei requisiti e alla descrizione delle linee guida utilizzate per lo sviluppo della GUI (Graphical User Interface). Infine presenteremo l'implementazione della GUI e la modalità d'installazione del software.

4.1 Implementazione in GLPK

Il modello descritto nel capitolo precedente è basato su una implementazione inizialmente sviluppata da M. Marzolla utilizzando GLPK, una libreria software, scritta in ANSI C, ottimizzata per la risoluzione di problemi di programmazione lineare. Questa libreria implementa il metodo del simplesso e un metodo del punto interno per la soluzione di problemi lineari, mentre utilizza metodi di branch and bound per risolvere problemi più complessi, come la programmazione lineare intera e mista. La libreria è open source e liberamente scaricabile dal sito del progetto[8]. GLPK utilizza un sottoinsieme di istruzioni di AMPL[9] (A Mathematical Programming Language) per la definizione del modello. Tipicamente la struttura di un programma scritto con GLPK ha due componenti fondamentali :

- *un file del modello:* descrive la struttura logica del modello (variabili, insiemi, vincoli). La sua estensione deve essere .mod
- *un file dei dati:* contiene i dati da dare in input al modello. La sua estensione deve essere .dat

Questa divisione è molto importante, poiché possiamo tenere separati i file riguardanti il modello da quelli relativi ai dati, così da poter modificare i dati in input senza dover necessariamente modificare il modello.

[8] http://www.gnu.org/software/glpk/

[9] linguaggio sviluppato dai laboratori bell per descrivere e risolvere grossi e complicati problemi di programmazione matematica.

Gli elementi fondamentali di un programma in GLPK sono:

- gli insiemi: descrivono l'ambiente in cui il problema deve essere circoscritto, cioè il suo dominio e la sua dimensione. Gli insiemi vengono indicati con la parola chiave *set*.
- i dati: rappresentano valori numerici che definiscono in dettaglio il problema che si vuole affrontare. Vengono dichiarati una volta sola, al contrario delle variabili che vengono modificate dal risolutore. I dati sono definiti dalla parola chiave *param*.
- le variabili: sono le grandezze che descrivono la soluzione al problema. Il loro valore deve essere determinato dal risolutore. Vengono dichiarate con la parola chiave *var*.
- la funzione obiettivo: specifica la grandezza del problema di cui si vuole trovare il valore ottimale. Viene introdotta dalle parole chiave *maximize* e *minimize*.
- i vincoli: servono per distinguere le soluzioni ammissibili da quelle inammissibili. Vengono introdotti dalla parola chiave *subject to* o *s.t.* .

Il file del modello e il file dei dati sono contenuti in Appendice A .

4.2 Analisi dei requisiti

Lo scopo di questo software è permettere alla segreteria e ai professori, afferenti al corso di studi in Informatica della Scuola di Scienze, di gestire l'orario delle lezioni, impostando vari parametri, che costituiranno il file dati da dare in input al risolutore. Il sistema è un'applicazione web based con architettura client server ed è composto da due attori principali: gli impiegati della segreteria studenti e i docenti.

4.2.1 Requisiti

Di seguito elencheremo i requisiti della nostra applicazione dividendoli per attori così da renderne più chiara la lettura.

- **Requisiti per l'attore segreteria:**

[R1] il sistema deve consentire l'inserimento di un nuovo orario.

[R1.1] Un nuovo orario è caratterizzato da aule, corsi e professori ed ha i seguenti attributi: numero massimo di blocchi, numero massimo di slot

consecutivi, numero massimo di slot in un giorno. Questi ultimi identificano rispettivamente: il numero massimo di sessioni in cui un professore può dividere un corso, il numero massimo di timeslot consecutivi che ci possono essere in una sessione e il numero massimo di ore che possono esserci in una giornata.

[R2] il sistema deve consentire all'utente di assegnare le aule ad un determinato corso affinché possano essere evitati i sovraffollamenti.

[R3] il sistema deve consentire l'inserimento di una nuova aula all'interno dell'orario.

[R3.1] ogni aula è caratterizzata da Nome, Ubicazione, Capienza, che indicano rispettivamente il nome dell'aula, il suo indirizzo, e la capienza in termini di posti disponibili al suo interno.

[R4] il sistema deve consentire l'inserimento di un nuovo corso all'interno dell' orario.

[R4.1] ogni corso è caratterizzato da NomeCorso, IdCorso, NrCrediti, Professore, anno di corso. Questi valori indicano rispettivamente: il nome del corso che si vuole inserire, un identificativo univoco del corso (formato da cinque caratteri), il numero di crediti richiesti dal corso, il nome del professore cui è stato assegnato e l'anno accademico in cui si terrà il corso.

[R5] il sistema deve consentire all'utente di salvare le informazioni inserite e calcolare l'orario sia per quanto riguarda la soddisfacibilità che la soluzione ottima.

[R6] il sistema deve consentire all'utente di interrompere in qualsiasi momento la computazione dell'orario.

[R7] il sistema deve consentire all'utente di visualizzare l'orario ed esportarlo in formato stampabile.

[R8] il sistema deve consentire all'utente di eliminare i dati inseriti all'interno dell'orario

- **Requisiti per l'attore professore:**

[R9] il sistema deve permettere al professore di inserire le informazioni riguardanti il corso che gli è stato assegnato.

[R9.1] ogni corso è caratterizzato da OreSettimanali e Suddivisione in blocchi, che rappresentano rispettivamente le ore settimanali che il professore

vuole dedicare a quel determinato corso, e la suddivisione in sessioni, con relativa durata di ogni sessione.

[R9.2] il docente deve poter esprimere le sue preferenze indicando, in base ai propri impegni personali o accademici, quali sono le ore più o meno consone allo svolgimento della lezione

[R9.3] un professore deve poter inserire ed eliminare le preferenze obbligatorie, cioè deve poter indicare l'eventuale necessità di tenere una lezione in un determinato giorno, in una determinata aula e in un determinato timeslot.

4.2.2 Struttura del database

Il database è stato progettato in base ai requisiti descritti e servirà per memorizzare i dati inseriti dai due attori principali, tali dati saranno poi dati in input al risolutore. La composizione del database sarà organizzata in sei tabelle di base e ogniqualvolta verrà inserito un nuovo professore sarà creata una nuova tabella, per memorizzarne le preferenze riguardanti i suoi corsi. Le tabelle sono :

LogIn: rappresenta la tabella che conterrà le informazioni riguardanti gli utenti che avranno accesso al sistema. La tabella LogIn si compone dei seguenti campi:

Nome: nome dell'utente registrato all'interno del sistema.

Cognome: cognome dell'utente registrato all'interno del sistema.

Username: username univoco utilizzato dall'utente per accedere al sistema

Password: password utilizzata, unitamente alla username, per avere accesso al sistema

Ruolo: per distinguere i due attori del sistema, ovvero i docenti e gli impiegati della segreteria.

Aule: questa tabella contiene tutte le informazioni relative alle aule ed è composta dai seguenti campi:

Nome: nome univoco dell'aula.

Ubicazione: indirizzo per raggiungere l'aula.

Capienza: numero di posti che possono essere occupati dagli studenti.

Corsi: questa tabella contiene tutte le informazioni riguardanti i corsi. I campi che la compongono sono:

Nomecorso: nome del corso per esteso.

Idcorso: identificativo univoco di un corso, di massimo cinque caratteri.

Professore: cognome del professore a cui è stato assegnato il corso.

Annocorso: anno accademico cui è rivolto il corso.

Oresett: rappresenta le ore settimanali che li professore decide di dedicare al corso.

Blocchi: rappresenta la suddivisione in sessioni del corso.

Crediti: in questo campo vengono inseriti il numero di crediti del corso.

Status: rappresenta lo stato del corso, che vale 0 se il professore non ha ancora espresso delle preferenze, altrimenti varrà 1.

DisponibilitàAule: questa tabella contiene l'associazione tra i corsi e le aule. Per ogni aula che viene inserita all'interno del sistema sarà aggiunta una nuova colonna alla tabella e quando viene inserito un nuovo corso allora sarà inserita una nuova riga. In una casella relativa ad una determinata aula, e ad un determinato corso, è possibile indicare la disponibilità dell'aula grazie ai parametri 1 e 0: ci sarà 1 se è possibile tenere il corso in quell'aula, in caso contrario, ci sarà 0.

ImpOrario: questa tabella contiene le impostazioni principali dell'orario ed è composta da quattro campi :

Id : identificativo univoco dell'orario.

Maxb: numero massimo di sessioni in cui il professore può dividere il corso.

Maxslotsperday: numero massimo di ore di una stessa lezione, in uno stesso giorno.

Slotsperday : numero di ore di lezione in un giorno.

Obbligatori: questa tabella contiene le preferenze obbligatorie dei professori e si compone di sei campi:

Idcorso: identificativo del corso.

Aule: nome dell'aula in cui il professore decide di tenere il corso.

Giorno: giorno in cui svolgere il corso.

Blocco: numero della sessione che si vuole fissare.

Initslot: ora d'inizio della lezione.

Selected: 1 se viene selezionato dalla segreteria per il calcolo dell'orario, altrimenti varrà 0.

4.2.3 Casi d'uso

Per ogni attore, segreteria e professore, è stato prodotto un diagramma dei casi d'uso, in cui vengono esposte le funzionalità più importanti di ognuno. Le attività principali per un attore segreteria, come mostrato in Figura 4.1, sono:

CUS.1) Effettuare il login

CUS.2) Effettuare il logout

CUS.3) Inserire/Modificare i corsi

CUS.4) Inserire/Modificare Aule

CUS.5) Inserire/Modificare le impostazioni dell orario

CUS.6) Calcolare Orario

CUS.7) Visualizzare Orario

Invece le attività principali per un professore vengono presentate in Figura 4.2 e sono:

CUSP.1) Effettuare il logIn

CUSP.2) Effettuare il logOut

CUSP.3) Inserire modificare le impostazioni dell'orario

CUSP.4) Inserire modificare preferenze orario

CUSP.5) Inserire e modificare mandatory

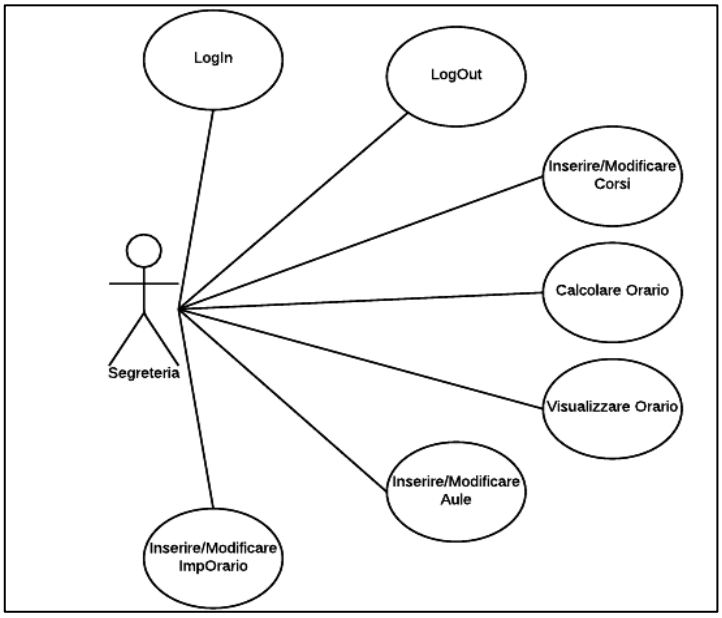

Figura 4.1 Diagramma dei Casi d'Uso per l'attore Segreteria

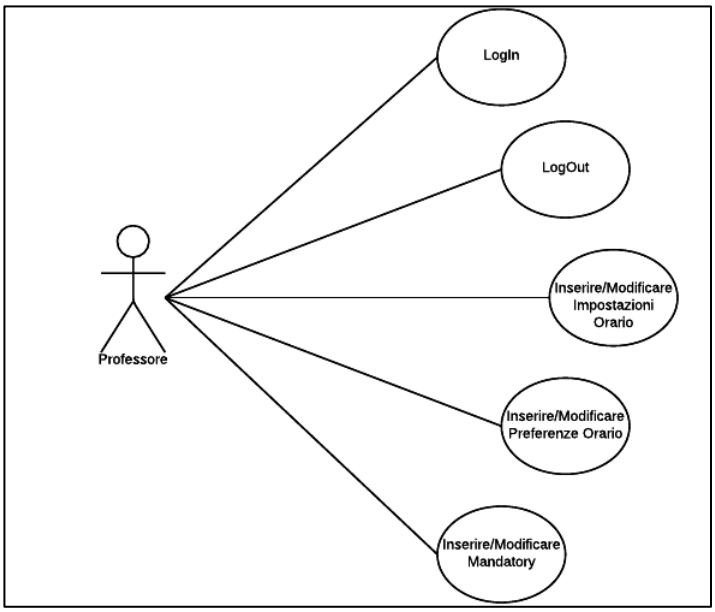

Figura 4.2 Diagramma dei Casi d'Uso per l'attore Professore

Di seguito mostriamo nel dettaglio i casi d'uso sotto forma di tabelle esplicative:

Nome Caso d'uso: CUS.1 LogIn/CUSP.1 LogIn
Attore: Segreteria/Professore
Precondizione: l'attore deve essere registrato precedentemente all'interno del sistema e deve trovarsi nella pagina di login.

Sequenza Eventi:

1. Una volta collegatosi alla home page, all'utente viene richiesto di inserire username e password
2. Dopo aver inserito username e password, l'utente preme il bottone *Login* e, nel caso di corretto inserimento delle credenziali, l'utente può accedere al sistema, in caso contrario verrà visualizzato un messaggio d'errore.

Nome Caso d'uso : CUS.2 LogOut/CUSP.2 LogOut
Attore: Segreteria/Professore
Precondizione: l'attore deve aver effettuato il login all'interno del sistema
Sequenza Eventi: *1.* L'utente ha completato le sue operazioni, decide di uscire dal sistema e preme il bottone *logout* 2. All'utente viene presentata la pagina di login

Nome Caso d'uso : CUS.3 Inserire Modificare Corsi
Attore: Segreteria
Precondizione: l'attore deve aver effettuato il login all'interno del sistema e deve essere nella pagina relativa ai corsi
Sequenza Eventi: 1. L'utente vuole eliminare un corso già presente nel sistema e inserirne uno nuovo 2. L'utente elimina il corso desiderato cliccando sul bottone relativo all'eliminazione del corso 3. L'utente riempie i campi NomeCorso, idcorso, professore, numero crediti e anno corso *4.* L'utente clicca sul bottone *Salva* 5. Se ha inserito tutti i dati correttamente il sistema segnalerà il corretto inserimenti del corso, altrimenti il sistema notificherà l'errore all'utente

Nome Caso d'uso : CUS.4 Inserire modificare aule
Attore: Segreteria
Precondizione: l'attore deve aver effettuato il login all'interno del sistema e deve trovarsi nella pagina relativa alle aule
Sequenza Eventi: 1. L'utente vuole eliminare un'aula già presente nel sistema e inserirne una nuova 2. L'utente elimina l'aula desiderata cliccando sul bottone relativo all'eliminazione dell'aula 3. L'utente riempie i campi Nome, Ubicazione, Capienza *4.* L'utente clicca sul bottone *Salva* 5. Se ha inserito tutti i dati correttamente il sistema segnalerà il corretto inserimento dell'aula, altrimenti il sistema notificherà l'errore all'utente

Nome Caso d'uso : CUS.5 Inserire modificare impostazioni dell' orario
Attore: Segreteria
Precondizione: l'attore deve aver effettuato il login all'interno del sistema e deve trovarsi nella pagina relativa all'orario, dopo aver già inserito almeno un corso e un'aula
Sequenza Eventi: 1. L'utente vuole modificare le impostazioni dell'orario 2. Riempie i campi: numero massimo di blocchi, numero massimo di slot consecutivi e numero massimo di slot in un giorno 3. Compila la tabella in cui vengono assegnate le aule ai corsi 4. Preme sul bottone *Salva* e, in caso di successo, il sistema segnala il corretto inserimento dei dati da parte dell' utente

Nome Caso d'uso : CUS.6 Calcolare orario
Attore: Segreteria
Precondizione: l'attore deve aver effettuato il login all'interno del sistema e deve trovarsi nella pagina relativa all'orario. Inoltre l'attore deve aver inserito i corsi, le aule, le impostazioni

dell'orario e l'associazione aule-corsi. Tutti i professori devono aver inserito le loro preferenze all'interno dell'orario.
Sequenza Eventi: 1. L'utente, dopo essersi accertato di aver inserito tutti i dati nel sistema e che tutti i professori abbiano inserito le loro preferenze all'interno dell'orario, decide di calcolare l'orario 2. Preme sul bottone *Calcola SAT,* nel caso voglia calcolare la soddisfacibilità, oppure preme sul bottone *Calcola Ottimo* per trovare la soluzione ottima 3. Il sistema mostra una finestra con il tempo trascorso e un bottone che permette all'utente di interrompere la computazione in qualsiasi momento 4. Nel caso in cui il problema non abbia soluzione, il sistema lo segnalerà all'utente 5. L'utente attende il termine della computazione

Nome Caso d'uso : CUS.7 Visualizzare Orario
Attore: Segreteria
Precondizione: l'attore deve aver effettuato il login all'interno del sistema e deve trovarsi nella pagina relativa all'orario. Inoltre l'attore deve aver inserito i corsi, le aule, le impostazioni dell'orario e l'associazione aule-corsi. Tutti i professori devono aver inserito le loro preferenze all'interno dell'orario. L'attore deve aver premuto sul bottone Calcola Ottimo o Calcola SAT
Sequenza Eventi: 1. Alla fine della computazione il sistema mostra all'utente l'orario che ha calcolato, nel caso in cui abbia trovato una soluzione 2. L'utente può scegliere quale orario visualizzare e in che formato esportarlo

Nome Caso d'uso : CUP.3 Inserire modificare le impostazioni dell'orario
Attore: Professore
Precondizione: La segreteria deve aver comunicato all'utente che gli è stato assegnato un corso.

L'attore deve aver effettuato il login all'interno del sistema e deve trovarsi nella pagina relativa alle impostazioni dell'orario
Sequenza Eventi: 1. L'utente vuole modificare le impostazioni dell'orario 2. Compila i campi: ore settimanali e suddivisione in blocchi 3. Clicca sul pulsante *Salva* 4. In caso di successo il sistema segnala il corretto inserimento dei dati da parte dell' utente

Nome Caso d'uso : CUP.4 Inserire modificare preferenze dell' orario
Attore: Professore
Precondizione: La segreteria deve aver comunicato all'utente che gli è stato assegnato un corso. L'attore deve aver effettuato il login all'interno del sistema, deve aver già inserito le impostazioni dell'orario e deve trovarsi nella pagina relativa alle impostazioni dell'orario
Sequenza Eventi: 1. L'utente vuole modificare le preferenze dell'orario 2. Tramite una tabella, in accordo alle proprie preferenze, l'utente può indicare per ogni timeslot una lezione. L'utente può decidere tra cinque opzioni che indicano il grado di preferenza per quella specifica ora 3. Una volta terminato preme sul bottone *Salva* 4. In caso di successo il sistema segnala il corretto inserimento dei dati da parte dell' utente, altrimenti il sistema ripropone la stessa pagina evidenziando gli errori commessi

Nome Caso d'uso : CUP.5 Inserire modificare Obbligatorio
Attore: Professore
Precondizione: La segreteria deve aver comunicato all'utente che gli è stato assegnato un corso. L'attore deve aver effettuato il login all'interno del sistema, deve aver già inserito le impostazioni dell'orario e deve trovarsi nella pagina relativa alle impostazioni dell'orario
Sequenza Eventi: 1. L'utente vuole eliminare un vincolo obbligatorio e ne vuole inserire uno nuovo 2. Preme sul bottone *Elimina* 3. Il sistema avvisa l'utente dell'avvenuta cancellazione 4. L'utente compila i campi: aula, giorno, ora 5. Clicca sul bottone *Salva* 6. In caso di successo il sistema segnala il corretto inserimento dei dati da parte dell' utente

4.3 Implementazione dell'interfaccia grafica

In questo paragrafo descriveremo le caratteristiche del software che ha costituito il punto principale della nostra ricerca, partendo dalle linee guida utilizzate per un corretto sviluppo della GUI, passando per le tecnologie utilizzate per la realizzazione del software, e concluderemo con un'analisi dell'interfaccia realizzata.

4.3.1 Linee guida per l'implementazione dell'interfaccia

Per l'implementazione dell'interfaccia grafica sono state adottate le seguenti linee guida, catalogate a seconda dell'ambito d'interesse[21]:

Navigazione :

1. Chi usa il sistema deve sempre sapere in quale pagina si trova.
2. Deve essere chiaramente mostrato il link alla home page del software.
3. Corretta organizzazione delle pagine direttamente accessibili dalla home page.

Funzionalità:

4. Le funzioni sono chiaramente etichettate e le funzioni essenziali sono disponibili senza abbandonare la GUI.
5. Chi usa il sistema non deve avere difficoltà nell'eseguire i task che gli sono stati assegnati.

Controllo Utente:

6. Le azioni devono essere facilmente reversibili.
7. Le pagine devono riflettere le intenzioni dell' utente.
8. Riduzione degli elementi che un utente deve memorizzare.

Contenuto Linguaggio e Consistenza:

9. Le informazioni e i task importanti sono messi in evidenza.
10. Le informazioni e i task relativi tra loro sono raggruppati nella stessa pagina o menù oppure nella stessa area di una pagina.
11. Il linguaggio è semplice corretto e chiaro.
12. I link sono coincisi espressivi e visibili.
13. Consistenza terminologia: parole e frasi simili descrivono i medesimi contenuti.
14. Il titolo della pagina riflette il contenuto della pagina stessa.
15. Il sito ha un design che richiede il minimo apprendimento.

Feedback utente e sistema:

16. Sono fornite schermate di feedback dopo la compilazione di un form.

Accessibilità:

17. Le etichette e i link sono correlate al contenuto
18. La GUI utilizza correttamente gli standard correnti del web e i CSS sono usati per il layout e stile.
19. La GUI è stata testata su diverse piattaforme.

Prevenzione e correzione degli errori:

20. I messaggi d'errore sono chiari e visibili
21. I messaggi d'errore prevedono un chiaro punto d'uscita.

Architettura e chiarezza visuale, layout della pagina:

22. Le pagine non sono troppo dense.
23. Il design e il layout del sito sono chiari e concisi.

24. I colori utilizzati per i link sono facilmente visibili e comprensibili. Gli oggetti non cliccabili non devono avere caratteristiche simili a quelle degli oggetti cliccabili.
25. Il testo in grassetto e corsivo sono usati con parsimonia
26. Design fluido.

4.3.2 Tecnologie utlizzate

Il sistema è stato implementato grazie alle moderne tecnologie per il web. Le tecnologie utilizzate sono le seguenti :

- PHP: nella versione 5.6.4, linguaggio di scripting server-side.
- Javascript: linguaggio di scripting client-side.
- JQuery: nella versione 2.1.4, framework javascript per applicazioni web.
- Mysql: nella versione 5.6, linguaggio per l'implementazione di database relazionali.
- CSS: nella versione 3, linguaggio usato per definire la formattazione dei documenti html.

4.3.3 GUI

La prima schermata che viene presentata agli utenti che si collegano al sistema è la pagina del login. Da questa pagina, in Figura 4.3, è possibile effettuare l'accesso al sistema.

Figura 4.3 Pagina LogIn

Come mostrato nell'immagine 4.3, ci sono due campi di testo in cui inserire username e password. Una volta inseriti questi valori, basterà cliccare sul bottone *LogIn* per effettuare l'accesso. Se l'utente dimentica di compilare un campo e clicca sul bottone

LogIn il sistema avviserà l'utente con un messaggio d'errore, come mostrato in Figura 4.4. Se l'username e/o la password sono errati il sistema comunicherà all'utente che c'è stato un errore nell'inserimento dei dati.

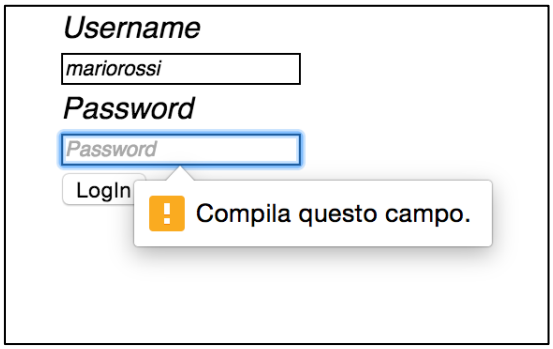

Figura 4.4 Errore: mancato inserimento del campo password

Adesso vedremo nel dettaglio le interfacce elaborate per gli utenti: segreteria e docenti. Iniziamo con la home page dell'utente segreteria in Figura 4.5. Per semplicità divideremo la pagina in quattro sezioni.

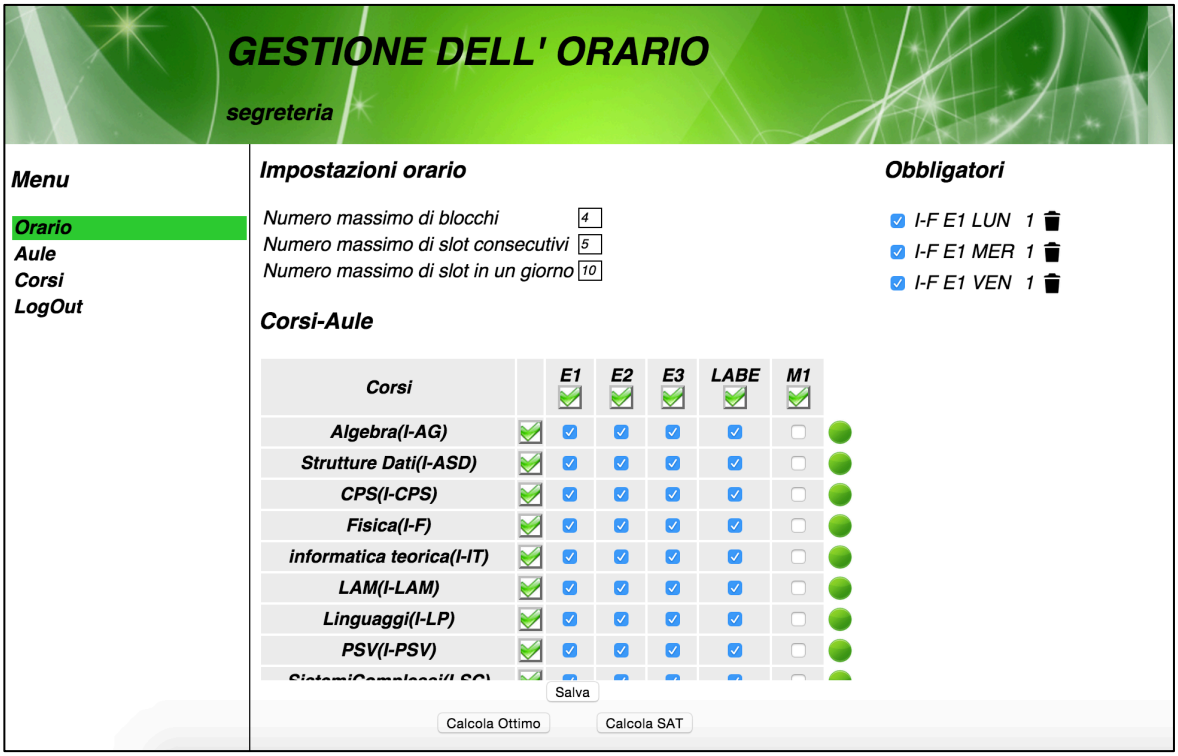

Figura 4.5 Home page per l'utente segreteria

Nella sezione 1 mostrata in Figura 4.6, l'utente può scegliere cosa visualizzare tra l'orario, i corsi e le aule, oppure può decidere di effettuare il logout.

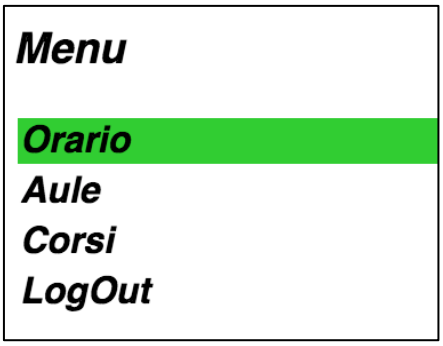

Figura 4.6 Sezione 1 home page segreteria

Nella sezione 2 in Figura 4.7, l'utente può inserire le informazioni riguardanti l'orario, modificando le caselle di testo, impostando il numero di timeslot per una giornata, il numero massimo di sessioni e il numero massimo di ore consecutive di lezione che un professore può svolgere in una giornata.

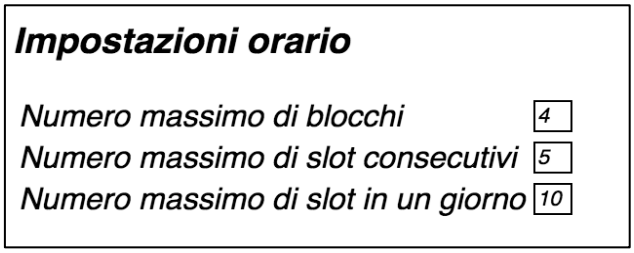

Figura 4.7 Sezione 2 della home segreteria

Nella sezione 3 in Figura 4.8, l'utente può associare i corsi alle aule semplicemente spuntando la casella corrispondente al corso e all'aula desiderata, oppure selezionare o deselezionare un'intera riga o un'intera colonna, cliccando bottone . Inoltre in questa sezione l'utente può controllare se tutti i professori hanno inserito le preferenze per l'orario, verificando che tutti i corsi siano contrassegnati dal "semaforo verde" . Nella sezione 4 mostrata in Figura 4.9, l'utente può scegliere quali vincoli, inseriti dai professori, usare nel calcolo dell' orario, oppure può decidere di eliminarli cliccando sul bottone . Per salvare le informazioni inserite basterà premere il pulsante *Salva*.

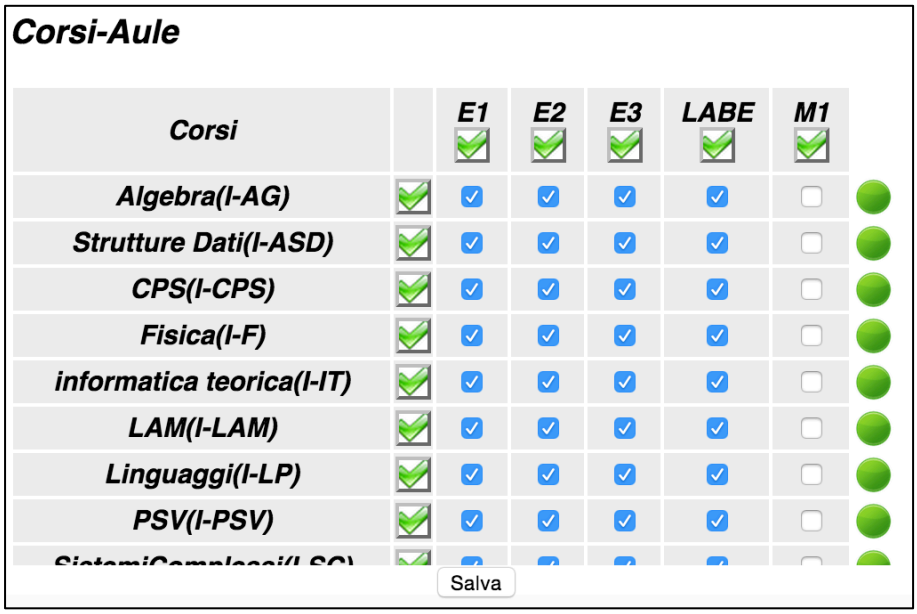

Figura 4.8 Sezione 3 home segreteria

Se l'utente ha inserito tutte le informazioni correttamente, viene mostrato un messaggio di conferma, altrimenti all'utente saranno segnalati gli errori.

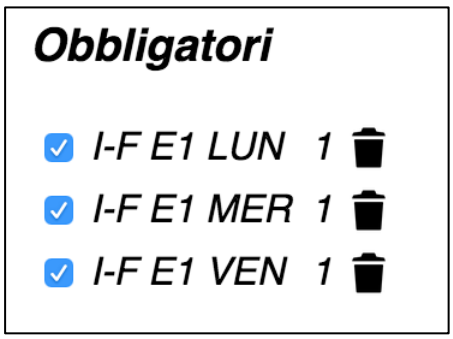

Figura 4.9 sezione 4 home segreteria

Da questa pagina, se e solo se tutti i professori hanno inserito le loro preferenze, è possibile calcolare l'orario. Ci sono due modi per farlo: trovare la soluzione ottima, e per questo basterà premere il pulsante *Calcola Ottimo*; trovare una soluzione soddisfacibile, e per questo invece basterà premere il bottone *Calcola SAT*. Se l'utente invece decide di inserire una nuova aula basterà selezionare, dalla home page, il link *Aule*. Una volta premuto il bottone verrà visualizzata la pagina relativa alle aule, come in Figura 4.10.

Figura 4.10 Pagina relativa all'inserimento di una nuova aula

Nella parte sinistra è presente una tabella che mostra le informazioni; nome, capienza e ubicazione, riguardanti le aule già presenti nel sistema. Se si vuole eliminare un'aula basta premere il bottone 🗑 all' interno della tabella. Un messaggio informerà l'utente dell'avvenuta cancellazione. Nella parte destra c'è un form per l'inserimento di una nuova aula. Da qui l'utente può inserire una nuova aula all'interno del sistema riempiendo i campi: Nome, Ubicazione e Capienza. Per salvare le informazioni relative all'inserimento basterà premere il bottone *Salva* e, se non ci sono stati errori, il sistema comunicherà all'utente l'avvenuto inserimento. In caso contrario, un messaggio di errore evidenzierà i campi che non sono stati compilati in modo corretto. Se invece l'utente vuole inserire o eliminare un corso, allora deve premere dalla home page il link *Corsi*. Così facendo si aprirà la pagina *corsi,* come mostrato in Figura 4.11. Nella parte sinistra della pagina sono elencati tutti i corsi già presenti nel sistema, presentati in ordine alfabetico, mostrando le seguenti informazioni: nome corso, Idcorso, professore, anno e stato. *Stato* indica se il professore ha inserito o meno le informazioni necessarie alla creazione dell'orario. Se c'è 🔴 , allora il professore non ha ancora inserito i dati, altrimenti la presenza di 🟢 indicherà l'avvenuto inserimento da parte del professore. Nella parte destra è possibile inserire un nuovo corso all'interno del sistema. Basterà riempire i campi: nome corso, id corso, numero crediti, selezionare un professore dall'elenco e un anno accademico. Per salvare i dati basterà premere il bottone *Salva* e in caso di successo, cioè se sono stati inseriti tutti i dati correttamente, il sistema mostrerà un messaggio d'inserimento avvenuto con successo.

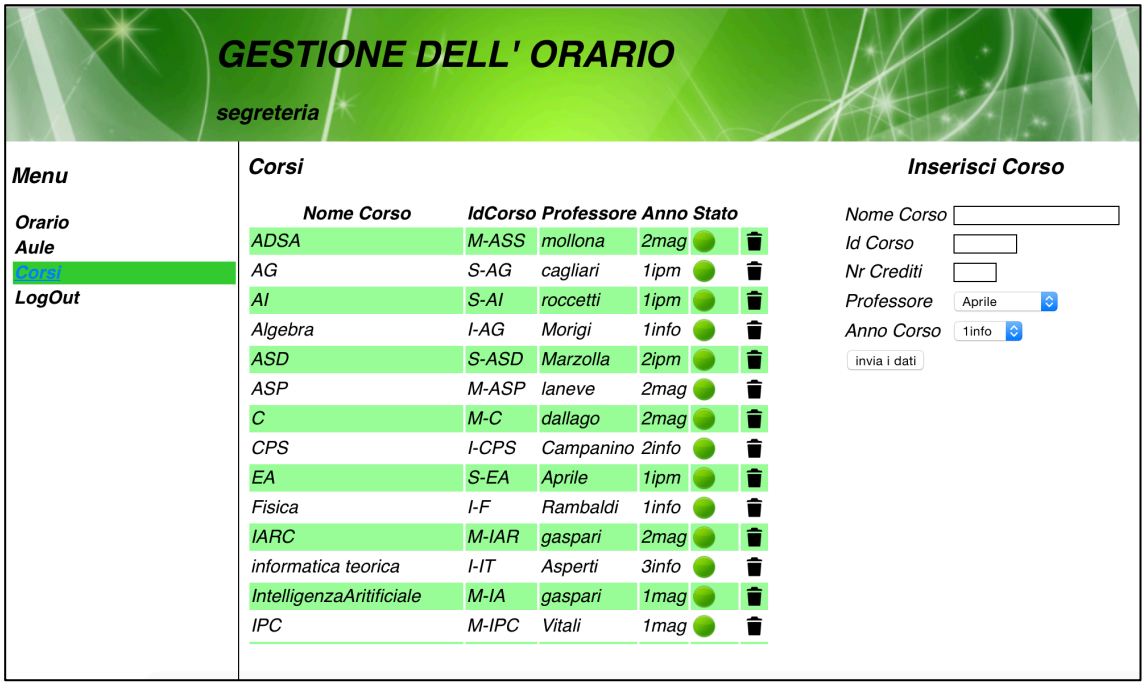

Figura 4.11 Pagina relativa all'inserimento di un nuovo corso

Quando l'utente vuole calcolare l'orario, ammesso che tutti i professori abbiano inserito le preferenze per l'orario, basterà premere dalla home page il bottone *Calcola SAT* per la soddisfacibilità, oppure il bottone *Calcola Ottimo* per l'ottimalità. Una volta premuto il bottone apparirà una finestra, come in Figura 4.12, che mostrerà il tempo d'esecuzione del programma. Sarà possibile interrompere in qualsiasi momento la computazione premendo il bottone oppure effettuando il LogOut. Se il problema non ha soluzione, il sistema lo comunicherà immediatamente all'utente e sospenderà il calcolo. Quando il sistema ha calcolato l'orario quest'ultimo verrà visualizzato in una nuova finestra, suddiviso per anni di corso, come in Figura 4.13. A questo punto l'utente può scegliere di esportarlo in formato pdf e in formato csv.

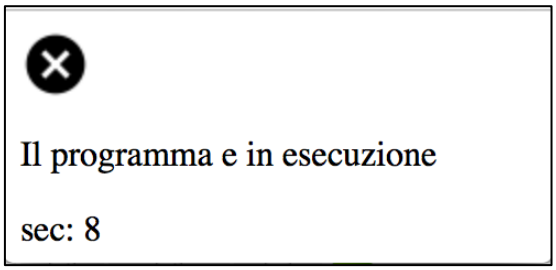

Figura 4.12 Finestra che mostra l'esecuzione della computazione

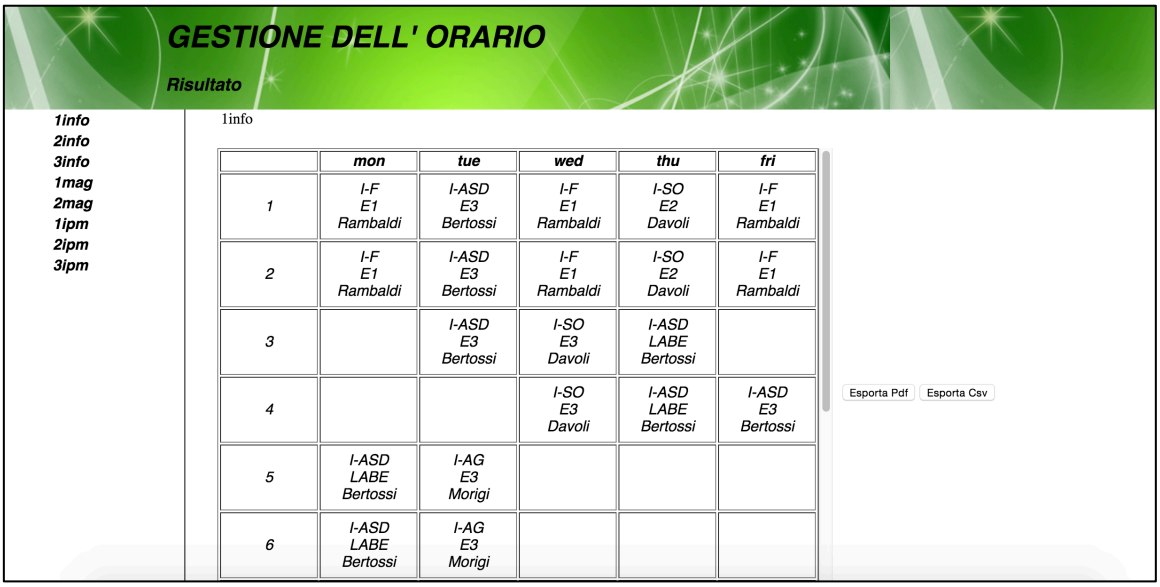

Figura 4.13 Pagina che visualizza il risultato del orario

Per l'esportazione in pdf l'utente deve cliccare sul bottone *Esporta Pdf* e poi può effettuare il download. Per esportare l'orario in formato csv, l'utente deve premere sul bottone *Esporta Csv* e partirà automaticamente il download. Adesso esaminiamo le caratteristiche dell'interfaccia relativa all'utente: professore. Una volta effettuato il login, il professore potrà visualizzare la home page professore, come in Figura 4.14

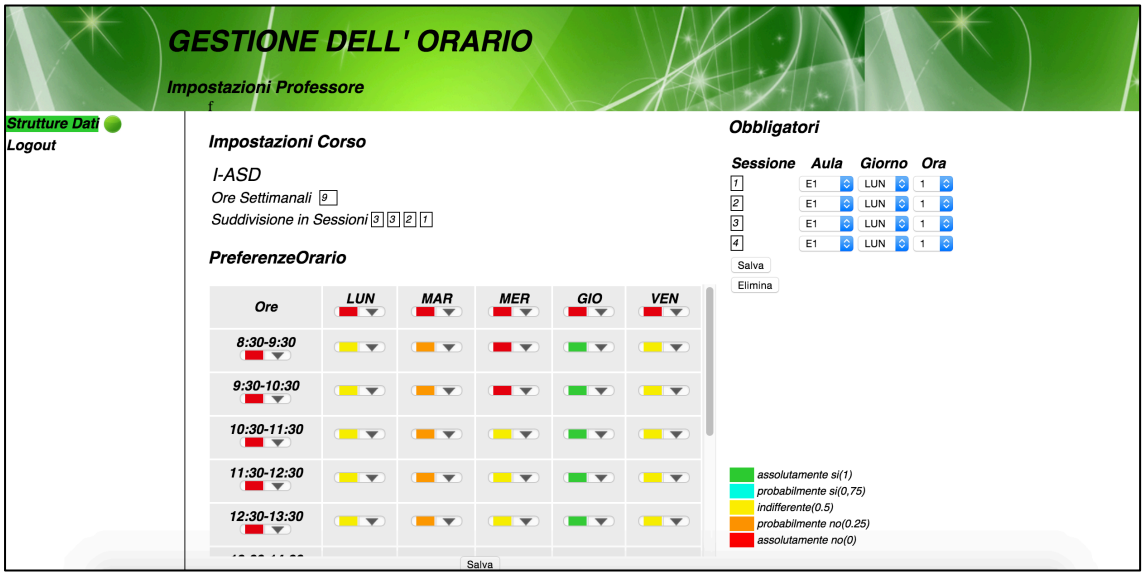

Figura 4.14 home page per l'utente professore

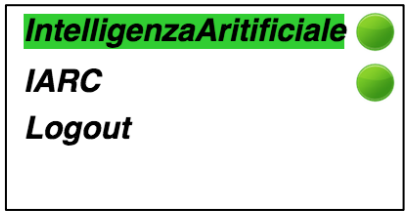

Figura 4.15 Sezione 1 homepage professore

Per semplicità divideremo la pagina in quattro sezioni. Dalla sezione 1 in Figura 4.15, il professore può scegliere quale corso modificare e/o inserire e controllare lo stato dei corsi.

Figura 4.16 Sezione 2 homepage professore

Nella sezione 2 in Figura 4.16, il professore può inserire il numero di ore di lezione che intende svolgere e scegliere come dividerle in sessioni. Nella sezione 3 in Figura 4.17 il professore può esprimere, per ogni slot e per ogni giorno, le sue preferenze, inoltre può scegliere di selezionare lo stesso valore per una stessa riga o per una stessa colonna.

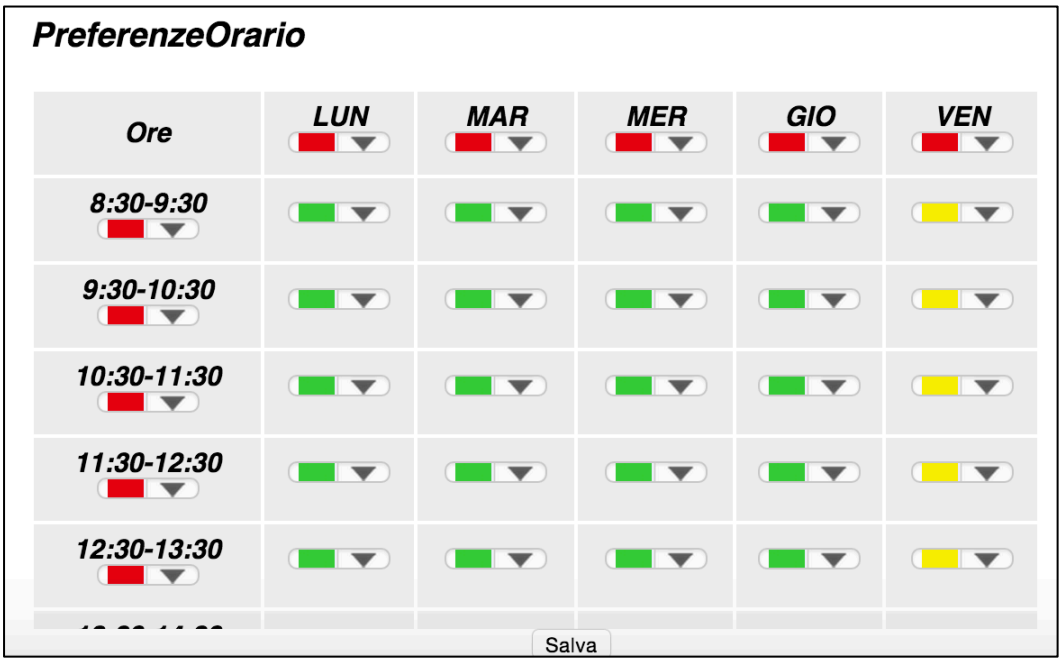

Figura 4.17 sezione 3 home page professore

Sono presenti cinque opzioni, rappresentate con cinque colori diversi, cui è stato assegnato un diverso valore di probabilità, come mostrato in Figura 4.18.

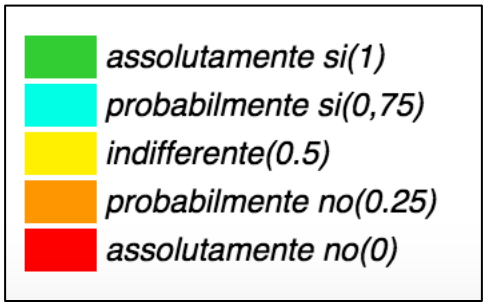

Figura 4.18 Legenda inserimento preferenze professore

Il professore può salvare le impostazioni inserite premendo il bottone *Salva*. Se l'utente ha inserito correttamente i dati, verrà mostrato un messaggio d'inserimento avvenuto con successo. Nella sezione 4 in Figura 4. il professore può inserire i vincoli "obbligatori", compilabili nel caso in cui un professore abbia particolari necessità e per cui egli debba svolgere una lezione solo: in un determinato giorno, in una determinata aula e in una determinata ora.

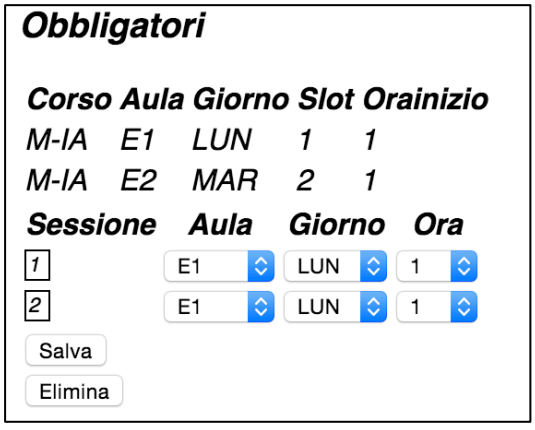

Figura 4.19 sezione 4 homepage professore

Per fare ciò il professore deve inserire, per ogni sessione, l'aula, il giorno e l'ora, selezionandole da un elenco. Una volta impostati tutti i parametri può premere il pulsante *Salva* per salvare le impostazioni nel sistema. Se non ci sono stati errori nell'inserimento verrà mostrato un messaggio di avvenuto salvataggio dei dati. Il salvataggio sovrascrive i dati precedentemente inseriti. Se si vogliono eliminare i dati

inseriti in precedenza, visualizzati in alto nella schermata in Figura 4.19, bisogna premere sul bottone Elimina.

4.4 Installazione e Configurazione del sistema

In questo paragrafo vedremo come installare e configurare il programma, utilizzando un web server. Il sistema è stato testato su Ubuntu 14.10 e successivi. I componenti necessari all'installazione sono: Apache, PHP, MySql, GLPK, Wkhtmltopdf. Si procederà dunque nel seguente modo:

1) Occorre innanzitutto installare Apache:
Apache è un server http open source per i moderni sistemi operativi. Per installare Apache digitare dal terminale:

```
sudo apt-get install apache2
```

Completata l'installazione, verrà creata una cartella /var/www/html

2) In un secondo momento, procederemo all'installazione di PHP:
PHP può essere installato su Ubuntu digitando il seguente comando :

```
sudo apt-get install php5 libapache2-mod-php5
```

3) Adesso è necessario installare MySql:
MySql è un Relational database managment system (RDMBS) composto da un client, a riga di comando, e da un server. Per installare MySql, digitare da terminale:

```
sudo apt-get install mysql-client mysql-server
```

Durante l'installazione verrà richiesta una password per identificare l'utente root.

4) Procediamo ora con l'installazione di phpmyamdin:
phpmyadmin è un software utile alla gestione dei database MySql. Per installarlo bisogna digitare dal terminale:

```
sudo apt-get install phpmyadmin
```

Durante l'installazione verrà chiesto di inserire la password del database MySql, inserita in precedenza. Al termine dell'installazione dobbiamo dire al nostro Apache dove si trova il file di configurazione di phpmyadmin, che ci permetterà di eseguirlo dal browser, quindi apriamo il file *apache2.conf* con il nostro editor preferito, ad esempio

con: `sudo nano /etc/apache2/apache2.conf` e aggiungiamo alla fine del file la seguente riga: *Include /etc/phpmyadmin/apache.conf* **.** Ora non ci resta che salvare il file e riavviare il server web con: `sudo /etc/init.d/apache2 restart.`
A questo punto possiamo accedere al programma dall'indirizzo: *http://localhost/phpmyadmin*

5) Dunque è possibile procedere con l'installazione di GLPK:
GLPK è il pacchetto necessario al sistema per risolvere il problema dell'orario. Per installare GLPK bisogna digitare da terminale:

```
sudo apt-get install glpk-doc glpk-utils
```

6) Ora è necessario installare wkhtmltopdf:
wkhtmltopdf è un pacchetto necessario alla generazione di file pdf partendo da file in formato html. Per richiamare questo pacchetto dal server bisogna per prima cosa installare xvfb. Per installarlo basterà eseguire il seguente comando da terminale:

```
sudo apt-get install xvfb
```

Dopodiché passiamo all'installazione di wkhtmltopdf con il seguente comando :

```
sudo apt-get install wkhtmltopdf.
```

7) È ora giunto il momento della configurazione:
Innanzitutto bisogna copiare il contenuto della cartella LessTime, presente nel cd in allegato, all'interno della root /var/www/html del nostro webServer. In secondo luogo sarà necessario cambiare i permessi della cartella, con il comando :

```
sudo chmod –R 777 /var/wwww/html
```

In seguito, importare il database presente nella sotto-cartella dumpDB. Per fare ciò bisogna accedere a phpmyadmin, digitando nel browser localhost/phpmyadmin, creare un nuovo database e importare il file startconfig.sql all'interno del database appena creato. Nella cartella dumpDB è inoltre presente un dump di un database contenente tutti i corsi del secondo semestre. Infine bisognerà editare il file config.php presente nella cartella LessTime specificando il nome del database creato in precedenza, username, password e indirizzo del database che è stato creato durante l'installazione. Per accedere alla pagina di login basterà aprire un qualsiasi browser web e digitare localhost/login.php.

4.5 In breve

In questo capitolo abbiamo analizzato l'implementazione del sistema descritto nei capitoli precedenti. È presente un'analisi dei requisiti e dei casi d'uso, con l'ausilio di diagrammi dei casi d'uso e di tabelle contenenti le informazioni per ogni caso d'uso. Sono presenti all'interno di questo capitolo le linee guida utilizzate per lo sviluppo della GUI e gli screenshot della stessa. Gli screenshot sono stati divisi per utenti e rappresentano la conseguenza delle scelte progettuali fatte durante l'analisi dei requisiti. Alla fine del capitolo abbiamo presentato una piccola guida all'installazione e alla configurazione del software sviluppato.

Conclusioni

In questa tesi abbiamo esaminato il problema dell'orario, indagandone le problematiche e le possibili soluzioni. Ci siamo concentrati, in particolar modo, sulla risoluzione del problema del timetabling relativo ai corsi di laurea triennali e magistrali in Informatica della scuola di Scienze dell'Università di Bologna, introducendo inoltre il vincolo per l'ora di pausa. Infine abbiamo sviluppato un'interfaccia grafica utile all'inserimento dei dati all'interno del modello, oltre che alla generazione di un orario che sia coerente con le direttive accademiche del corso di laurea in questione. Per lo sviluppo della GUI sono stati descritti i requisiti e indagati i casi d'uso affinché l'implementazione rispettasse coerentemente la progettazione del modello finale. Questa GUI facilita l'interazione tra gli utenti e il sistema; oltretutto consente la personalizzazioni dei vincoli, in base alle esigenze dei singoli utenti.

L'impostazione metodologica qui adottata ci ha consentito di raggiungere degli obiettivi posti all'inizio di questo studio, nonostante ciò non possiamo fare a meno di riconoscere alcuni dei limiti insiti nel nostro lavoro. Di certo il limite principale di questo modello è identificabile nella scelta del risolutore GLPK. Benché capace di trovare in modo efficace la soluzione ottima alla risoluzione del problema del timetabling, il risolutore GLPK richiede tempi di calcolo molto lunghi. Sarebbe quindi necessario provare ad adottare un risolutore più performante, come GUROBI o CPLEX, in modo tale da riuscire a trovare una soluzione ottima al problema in tempi ragionevoli. Un altro cambiamento auspicabile sarebbe modificare la GUI così da permettere all'utente di poter modificare l'orario una volta generato, invertendo e spostando gli orari delle lezioni a proprio piacimento prima della versione definitiva. Un'altra importante modifica che può essere effettuata al software consiste nella generazione di un orario basandosi su un orario generato in precedenza. Infine la nostra implementazione del timetabling, specifica per il problema degli orari del corso di laurea in Informatica per la Scuola di Scienze, potrebbe essere implementata per altri corsi di laurea, nonché per altre istituzioni che necessitino di questo software.

Bibliografia

[1] Abdullah, S. and Hamdan, A. R. (2008) A Hybrid Approach for University Course Timetabling. IJCSNS International Journal of Computer Science and Network Security, VOL.8 No.8.

[2] Alvarez, R., Crespo, E., and Tamarit, J. M. (2002) Design and implementation of a course scheduling system using tabu search. European Journal of Operational Research 137, pp. 512–523.

[3] Asmuni, H. (2008) Fuzzy Methodologies for Automated University Timetabling Solution Construction and Evaluation, Ph.D. Thesis. School of Computer Science University of Nottingham, Nottingham, UK.

[4] Dantzig George Bernard, A. O. (1955). The generalized simplex method for minimizing a linear form under linear inequality restraints. Journal od Mathematics , 183-190.

[5] Daskalaki, Sophia, Theodore Birbas, and Efthymios Housos. (2004) "An integer programming formulation for a case study in university timetabling." European Journal of Operational Research 153.1,117- 135

[6] Fering, A. (1986) Linear Programming An Introduction, quantitative applications in social sciences, Sage University Paper.

[7] Foundation, F. S. (s.d.). Tratto da glpk: https://www.gnu.org/software/glpk

[8] Golabpour, A., Mozdorani, S. H., Farahi, A., kootiani, M. and beige, H. (2008) A fuzzy solution based on Memetic algorithms for timetabling. IEEE, 978-0-7695-3556-2.

[9] Gunawan Aldy, Kien Ming Ng and Kim Leng Poh. A Mathematical Programming Model for A Timetabling Problem. World Congress in Computer Science, Computer Engineering and Applied Computing, pp. 42-47, 2006.

[10] Kimball George, Morse Philip; Methods of operations research, Dower Publications, USA, 2003

[11] Obit, J. H. (2010) Developing Novel Meta-heuristic, Hyper-heuristic and Cooperative Search for Course Timetabling Problems, Ph.D. Thesis. School of Computer Science University of Nottingham, Nottingham, UK.

[12] Robere, R. (2012). Interior Point Methods and Linear Programming.

[13] Sadaf, N. J. and Shengxiang, Y. (2008) A Memetic Algorithm for the University Course Timetabling Problem. IEEE, pp. 1082-3409.

[14] Schaef. Andrea. A Survey of Automated Timetabling. Artificial Intelligence Review 13(2), pp. 87-127, 1999.

[15] Sivarethinamohan R. (2008). Operations Research. NewYork: Tata McGraw-Hill Education.

[16] Tadei Roberto, F. D. (2010). Elementi di Ricerca Operativa. Bologna, Bologna, Italia: Società Editrice Esculapio.

[17] Tardos, J. K. (2012). Algorithm Design. New York, USA: Pearson.

[18] Tuga M., Berretta, R. and Mendes, A. (2007) A Hybrid Simulated Annealing with Kempe Chain Neighborhood for the University Timetabling Problem . IEEE, 0-7695-2841-4.

[19] Various. (2012). Optimization and Decision Support Design Guide: Using IBM ILOG Optimization Decision Manager. USA: IBMRedbooks.

[20] Wren A. (1996). Scheduling, Timetabling and Rostering - A Special Relationship? In the Practice and Theory of Automated Timetabling , 46- 75.

[21] Usability.gov http://guidelines.usability.gov/ è la principale risorsa per l'user experience (UX) best practice e linee guida, che servono a professionisti e studenti nei settori governativi e privati. Visualizzato in data 5/06/2015

APPENDICE A

MODELLO IN GLPK ALLA BASE DEL SOFTWARE

File con il modello con i vincoli per la pausa pranzo

---

timetable.mod

```
/*************************************************************/
                          A TIMETABLING PROBLEM
                          =====================

This script solves a timetabling problem using Mixed Integer Linear
Programming (MILP).

Execute with

glpsol --math timetable-0.9.mod --data timetable-0.9-2sem.dat [--cuts]
[--pcost] [--minisat]

/*************************************************************/

############################################################
##
## MODEL PARAMETERS
##
############################################################
##Spesso è utile poter lavorare con insiemi pluridimensionali, vale a
##dire insiemi i cui elementi sono n-uple (si tratta sempre di n-uple
##ordinate). Questo viene effettuato #con la parola chiave dimen
##set Data dimen 3; /* Course, Professor, Year */

##Le parole chiave setof e in consentono di definire un insieme
##monodimensionale come la proiezione su un indice di un insieme
##pluridimensionale.

set Courses := setof {(c,p,y) in Data} c;
```

```
/* Set of Courses. Each course is taught by exactly one professor;
furthermore, each course
is held on a specific academic year. */

set Professors := setof {(c,p,y) in Data} p;
/* set of Professors. Each professor teaches one or more courses. */

set Years := setof {(c,p,y) in Data} y;
/* Academic years. The academic year is used to automatically identify
static conflicts: courses from the same academic year must not
overlap. */

set Rooms;
/* set of available rooms */

set Days := {1 .. 5};
/* Days of the week (1=mon, 5=fri) */

/*con la parola param indichiamo i dati che inseriremo all interno del
nostro programma in questo caso definiamo
il massimo numero di slot che ci possono essere in un giorno*/
param slots_per_day integer, > 0, default 9;

set Slots := {1 .. slots_per_day};
/* Teaching slots within a day. A teaching slot corresponds to one
hour of lecture; slot 1 corresponds to 9:30-10:30, slot 9 corresponds
to 17:30-18:30 */

param maxb integer, > 0, default 3;
/* Maximum number of blocks which can be requested by a professor. A
block is defined as a set of consecutive teaching slots. For example,
if a specific course requires a total of 6 slots per week, then the
professor may request two blocks of {3, 3} hours, or three blocks of
{2, 2, 2} hours, and so on. */

param max_slots_per_day integer, > 0, default 3;
/* Each professor will not be assigned more than max_slots_per_day
teaching slots per day. */

param ns{c in Courses, 1..maxb}, integer, >= 0, <= max_slots_per_day,
default 0;
/* ns[c,b] is the number of consecutive slots requested by the b-th
block of course c. For example, if course c requires a weekly workload
of 6 slots divided in two blocks of {3, 3} slots each, then ns[c] =
{3, 3, 0} (trailing zeros are ignored). */

param nb{c in Courses} integer := card( {b in 1..maxb: ns[c,b] > 0} );
/* nb[c] is the total number of blocks requested by course c. */

set BlockIdx{c in Courses} := {1 .. nb[c]};
/* The size (number of slots) of blocks requested by course c are
ns[j] for each j in BlockIdx[c] */

param room_course{Rooms, Courses} binary, default 1;
/* room_course[r,c] = 1 iff room r can be used for course c */

param room_available{Rooms, Days, Slots} binary, default 1;
```

```
/* room_available[r,d,s] := 1 iff room r is available on slot s of day
d. */

set Teaches within (Professors cross Courses) := setof {(c,p,y) in
Data} (p,c);
/* (p,c) belongs to this set if professor p teaches course c.  */

set CoursesTaught{p in Professors} within Courses := {c in Courses:
(p,c) in Teaches};
/* CoursesTaught[p] is the set of courses taught by professor p */

param nslots{c in Courses} > 0, integer := sum{l in 1..maxb} ns[c,l];
/* nslots[c] is the total number of teaching slots required by course
c each week. calcolato come somma degli slot
consecutivi richiesti*/

set year within (Courses cross Years) := setof {(c,p,y) in Data}
(c,y);
/* year(c,y) belongs to this set iff course c is held during academic
year y. FIXME: should this be a parameter rather than a set?? */

set BasicConflicts within (Courses cross Courses) := (setof {(c1,y) in
year, (c2,y) in year: c1 != c2} (c1,c2)) union (setof {(p,c1) in
Teaches, (p,c2) in Teaches: c1 != c2} (c1,c2));
/* BasicConflicts is the set of pairs of courses which must not be
scheduled at the same time. This set is initialized with a basic set
of conflicts that includes: (i) courses that belongs to the same
academic year, and (ii) courses that are taught by the same professor
*/

set ExtraConflicts within (Courses cross Courses);
/* Additional conficts, user defined */

set Conflicts within (Courses cross Courses) := setof {(c1, c2) in
Courses cross Courses: (c1,c2) in BasicConflicts union ExtraConflicts
or (c2, c1) in BasicConflicts union ExtraConflicts} (c1, c2);
/* all conflicts: ensure that if (c1,c2) conflict, also (c2,c1) do */

var schedule{Courses, Rooms, Days, Slots} binary;
/* schedule[c,r,d,s] is true if course c is assigned slot s on day d,
and lecture will be held in room r */

/*block_schedule è la variabile per lo scehduling dei blocchi*/

var block_schedule{c in Courses, Rooms, Days, l in BlockIdx[c]}
binary;
/* block_schedule[c,r,d,l] is true iff course c is assigned ns[c,l]
consecutive slots in room r on day d. Note that at most one block for
each course can be scheduled each day */

set pre_schedule dimen 5;
/* (course, room, day, l, s): the l-th block of course c is scheduled
to start on slot s at the given day and room */

param weights{p in Professors, d in Days, s in Slots} default 1.0;

param prof_available{p in Professors, d in Days, s in Slots} binary :=
if (weights[p,d,s] != 0 ) then 1 else 0;
```

```
/* prof_available[p,s,d] is true iff professor p is available on slot
s of day d. */

############################################################
##
## CONSISTENCY CHECKS
##
############################################################
/*Un modo più raffinato di eseguire controlli di coerenza sui dati
richiede l'uso
della parola chiave check , seguita dal simbolo : e da un'espressione
logica*/


check sum{c in Courses, r in Rooms, d in Days, s in Slots}
room_course[r,c] * room_available[r,d,s] >= sum{c in Courses}
nslots[c];
/* The total number of slots for all courses must be less than or
equal to the maximum number of slots available for that courses in all
rooms */

check {p in Professors}: sum{d in Days, s in Slots}
prof_available[p,d,s] >= sum{c in CoursesTaught[p]} nslots[c];
/* Each professor must guarantee enough availability to satisfy all
his teaching duties */

check {p in Professors}: card(Days)*max_slots_per_day >= sum{c in
CoursesTaught[p]} nslots[c];
/* max_slots_per_day must be set so that each professor can satisfy
his teaching requirements */

check {c in Courses}: card({(p,c) in Teaches}) = 1;
/* Each course must be taught by exactly one professor */

check {(c1,c2) in Conflicts} : (c2,c1) in Conflicts;
/* Conflicts matrix must be symmetric */

check {c in Courses} : (c,c) not in Conflicts;
/* A course must not confict with itself */

check {c in Courses, j in BlockIdx[c]} : ns[c,j] > 0;
/* The first nb[c] elements of the cth row of ns[] must be nonzero...
*/

check {c in Courses, j in nb[c]+1..maxb} : ns[c,j] = 0;
/* ...and the rest must be zero */

############################################################
##
## OBJECTIVE FUNCTION
##
############################################################
## If you want to solve the model using --minisat, comment the
objective function ##above and uncomment the one below:
 maximize obj:
  sum{c in Courses, d in Days, s in Slots, r in Rooms}
schedule[c,r,d,s];
##maximize obj:
```

```
##   sum{c in Courses, d in Days, s in Slots, r in Rooms, p in
Professors} ##schedule[c,r,d,s] * weights[p,d,s];
#############################################################
##
## CONSTRAINTS
##
#############################################################

## Each course should be scheduled at most once per slot
s.t. schedule_at_most_once 'schedule at most once per block'
{c in Courses, d in Days, s in Slots}:
      sum{r in Rooms} schedule[c,r,d,s] <= 1;

## The number of slots allocated for course c on day d and room r must
be equal to the total blocks for the same course and day
s.t. blocks_eq_schedule 'block_schedule and schedule must match'
{c in Courses, d in Days, r in Rooms}:
      sum{s in Slots} schedule[c,r,d,s] = sum{l in BlockIdx[c]}
ns[c,l] * block_schedule[c,r,d,l];

## A block of each course must be scheduled at most once per day
s.t. max_once_per_day 'Each course must be scheduled at most once per
day'
{c in Courses, d in Days}:
      sum{r in Rooms, l in BlockIdx[c]} block_schedule[c,r,d,l] <= 1;

## Each block must be instantiated exactly once per week
s.t. each_block_instantiated 'Each block must be instantiated exactly
once per week'
{c in Courses, l in BlockIdx[c]}:
        sum{r in Rooms, d in Days} block_schedule[c,r,d,l] = 1;

## Each room must be used by at most one course
s.t. at_most_one_course_per_room 'Each room must be used by at most
one course at a time'
{r in Rooms, d in Days, s in Slots}:
      sum{c in Courses} schedule[c,r,d,s] <= room_available[r,d,s];

## Conflicting courses must not overlap; note the predicate 'c1 < c2'
that is used only to reduce the size of this constraint
s.t. courses_dont_overlap 'Conflicting courses must not overlap'
{d in Days, s in Slots, (c1,c2) in Conflicts: c1 < c2}:
      sum{r in Rooms} schedule[c1,r,d,s] + sum{r in Rooms}
schedule[c2,r,d,s] <= 1;

## Professors must be available in their assigned slots
s.t. prof_av 'Professors must be available in their assigned slots'
{(p,c) in Teaches, d in Days, s in Slots}:
            sum{r      in      Rooms}      schedule[c,r,d,s]      <=
prof_available[p,d,s];

s.t. prof_av_2 'Professors must be available in their assigned slots'
{(p,c) in Teaches, d in Days, s in Slots, r in Rooms}:
            schedule[c,r,d,s] <= prof_available[p,d,s];

## Constraint per pranzoslot
 s.t. lunchhour 'ora di pranzo'
 {d in Days ,y in Years}:
```

```
  sum{r in Rooms, (c,y) in year} schedule[c,r,d,4] + sum{r in Rooms,
(c,y) in year} schedule[c,r,d,5] <= 1;

## Constraint pranzo come lezione
#LUNCH
 s.t. lunch1 'lunch between 13:30 and 15:30 trated as a lesson'
 {d in Days}:
      sum{r in Rooms} schedule["Lunch1",r,d,4] + sum{r in Rooms}
schedule["Lunch1",r,d,5] = 1;

## Rooms must be available
 s.t. room_av 'Rooms must be available'
 {r in Rooms, d in Days, s in Slots, c in Courses}:
      schedule[c,r,d,s] <= room_course[r,c] * room_available[r,d,s];

## Allocated blocks must be consecutive
# s.t. fill_blocks1 'Allocated blocks must be consecutive 1'
#  {r in Rooms, d in Days, c in Courses, l in 1..nb[c], t in
2..ns[c,l]: room_available[r,d,t] and room_course[r,c]}:
#     schedule[c,r,d,1]     -     schedule[c,r,d,t]     <=     1     -
block_schedule[c,r,d,l];

# s.t. fill_blocks2 'Allocated blocks must be consecutive 2'
# {r in Rooms, d in Days, c in Courses, s in Slots, l in 1..nb[c], t
in 2..ns[c,l]: s+ns[c,l] in Slots and room_available[r,d,s] and
room_course[r,c]}:
#     -schedule[c,r,d,s] + schedule[c,r,d,s+1] - schedule[c,r,d,s+t]
<= 1 - block_schedule[c,r,d,l];

s.t. fill_blocks 'Allocated blocks must be consecutive'
{r in Rooms, d in Days, c in Courses, s1 in Slots, s2 in Slots, s3 in
Slots: s1<s2 and s2<s3}:
      schedule[c,r,d,s1] - schedule[c,r,d,s2] + schedule[c,r,d,s3] <=
1;

## Each professor should not teach more than max_slots_per_day
s.t. max_hours_per_professor 'Each professor should not teach more
than max_slots_per_day slots per day'
{p in Professors, d in Days}:
      sum{r in Rooms, c in CoursesTaught[p], l in BlockIdx[c]}
ns[c,l]*block_schedule[c,r,d,l] <= max_slots_per_day;

## mark preallocated slots
s.t. preschedule_slots1 'mark preallocated blocks'
{(c, r, d, l, s) in pre_schedule}:
      block_schedule[c,r,d,l] = 1;

s.t. preschedule_slots2 'mark preallocated slots'
{(c, r, d, l, s) in pre_schedule, t in Slots: t>=s and t<s+ns[c,l]}:
      schedule[c,r,d,t] = 1;

############################################################
##
## SOLVE MODEL
##
############################################################

solve;
```

File .dat con una particolare configurazione del file dati

```
data;

param maxb := 4;

param max_slots_per_day := 5;

param slots_per_day := 10;

set Rooms := "E1","E2","E3","LABE","M1";

set Data :=

("I-AG", "Morigi", "1info")
("I-ASD", "Bertossi", "1info")
("I-CPS", "Campanino", "2info")
("I-F", "Rambaldi", "1info")
("I-IT", "Asperti", "3info")
("I-LAM", "Bononi", "3info")
("I-LP", "Martini", "2info")
("I-PSV", "Davoli", "3info")
("I-SC", "babaoglu", "1mag")
("I-SC2", "ferretti", "1mag")
("I-SI", "Casadei", "3info")
("I-SO", "Davoli", "1info")
("I-TW", "Vitali", "2info")
("M-ASP", "laneve", "2mag")
("M-ASS", "mollona", "2mag")
("M-C", "dallago", "2mag")
("M-IA", "gaspari", "1mag")
("M-IAR", "gaspari", "2mag")
("M-IPC", "Vitali", "1mag")
("M-MC", "Spaletta", "1mag")
("M-MSC", "gorrieri", "2mag")
("M-MTI", "roccetti", "2mag")
("M-SM", "panzieri", "1mag")
("S-AG", "cagliari", "1ipm")
("S-AI", "roccetti", "1ipm")
("S-ASD", "Marzolla", "2ipm")
("S-EA", "Aprile", "1ipm")
("S-M", "fumagalli", "2ipm")
("S-PI", "messina", "1ipm")
("S-SM", "ghini", "2mag")
("S-SN", "loli", "2ipm")
("S-SO", "sangiorgi", "2ipm")
("S-TI", "fumagalli", "3ipm")

;
param ns :
          1 2 3 4  :=
"I-AG" 3 2 . .
"I-ASD" 3 3 2 1
"I-CPS" 3 2 . .
"I-F" 2 2 2 .
"I-IT" 3 2 . .
"I-LAM" 3 2 . .
```

```
"I-LP" 2 2 2 .
"I-PSV" 2 2 2 .
"I-SC" 3 3 . .
"I-SC2" 2 1 . .
"I-SI" 2 2 2 .
"I-SO" 2 2 . .
"I-TW" 3 3 . .
"M-ASP" 2 2 . .
"M-ASS" 3 2 . .
"M-C" 2 2 . .
"M-IA" 3 2 . .
"M-IAR" 3 3 . .
"M-IPC" 2 3 3 .
"M-MC" 3 3 . .
"M-MSC" 3 3 . .
"M-MTI" 3 2 . .
"M-SM" 3 3 . .
"S-AG" 2 3 . .
"S-AI" 3 3 . .
"S-ASD" 3 2 . .
"S-EA" 3 3 . .
"S-M" 3 2 . .
"S-PI" 2 2 . .
"S-SM" 2 3 . .
"S-SN" 3 3 . .
"S-SO" 5 3 1 .
;

//Preferenze scelte da ogni professore
param weights :=

["Bononi",*,*] (tr) :
           1          2          3          4          5 :=
1          0          1          1          1          1
2          0          1          1          1          1
3          0          1          1          1          1
4          0          1          1          1          1
5          0          1          1          1          1
6          0          0.5        1          1          1
7          0          0.5        1          1          1
8          0          0.5        1          1          1
9          0          0.5        1          1          1
10         0          1          1          1          1
["Martini",*,*] (tr) :
           1          2          3          4          5 :=
1          1          1          0.75       1          1
2          1          1          0.75       1          1
3          1          1          0.75       1          1
4          1          1          0.75       1          1
5          1          1          0.75       1          1
6          1          1          0.75       1          1
7          1          1          0          1          1
8          1          1          0          1          1
9          1          1          0          1          1
10         1          1          0          1          1

["Davoli",*,*] (tr) :
           1          2          3          4          5 :=
```

```
1           1           1           1           1           0
2           1           1           1           1           0
3           1           1           1           1           0
4           1           1           1           1           0
5           1           1           1           1           0
6           1           1           1           1           0
7           1           1           1           1           0
8           1           1           1           1           0
9           1           1           1           1           0
10          1           1           1           1           0

["babaoglu",*,*] (tr) :
            1           2           3           4           5 :=
1           1           1           1           1           0
2           1           1           1           1           0
3           1           1           1           1           0
4           1           1           1           1           0
5           1           1           1           1           0
6           1           1           1           1           0
7           1           1           1           1           0
8           1           1           1           1           0
9           1           1           1           1           0
10          1           1           1           1           0

["ferretti",*,*] (tr) :
            1           2           3           4           5 :=
1           1           1           1           0           0
2           1           1           1           0           0
3           1           1           1           0           0
4           1           1           1           0           0
5           1           1           1           0           0
6           1           1           1           0           0
7           1           1           1           0           0
8           1           1           1           0           0
9           1           1           1           0           0
10          1           1           1           0           0

["Casadei",*,*] (tr) :
            1           2           3           4           5 :=
1           1           0           1           0           1
2           1           0           1           0           1
3           1           0           1           0           1
4           1           0           1           0           1
5           1           0           1           0           1
6           1           0           1           0           1
7           1           0           1           0           1
8           1           0           1           0           1
9           1           0           1           0           1
10          1           0           1           0           1

["Vitali",*,*] (tr) :
            1           2           3           4           5 :=
1           1           1           1           1           1
2           1           1           1           1           1
3           1           1           1           1           1
4           1           1           1           1           1
5           1           1           1           1           1
6           1           1           1           1           1
```

```
7          1          1          1          1          1
8          1          1          1          1          1
9          1          1          1          1          1
10         1          1          1          1          1
;
//Associazione corsi-aule
param room_course :=
[*,*] (tr) :
          "E3" "E1" "E2" "M1" "LABE" :=
"I-AG"    1    1    1    0    1
"I-ASD"   1    1    1    0    1
"I-CPS"   1    1    1    0    1
"I-F"1    1    1    0    1
"I-IT"    1    1    1    0    1
"I-LAM"   1    1    1    0    1
"I-LP"    1    1    1    0    1
"I-PSV"   1    1    1    0    1
"I-SC"    1    1    1    0    1
"I-SC2"   1    1    1    0    1
"I-SI"    1    1    1    0    0
"I-SO"    1    1    1    0    0
"I-TW"    1    1    1    0    0
"M-ASP"   1    1    1    0    0
"M-ASS"   1    1    1    0    1
"M-C"1    1    1    0    1
"M-IA"    1    1    1    0    1
"M-IAR"   1    1    1    0    1
"M-IPC"   1    1    1    0    1
"M-MC"    1    1    1    0    1
"M-MSC"   1    1    1    0    1
"M-MTI"   1    1    1    0    0
"M-SM"    1    1    1    0    0
"S-AG"    1    1    1    0    1
"S-AI"    1    1    1    0    1
"S-ASD"   1    1    1    0    1
"S-EA"    1    1    1    0    1
"S-M"1    1    1    0    1
"S-PI"    1    1    1    0    1
"S-SM"    1    1    1    0    0
"S-SN"    1    1    1    0    0
"S-SO"    1    1    1    0    0
"S-TI"    1    1    1    0    0
;
set ExtraConflicts :=
;
set pre_schedule :=
("I-F","E1",1,1,1)
;
end;
```