ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

CAMPUS DI CESENA
SCUOLA DI SCIENZE

Corso di Laurea Magistrale in Ingegneria e Scienze Informatiche

STUDIO DI UN APPLICATIVO PER LA PIANIFICAZIONE DELL'ORARIO UNIVERSITARIO

Elaborata nel corso di: Project Management

Tesi di Laurea di:
ELIA BRACCI

Relatore:
Prof. MARCO ANTONIO BOSCHETTI

Correlatore:
ALESSANDRO FURLATI

ANNO ACCADEMICO 2019–2020
SESSIONE UNICA

PAROLE CHIAVE

Timetabling

Constraint-based Solver

Local-Search

Iterative Forward Search

UniTime

UP2.0

Alle persone a me care, a tutte quelle che mi sono state vicine in questo periodo particolare che stiamo vivendo.
A chi mi ha accompagnato durante questo percorso di crescita, a chi non ha mai smesso di credere in me.

Indice

Elenco delle figure

Listings

Introduzione

In Italia sono presenti 96 atenei distribuiti in tutte le regioni della penisola. Nell'anno accademico 2019/2020 ci sono stati 1.730.563 studenti iscritti a corsi universitari [1] e il personale di Ateneo ammontava a 154.181 di cui 98.461 personale docente e ricercatore [2]. Questi dati evidenziano l'elevato numero di persone e atenei che il nostro sistema universitario coinvolge e il loro costante aumento ha fatto in modo che, con il passare degli anni, le esigenze delle università siano aumentate sempre di più, richiedendo un maggiore sforzo nella corretta pianificazione dell'orario e della gestione delle risorse di ogni ateneo. Durante l'attività di pianificazione dell'orario didattico, il personale di ateneo svolge un ruolo di fondamentale importanza: quello di raccogliere tutte le informazioni necessarie per poi effettuare una corretta pianificazione dell'offerta formativa che rispetti tutti i vincoli. Esempi di vincoli possono essere le preferenze di ciascun docente, le disponibilità delle aule, i curriculum di ogni studente e così via dicendo. Queste operazioni, ovvero la raccolta delle informazioni e la successiva pianificazione, richiedono al personale di ateneo un enorme sforzo e, nonostante ciò, spesso è pressoché impossibile realizzare un orario che non contenga conflitti, visto l'enorme quantitativo di variabili da tenere in considerazione. Da questa problematica nasce l'esigenza di Cineca, un consorzio universitario italiano formato da 2 Ministeri, 22 Istituzioni pubbliche Nazionali e 69 università italiane [3], di offrire a queste ultime un servizio che sia in grado di automatizzare ed ottimizzare la pianificazione dell'orario in base a tutti i vincoli definiti a priori dal personale di ateneo.

Ad oggi, Cineca mette già a disposizione delle università aderenti un sistema manuale di pianificazione della didattica, offerto sotto forma di applicativo web che prende il nome di UP2.0. Tuttavia, il consorzio ha intenzione di migliorare e incrementare le funzionalità del loro prodotto.

Prima di procedere con la progettazione e la successiva implementazione di un algoritmo ad hoc per la risoluzione del problema sopra descritto, è stata effettuata un'analisi del mercato per evidenziare soluzioni già esistenti. Qui entra in gioco UniTime, un sistema per la pianificazione della didattica ritenuto idoneo a soddisfare le esigenze di Cineca.

UniTime è un sistema open source progettato e sviluppato da ricercatori americani per facilitare la pianificazione della didattica a tutte le università e college che hanno deciso di aderire al progetto.

Lo scopo principale di questa tesi è quello di comprendere se UniTime sia in grado di soddisfare veramente le esigenze di Cineca, così da offrire una qualità migliore del servizio a tutte le università che usano la sua soluzione, per agevolare il lavoro del personale di ateneo ed ottenere una maggiore qualità della pianificazione degli orari.

Il sistema UniTime viene offerto sotto forma di piattaforma web, ma viste le esigenze di Cineca di integrare l'algoritmo utilizzato dal sistema e di non avvalersi dell'interfaccia web, la tesi si concentrerà maggiormente sull'esecuzione del risolutore da riga di comando. Nonostante ciò, il documento contiene qualche riferimento agli studi effettuati sull'applicativo web, svolti per capirne il funzionamento e il modello dati su cui si basa.

La tesi è strutturata come segue: il primo capitolo contiene un'analisi generale sul sistema universitario italiano, dove opera Cineca, e sul sistema universitario americano, paese di riferimento di UniTime; segue un capitolo di descrizione del prodotto di Cineca UP2.0, per definirne le funzionalità e i limiti del prodotto stesso. La tesi continua con un capitolo dedicato al sistema UniTime, la sua struttura, le sue funzionalità ed un capitolo interamente dedicato all'algoritmo utilizzato per la risoluzione dei problemi di pianificazione degli orari. Infine, si conclude con i capitoli sull'utilizzo e l'integrazione di UniTime con il modello dati del prodotto di Cineca, i risultati ottenuti e le conclusioni.

Capitolo 1

Sistemi Universitari a Confronto

1.1 Sistema Universitario Italiano

L'università è la sede primaria di libera ricerca e di libera formazione nell'ambito del suo ordinamento ed è luogo di apprendimento ed elaborazione critica delle conoscenze; opera, ispirandosi al principio di autonomia e responsabilità, combinando in modo organico ricerca e didattica, per il progresso culturale, civile ed economico della Repubblica (Legge 240/2010, art1, comma 1 e 2).

Il sistema universitario italiano [4] prevede:

- Sessantasette (67) università statali;
- Ventinove (29) università non statali legalmente riconosciute;
- Nove (9) istituti superiori a ordinamento speciale;
- Undici (11) università telematiche.

Il sistema [5] si articola sui 3 cicli del Processo di Bologna, un accordo intergovernativo di collaborazione nel settore dell'istruzione superiore istituito nel 1999 [6]. L'iniziativa di questo processo era stata lanciata nella Conferenza di Bologna [7], una conferenza in cui 29 paesi hanno espresso la loro volontà di impegnarsi per migliorare la competitività dell'Istruzione

Superiore Europea, confermando gli obiettivi della Dichiarazione della Sorbona del 1998 [8]. Quest'ultima, firmata dai ministri di Francia, Germania, Italia e Regno Unito, aveva come compito principale quello di costruire uno Spazio Europeo dell'Istruzione Superiore che si basasse su principi e criteri condivisi tra i paesi firmatari, ovvero:

- libertà accademica, autonomia istituzionale e partecipazione di docenti e studenti al governo dell'istruzione superiore;
- qualità accademica, sviluppo economico e coesione sociale;
- incoraggiamento alla libera circolazione di studenti e docenti;
- sviluppo della dimensione sociale dell'istruzione superiore;
- massima occupabilità e apprendimento permanente dei laureati;
- considerazione di studenti e docenti quali membri della medesima comunità accademica;
- apertura all'esterno e collaborazione con sistemi di istruzione superiore di altre parti del mondo.

Da questi obiettivi successivamente ne derivano quelli della Conferenza di Bologna e sono:

- adottare un sistema di titoli di studio facilmente leggibili e comparabili;
- adottare un sistema comune basato su due cicli principali. L'accesso al secondo ciclo richiede il completamento del primo ciclo con durata minima di tre anni;
- istituire un sistema di crediti per promuovere la mobilità studentesca come l'ECTS, il sistema europeo di accumulazione e trasferimento dei crediti utilizzato come strumento dello spazio europeo dell'istruzione superiore per rendere più trasparenti gli studi e i corsi. Inoltre, questo sistema permette agli studenti di spostarsi da un paese all'altro ottenendo il riconoscimento dei titoli e dei periodi di studio all'estero [9];

- promuovere la mobilità degli studenti, dei docenti, dei ricercatori e del personale amministrativo;

- promuovere la cooperazione europea in garanzia della qualità dei corsi e dei titoli di studio;

- promuovere un'unica dimensione Europea necessaria per l'istruzione superiore, con particolare riguardo allo sviluppo dei curriculum universitari, alla cooperazione tra gli istituti e allo sviluppo di programmi di studio e ricerca integrati.

Gli stati firmatari della dichiarazione sono: Austria, Belgio, Bulgaria, Repubblica Ceca, Estonia, Danimarca, Francia, Finlandia, Germania, Ungheria, Grecia, Irlanda, Islanda, Lettonia, Italia, Lussemburgo, Lituania, Malta, Paesi Bassi, Norvegia, Polonia, Portogallo, Romania, Repubblica Slovacca, Slovenia, Spagna, Svezia, Confederazione Svizzera e Regno Unito.

I principali titoli italiani sono la Laurea (1° ciclo), la Laurea Magistrale (2° ciclo) e il Dottorato di Ricerca (3° ciclo). Il sistema italiano offre anche altri corsi accademici con i relativi titoli.

1.1.1 Primo Ciclo

I Corsi di Laurea costituiscono esclusivamente il primo ciclo e hanno l'obiettivo di assicurare agli studenti un'adeguata padronanza di metodi e contenuti scientifici generali e l'acquisizione di specifiche conoscenze professionali. Per poter accedere a questo primo ciclo è richiesto come requisito minimo il diploma finale di scuola secondaria che viene rilasciato al completamento di 13 anni di scolarità complessiva e a seguito del superamento del relativo esame di Stato oppure un titolo estero comparabile. Tuttavia, in base alla casistica di interesse, l'ammissione può essere subordinata alla verifica di ulteriori requisiti. I Corsi di Laurea del primo ciclo hanno durata triennale e per poter conseguire il titolo di Laurea, ogni studente dovrà aver acquisito 180 Crediti Formativi Universitari (CFU), equivalenti ai crediti ECTS. Inoltre, per il conseguimento del titolo può essere richiesto un periodo di tirocinio e la discussione di una tesi oppure la preparazione di un elaborato finale. Il possedimento del titolo di Laurea dà accesso alla Laurea Magistrale e tutti gli altri corsi del secondo ciclo.

1.1.2 Secondo Ciclo

I Corsi di Laurea Magistrale sono i principali corsi che costituiscono il secondo ciclo. Essi offrono una formazione di livello avanzato per l'esercizio di attività di elevata qualificazione in ambiti specifici. L'accesso a questi corsi è determinato dal possesso di una Laurea o di un titolo estero comparabile e l'ammissione è soggetta a specifici requisiti decisi dalle singole università. I corsi, di durata biennale, richiedono allo studente l'acquisizione di 120 crediti CFU e l'elaborazione, con relativa discussione, di una tesi di ricerca per poter conseguire il titolo di Laurea Magistrale. Alcuni corsi, come ad esempio Medicina e chirurgia, Medicina veterinaria, Odontoiatria e protesi dentaria, Farmacia e Farmacia industriale, Architettura e Ingegneria edile-Architettura, Giurisprudenza, Scienze della formazione primaria, sono definiti "Corsi di Laurea Magistrale a ciclo unico" ed hanno una durata complessiva di 5 anni (6 anni per Medicina e Chirurgia e per Odontoiatria e protesi dentaria). Per questa specifica tipologia è sufficiente il diploma di scuola secondaria superiore o un titolo estero equivalente come requisito di accesso. Tuttavia, l'ammissione è subordinata a una prova di selezione e per poter conseguire il titolo di Laurea Magistrale lo studente deve quindi aver acquisito 300 CFU (360 per Medicina e Chirurgia e per Odontoiatria e protesi dentaria) ed aver elaborato e discusso una tesi di ricerca. Il conseguimento del titolo di Laurea Magistrale dà accesso al Dottorato di Ricerca e agli altri corsi di terzo ciclo.

1.1.3 Terzo Ciclo

I principali corsi del terzo ciclo sono quelli di Dottorato di Ricerca; essi hanno l'obiettivo di far acquisire una corretta metodologia per la ricerca scientifica avanzata, adottano tecniche innovative e nuove tecnologie, prevedono stage all'estero e la frequenza di laboratori di ricerca. Per poter essere ammessi a questa tipologia di corsi è richiesta una Laurea Magistrale, o un titolo estero equivalente, e il superamento di un concorso. La durata degli studi è di minimo 3 anni e il dottorando deve elaborare una tesi originale di ricerca e discuterla durante l'esame finale.

1.1.4 Altri Corsi

Nel sistema universitario italiano sono previsti anche ulteriori corsi, quali:

- Corsi di Specializzazione: corsi di 3° ciclo con l'obiettivo di fornire conoscenze e abilità per l'esercizio di attività professionali di alta qualificazione, particolarmente nel settore delle specialità mediche, cliniche e chirurgiche. L'ammissione a questi corsi di specializzazione è subordinata al possesso di una Laurea Magistrale (o un titolo estero equivalente) e al superamento di un concorso. La durata degli studi varia da 2 (120 CFU) a 6 anni (360 CFU) in rapporto al settore disciplinare ed il titolo finale rilasciato è il Diploma di Specializzazione;

- Corsi di Master universitario di primo livello: sono corsi del secondo ciclo di perfezionamento scientifico o di alta formazione permanente e ricorrente. Vi si accede con una Laurea o con un titolo estero comparabile e la durata minima è annuale (60 CFU). Tale corso non consente l'accesso a corsi di Dottorato di Ricerca e di 3°ciclo, perché non ha ordinamento didattico nazionale e il titolo è rilasciato sotto la responsabilità autonoma della singola università. Il titolo finale è il Master universitario di primo livello;

- Corsi di Master Universitario di secondo livello: corsi di 3° ciclo di perfezionamento scientifico o di alta formazione permanente e ricorrente. Vi si accede con una Laurea Magistrale o con un titolo estero comparabile ed hanno una durata minima annuale (60 CFU). Questa tipologia di corsi non consente l'accesso a corsi di Dottorato di Ricerca e di 3° ciclo, perché il corso non ha ordinamento didattico nazionale e il titolo è rilasciato sotto la responsabilità autonoma della singola università. Il titolo finale è il Master universitario di secondo livello.

1.1.5 Crediti Formativi Universitari (CFU)

I corsi di studio sono strutturati in crediti e a ciascun Credito Formativo Universitario (CFU) corrispondono normalmente 25 ore di lavoro dello studente, ivi compreso lo studio individuale. La quantità media di lavoro accademico svolto in un anno da uno studente a tempo pieno è convenzionalmente fissata in 60 CFU. Come accennato in precedenza in questa sezione, i crediti formativi universitari sono equivalenti ai crediti ECTS.

1.1.6 Classi dei Corsi di Studio

I corsi di studio di Laurea e di Laurea Magistrale che condividono obiettivi e attività formative sono raggruppati in "classi". I contenuti formativi di ciascun corso di studio sono fissati autonomamente dalle singole università; tuttavia, le università devono obbligatoriamente inserire alcune attività formative ed il corrispondente numero di crediti determinate a livello nazionale. Tali requisiti sono stabiliti in relazione a ciascuna classe e i titoli di una stessa classe hanno lo stesso valore legale.

1.1.7 Titoli Accademici

La Laurea dà diritto alla qualifica accademica di "Dottore"; la Laurea Magistrale dà diritto a quella di "Dottore magistrale"; il Dottorato di Ricerca conferisce il titolo di "Dottore di ricerca" o "PhD".

1.1.8 Titoli Congiunti

Le università italiane possono istituire corsi di studio in cooperazione con altre università, italiane ed estere, al termine dei quali sono rilasciati titoli congiunti o titoli doppi/multipli.

1.2 Sistema Universitario Americano

Prima di analizzare nello specifico il sistema universitario americano è bene effettuare un'inquadratura generale del sistema scolastico americano. Il sistema scolastico americano è gestito dal settore pubblico ed è obbligatorio fino ai 18 anni (16 in alcuni Stati). Negli Stati Uniti sono presenti le scuole più costose del mondo per quanto riguarda le università, sia che si parli di scuole pubbliche o private; le tasse annuali di iscrizione si aggirano intorno ai 20,000 dollari per le scuole pubbliche, mentre le università private possono superare i 60,000 dollari annui. Fino all'high school la scuola è del tutto gratuita, compresi i libri e l'iscrizione, ma vi possono essere delle tasse pagate dai residenti del distretto.

Il sistema scolastico è diviso in diversi moduli a seconda dello Stato e nel caso in cui i ragazzi non frequentino la scuola lo Stato esorta i genitori ad obbligarli a frequentarla.

Ogni Stato federale ha una limitata autonomia per quanto riguarda l'educazione scolastica; devono attenersi a delle caratteristiche generali proposte dalla Costituzione degli Stati Uniti relative a fondi, controlli, insegnanti e le loro certificazione, libri di testo, biblioteche.

L'età di accesso alla scuola varia da stato a stato dai cinque anni a otto anni terminando con i 18 anni [10].

1.2.1 Cicli di Istruzione

In America sono presenti 3 livelli di istruzione che seguono la scuola dell'infanzia (chiamata Pre-School, Nursery School o Head Start) e sono:

- Elementary School o Elementary Level: rappresentano il primo livello di istruzione e corrisponde alla nostra scuola elementare con durata di 5 anni;
- Middle School e High School (Secondary Level): rappresentano il secondo livello di istruzione. La Middle School corrisponde alla scuola media inferiore italiana (o scuola secondaria di primo grado) e si divide in tre anni chiamati: sesto, settimo e ottavo; mentre la High School corrisponde alla scuola secondaria di secondo grado italiana, ma con durata di 4 anni che prendono il nome di nono, decimo, undicesimo e dodicesimo;
- College o University: rappresentano il terzo livello di istruzione e consistono in 4 anni di studio, a differenza delle università italiane le quali hanno una durata che varia dai 3 ai 6 anni in base alla tipologia di laurea, come visto nella sezione 1.1.

Nelle prossime sezioni verrà effettuato un excursus sul terzo livello di istruzione del sistema scolastico americano.

1.2.2 College e Università

Come accennato nella sezione precedete, sezione 1.2.1, il terzo livello è rappresentato dal livello universitario che include i college e le università con una durata complessiva di quattro anni di studio. Gli istituti possono essere sia privati che pubblici. La differenza sostanziale tra college e università risiede nella mole degli istituti e negli ambiti in cui essi spaziano: il college propone una scelta di materie in uno o due campi di studio, mentre l'università propone una gamma ben più ampia di opzioni, che comprendono quasi tutti i campi della conoscenza. Un'altra differenza importante sta nel fatto che il college si conclude in 4 anni (Bachelor), mentre al termine dell'università si può proseguire con Master o Dottorati.

In generale negli Stati Uniti con la parola *University* si intendono i luoghi dove è anche possibile ricevere una formazione più avanzata, come ad esempio master, dottorati di ricerca e altre specializzazioni, mentre il college è paragonabile alla tipologia delle università europee [11].

Le università sono a pagamento e un anno di corso costa dai 15.000-20.000 dollari per le grandi università pubbliche, ai 70.000 dollari per quelle private, importo che copre le spese di alloggio e le spese per i corsi frequentati. Tuttavia, sono disponibili borse di studio per gli studenti meritevoli.

Titoli di studio universitari

Negli Stati Uniti il titolo di studio più diffuso è il Bachelor's degree che richiede 4 anni di studio ad eccezione di architettura che ne richiede 5. I corsi da seguire per conseguire un Bachelor si dividono in 3 categorie:

- general education: corsi di cultura generale e abilità di base;
- field of concentration o major: corsi di specializzazione;
- electives: corsi a libera scelta.

La scelta del major può essere posticipata di uno o due anni senza che questo pregiudichi il conseguimento del Bachelor. Il corso di Bachelor è diviso in due bienni:

- lower division: i primi due anni si fanno studi di base detti Freshman e Sophomore years, rispettivamente primo e secondo anno;

- upper division: corsi avanzati e specialistici detti junior e senior years.

Normalmente, il Bachelor non richiede la presentazione di una tesi. Inoltre, vi è un titolo di studio più breve, ovvero l'Associate Degree della durata complessiva di 2 anni e che si consegue presso college biennali detti community college, i cosiddetti college della comunità. Essi risultano di facile accesso ed hanno un costo relativamente basso. Esistono poi l'Associate of Applied Science o altri titoli di associate, denominati con la disciplina di specializzazione, che mirano all'occupazione immediata in ambito tecnico o semiprofessionale. Questa tipologia di corsi è focalizzata sulla disciplina scelta dallo studente ed è poco adatta a consentire un proseguimento accademico. I college biennali offrono spesso studi orientati ad acquisire specifiche competenze lavorative ed attestati al termine con Certificates o Diplomas.

Gli studi per sostenere il Bachelor's degree o l'Associate Degree sono detti di livello undergraduate. Tuttavia, esistono titoli di livello più avanzato, ovvero di livello graduate. Per poter accedere a questi corsi occorre aver conseguito il Bachelor, che è ritenuto equivalente alla nostra laurea triennale. In alcuni istituti non è necessario aver completato il Bachelor per essere ammessi. Gli studenti intenzionati ad effettuare un passaggio di corso da undergraduate a graduate hanno la possibilità di iscriversi ad una disciplina diversa da quella studiata precedentemente nel Bachelor.

I titoli di livello più avanzato di Bachelor possono essere divisi in tre categorie:

- master degree;
- first professional advanced degrees;
- doctor's degree.

Il master degree è un corso di studi che ha una durata pari ad uno o due anni dopo il conseguimento del Bachelor. Questi corsi si concentrano su un argomento specifico e sono orientati ad uno sbocco professionale, come ad esempio il Master of Arts, il Master of Science e il Master of Business Administration che è uno dei più diffusi e dei più utili nel mondo del lavoro volto a formare quadri e manager aziendali. Un altro master di significativa importanza è il Master of Education, un master relativo all'insegnamento scolastico. Altri master importanti sono il Master of Engineering, il Master of Architecture, il Master of Music, il Master of Journalism, ecc.

First Professional Advanced Degrees Per poter accedere alle professioni mediche e legali è necessario frequentare i First Professional Advanced Degrees, offerti da scuole professionali che spesso sono delle facoltà all'interno delle università stessa oppure istituti autonomi. L'ammissione è subordinata ad un esame molto selettivo ed i titoli più importanti che possono essere conseguiti sono il Juris Doctor e il Doctor of Medicine. Il Juris of Doctor server ad istruire avvocati o magistrati, ha una durata complessiva di tre anni e per potervi accedere è necessario aver completato il Bachelor; mentre il Doctor of Medicine ha una durata di tre anni di undergraduate più ulteriori quattro anni graduate. Altri titoli della stessa tipologia sono il Doctor of Dental Medicine, il Doctor of Pharmacy e tutti gli altri titoli in discipline teologiche per poter svolgere cariche ecclesiastiche.

Doctor's Degree I Doctor's Degree sono i titoli di dottorato in cui lo studente effettua studi avanzati e di ricerca in una specifica disciplina. Per poter accedere ad uno di questi corsi è necessario essere in possesso del Bachelor e, per certi corsi, anche di un master o un titolo professionale nella stessa disciplina. Lo studente deve scrivere e discutere una tesi a seguito di una ricerca autonoma con la supervisione di un docente. Molti istituti offrono la possibilità di seguire degli studi congiunti detti Joint o Dual Degrees che portano a conseguire due titoli diversi, per esempio un titolo combinato in legge e amministrazione aziendale.

1.3 Università Italiana VS Università Americana

In questa sezione verranno analizzate differenze ed analogie tra i due sistemi sopra descritti, ovvero il sistema universitario italiano, sezione 1.1, e il sistema universitario americano, sezione 1.2.

1.3.1 Differenze

In Italia si tende a fornire una buona formazione generale già alle scuole secondarie superiori e si punta alla specializzazione quando si frequenta l'università, tanto che in passato nemmeno esisteva il dottorato perché si riteneva che la preparazione universitaria fosse sufficiente all'abilitazione in campi quali l'insegnamento e la ricerca [12].

In America, invece, l'università garantisce una formazione più generale volta a colmare le lacune del liceo. I primi corsi che si frequentano offrono una panoramica della materia, senza approfondirla. Ogni corso di laurea, inoltre, prevede dei core program, ossia dei corsi obbligatori che pongono le basi necessarie per proseguire il proprio percorso e questo sistema è adottato anche da tutte le università italiane.

Un'altra differenza tra università italiana e americana è che, oltre al principale settore di studi in cui ci si intende specializzare (major), si può optare anche per un percorso secondario (minor) con corsi non per forza correlati alla principale area di studio. In questo modo uno studente può avere, ad esempio, un major in economia e un minor in medicina. Questo sistema fa sì che un ragazzo possa avere una cultura approfondita nel settore in cui intende specializzarsi, ma allo stesso tempo di perseguire altre sue passioni che può sviluppare in un percorso secondario.

Se alcune università richiedono solo il major, altre esortano gli studenti ad ottenere anche il minor; altre ancora impongono l'ottenimento di entrambi, ma su tematiche correlate. In questo caso bisogna quindi scegliere materie appartenenti allo stesso ambito: sociale, storico, umanistico, scientifico, ecc.

Il sistema di valutazione

Il sistema di valutazione degli studenti universitari in America è molto diverso da quello italiano. Le differenze si concentrano principalmente su quattro aspetti:

- voti;
- prove ed esami;
- bocciatura;
- periodo degli esami.

Voti Se in Italia si usano i trentesimi, in USA il sistema di valutazione prevede le lettere. Il voto più alto è la A+ e corrisponde al 30 cum laude, mentre un voto insufficiente è segnato con la lettera F, corrispettivo della gamma dei voti inferiori al 18.

SISTEMA DI VALUTAZIONE ITALIANO	SISTEMA DI VALUTAZIONE USA	VALUTAZIONE
30 cum laude	A+	outstanding
30	A-	excellent
29 28 27	B+ B B-	very good to good
26 25-24 23	C+ C C-	satisfactory to adequate
22-21 20-19	D+ D	poor to barely adequate
18	D-	minimum passing grade
<18	F	fail

Figura 1.1: Tabella Voti Universitari Americani e Italiani a Confronto

Prove ed esami La seconda grande differenza con l'Italia è che la maggior parte dei corsi non prevede solo un esame finale ma anche degli step intermedi. Le prove si dividono in assignment, mid-term exam e final exam:

- assignment: sono elaborati scritti chiesti dal professore di settimana in settimana, talvolta ogni mese. Sono valutati con un voto che corrisponde a una percentuale sul voto finale dell'esame. Questo vale soprattutto per le materie scientifiche in cui sono previsti esercizi e attività da svolgere. Negli ultimi anni numerosi corsi nelle università italiane hanno adottato questa metodologia;

- mid-term exam: è la prova che vuole certificare le nozioni apprese fino a un certo punto del corso, un preludio all'esame finale paragonabile ad una prova parziale nel sistema italiano. Non tutti i corsi lo prevedono;

- final exam: è la prova finale che definisce il voto definitivo del corso seguito. La particolarità rispetto al sistema di valutazione italiano è che ogni studente è valutato esclusivamente con verifiche scritte. L'unico momento in cui ci si allena nell'esposizione orale è in classe, ma non si tratta di esami veri e propri bensì di esercitazioni.

Per quanto riguarda gli ultimi anni ci sono alcune specificità. Superati gli esami scritti relativi ai corsi più generali, viene richiesto allo studente un ulteriore impegno nella produzione di temi o relazioni settimanali. Alla fine di ogni corso è necessario consegnare un documento di circa una trentina di pagine da presentare alla classe. Soprattutto per i corsi letterari sono previste letture obbligatorie da discutere con i colleghi di facoltà. Inoltre, l'intervento e la partecipazione in classe sono ritenuti obbligatori e sono considerati nella valutazione finale di ogni singolo studente.

Bocciatura La terza grande differenza con l'Italia è che non è possibile ridare un esame. Nel caso in cui non si dovesse superare un corso con una votazione sufficiente, questo viene registrato e sarà necessario pagare nuovamente la tuition per poter seguire il corso per una seconda volta e superare con esito positivo tutte le prove previste. Per questo motivo, in America la bocciatura è considerata in modo molto negativo. Nel caso in cui si avessero problemi di salute il giorno dell'esame finale, è possibile spostare la data con un certificato medico che dimostri le proprie condizioni. Tendenzialmente in Italia le università offrono la possibilità di poter ridare l'esame. Tuttavia, alcune università adottano un sistema simile a quello americano, ovvero che non offrono la possibilità di poter ripetere l'esame, ma

questo avviene solo nel caso in cui l'esame sia stato superato; in alternativa lo studente potrà riprovarlo nella prossima sessione disponibile.

Periodo degli esami La quarta differenza con l'Italia riguarda il periodo dedicato agli esami. Quest'ultimo, nelle università americane tendenzialmente è una finestra di una decina di giorni fissata a priori da un organo amministrativo di ciascuna università, durante la quale si devono sostenere tutti gli esami. Ciascuno studente può effettuare fino a due esami al giorno; qualora ce ne fossero più di due è possibile fare richiesta di spostarne uno.

Offerta Formativa

Come precedentemente accennato, tutti i bachelor prevedono lo studio di una disciplina prevalente, denominata major, come ad esempio fisica, matematica, lettere, eccetera. Ma a differenza del sistema italiano, lo spazio dedicato alla disciplina scelta non riempie interamente il curriculum, ma è tipicamente pari a solo il 50-60% circa del totale che corrisponde all'equivalente di due anni di studi distribuiti includendo anche i corsi propedeutici di altre discipline. Il resto del curriculum è composto da insegnamenti di cultura generale, abilità di base o nozioni di altre discipline anche molto lontane da quella del major, offrendo così allo studente un'ampia gamma di scelte libere. Rispetto ai corsi di laurea italiani, quelli americani sono caratterizzati da una grande flessibilità del curriculum. Lo studente statunitense gode di molta libertà nello stabilire autonomamente tutto il proprio piano di studi scegliendo, all'interno dell'intera offerta didattica dell'università, sia i corsi da sostenere che la loro collocazione temporale, con il solo vincolo di raggiungere entro la fine del quadriennio i requisiti curriculari legati al major ed alla formazione generale. Questa flessibilità permette all'università di soddisfare le diverse esigenze, talvolta contraddittorie dei suoi studenti, senza dover impiegare un numero enorme di docenti e senza attivare una moltitudine di corsi di laurea diversi, come accade in Italia. Certi studenti sono più orientati a studi eclettici e multidisciplinari, altri invece sono più focalizzati su un singolo argomento; qualcuno è più orientato ad acquisire presto tutte le competenze professionali utili al lavoro, mentre qualcun altro è mosso da interessi culturali; alcuni risultano essere più propensi a studi teorici e formalizzazioni, altri invece si specializzano in attività pratiche e operative [13].

Divisione in Classi

L'elasticità della scelta formativa analizzata nella sottosezione precedente comporta un numero elevato di studenti per ciascun insegnamento. Questa peculiarità porta le università americane, a differenza di quelle italiane, a dividere ciascun insegnamento in classi. Studenti che hanno scelto lo stesso curriculum e quindi di conseguenza gli stessi insegnamenti possono ritrovarsi a frequentare classi diverse, mantenendo nell'eventualità lo stesso docente. Questa situazione in Italia è difficile che si verifichi. I piani di studi stabiliti a priori e con una scarsa flessibilità, data da un basso numero di esami a scelta, non portano ad effettuare una divisione degli studenti in classi per ciascun insegnamento. È possibile però che siano presenti alcune eccezioni, come ad esempio l'Università Bocconi. Quest'ultima adotta un sistema di divisione in classi nonostante abbia anch'essa una scarsa flessibilità sulla scelta del curriculum da parte di ciascuno studente.

1.3.2 Analogie

Oltre alle differenze elencate nella sezione precedente, sezione 1.3.1, i due sistemi universitari hanno comunque una struttura similare sotto numerosi punti di vista. Entrambi i sistemi analizzati utilizzano i crediti formativi per ciascun insegnamento, i curriculum per definire il piano di studi degli studenti ed hanno una gestione similare delle risorse come ad esempio docenti, aule, ecc.

Capitolo 2

Problema della Pianificazione dell'Orario

2.1 Definizione del Problema

L'approfondimento del capitolo 1 relativo al sistema universitario italiano e americano ha messo in luce quale sia la struttura che li definisce e ne ha mostrato analogie e differenze. Nonostante ciò, in entrambe le situazioni rimane evidente un problema che li accomuna, quello della pianificazione degli insegnamenti e dell'allocazione delle risorse. L'elevato numero di studenti, di aule e di docenti per ciascuna università comportano seri problemi nella ricerca di un orario che sia ottimale e minimizzi i conflitti.

Nelle sezioni successive verranno analizzate nel dettaglio le differenti problematiche che si possono verificare nella pianificazione delle risorse.

2.2 Gestione delle Risorse

2.2.1 Studenti

Ogni studente ha la possibilità, oltre agli insegnamenti obbligatori, di scegliere esami a scelta che definiscono il curriculum. Per questo è importante che gli insegnamenti comuni a tutto il corso di studi non si sovrappongano nel tempo e che i conflitti di orario degli esami a scelta siano minimizzati se non nulli.

2.2.2 Docenti

Ciascun docente può essere titolare di uno o più insegnamenti; ciò comporta che nell'orario che si dovrà generare bisognerà tenere in considerazione che due insegnamenti tenuti dallo stesso docente non si dovranno sovrapporre nel tempo, causando l'impossibilità della persona di partecipare ad entrambe le programmazioni contemporaneamente.

Indisponibilità

L'indisponibilità dei docenti è uno dei vincoli di maggiore importanza da considerare durante la pianificazione; è bene perciò conoscere a priori le disponibilità del docente. Esempi di indisponibilità sono impegni personali oppure insegnamenti già pianificati in altre università.

Preferenze

Un docente ha delle preferenze di orario o di giorni. Queste preferenze possono essere determinate da una serie di fattori esterni come, ad esempio, la distanza del docente dall'ateneo.

2.2.3 Aule

Ogni campus universitario è composto da un elevato numero di aule; rimane perciò complessa la gestione di questa risorsa. Durante l'attività di pianificazione dell'orario è importante considerare come vincoli tutte le peculiarità/caratteristiche di questa risorsa, per evitarne la sovrapposizione, ovvero la situazione in cui due insegnamenti siano tenuti nella stessa aula, lo stesso giorno, allo stesso orario, e per garantire il corretto svolgimento di tutte le lezioni. Nel caso di studio analizzato, aule e stanze sono due termini che assumono lo stesso significato e, quindi, sono intercambiabili tra loro.

Indisponibilità

Ciascun'aula può essere indisponibile in maniera temporanea o definitiva. Questa situazione deve poter essere gestita e considerata come vincolo durante la pianificazione dell'orario.

Caratteristiche

Ogni aula è definita da diverse caratteristiche, quali:

- capienza: ovvero il numero di postazioni presenti all'interno dell'aula, che dovrà essere considerato durante la pianificazione tenendo conto del numero di studenti che seguono l'insegnamento di riferimento;
- superficie: definita in metri quadri per garantire la corretta dimensione dell'aula e la distanza tra i vari studenti, ad esempio per il corretto svolgimento delle prove d'esame o nella situazione attuale di emergenza sanitaria COVID-19;
- tipologia: esempi di tipologie possono essere laboratorio, aula magna ecc.;
- posizione: ovvero la posizione geografica dell'aula interessata, definita come latitudine e longitudine oppure x e y. Questa informazione è utile per definire la distanza tra un'aula e l'altra in modo da evitare di collocare due lezioni adiacenti in aule troppo distanti tra loro;
- dotazioni: ovvero le attrezzature presenti nell'aula come i computer e il relativo numero di postazioni, la lavagna, il proiettore, ecc.;
- dipartimenti: ciascun'aula può essere utilizzata da uno o più dipartimenti; è bene conoscere questa informazione a priori per poter effettuare una corretta pianificazione.

Durante la pianificazione dell'orario le caratteristiche sopra elencate dovranno essere tutte prese in considerazione come vincoli da rispettare.

Nei capitoli successivi verranno analizzate nello specifico le soluzioni attualmente utilizzate e utilizzabili per la pianificazione dell'orario nel rispetto dei vincoli relativi alla gestione delle risorse sopra descritte.

Capitolo 3

University Planner 2.0

3.1 UP2.0

UP, acronimo di University Planner, è l'applicazione Cineca per la gestione dei calendari e dell'occupazione delle risorse in un ateneo universitario. In UP possono essere inseriti eventi per la prenotazione ed allocazione di spazi, aule e risorse [14]. L'applicazione web è utilizzata da circa 20 università della penisola, che ne sfruttano la potenzialità per calendarizzare eventi, insegnamenti ed esami. UP2.0 si caratterizza come un nuovo applicativo e non come un aggiornamento della versione precedente, in quanto è stata riorganizzata tutta la sua architettura; nelle prossime sottosezioni verrà analizzato il prodotto nello specifico, evidenziandone caratteristiche, novità e funzionalità.

Figura 3.1: Quadro Generale University Planner

3.1.1 Descrizione del Prodotto

Cosa si può fare

UP2.0 mette a disposizione dell'utente finale numerose funzionalità per una perfetta pianificazione dell'orario e gestione dell'ateneo. Le funzionalità principali sono:

- pianificare per diverse viste (per calendario evento, aula, a lista);
- gestire le variazioni giornaliere (sospensioni, variazioni orario, annullamenti);
- notificare la variazione a docenti e operatori o a sistemi esterni;
- gestire periodi o date di indisponibilità;
- tracciare i cambiamenti di orario;
- controllare le sovrapposizioni di orario docenti/aule;
- prenotare aule per eventi con workflow di approvazione;
- produrre pagine web con diverse viste per la visualizzazione dell'orario;
- produrre per monitor negli edifici/aule;
- produrre reportistica di controllo a scopo statistico.

Caratteristiche Principali

Questa nuova tecnologia, ovvero UP2.0, ha portato ad avere numerosi vantaggi rispetto alla versione precedente che possono essere riassunti nei seguenti punti:

- utilizzo di nuove tecnologie più adatte;
- non richiede installazione client;
- migliori performance nel recupero dati;
- architettura Saas efficiente e scalabile;
- gli aggiornamenti non fermano il servizio (100% up time).

Novità UP2.0

Oltre ai vantaggi, questa piattaforma innovativa ha portato a numerose novità che la differenziano e caratterizzano rispetto alla precedente:

- unica interfaccia web per utente e le funzionalità dell'amministratore;
- funzionalità concentrate in un'unica area di lavoro;
- gestione delle "indisponibilità" immediata;
- flusso prenotazione aule più efficiente (grazie al sistema di notifica);
- possibilità di condividere "filtri" operativi fra più utenti.

3.2 Funzionalità

3.2.1 Ciclo Gestionale della Didattica

Il ciclo della gestione della didattica si suddivide in due fasi principali. La prima fase, ovvero quella di Planning, ha il compito di programmare la didattica e pianificare l'orario delle lezioni, definire il calendario degli esami allocando per ciascun appello l'aula che si ritiene più adatta all'esigenza. La seconda fase, fase di esercizio, non è altro che la fase di gestione in cui vengono monitorati gli orari pianificati, tenendo conto di eventuali variazioni di orario o prenotazioni di aule.

Figura 3.2: Ciclo Gestionale della Didattica

3.2.2 Workflow Pianificazione delle Lezioni

Il workflow relativo alla pianificazione delle lezioni ha inizio con la fase di planning vista nella sottosezione 3.2.1, quindi dalla programmazione didattica. Una volta terminata questa fase e ottenuta l'offerta formativa da pianificare, si passa all'allocazione delle aule, generando un calendario settimana per settimana e si termina con la pubblicazione dell'orario. Infine, tramite l'utilizzo di sondaggi, App Studenti, Course Catalog e così via dicendo, viene effettuata un'ultima fase di consultazione sulla pianificazione eseguita.

Figura 3.3: Workflow Pianificazione delle Lezioni

3.2.3 Gestione Eventi

Oltre alla classica pianificazione dell'offerta formativa, vista nelle sottosezioni precedenti, UP2.0 offre la possibilità di pianificare e gestire eventi che non siano le lezioni didattiche. In questa casistica entrano in gioco due soggetti di fondamentale importanza per il corretto svolgimento di questa pratica:

- prenotatore: ovvero colui che si occupa di inserire l'evento e richiedere la disponibilità dell'aula in uno slot;
- pianificatore: ovvero colui che conferma o rifiuta la prenotazione dopo un'attenta fase di verifica dell'elenco delle prenotazioni già effettuate.

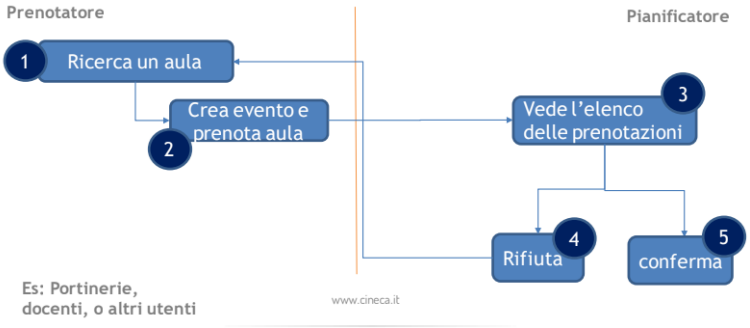

Figura 3.4: Gestione Eventi

3.2.4 Pianificazione Esami

Un'altra funzione di fondamentale importanza di UP2.0 è quella di pianificare gli esami. In questa situazione possono verificarsi due possibili scenari:

- gli esami vengono pianificati nella segreteria studenti e importati in UP per l'assegnazione delle aule:
  - viene generato un calendario di massima che può variare nelle date direttamente su UP;
  - viene fatto un calendario definitivo e su UP viene assegnata un'aula a ciascun esame;
- gli esami vengono pianificati e inserite le aule direttamente dalla segreteria studenti:
  - in questo scenario su UP vengono direttamente o confermate o modificate le aule.

Al termine di queste due possibili situazioni verificabili, vengono pubblicati gli appelli su UP e restituiti alla segreteria studenti dell'ateneo di riferimento.

Figura 3.5: Workflow Pianificazione Esami

Demo del Prodotto

Come potete vedere nella figura 3.6, una volta definita l'offerta formativa, per procedere con la pianificazione dell'orario della settimana desiderata sarà sufficiente trascinare l'insegnamento nello slot orario che si preferisce, assegnandogli l'aula designata.

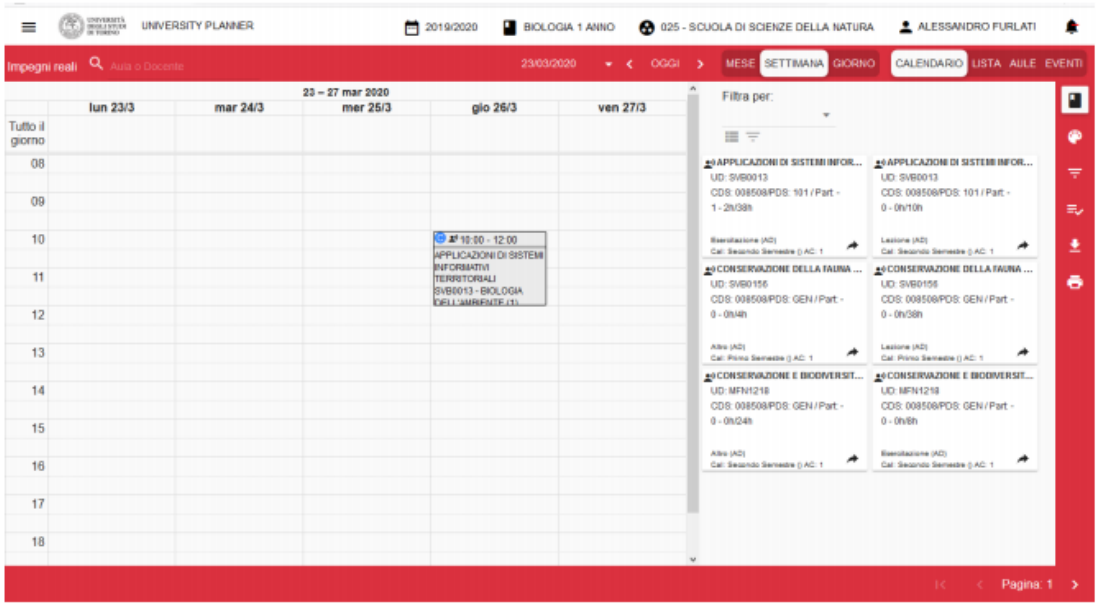

Figura 3.6: Demo UP2.0

3.3 Limiti di UP2.0

UP2.0 è una piattaforma ad alte potenzialità che offre agli utilizzatori numerose funzionalità, come descritto nella sezione 3.2. Tuttavia, nonostante i servizi offerti, questo strumento ha dei limiti. Il limite principale di questo prodotto è l'impossibilità di automatizzare la pianificazione dell'orario in base ai vincoli dati in input, ma l'intera pianificazione deve essere effettuata manualmente. Per questo motivo, nei capitoli successivi sarà analizzato nel dettaglio la soluzione UniTime, che permette di automatizzare questo procedimento offrendo una soluzione pronta all'uso o da impiegare come proposta iniziale su cui apportare delle modifiche manuali per soddisfare gli utenti.

Capitolo 4

UniTime

4.1 UniTime

UniTime è un sistema completo di pianificazione della didattica che supporta la creazione e lo sviluppo degli orari dei corsi e degli esami di un determinato ateneo. Questo strumento offre la possibilità di gestire e modificare gli orari pianificati, condividere le aule e gli edifici con eventi esterni alla didattica e pianificare gli insegnamenti di ciascuno studente nelle singole classi in base al curriculum scelto [15].

Si tratta di un sistema distribuito che consente ai responsabili di ciascuna università o dipartimento, che si occupano della pianificazione dell'offerta didattica, di coordinare e ridurre al minimo gli sforzi per costruire e modificare un orario che soddisfi le loro esigenze organizzative, permettendo allo stesso tempo di ridurre al minimo i conflitti che si possono verificare all'interno del curriculum di ciascuno studente. UniTime può essere utilizzato singolarmente per creare e gestire il programma delle lezioni e degli esami di una determinata università, oppure può essere interfacciato con un sistema informativo già esistente (ad esempio University Planner).

In origine, UniTime è stato sviluppato come strumento di ausilio ai docenti, agli studenti e al personale delle università del Nord America e dell'Europa con lo scopo di diminuire lo sforzo necessario per la pianificazione delle attività. Questo software è distribuito gratuitamente con una licenza open source, con la speranza che altri college e altre università possano avvantaggiare i loro studenti offrendo una miglior organizzazione delle lezioni oppure abbiano la volontà di contribuire alla ricerca e allo sviluppo di questo

progetto. Da Marzo 2015, UniTime è diventato un progetto sponsorizzato dalla Fondazione Apereo, una fondazione che si occupa di supportare i software dedicati all'istruzione e alla didattica [16].

Questo progetto è stato presentato alla conferenza *Open Apereo 2020 Online conference*, con il titolo di *UniTime: Comprehensive University Scheduling System - Using UniTime to improve institutional efficiency.*

Le funzionalità principali di UniTime si possono suddividere in quattro macro-sezioni:

- pianificazione dell'orario e gestione dei corsi;
- pianificazione degli esami;
- gestione degli eventi;
- pianificazione degli studenti.

4.1.1 Pianificazione dell'Orario e Gestione dei Corsi

Pianificazione dell'Orario

L'obiettivo principale della pianificazione dell'orario didattico è quello di collocare ciascun insegnamento in uno slot orario (o più slot in base alle esigenze) che non sia in conflitto con lo slot assegnato a qualsiasi altro insegnamento richiesto dagli studenti che lo frequentano. Questa procedura risulta relativamente facile se il numero di insegnamenti del corso di interesse è basso, ad esempio cinque o sei insegnamenti. Diventa notevolmente complicato con l'aumentare delle combinazioni di corsi che richiedono diversi slot/periodi di collocamento. In questa situazione, la disponibilità dei docenti, delle aule e la varietà dei vincoli di ciascuna casistica complicano ulteriormente il problema della pianificazione.

Le richieste degli studenti relative a ciascun insegnamento, la creazione di modelli storici, il mantenimento di una stabilità nei programmi di studio e la combinazione dei dati storici possono aiutare a minimizzare i conflitti.

I conflitti degli studenti sono calcolati tenendo in considerazione gli orari dei docenti, le preferenze delle aule ed altri vincoli, come ad esempio la scelta delle classi.

UniTime sfrutta i più recenti algoritmi di soluzione progettati ad hoc per la risoluzione di problemi complessi di predisposizione dell'orario. Inoltre,

fornisce un modello della struttura del corso completo in modo da semplificare la definizione delle relazioni tra i vari componenti dei corsi (edifici, aule, classi, insegnamenti, istruttori, ecc.) e per essere compatibile con le differenti tipologie di insegnamento (ad esempio lezione, laboratorio, seminario, conferenza, ecc.) e le diverse durate degli incontri. UniTime fornisce approcci centralizzati, distribuiti e ibridi per costruire il proprio programma di lezioni in base alle esigenze di ciascun ateneo.

Questo modulo, ovvero quello relativo all'organizzazione dei corsi, può anche essere utilizzato per testare scenari differenti rispetto alla soluzione trovata, come la disposizione di un minor numero di classi o l'apporto di modifiche dei requisiti del corso.

Gestione dei Corsi

Anche la miglior pianificazione può richiedere modifiche a seguito della mutazione delle esigenze o delle disponibilità di risorse. Un aumento della domanda per un corso, la partenza di un docente o la perdita di un'aula richiederà di apportare un cambiamento nella soluzione trovata. Per questo, UniTime offre agli utenti la possibilità di cercare facilmente alternative che abbiano un impatto minimo sull'orario complessivo già pianificato, di apportare modifiche all'orario delle lezioni e di comunicare le modifiche effettuate agli studenti interessati.

I cambiamenti di classe, dopo la pubblicazione dell'orario, fanno uso di un risolutore interattivo che suggerisce un insieme limitato di soluzioni alternative di tempo e/o di aula che consentiranno di soddisfare le esigenze richieste dalla modifica.

Inizialmente il risolutore cerca possibili assegnamenti differenti che si adattino al meglio a tutti i requisiti e le preferenze impostate sulle classi, senza influire sull'assegnazione di più di due classi aggiuntive.

L'utente può quindi selezionare l'insieme di modifiche che meglio soddisfa i suoi bisogni tra le opzioni presentate. Ogni opzione fornisce informazioni su eventuali conflitti tra studenti nella soluzione alternativa e su qualsiasi altra preferenza e/o vincolo che potrebbe essere stato violato.

Nel caso in cui non sia possibile trovare una soluzione che interessi solamente due classi, la ricerca potrà essere ulteriormente regolata per consentire ulteriori modifiche. Queste potranno essere limitate per consentire solamente scambi di aule o scambi di tempo e quindi di slot orari.

4.1.2 Pianificazione degli Esami

UniTime crea una pianificazione degli esami per ciascun trimestre che riduce al minimo il numero di posizionamenti degli esami in conflitto tra i vari studenti.

Inoltre, offre la possibilità di ridurre al minimo il numero di esami consecutivi o di studenti con più esami nello stesso giorno. Questa funzionalità risulta particolarmente utile per i college e per le università che aggiornano frequentemente le classi o hanno un gran numero di corsi suddivisi in sezioni che non si adattano alla pianificazione degli esami trovata.

UniTime può essere utilizzato per regolare sia gli esami intermedi che gli esami finali.

4.1.3 Gestione degli Eventi

La maggior parte dei college e delle università utilizza le proprie strutture per la programmazione di più lezioni. Relatori ospiti, riunioni, sessioni di studio e altre attività devono essere programmate in diversi spazi del campus. UniTime crea automaticamente eventi per tutte le lezioni di ciascuna classe e per gli esami nel calendario degli eventi per la sessione accademica di riferimento.

È possibile aggiungere eventi aggiuntivi alle stanze utilizzate dalle classi o qualsiasi altra struttura dell'ateneo presente all'interno del sistema UniTime, utilizzando l'interfaccia web *Eventi*. Questa interfaccia può essere messa a disposizione di tutto il personale e degli studenti per la richiesta di eventi e/o la ricerca di orari e luoghi disponibili. Inoltre, è possibile assegnare diversi gestori di eventi per controllare l'assegnazione di diversi gruppi di spazi. Gli avvisi relativi ai cambiamenti di stato vengono inviati automaticamente agli organizzatori dell'evento e sono subito disponibili nella pagina web *I miei eventi*.

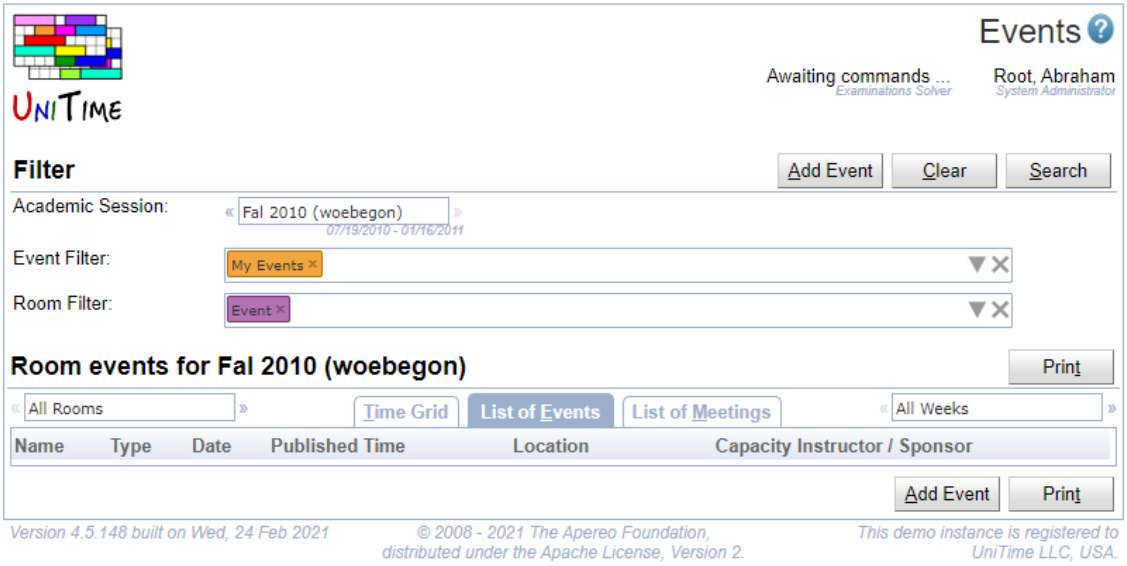

Figura 4.1: Pagina i Miei Eventi

4.1.4 Pianificazione degli Studenti

Nella maggior parte degli atenei, il processo di pianificazione e programmazione dei singoli studenti viene spesso sottovalutato. La priorità viene data alle lezioni che vengono esposte nelle bacheche all'interno dell'università e gli studenti fanno la fila per potersi iscrivere ad un corso o ad una particolare classe caratterizzata dall'orario che preferiscono. Sebbene questo sia un processo facile e ben compreso, spesso è difficile per alcuni studenti trovare combinazioni praticabili necessarie per diplomarsi o fare progressi sul loro curriculum.

Il processo di organizzazione degli studenti è essenzialmente quello di abbinare gli insiemi di classi richieste da ogni studente agli spazi di classe disponibili in modo che tutti (o il maggior numero possibile) i requisiti educativi siano soddisfatti. Potrebbero esserci anche preferenze di tempo di una determinata classe o del singolo studente che complicano il problema. L'approccio più diretto per assicurarsi che ogni studente sia in grado di frequentare tutti i corsi necessari è costruire l'orario delle lezioni dopo aver raccolto tutti i requisiti. UniTime può essere utilizzato per costruire un orario basato sulla domanda e ottimizzare il numero di studenti che

partecipano senza conflitti ai corsi richiesti. Tuttavia, spesso questo non è pratico; quindi, è auspicabile soddisfare il maggior numero possibile di esigenze degli studenti con un orario esistente.

Quando si adattano i singoli studenti ad un orario preesistente in tempo reale, la difficoltà principale è garantire che le scelte effettuate per i primi membri della classe, non precludano ai nuovi partecipanti di frequentare tutti i corsi richiesti. Questo è un problema solo quando vengono offerte più sezioni di corsi in momenti diversi. Se tutti i corsi vengono presentati in una sola volta, gli studenti non sono più in grado di frequentare i corsi prescelti, perciò è necessario un orario migliore.

Utilizzando le conoscenze basate sui requisiti del corso curriculare o sulle richieste storiche dei corsi o sull'orario esistente, UniTime è in grado di determinare le necessità per le singole sezioni del corso. Se il numero di studenti che richiedono di partecipare a quella determinata classe, è maggiore al numero di posti disponibili per la classe di riferimento, questi posti saranno riservati agli studenti la cui scelta permette una pianificazione dell'orario priva di conflitti. Ciò comporta la limitazione delle scelte di orario e di classe di alcuni studenti, ma solo quanto è necessario per consentire a questi alunni di potersi iscrivere a tutte le classi richieste. Poiché si tratta di un processo stocastico, non è possibile garantire che tutti i conflitti siano risolti, ma i test basati sulle richieste effettive degli studenti mostrano riduzioni significative delle esigenze del corso non soddisfatte.

4.2 Struttura Applicazione

UniTime è un'applicazione web sviluppata in Java, JEE e js hostata su Apache Tomcat, un server web (nella forma di contenitore servlet) open source sviluppato dalla Apache Software Foundation. Tomcat implementa le specifiche JavaServer Pages (JSP) e servlet, fornendo quindi una piattaforma software per l'esecuzione di applicazioni web sviluppate in linguaggio Java [17]. È possibile installare UniTime su qualsiasi macchina munita di Apache Tomcat.

4.3 Struttura Base di Dati

Il database utilizzato da UniTime è MySQL, un relational database management system (RDBMS) composto da un client a riga di comando e un server [18]. MySQL è un software libero rilasciato a doppia licenza, compresa la GNU General Public License, sviluppato per essere il più possibile conforme agli standard ANSI SQL e ODBC SQL e che supporta numerosi linguaggi di programmazione quali ODBC, Java, Mono, .NET, PHP, Python e molti altri.

Il database di UniTime, che prende il nome di *timetable*, è composto da 219 tabelle.

Vista la grande dimensione di questa base di dati e l'impossibilità di analizzarne nel dettaglio ogni sua tabella, nelle sezioni successive verranno analizzate solamente quelle ritenute più importanti per comprendere al meglio lo strumento UniTime.

Tabelle ed Entità Principali

Di seguito è riportata una tabella contenente le principali entità del modello dati di UniTime e le rispettive tabelle.

Entità	**Nome Tabella**	**Contenuto**
Area Accademica	academic_area	Tabella che contiene le informazioni relative alle aree accademiche presenti nel sistema
Aule	room	Tabella che contiene l'elenco di tutte le aule
Classi	class_	Tabella che contiene al suo interno le classi
Classificazione Accademica	academic_classification	Tabella che contiene al suo interno le classificazioni accademiche
Curriculum	curriculum	Tabella che contiene l'elenco di tutti i curriculum
Dipartimenti	department	Tabella che contiene le informazioni relative a tutti i dipartimenti
Edifici	building	Tabella che contiene l'elenco di tutti gli edifici
Insegnamenti	instructional_offering	Tabella che contiene gli insegnamenti creati (instructional offering)
Sessione Accademica	sessions	Tabella che contiene l'elenco di tutte le sessioni accademiche
Staff	staff	Tabella che contiene l'elenco di tutto il personale (insegnanti ecc.)
Studenti	student	Tabella che contiene l'elenco di tutti gli studenti
Time Pattern	time_patter	Tabella che contiene al suo interno tutti i time pattern

La tabella contiene solamente la tabella principale di ciascuna entità. Ciascuna di essa verrà analizzata nei capitoli successivi.

4.4 Libreria Algoritmi di Ottimizzazione

La libreria del risolutore di problemi con vincoli di UniTime, che prende il nome di *Constraint Solver Library*, contiene un framework basato sulla ricerca locale che consente la modellazione di un problema utilizzando i principi della programmazione con vincoli, ovvero variabili - valori - vincoli.

La ricerca della soluzione si basa su un algoritmo di ricerca iterativo. Questo algoritmo è simile ai metodi di ricerca locale; tuttavia, a differenza delle classiche tecniche di ricerca locale, opera su soluzioni fattibili, anche se non necessariamente complete. In queste soluzioni alcune variabili possono essere lasciate non assegnate, ma tutti i vincoli rigidi sulle variabili assegnate devono essere soddisfatti. Tali soluzioni sono più facili da visualizzare e più significative per gli utenti rispetto a soluzioni complete ma irrealizzabili. A causa del carattere iterativo dell'algoritmo, il risolutore può anche essere avviato, arrestato o continuare facilmente da qualsiasi soluzione possibile, completa o incompleta.

Il framework supporta anche gli aspetti dinamici del problema della perturbazione minima, consentendo di mantenere il numero di modifiche alla soluzione (perturbazioni) il più piccolo possibile.

Nel capitolo successivo verrà analizzato nel dettaglio l'algoritmo utilizzato e la libreria che ne sfrutta le capacità.

Capitolo 5

Algoritmo UniTime

In questo capitolo verrà analizzato l'algoritmo utilizzato dalla piattaforma UniTime e la libreria in cui esso è contenuto, ovvero la sopracitata *Constraint Solver Library*.

5.1 Constraint Solver Library

La libreria utilizzata da UniTime, chiamata appunto Constraint Solver Library, ovvero Libreria di Risoluzione di Vincoli e sviluppata interamente in Java, è composta da 4 moduli [19]:

- ifs.jar: acronimo di iterative forward search. Questo eseguibile si occupa di risolvere e ottimizzare problemi semplici di ricerca locale con vincoli;
- coursett.jar: eseguibile che si occupa di risolvere i problemi della pianificazione dell'orario;
- studentsct.jar: eseguibile che si occupa della pianificazione nelle classi degli studenti;
- examtt.jar: eseguibile per la risoluzione dei problemi inerenti alla pianificazione degli esami.

I moduli sopra descritti sono inclusi all'interno dell'eseguibile principale chiamato *CPSolver.jar*. Ciascun modulo ha funzionalità differenti, ma la logica di risoluzione dei problemi di ognuno di essi è la medesima, ovvero

l'utilizzo di un algoritmo di ricerca locale di risoluzione dei vincoli, composto da tre componenti principali: le variabili (ad esempio aule, istruttori e classi), i valori (ovvero le possibili soluzioni che possono essere selezionate) e i vincoli (ad esempio le preferenze) da rispettare. Come accennato nel capitolo 2, il nostro caso di studio si concentrerà sulla risoluzione dei problemi di pianificazione dell'orario e quindi dell'utilizzo dell'eseguibile *coursett*. La versione utilizzata è la 1.3, rilasciata in data 28 Gennaio 2021. Nelle sezioni successive verrà analizzato l'algoritmo di cui l'eseguibile si avvale per la ricerca delle soluzioni.

5.2 Studio dell'Algoritmo

In questa sezione verrà analizzato l'algoritmo utilizzato da UniTime per la pianificazione dell'orario, proposto e sviluppato nella tesi di ricerca di Tomáš Müller con titolo *Constraint-based Timetabling* [20] pubblicata nel 2005; per questo motivo in questa sezione verranno fatti riferimenti alla tesi di ricerca del Dott. Müller.

L'algoritmo utilizzato è di tipologia Iterative Forward Search (IFS) e si basa su tecniche di ricerca locale. La ricerca locale è un algoritmo euristico di ricerca in grado di risolvere un problema elaborando solamente un sottoinsieme dello spazio di ricerca. Questa tipologia di algoritmo si differenzia dagli algoritmi di ricerca tradizionali perché per individuare una possibile soluzione non analizza tutti i cammini multipli, ma solamente quelli locali a partire da un nodo definito nodo corrente. Vista la bassa complessità spaziale, l'algoritmo è in grado di utilizzare una quantità di memoria limitata. L'utilizzo di questi algoritmi avviene principalmente negli spazi di ricerca molto vasti oppure negli spazi a rete [21]. Tuttavia, a differenza dei classici algoritmi di local-search, l'algoritmo progettato da Müller funziona sulla ricerca di una soluzione fattibile, anche se non necessariamente completa e di conseguenza lasciando alcune variabili senza assegnamento. Nonostante ciò tutti i vincoli sulle variabili devono essere soddisfatti. Lavorare su soluzioni incomplete ha diversi vantaggi rispetto a lavorare su soluzioni complete ma non fattibili, cosa che di solito accade quando si utilizzano algoritmi di ricerca locale. Un altro vantaggio dovuto dalla scelta di utilizzare un algoritmo iterativo di questa tipologia con le caratteristiche sopra descritte è la possibilità di avviare, interrompere o riavviare la

ricerca di una soluzione da parte del solver, ripartendo all'eventualità dalla soluzione, completa o incompleta, precedentemente trovata. L'IFS di UniTime è scritto interamente in Java. Il codice aggiornato è accessibile all'url `https://github.com/UniTime/cpsolver` e descritto dettagliatamente nell'appendice A e B della tesi di ricerca citata all'inizio della sessione. Oltre alla ricerca di una soluzione, l'algoritmo ha come scopi principali quelli di ottimizzare la soluzione stessa e di soddisfare tutti i vincoli richiesti.

Nella sottosezione successiva verrà analizzato brevemente lo studio che ha portato alla scelta di un algoritmo IFS, descrivendo le principali tipologie di algoritmo alternative.

5.2.1 Ricerche Correlate

Come precedentemente accennato, gli algoritmi di ricerca locale, come ad esempio il min-conflict o il tabu search, eseguono un'esplorazione incompleta dello spazio di ricerca, proponendo tal volta soluzioni complete ma non realizzabili. Le soluzioni proposte da questa tipologia di algoritmi non garantiscono il rispetto di tutti i vincoli ma, a differenza dei tradizionali algoritmi di ricerca sistematici, non soffrono del problema dell'errore precoce vista la possibilità di scartare una soluzione non appena si ritiene che essa porti ad un vicolo cieco. Inoltre, sono in grado di garantire performance nettamente migliori soprattutto nei problemi di ottimizzazione. Nelle successive sezioni verranno descritte brevemente alcune tipologie di algoritmi di ricerca locale e approcci ibridi.

Approcci di Ricerca Locale

I termini local search e neighbour search esprimono l'idea di come questi algoritmi lavorino sulla modifica di una soluzione per ottenere una migliore assegnazione di ciascuna variabile ad ogni iterazione. Esistono due schemi base per gli algoritmi di ricerca locale e sono hill-climbing e min-conflict. Entrambi iniziano la loro computazione da un nodo scelto in maniera casuale o euristica e procedono con l'assegnamento di un valore a ciascuna variabile fino al raggiungimento di una soluzione fattibile o del timeout di esecuzione. Tuttavia, queste due tipologie si differenziano da come i vicini di ciascun nodo vengono selezionati. Nei successivi paragrafi verranno analizzate entrambe le tipologie.

Hill-climbing Algorithm L'algoritmo hill-climbing seleziona sempre l'assegnazione migliore (valore più alto) tra tutti i nodi vicini, riducendo al minimo il numero di vincoli non rispettati. Nel caso di problemi di ottimizzazione, l'algoritmo selezionerà tra gli assegnamenti con il minor numero di vincoli hard violati, i vicini che minimizzano la funzione obiettivo e che quindi riducono il numero di vincoli soft violati. Nel caso in cui non ci dovesse essere un assegnamento migliore rispetto a quello selezionato, la ricerca si ferma in una soluzione ottima locale. Nell'iterazione successiva, l'algoritmo ricomincerà la ricerca a partire da un altro assegnamento scelto in maniera casuale. Il termine hill-climbing indica la capacità di questo algoritmo di scalare i nodi verso quelli con valori maggiori.

Min-conflict Algorithm L'algoritmo min-conflict seleziona l'assegnamento migliore solamente all'interno di un set ridotto di nodi adiacenti. Solitamente la scelta dell'assegnamento viene effettuata scegliendo un valore che sia in grado di ridurre al minimo il numero di conflitti violati. In alternativa, l'assegnamento viene effettuato in maniera casuale scegliendo tra i valori che non variano il numero di vincoli violati. Nel caso in cui non ci siano valori che soddisfino questi requisiti, il valore della variabile rimane inalterato. Tuttavia, è difficile per questa tipologia di algoritmi trovare una soluzione che abbia minimizzato il numero di conflitti a livello locale e quindi spesso la configurazione trovata risulta una soluzione limitata localmente.

Min-conflict Random Walk Algorithm Come precedentemente accennato, l'utilizzo puro di un algoritmo min-conflict comporta spesso l'impossibilità di trovare una soluzione ottimale a livello locale, per questo sono state apportate modifiche per far sì che l'algoritmo sia più efficiente. La strategia più utilizzata è quella chiamata random-walk, strategia in cui l'algoritmo sceglie casualmente un valore con probabilità p e applica l'euristica min-conflict con probabilità *1 - p*. Spesso questa strategia è utilizzata anche negli algoritmi hill-climbing, sempre con probabilità p ma con assegnamento casuale dei nodi vicini.

Tabu Search Algorithm L'algoritmo tabu-search è un altro metodo per evitare numerosi cicli di iterazione ed evitare di rimanere intrappolati in soluzioni minime (minimi conflitti) locali. L'algoritmo si basa sulla nozione di tabu list, ovvero un file di memoria a breve termine contenente le coppie

variabile - valore e una cronologia selettiva delle configurazioni trovate nelle iterazioni precedenti. Un esempio di strategia di tabu-search è prevenire che configurazioni precedentemente trovate e presenti nella tabu list siano riproposte nelle prossime k iterazioni. La variabile k, chiamata tabu tenue, rappresenta la grandezza della tabu list. Adottando questa strategia non solo si impedisce che l'algoritmo rimanga intrappolato in cicli a breve termine, ma ci si assicura anche che la soluzione non venga trovata solamente a livello locale ma in tutto lo spazio di ricerca del problema.

Le restrizioni di questo algoritmo possono essere sovrascritte rispettando determinate condizioni, chiamate *aspiration criteria*; queste condizioni definiscono le regole che determinano se la successiva configurazione sarà da considerare come possibile seppure presente all'interno della tabu list. Un esempio di *aspiration criteria* consiste nel rimuovere una configurazione all'interno della tabu list se questa configurazione porterà poi ad ottenere una soluzione migliore rispetto a quelle ottenute finora.

Approcci Ibridi

Spesso negli anni è stata adottata l'idea di combinare tra loro i sistemi di ricerca sistematica con i sistemi di ricerca locale, dando vita ad algoritmi ibridi. Quest'ultimi sono in grado di offrire ottimi risultati soprattutto su problemi di grandi dimensioni e possono essere suddivisi in tre categorie principali:

- effettuare una ricerca locale prima o dopo una ricerca sistematica;
- inserire un algoritmo di ricerca locale in un punto preciso della ricerca sistematica per migliorarne l'efficienza;
- effettuare una ricerca locale su tutto lo spazio del problema ed integrarla con una ricerca sistematica per la ricerca dei vicini oppure per eliminare una determinata sezione dello spazio della ricerca.

Di seguito verranno descritti alcuni degli approcci ibridi più utilizzati.

Decision Repair Algorithm Uno degli algoritmi ibridi più utilizzati è il decision repair, che rientra nella terza categoria di algoritmi ibridi definite nell'elenco del paragrafo precedente. L'algoritmo in questione ha come funzione principale quella di estendere ripetutamente un set di assegnamenti,

chiamate decisioni, che soddisfino tutti i requisiti come avviene negli algoritmi di ricerca sistematici. Successivamente, l'algoritmo effettua una ricerca locale per cercare di riparare o migliorare gli assegnamenti precedentemente trovati fino al raggiungimento di un cosiddetto punto morto. Al termine di questa operazione in cui tutte le decisioni sono state riparate, l'algoritmo riprende l'esecuzione fino al raggiungimento del punto morto successivo.

Constrained Local Search Algorithm Un altro approccio molto utilizzato è il constrained local search. Questa tipologia di algoritmo si basa sulla randomizzazione delle soluzioni di un algoritmo di ricerca sistematica trovate tramite backtracking. Il backtracking è una tecnica per trovare soluzioni a problemi in cui devono essere soddisfatti dei vincoli, enumerando tutte le possibili soluzioni e scartando quelle che non soddisfano i vincoli [22]. La randomizzazione fa in modo che la scelta delle variabili su cui fare backtracking sia totalmente casuale.

L'algoritmo ha al suo interno un parametro di tipo intero chiamato noise level che indica di quante variabili il backtracking tornerà indietro. Il constrained local search parte da una soluzione fattibile parziale e la estende ad ogni iterazione fino al raggiungimento di una soluzione completa o al raggiungimento di un punto morto in cui l'algoritmo non può più procedere. Quest'ultima situazione, ovvero il punto morto, si verifica nel caso in cui ci sia qualche variabile del problema che abbia un valore non consistente, ovvero che non rispetta i vincoli. In questo caso l'algoritmo procede con la rimozione dei valori di un determinato numero di variabili in base al valore del noise level e ricomincia la sua esecuzione dalla nuova soluzione parziale ottenuta.

Constructive Backtracking-free Algorithm Il constructive backtracking-free è un approccio similare all'algoritmo descritto nel paragrafo precedente. Questa tipologia di algoritmo estende iterativamente una soluzione parziale ammissibile fino al raggiungimento di un vicolo cieco. Al termine di questa operazione, effettua una fase di ricerca locale in cui apporta modifiche agli assegnamenti effettuati sull'attuale soluzione parziale e ricomincia con la computazione fino al raggiungimento del punto morto successivo. L'algoritmo può terminare con due esiti diversi, positivo oppure negativo. L'esito positivo dell'esecuzione porta ad avere una soluzione ammissibile completa, mentre l'esito negativo si avrà nel caso in cui sia stato raggiunto

un numero predeterminato di fasi di ricerca locale senza aver ottenuto una soluzione completa ammissibile. Le differenze rispetto agli algoritmi visti in precedenza sono essenzialmente due. La prima sta nel fatto che l'algoritmo effettua una fase di ricerca locale a partire da una soluzione parziale su tutto lo spazio di ricerca e non limitandosi ad effettuare un numero prefissato di modifiche agli assegnamenti. La seconda, invece, sta nel fatto che la ricerca locale ha come obiettivo principale quello di riuscire a completare gli assegnamenti parziali. Tuttavia, la fase di local search non è guidata solo dalla fattibilità e ottimalità dell'attuale soluzione, ma anche dal fattore look-ahead ovvero prevedere gli effetti della scelta di un determinato assegnamento ad una determinata variabile da effettuare nelle successive iterazioni.

A differenza degli approcci sopra descritti, l'algoritmo progettato da Müller ha un funzionamento similare ad un algoritmo di ricerca locale. Infatti, esso non effettua un ulteriore fase di ricerca locale al raggiungimento di un punto morto, ma applica gli stessi identici passaggi locali effettuati precedentemente durante la ricerca. Ad ogni iterazione, la soluzione parziale ottenuta può essere estesa dall'assegnazione arbitraria di un valore ad una variabile e la coerenza della soluzione è determinata dalla possibilità di eliminare variabili precedentemente assegnate per rendere coerente la nuova soluzione parziale. Questa peculiarità dell'algoritmo di UniTime permette che esso sia facile da implementare o da integrare/estendere con ulteriori algoritmi euristici o con altre tecniche come ad esempio algoritmi di filtraggio. Grazie a questa scelta implementativa, l'algoritmo implementato da Müller si è rivelato flessibile e capace di adattarsi a problematiche completamente differenti tra loro, riuscendo sempre ad ottenere ottime performance e ottimi risultati.

Nella successiva sezione verrà riportato ed analizzato lo pseudocodice dell'algoritmo di riferimento.

5.3 Iterative Forward Search Algorithm

Come precedentemente accennato, l'algoritmo utilizzato dalla libreria di UniTime si basa su un algoritmo IFS (iterative forward search) e a differenza dei classici algoritmi di ricerca locale, si occupa di trovare una soluzione, anche parziale, ma fattibile che rispetti tutti vincoli del problema. Questo comporta la possibilità di avere una soluzione con variabili non assegnate ma di possibile realizzazione e di facile comprensione da parte degli utenti finali che utilizzeranno il solver.

Di seguito è riportato lo pseudocodice dell'algoritmo di UniTime. Tuttavia, il codice mostrato non è quello presente nella tesi di ricerca di Müller ma preso dalla pubblicazione, sempre dello stesso, del 2016 intitolata *University Course Timetabling - Solver Evolution* [23] che ne contiene una versione aggiornata.

```
1  function IFS(ω)
2      σ = ω
3      while canContinue(ω) do
4          ω = selectVariable(ω)
5          d = selectValue(ω, υ)
6          υ = hardConflicts(ω, υ/d)
7          ω = ω\υ ∪ {υ/d}
8          if better(ω, σ) then σ = ω
9      end while
10     return σ
11 end function
```

Listing 5.1: Pseudocodice dell'Algoritmo di Iterative Forward Search

Nel caso di studio di interesse, ovvero la pianificazione dell'orario, le variabili da assegnare sono le classi per questo nella spiegazione dell'algoritmo i termini classe e variabile potranno essere intercambiati l'un l'altro.

Come si può notare nello pseudocodice sopra riportato, la ricerca è effettuata iterativamente a partire da una timetable iniziale, definita con ω che può essere anche vuota. Durante ogni iterazione, si procede come prima cosa con la selezione di una classe, una variabile, riga 4, solitamente non assegnata o già assegnata nel caso in cui tutte le variabili siano già state assegnate, ma la pianificazione ottenuta non è abbastanza ottima. Un esempio può essere la situazione in cui ci siano dei vincoli soft non sod-

disfatti. Successivamente, una volta selezionata la classe da assegnare si procede con l'assegnamento dei valori alla classe; rispettivamente vengono selezionati lo slot orario e il numero di aule desiderato, come è possibile vedere a riga 5. Alle volte, seppur venga selezionato il valore migliore per una determinata variabile, questo assegnamento può causare conflitti con classi già assegnate. Per questo motivo, i conflitti tra le varie classi vengono computati, riga 6, e vengono rimosse dalla pianificazione le classi in conflitto diventando variabili non assegnate. Infine, viene assegnato alla classe il posizionamento precedentemente selezionato. Successivamente, alla riga 7, l'algoritmo tenta di passare da una timetable parziale ma realizzabile, ad un'altra pianificazione tramite l'assegnazione del valore selezionato ad una determinata classe. Infine, l'algoritmo verifica se la soluzione trovata in quella iterazione sia migliore della soluzione precedente, riga 8, la memorizza nella variabile σ restituendola come risultato della funzione a riga 10. L'esecuzione dell'algoritmo termina al raggiungimento di una timetable desiderata in cui tutte le variabili siano assegnate e rispettino tutti i vincoli, oppure al raggiungimento del timeout.

L'algoritmo è composto da cinque elementi principali e sono:

- *termination condition*: ovvero la condizione per la quale l'algoritmo dovrà terminare l'esecuzione. Questa condizione è definita dalla funzione *canContinue*, riga 3;

- *solution comparator*: ovvero la funzione *better*, riga 8, che si occupa di comparare tra loro le due timetable trovate, quella ritenuta migliore e quella generata dall'attuale iterazione. Questa procedura viene effettuata tenendo in considerazione diversi aspetti, come ad esempio il numero di variabili non assegnate;

- *variable selection*: ovvero la scelta della variabile da assegnare, riga 4 funzione *selectVariable*. Come precedentemente accennato, tendenzialmente viene selezionata una variabile non ancora assegnata oppure nel caso in cui non ce ne siano, verrà selezionata una variabile già assegnata se si dovesse ritenere che la soluzione attuale non sia ottima;

- *value selection*: ovvero la scelta del valore/i da assegnare alla variabile precedentemente scelta, riga 5 funzione *selectValue*;

- *conflicting assignments*: ovvero la funzione che si occupa di calcolare i conflitti tra gli assegnamenti effettuati, funzione *hardConflicts* riga 6.

L'algoritmo sfrutta una tecnica di apprendimento chiamata conflict-based statistics (CBS), tecnica sviluppata da Tomáš Müller, Roman Barták e Hana Rudová e consultabile nella pubblicazione con titolo *Conflict-based Statistics* [24]. Questa tecnica ha come obiettivi principali quelli di evitare cycling e migliorare la qualità della soluzione finale. Brevemente, il funzionamento di questa tecnica si basa sull'utilizzo di una struttura dati che memorizza al suo interno tutti i conflitti hard riscontrati durante la fase di ricerca, memorizzando in oltre la loro frequenza e gli assegnamenti che hanno portato a generare questi conflitti. Durante la scelta del valore da assegnare alla variabile, ogni conflitto avrà un peso differente in base alla frequenza in cui esso si è ripetuto. Spesso un'inconsistenza dei dati in input porta all'impossibilità di trovare una soluzione completa. Il contenuto della struttura dati sopracitata è di notevole aiuto all'utente finale per capire quali vincoli siano da rivedere e da alleggerire per poter ottenere una soluzione realizzabile in cui tutte le variabili siano assegnate.

Un'importante considerazione da effettuare sul solver è che mentre l'algoritmo si basa su tecniche di local search, l'intera libreria del solver è stata implementata con orientamento a variabili (ovvero le classi), valori, assegnamenti e vincoli. Questi ultimi possono essere hard o soft in base alla loro importanza all'interno del problema. Il risolutore non permette in nessun modo e in nessuna situazione che i vincoli di tipologia hard siano violati.

Nei capitoli successivi verrà analizzato e descritto come il solver è stato integrato all'interno del progetto, l'utilizzo che ne è stato fatto e i risultati ottenuti dall'esecuzione delle diverse problematiche.

Capitolo 6

Utilizzo e Integrazione UniTime

In questo capitolo verrà fatta luce sull'utilizzo e l'integrazione dell'applicativo UniTime con i dati esportati dalla piattaforma di Cineca UP2.0.

6.1 Raw Data Cineca

Prima di affrontare nel dettaglio l'integrazione sopraccitata è bene analizzare i dati messi a disposizione da UP2.0 e da Cineca per poter creare un ambiente di test all'interno della piattaforma UniTime. I dati messi a disposizione sono dati esempio esportati dal dipartimento di Lettere e Lingue, corso di studi di Antropologia Culturale ed Etnologia, dell'Università di Bologna che, pur non essendo totalmente completi e integrati, sono stati sufficienti per poter effettuare tutte le prove necessarie sulla pianificazione dell'orario. I dati fondamentali necessari all'algoritmo di UniTime sono: aule, dipartimenti, insegnamenti, docenti e preferenze. Questa ultima voce si riferisce alle preferenze di ciascun insegnamento, quali orario, durata dell'incontro, minuti settimanali e così via dicendo. Queste informazioni non sono state esportate da UniTime ma inserite manualmente attraverso l'ausilio di uno script che analizzeremo nelle sezioni successive.

6.1.1 Aule

Le aule assumono nell'operazione di pianificazione dell'orario un ruolo di fondamentale importanza; per questo motivo è necessario raccogliere più informazioni possibili e dettagliate. I dati con rilevanza maggiore raccolti sono il codice, la descrizione, il tipo di aula, la capienza, il numero di postazioni, il numero di PC, i metri quadri, la via (comprensiva di numero civico), l'edificio di appartenenza, l'unità organizzativa (ovvero il dipartimento di appartenenza di quell'aula) e lo stato dell'aula. Quest'ultimo assume i valori true o false in base allo stato di attività dell'aula di riferimento. L'immagine successiva, figura 6.1, rappresenta un esempio di un file csv esportato dalla piattaforma di UP2.0 e contenente le informazioni sopra descritte sulle aule.

Codice	Descrizione	Piano	Capienza	Tipo aula	Numero p	Numero P	Telefono	Metri quadri	Note	Url mappe	Via	Numero c	Edificio	UnitÃ org	Contesti c	Servizi	Attivo	Disabilitat	Disabilitato il
100_WP01_003	AULA 3	100_WP01 - Piano Primo	26	AULA - AULA				41.57			Piazza S.Giovanni in		100 - San (	742482 - A	BO_LET - FILIERA_LE		true		
100_WP01_004	AULA 4	100_WP01 - Piano Primo	21	AULA - AULA				32.38			Piazza S.Giovanni in		100 - San (	742482 - A	BO_LET - FILIERA_LE		true		
100_WP01_005	AULA 5	100_WP01 - Piano Primo	24	AULA_INFORMATICA - AULA_INFORMATI				45.02			Piazza S.Giovanni in		100 - San (	742482 - A	BO_LET - FILIERA_LE		true		
100_WP01_007	SALA RIUNIONI	100_WP01 - Piano Primo		SALA_RIUNIONI - SALA_RIUNIONI				23.91			Piazza S.Giovanni in		100 - San (	742482 - A	BO_LET - FILIERA_LE		true		
100_WP01_037	AULA GIORGIO MORANDI	100_WP01 - Piano Primo	30	AULA_INFORMATICA - AULA_INFORMATI				65.53			Piazza S.Giovanni in		100 - San (	795470 - D	BO_LET - F	Accesso d	true		
100_WP01_038	AULA VITO FUMAGALLI	100_WP01 - Piano Primo	40	AULA - AULA				60.980.000.000.000			Piazza S.Giovanni in		100 - San (	795470 - D	BO_LET - FILIERA_LE		true		
100_WP01_083	AULA LUCIO GAMBI	100_WP01 - Piano Primo	40	AULA - AULA				52.800.000.000.000			Piazza S.Giovanni in		100 - San (	795470 - D	BO_LET - F	Accesso d	true		
100_WP01_111	SEMINARI 2	100_WP01 - Piano Primo	40	AULA - AULA				58.82			Piazza S.Giovanni in		100 - San (	795470 - D	BO_LET - FILIERA_LE		true		
100_WP02_039	AULA STEFANO TORRESANI	100_WP02 - Piano Secondo	25	AULA - AULA				17.57			Piazza S.Giovanni in		100 - San (	795470 - D	BO_LET - FILIERA_LE		true		
100_WP02_066	AULA SPECOLA	100_WP02 - Piano Secondo	40	AULA - AULA				55.120.000.000.000			Piazza S.Giovanni in		100 - San (	795470 - D	BO_LET - FILIERA_LE		true		
100_WP02_112	AULA SEMINARI	100_WP02 - Piano Secondo	30	AULA - AULA				31.810.000.000.000			Piazza S.Giovanni in		100 - San (	795470 - D	BO_LET - FILIERA_LE		true		
100_WP03_001	AULA SPECOLA	100_WP03 - Piano Terzo		AULA - AULA				32.43			Piazza S.Giovanni in		100 - San (	795470 - D	BO_LET - FILIERA_LE		true		
100_WPTE_023	LABORATORIO INFORMATICO	100_WPTE - Piano Terra	10								Piazza S.Giovanni in		100 - San (	807473 - C	BO_LET - FILIERA_LE		true		
100_WPTE_078	AULA GIORGIO PRODI	100_WPTE - Piano Terra	130	AULA - AULA				18.039.000.000.000			Piazza S.Giovanni in		100 - San (	795470 - D	BO_LET - FILIERA_LE		true		
100_WPTE_083	AULA OVIDIO CAPITANI	100_WPTE - Piano Terra	55	AULA - AULA				65.65			Piazza S.Giovanni in		100 - San (	795470 - D	BO_LET - FILIERA_LE		true		
100_WPTE_084	AULA GIORGIO GUALANDI	100_WPTE - Piano Terra	30	AULA - AULA				44.99			Piazza S.Giovanni in		100 - San (	795470 - D	BO_LET - FILIERA_LE		true		
100_WS01_001	AULA 1	100_WS01 - Piano Primo Interrato	22	AULA - AULA				38.22			Piazza S.Giovanni in		100 - San (	742482 - A	BO_LET - FILIERA_LE		true		
100_WS01_002	AULA 2	100_WS01 - Piano Primo Interrato	40	AULA - AULA				62.58			Piazza S.Giovanni in		100 - San (	742482 - A	BO_LET - FILIERA_LE		true		
100_WS01_042	LABORATORIO INFORMATICO	100_WS01 - Piano Primo Interrato									Piazza S.Giovanni in		100 - San (	795470 - D	BO_LET - FILIERA_LE		true		
100_WS01_057	AULA GRANDE	100_WS01 - Piano Primo Interrato	100	AULA - AULA				171.19			Piazza S.Giovanni in		100 - San (	795470 - D	BO_LET - FILIERA_LE		true		
101_WA02_034	AULA MONDOLFO	101_WA02 - Piano Secondo Ammezzato	30	AULA - AULA				75.67			Via Zamboni, 38		101 - Pala:	795400 - F	BO_LET - FILIERA_LE		true		
101_WP01_003	AULA II	101_WP01 - Piano Primo	122	AULA - AULA				110.42			Via Zamboni, 38		101 - Pala:	1132183 -	BO_LET - FILIERA_LE		true		
101_WP01_005	AULA III	101_WP01 - Piano Primo	230	AULA - AULA				169.66			Via Zamboni, 38		101 - Pala:	1132183 -	BO_LET - FILIERA_LE		true		
101_WP01_008	AULA I	101_WP01 - Piano Primo	72	AULA - AULA				75.62			Via Zamboni, 38		101 - Pala:	1132183 -	BO_LET - FILIERA_LE		true		
101_WP01_041	LABORATORIO INFORMATICO	101_WP01 - Piano Primo	10	AULA_INFORMATICA - AULA_INFORMATI				19.23			Via Zamboni, 38		101 - Pala:	795400 - F	BO_LET - FILIERA_LE		true		

Figura 6.1: CSV Contenente le Informazioni sulle Aule

6.1.2 Dipartimenti

Un dipartimento universitario è una struttura organizzativa che all'interno delle università italiane promuove e coordina le attività di uno o più settori della ricerca scientifica che siano omogenei per fini e per metodo [25]. L'informazione riguardante i dipartimenti è fondamentale per garantire ai dati una maggiore coerenze e completezza oltre che a fornire il contesto di riferimento di ciascuna risorsa analizzata (aula, insegnamento, corso ecc.). Per quel che riguarda UP2.0 e l'integrazione con UniTime, i dati esportati sono codice, descrizione e ore accademiche definite in minuti. L'immagine

sottostante, figura 6.2, rappresenta un esempio di csv esportato contenente le informazioni sui dipartimenti.

Codice	Descrizione	Ore accademiche (in minuti)
12	DIPARTIMENTO PROVA MANCANTE	
6	LETTERE-LINGUE	

Figura 6.2: CSV Contenente le Informazioni sui Dipartimenti

Oltre ai dipartimenti è necessario mantenere anche le informazioni relative all'associazione tra ciascun corso e i dipartimenti di riferimento, per poter poi ricavare il dipartimento di appartenenza di ciascun insegnamento. In sintesi, ciascun insegnamento è relativo ad un corso di studi che appartiene ad un dipartimento. Come è possibile vedere nell'immagine sottostante, figura 6.3, il csv contiene il codice del corso di studi, la sua descrizione e l'unità organizzativa di appartenenza. Le colonne non citate non sono state ritenute importanti per la pianificazione dell'orario e quindi non utilizzate.

Codice	Descrizione	Descrizione (EN)	Acronimo	Tipo corso	Unita organizzativa	Ante Riforma
964	ANTROPOLOGIA CULTURALE ED ETNOLOGIA			LM - LAUREA MAGISTRALE	6	No
966	CINEMA, TELEVISIONE E PRODUZIONE MULTIMEDIALE			LM - LAUREA MAGISTRALE	6	No
8837	DISCIPLINE DELLA MUSICA E DEL TEATRO			LM - LAUREA MAGISTRALE	6	No
8845	SCIENZE STORICHE E ORIENTALISTICHE			LM - LAUREA MAGISTRALE	6	No
9071	ARTI VISIVE			LM - LAUREA MAGISTRALE	6	No
9220	ITALIANISTICA, CULTURE LETTERARIE EUROPEE, SCIENZE LINGUISTICHE			LM - LAUREA MAGISTRALE	6	No
9228	CORSO PROVA MANCANTE			L - LAUREA	12	No

Figura 6.3: CSV Contenente le Informazioni sull'Associazione tra Corsi e Dipartimenti

6.1.3 Docenti

I docenti rappresentano una delle risorse di maggiore importanza. Nella sezione 2.2.2 sono stati descritti i principali problemi di questa risorsa e il ruolo che essa ha all'interno della pianificazione: per questo è necessaria una corretta, completa e integrata profilazione. Il csv esportato da UP2.0 contiene al suo interno le informazioni riguardanti il cognome, il nome, il nome utente, la matricola, la mail istituzionale, il contesto di appartenenze, due colonne con valori booleani che ne definiscono il ruolo (docente y/n) e lo stato di attività (attivo y/n) ed infine l'unità organizzativa, ovvero il

dipartimento di appartenenza. La figura sottostante riporta un esempio di quanto appena descritto.

Cognome	Nome	Nome utente	Matricola	Mail	Ruoli	Contesti c	Docente	Attivo	Unita Organizzativa
Cognome1	Nome1	nome1.cognome1@unibo.it	1	nome1.cognome1@unibo.it			true	true	6
Cognome2	Nome2	nome2.cognome2@unibo.it	2	nome2.cognome2@unibo.it			true	true	6
Cognome3	Nome3	nome3.cognome3@unibo.it	3	nome3.cognome3@unibo.it			true	true	6
Cognome4	Nome4	nome4.cognome4@unibo.it	4	nome4.cognome4@unibo.it			true	true	6
Cognome5	Nome5	nome5.cognome5@unibo.it	5	nome5.cognome5@unibo.it			true	true	6
Cognome6	Nome6	nome6.cognome6@unibo.it	6	nome6.cognome6@unibo.it			true	true	6
Cognome7	Nome7	nome7.cognome7@unibo.it	7	nome7.cognome7@unibo.it			true	true	6
Cognome8	Nome8	nome8.cognome2@unibo.it	8	nome8.cognome2@unibo.it			true	true	6
Cognome9	Nome9	nome9.cognome9@unibo.it	9	nome9.cognome9@unibo.it			true	true	6
Cognome10	Nome10	nome10.cognome10@unibo.it	10	nome10.cognome10@unibo.it			true	true	6

Figura 6.4: CSV Contenente le Informazioni sui Docenti[1]

6.1.4 Insegnamenti

I dati relativi agli insegnamenti rappresentano il fulcro principale di tutta la pianificazione dell'orario; è perciò necessario che questi siano dettagliati, precisi, integrati ma soprattutto completi. Essi, infatti, contengono tutte le informazioni riguardanti le ore pianificate, il codice dell'insegnamento, l'evento (il nome dell'insegnamento) e la sua tipologia, l'anno di corso (aggiunto successivamente per aver la possibilità di suddividere l'orario pianificato per anno di corso), la tipologia dell'attività (insegnamento, seminario, ecc.), l'unità organizzativa (dipartimento), la data di inizio e di fine, il corso di studi, il docente responsabile e altre informazioni non rilevanti per il nostro caso di studio. Nell'immagine sottostante, figura 6.5, è possibile vedere un csv di esempio sugli insegnamenti.

Impegni	Ore pianificate	Monte ore	Codice	Evento	Tipo even	anno di corso	Tipo attivita	Unita	org.	Percorso	Fattore di	Partizione	Sede	Calendario	data ini	data fine	Corso di s	Responsabili	dipartime
12	36	Monte ore superato	28304	ANTROPOLOGIA APPLICATA (1) (LM)	ATTIVITA'	1	Insegnamento	6		000-000			BO	2019_0964	11/11/2019	20/12/2019	964	Docente1	6
33	66	Monte ore superato	28346	ANTROPOLOGIA DEL CORPO E DELLA MALATTIA (LM)	ATTIVITA'	1	Insegnamento	6		000-000			BO	2019_0964	23/09/2019	20/12/2019	964	Docente2	6
11	33	Monte ore superato	69514	ANTROPOLOGIA DELL'EDUCAZIONE (1) (LM)	ATTIVITA'	1	Insegnamento	6		000-000			BO	2019_0964	23/09/2019	31/10/2019	964	Docente3	6
49	98	Monte ore superato	81691	ANTROPOLOGIA POLITICA (LM)	ATTIVITA'	1	Insegnamento	6		000-000			BO	2019_0964	23/09/2019	20/12/2019	964	Docente4	6
15	30	Monte ore raggiunto	39426	ANTROPOLOGIA STORICA (1) (LM)	ATTIVITA'	1	Insegnamento	6		000-000			BO	2019_0964	11/11/2019	20/12/2019	964	Docente5	6
13	39	Monte ore superato	84268	ANTROPOLOGIA VISIVA (1) (LM)	ATTIVITA'	1	Insegnamento	6		000-000			BO	2019_0964	06/04/2020	22/05/2020	964	Docente6	6
18	36	Monte ore superato	69515	DEMOGRAFIA STORICA (1) (LM)	ATTIVITA'	2	Insegnamento	6		000-000				2019_0964	11/11/2019	20/12/2019	964	Docente7	6
18	36	Monte ore superato	28341	ISLAMISTICA (1) (LM)	ATTIVITA'	2	Insegnamento	6		000-000			BO	2019_0964	03/02/2020	03/04/2020	964	Docente8	6
		Monte ore non raggiu	27928	LABORATORIO (1) (LM) / (G.A)	ATTIVITA'	2	Laboratorio	6		000-000	G.A	G.A\G.A	BO	2019_0964	23/09/2019	31/10/2019	964	Docente9	6
		Monte ore non raggiu	27928	LABORATORIO (1) (LM) / (G.B)	ATTIVITA'	2	Laboratorio	6		000-000	G.B	G.B\G.B	BO	2019	0964	23/09/2019	31/10/2019	964	Docente10	6

Figura 6.5: CSV Contenente le Informazioni sugli Insegnamenti

[1]Nel rispetto della privacy i nomi dei docenti sono stati sostituiti da nominativi generici.

6.2 UniTime Wrapper Application

Per poter facilitare il procedimento di import dei dati messi a disposizione da Cineca, sezione 6.1, trasformandoli in file xml strutturati come vedremo nelle sezioni successive, è stato progettato e sviluppato uno script ad hoc utilizzando il linguaggio di programmazione python. Il software mette a disposizione dell'utente una piccola interfaccia grafica che offre la possibilità di effettuare delle personalizzazioni, come la scelta della durata delle lezioni e del numero di giorni per settimana. Inoltre, permettere di scegliere se generare i file xml per importare i dati sulla piattaforma UniTime oppure il singolo file xml da passare al solver da riga di comando. Un'altra funzionalità che offre lo script è quella di trasformare una soluzione ottenuta dal solver da riga di comando in file csv, uno per ogni anno della pianificazione dell'orario effettuata, in maniera da rendere la soluzione comprensibile dall'utente finale.

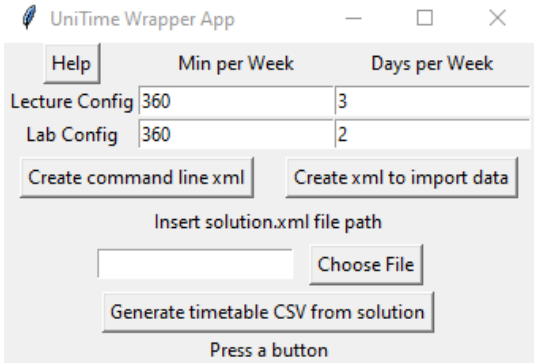

Figura 6.6: Interfaccia Grafica UniTime Wrapper Application

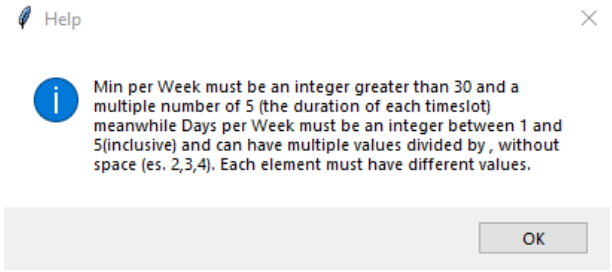

Figura 6.7: Suggerimento per Input Dati UniTime Wrapper Application

Ogni funzionalità offerta dall'applicativo sviluppato è strutturata ed eseguita in maniera asincrona sfruttando le coroutine di python, più precisamente la libreria *asyncio*. Quest'ultima permette di eseguire in maniera concorrente le diverse funzioni ed ottenere performance nettamente migliori in termini di speed up ed efficienza.

```
1  import asyncio
2  ...
3  async def create_import_data_xml(lecture_days_per_week,
       lecture_min_per_week, lab_days_per_week, lab_min_per_week):
4      staff_mapping = asyncio.create_task(
5          staff_csv_to_xml(staff_csv_path)
6      )
7
8      subject_area_mapping =
           asyncio.create_task(subject_areas_csv_to_xml(subject_area_csv_path))
9
10     staff_rows = await staff_mapping
11
12     subject_area_rows = await subject_area_mapping
13
14     courses_mapping = asyncio.create_task(
15         courses_csv_to_xml(courses_csv_path, staff_rows,
               subject_area_rows, lecture_days_per_week,
               lecture_min_per_week, lab_days_per_week,
               lab_min_per_week))
16
17     await asyncio.gather(
18         rooms_csv_to_xml(building_and_rooms_csv_path),
19         departments_csv_to_xml(departments_csv_path)
20     )
21
22     await courses_mapping
```

Listing 6.1: Esempio Codice Coroutine

Nelle successive sezioni verrà analizzato nel dettaglio il codice principale dell'applicazione, spiegando il funzionamento di ogni parte che la compone.

6.3 Utilizzo UniTime Tramite Piattaforma Web

In questa sezione verrà analizzato come i dati esportati ed offerti da Cineca, sezione 6.1, siano stati importati all'interno della piattaforma web di UniTime, evidenziando i problemi riscontrati, e come i dati siano stati utilizzati.

Figura 6.8: Home Applicazione Web UniTime

Figura 6.9: Menu Applicazione Web UniTime

La demo del prodotto è accessibile all'url `https://demo.unitime.org/` con le credenziali pubbliche username *admin* e password *admin*. Per le prove e gli studi effettuati la piattaforma UniTime è stata installata in locale seguendo la guida confrontabile all'url `http://help.unitime.org/Timetabling_Installation`.

6.3.1 Struttura Dati XML

Prima di analizzare come i dati di UP2.0 siano stati utilizzati è bene fare luce su quali siano le informazioni necessarie alla piattaforma web di UniTime per poter sfruttare l'algoritmo di pianificazione dell'orario. Il modello dati di UniTime ha una complessità tale da permettere la definizione di qualsiasi struttura dati necessaria. I punti cardine di questo modello per poter creare in toto una situazione reale di un determinato ateneo sono [26]:

- area accademica: ovvero il campus e/o l'università di riferimento;

```
<academicAreas campus="woebegon" term="Fal" year="2010">
    <!-- Since UniTime 3.4, only title is needed (instead
        of shortTitle and longTitle). If title is not
        present, longTitle is used instead. -->
    <academicArea externalId="A" abbreviation="A"
        title="The Woebegon's Only Academic Area"/>
    <academicArea abbreviation="TEST" title="Test Area"/>
</academicAreas>
```

Listing 6.2: Struttura XML Area Accademica

- sessione accademica: il semestre per il quale si vuole effettuare la pianificazione dell'orario;

```
<sessionSetup term="Fal" year="2010" campus="TEST"
    dateFormat="yyyy/M/d" created="Fri Jun 23 15:21:28
    CEST 2017">
<!-- Element session is optional, needed to create
    and/or update academic session information -->
    <session startDate="2010/8/23" endDate="2010/12/19"
        classEndDate="2010/12/12"
        examStartDate="2010/12/13"
```

```
            eventStartDate="2010/7/19"
            eventEndDate="2011/1/16">
        <holidays>
            <holiday date="2010/9/6"/>
            <break startDate="2010/10/11"
                endDate="2010/10/12"/>
            <break startDate="2010/11/24"
                endDate="2010/11/27"/>
        </holidays>
        </session>
    </sessionSetup>
```

Listing 6.3: Struttura XML Sessione Accademica

- time pattern: il pattern definito come numero giorni a settimana per durata dell'incontro (es. 2 x 120, due giorni a settimana della durata di 120 minuti ciascuno). Per ciascun time pattern è necessario definire quali siano le combinazioni di giorni possibili e l'orario di inizio della lezione evidenziabile come soluzione;

```
    <timePattern name="2 x 50" nbrMeetings="2"
        minsPerMeeting="50" type="Standard" visible="true"
        nbrSlotsPerMeeting="12" breakTime="10">
        <days code="MW"/>
        <days code="MF"/>
        <days code="TTh"/>
        <days code="WF"/>
        <time start="0730"/>
        <time start="0830"/>
        <time start="0930"/>
        <time start="1030"/>
        <time start="1130"/>
        ...
    </timePattern>
```

Listing 6.4: Struttura XML Time Pattern

- date pattern: ovvero le settimane che definiscono il semestre di riferimento;

```
<datePattern name="Full Term" type="Standard"
    visible="true" default="true">
    <dates fromDate="2010/8/23" toDate="2010/8/28"/>
    <dates fromDate="2010/8/30" toDate="2010/9/4"/>
    <dates fromDate="2010/9/7" toDate="2010/9/11"/>
    <dates fromDate="2010/9/13" toDate="2010/9/18"/>
    ...
</datePattern>
```

Listing 6.5: Struttura XML Date Pattern

- academic classification: ovvero la classificazione accademica che descrive i differenti anni accademici;

```
<academicClassifications campus="woebegon" term="Fal"
    year="2010">
    <academicClassification externalId="01" code="01"
        name="Junior Year"/>
    <academicClassification externalId="02" code="02"
        name="Senior Year"/>
</academicClassifications>
```

Listing 6.6: Struttura XML Classificazione Accademica

- dipartimenti: ovvero i dipartimenti dell'università di riferimento;

```
<departments campus="woebegon" term="Fal" year="2007">
    <department externalId="0S4ZF" abbreviation="BSCmpt"
        name="Business Services Computing"
        deptCode="1068"/>
    <department externalId="0SOPX" abbreviation="PFComp"
        name="Physical Facilities Computing"
        deptCode="1678"/>
</departments>
```

Listing 6.7: Struttura XML Dipartimenti

- settori scientifici disciplinari (ssd): dal punto di vista formale, il settore scientifico disciplinare serve a riconoscere l'ambito della ricerca di un determinato insegnamento e a capire qual è l'ssd che caratterizza ciascun corso di laurea. Inoltre, gli ssd sono utilizzati per costruire una proposta didattica consona. All'estero i settori scientifici disciplinari non esistono, ma vengono utilizzati settori più ampi che racchiudono diversi ambiti di ricerca. Tuttavia, UniTime permette di mappare queste informazioni sfruttando la risorsa *subject area*;

```
<subjectAreas campus="puWestLafayetteTrdtn" term="Fal"
    year="2007">
    <subjectArea externalId="10SDPVN" abbreviation="A&AE"
        title="Aeronautics And Astronautics Engineering"
        schedBookOnly="false" pseudoSubjArea="false"
        department="1282"/>
    <subjectArea externalId="19PXO" abbreviation="G S"
        title="General Studies-Lafayette"
        schedBookOnly="false" pseudoSubjArea="false"
        department="1357"/>
</subjectAreas>
```

Listing 6.8: Struttura XML SSD

- aule ed edifici: ovvero tutte gli edifici e le aule di interesse per la pianificazione dell'orario con i rispettivi dipartimenti che ne potranno fare uso e le rispettive caratteristiche;

```
<buildingsRooms campus="woebegon" term="Fal" year="2007">
    <building externalId="14C5A" abbreviation="ABE"
        locationX="449" locationY="392" name="Agricultural
        and Biological Engineering">
    <room externalId="14P3DDEB" locationX="447"
        locationY="392" area="1265.3" roomNumber="204"
        roomClassification="classroom" capacity="58"
        instructional="True"
        scheduledRoomType="genClassroom">
        <roomDepartments>
            <assigned departmentNumber="1979"
                percent="100"/>
```

```
            <scheduling departmentNumber="1128"
                percent="50"/>
        </roomDepartments>
        <roomFeatures>
            <roomFeature feature="seatingType"
                value="Tablet Arm Chairs"/>
            <roomFeature feature="dvd" value="DVD"/>
            <roomFeature feature="computerProjection"
                value="Computer Projection"/>
        </roomFeatures>
    </room>
    </building>
</buildingsRooms>
```

Listing 6.9: Struttura XML Aule ed Edifici

- staff: lo staff universitario e i docenti;

```
<staff campus="PWL" term="Fal" year="2010">
    <staffMember externalId="101" firstName="George"
        lastName="Newman" acadTitle="Prof."
        positionType="PROF" department="0101"/>
    <staffMember externalId="103" firstName="Josef"
        lastName="Nowak" positionType="INSTRUCTOR"
        department="0101"/>
</staff>
```

Listing 6.10: Struttura XML Staff

- curriculum: ovvero l'offerta formativa che definisce il curriculum di ciascun anno accademico (classificazione accademica);

```
<curriculum>
    <academicArea abbreviation="A"/>
    <department code="0101"/>
    <classification enrollment="2">
        <academicClassification code="01"/>
        <course subject="BIOL" courseNbr="101"/>
        <course subject="CALC" courseNbr="101"/>
    </classification>
```

```
        <classification enrollment="2">
            <academicClassification code="02"/>
            <course subject="GER" courseNbr="101"
                share="0.5"/>
        </classification>
    </curriculum>
```

Listing 6.11: Struttura XML Curriculum

- insegnamenti: ovvero l'offerta formativa messa a disposizione dell'università. Per ciascun insegnamento è necessario definirne la configurazione, ovvero le differenti parti che strutturano il corso, ad esempio lezione e laboratorio, e le rispettive classi. Inoltre, è possibile inserire gli esami previsti e la loro tipologia;

```
    <offerings campus="PuWL" year="2006" term="Fall">
        <offering id="10234" offered="true" action="update">
        <course id="10234" subject="A&AE" courseNbr="101"
            controlling="true" title=""/>
        <config name="1" limit="20">
            <subpart type="Lec" suffix="" minPerWeek="100"/>
            <subpart type="Lab" suffix="" minPerWeek="100"/>
            <class id="1006" type="Lab" suffix="1"
                limit="15"/>
            <class id="1007" type="Lab" suffix="2" limit="5">
                <date startDate="08/20" endDate="09/22"/>
                <instructor id="23344" lname="" mname=""
                    fname="" share="0.8" lead="true"/>
                <instructor id="23435" lname="" mname=""
                    fname="" share="0.2"/>
            </class>
        </config>
        <exam id="1223" name="A&AE 101" size="30"
            seatingType="exam" length="120" type="final">
            <instructor id="23344"/>
        </exam>
        </offering>
    </offerings>
```

Listing 6.12: Struttura XML Insegnamenti

- preferenze: ovvero le preferenze di ciascuna risorsa:
  - istruttore: preferenze dell'istruttore possono essere i giorni, gli orari, le aule, ecc. indicate con un grado di preferenza definito come: R required, P prohibited, -2 strongly preferred, -1 preferred, 1 discouraged e 2 strongly discouraged;

```
<instructor externalId="100" firstName="JOE"
    lastName="DOE" department="0101">
    <teachingPref maxLoad="20.0" level="0"/>
    <featurePref feature="Comp" level="-2"/>
    <timePref level="R">
        <pref level="P" day="M" start="0730"
            stop="0830"/>
        <pref level="-2" day="M" start="0930"
            stop="1430"/>
    </timePref>
    <distributionPref type="MAX_HRS_DAY(6)"
        structure="AllClasses" level="-2"/>
    <coursePref subject="ALG" course="101"
        level="-2"/>
</instructor>
```

Listing 6.13: Struttura XML Preferenze Istruttore

  - insegnamento: preferenze dell'insegnamento possono essere i giorni, gli orari, le aule, ecc. indicate con un grado di preferenza definito come: R required, P prohibited, -2 strongly preferred, -1 preferred, 1 discouraged e 2 strongly discouraged;

```
<subpart subject="BIOL" course="201" type="Lab">
    <timePref pattern="1 x 100" level="R">
        <pref level="P" days="M" time="0730"/>
        <pref level="P" days="T" time="0730"/>
        <pref level="P" days="W" time="0730"/>
    </timePref>
    <groupPref group="Classroom" level="R"/>
</subpart>
```

Listing 6.14: Struttura XML Preferenze Insegnamenti

- studenti: ovvero gli studenti dell'ateneo e la loro rispettiva classificazione accademica.

```
<student externalId="1020" firstName="Urshula"
    lastName="Student">
    <studentAcadAreaClass>
        <acadAreaClass academicArea="A"
            academicClass="02"/>
    </studentAcadAreaClass>
</student>
```

Listing 6.15: Struttura XML Studenti

Per modellare la casistica presa in esempio e poter procedere con la pianificazione dell'orario è necessario importare o inserire manualmente sulla piattaforma web di UniTime tutti i dati sopra descritti.

6.3.2 Import dei Dati

L'import dei dati all'interno dell'applicazione web di UniTime è stato possibile grazie all'interfaccia di data exchange messa a disposizione dalla piattaforma stessa e accessibile dalla voce Administrator del menu, come visibile nella figura 6.10. Questa interfaccia non solo permette di importare all'interno del sistema nuovi dati in formato xml, ma permette anche di esportare tutti o solo alcuni dei dati già presenti.

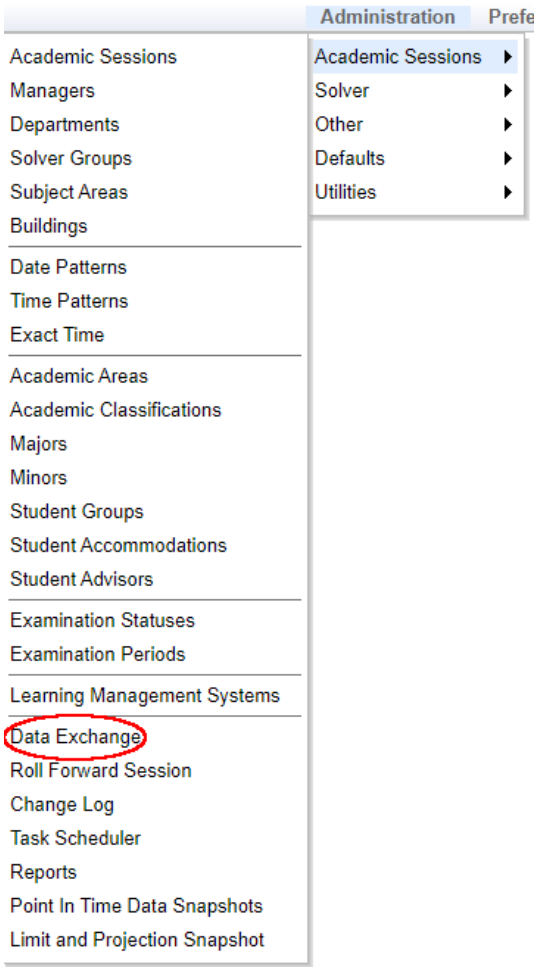

Figura 6.10: Data Exchange Menu Administration UniTime

Figura 6.11: Pagina Data Exchange UniTime

Per poter importare al meglio i dati è necessario generare un file xml per ogni singola risorsa, aule, docenti, insegnamenti, ecc., che si desidera inserire all'interno del sistema. Per adattare il modello dati di UP a quello di UniTime sono state apportate delle modifiche concettuali nella lettura dei dati. Ad esempio, la *subject area* di UniTime è stata intesa come il corso di studio e non il settore scientifico disciplinare, l'unità organizzativa è stata associata al dipartimento e per semplificare la pianificazione dell'orario è stata creata solamente una classe per ciascun insegnamento, *instructional offering*, presente nell'offerta formativa. Nel caso in cui un insegnamento si suddividesse in due moduli differenti oppure avesse due tipologie di incontro, ad esempio lezione e laboratorio, si è provveduto alla creazione di una classe per ogni singola tipologia. La risorsa rappresentata dagli studenti non è stata inizialmente presa in considerazione. Di seguito verrà analizzato il codice utilizzato per la generazione di un file xml per ogni risorsa da importare all'interno del sistema.

Import Docenti

L'import dei docenti, ovvero dello staff universitario, ha come obiettivo principale quello di trasformare il csv contenente i dati di Cineca, sezione 6.1.3, in un xml strutturato come visto nella precedente sezione da importare sulla piattaforma di UniTime. Di seguito è riportato il codice python che si occupa di effettuare questo procedimento.

```
1  async def staff_csv_to_xml(file_name):
2      rows = read_csv_file(file_name)
3
4      staff = ET.Element('staff', campus='Unibo', term='Fall',
           year='2020')
5
6      for row in rows:
7          ET.SubElement(staff, 'staffMember',
               externalId=row['Matricola'], firstName=row['Nome'],
               lastName=row['Cognome'], positionType='INSTRUCTOR',
               department=row['Unita Organizzativa'],
               email=row['Mail'])
8
9      tree = ET.ElementTree(staff)
10     tree.write('ScriptOutput\\Staff.xml')
11     return rows
```

Listing 6.16: Codice Generazione XML Docenti

La funzione prende a parametro il nome del csv contente le informazioni relative ai docenti. La chiamata al metodo *read_csv_file*, riga 2, restituisce le righe presenti nel csv; per ognuna di queste viene creata una voce *staffMember*, riga 7, all'interno della radice *staff*, riga 4, precedentemente creata. L'elemento *staffMember* contiene al suo interno la matricola del docente, il nome, il cognome, il dipartimento di appartenenza, la mail e la posizione ovvero il ruolo. Nel caso di studio analizzato la posizione è sempre *INSTRUCTOR* poiché tutti i membri dello staff universitario inseriti sono solamente docenti. Al termine di questa procedura la funzione si occupa di scrivere il file xml generato, riga 9 e 10, e ritorna le righe del csv dei docenti che verranno utilizzate nelle prossime procedure per evitare di rieseguire la lettura del file ogni qualvolta ci sia la necessità.

Import Aule ed Edifici

L'import delle aule e degli edifici viene effettuato a partire dal file csv esportato da UP2.0, sezione 6.1.1, che contiene al suo interno le informazioni di ciascun'aula e del relativo edificio. Di seguito è riportato il codice python che svolge questo procedimento.

```
1  async def rooms_csv_to_xml(file_name):
2      rows = read_csv_file(file_name)
3      current_building = ''
4      current_building_full_name = ''
5      location = ''
6
7      building_rooms = ET.Element('buildingsRooms', campus='Unibo',
           term='Fall', year='2020')
8
9      for row in rows:
10         # presumo che le aule nello stesso edificio abbiano
11         # lo stesso indirizzo
12         if row['Edificio'] != current_building_full_name:
13             location = get_room_location_lat_lon(row['Via'])
14             current_building_full_name = row['Edificio']
15             minus_index = current_building_full_name.find('-')
16             building_external_id =
                   current_building_full_name[:minus_index - 1]
17             building_name = current_building_full_name[minus_index
                   + 2:]
18             current_building = ET.SubElement(building_rooms,
                   'building', externalId=building_external_id,
                   abbreviation=building_external_id,
                   locationX=location[0], locationY=location[1],
                   name=building_name)
19
20         add_room_to_building(current_building, row, location)
21
22         tree = ET.ElementTree(building_rooms)
23         tree.write('ScriptOutput\\Rooms.xml')
```

Listing 6.17: Codice Generazione XML Aule ed Edifici

Dato che all'interno del file non vi sono distinzioni tra aule ed edifici ma bensì solamente le aule con il corrispettivo edificio, si è proceduto in primis con la creazione, se non già precedentemente creato, degli elementi *building* (riga 18), contenenti tutte le informazioni relative all'edificio quali: posizione geografica (latitudine e longitudine oppure x e y), nome, abbreviazione e id. Tuttavia, il file csv non contiene al suo interno le informazioni relative alla posizione geografica espressa in coordinate ma bensì il nome della via e il numero civico. Per velocizzare ed automatizzare questo procedimento di conversione è stata sfruttata la libreria *geopy.geodecoders*, più precisamente il nome *Nomatim*. Questo ha permesso di poter trasformare di volta in volta l'indirizzo in coordinate geografiche definite come latitudine e longitudine.

```
1 from geopy.geocoders import Nominatim
2
3 app = Nominatim(user_agent="data_import")
4
5 def get_room_location_lat_lon(route):
6     try:
7         location = app.geocode(route).raw
8         return location['lat'], location['lon']
9     except:
10        return '0', '0'
```

Listing 6.18: Codice Funzione get_room_location_lat_lon

Successivamente, una volta creato l'elemento *building*, la funzione *add_room_to_building* ha il compito di inserire all'interno dell'edificio ciascun'aula di appartenenza, aggiungendo le informazioni relative all'id, il numero della stanza, la classificazione (restituita dal metodo *get_room_classification*, ad esempio aula, laboratorio, aula magna, ecc.), la capacità, l'area in metri quadri, la posizione, eccetera. Al termine di questa procedura vengono definite per ogni stanza le *features*, ovvero le caratteristiche della singola aula e i dipartimenti che ne potranno fare uso, come precedentemente visto nella sezione 2.2.3.

```
1 def add_room_to_building(building, row, location):
2     room_classification = get_room_classification(row)
3     capacity = row['Capienza'] if row['Capienza'] != '' else '0'
4     area = row['Metri quadri'] if row['Metri quadri'] != '' else
          '0'
5     underscore_last_occurence = str(row['Codice']).find('_')
6     room = ET.SubElement(building, 'room',
          externalId=row['Codice'],
          roomNumber=row['Codice'][underscore_last_occurence + 1:],
          roomClassification=room_classification[0],
          capacity=capacity, instructional=room_classification[2],
          locationX=location[0], locationY=location[1], area=area,
          scheduledRoomType=room_classification[1],
          displayName=row['Descrizione'])
7
8     # room features
9     ET.SubElement(room, 'roomFeatures',
          workstationsNumbers=row['Numero postazioni'],
10                PCNumbers=row['Numero PC'],
                      services=row['Servizi'], floor=row['Piano'])
11    # room departments
12    room_departments = ET.SubElement(room, 'roomDepartments')
13
14    ET.SubElement(room_departments, 'assigned',
          departmentNumber='6', percent='100')
15    ET.SubElement(room_departments, 'scheduling',
          departmentNumber='6', percent='50')
16    ET.SubElement(room_departments, 'scheduling',
          departmentNumber='12', percent='50')
```

Listing 6.19: Codice Funzione add_room_to_building

Import Settori Scientifici Disciplinari

Come precedentemente accennato nella sottosezione 6.3.2 come subject area, ovvero SSD - settore scientifico disciplinare, sono stati mappati i corsi di studio. Il csv contente le informazioni riguardanti i corsi di studio e il relativo dipartimento è strutturato come visto nella sezione 6.2 mentre di

seguito è riportato il codice che si occupa di generare l'xml da importare sulla piattaforma UniTime.

```
1  async def subject_areas_csv_to_xml(file_name):
2      rows = read_csv_file(file_name)
3
4    subject_areas = ET.Element('subjectAreas', campus='Unibo',
         term='Fall', year='2020')
5
6      subject_areas_map = {}
7
8      for row in rows:
9          description = row['Descrizione']
10         # verifico che non sia gia' presente un corso di studi
11         # con la stessa descrizione, altrimenti aggiungo un numero
12         # in fondo alla descrizione
13         if description in subject_areas_map:
14             subject_areas_map[description] =
                   subject_areas_map[description] + 1
15         else:
16             subject_areas_map[description] = 0
17
18         current_subject_area_instances = '' if
               subject_areas_map[description] == 0 else
               subject_areas_map[description]
19         abbreviation = str(row['Descrizione'])[:37] +
               str(current_subject_area_instances)
20         title = row['Descrizione'] +
               str(current_subject_area_instances)
21         ET.SubElement(subject_areas, 'subjectArea',
               externalId=row['Codice'], abbreviation=abbreviation,
               title=title, department=row['Unita organizzativa'])
22
23     tree = ET.ElementTree(subject_areas)
24     tree.write('ScriptOutput\\SubjectAreas.xml')
25     return rows
```

Listing 6.20: Codice Generazione XML Subject Area

Vista la possibilità che due o più corsi di studio abbiano la stessa descrizione ma codici diversi, le righe 12, 13 e 14 si occupano di verificare che non sia già stato generato un elemento con quella descrizione; in alternativa viene aggiunto alla fine della descrizione un numero progressivo che parte da 1 e aumenta ogni qualvolta si riscontrasse questa determinata casistica.

Import Dipartimenti

L'import dei dipartimenti a partire dal file di UP2.0, sezione 6.2, comporta la creazione all'interno del file xml di un elemento *department* sotto la radice precedentemente creata *departments*, per ogni dipartimento presente nell'università che si intende modellare. Questo elemento contiene al suo interno le informazioni relative al codice del dipartimento, il nome, l'abbreviazione ed un id esterno. Di seguito il codice che riporta il procedimento sopra descritto.

```
1  async def departments_csv_to_xml(file_name):
2      rows = read_csv_file(file_name)
3      departments = ET.Element('departments', campus='Unibo',
           term='Fall', year='2020')
4
5      for row in rows:
6          description = str(row['Descrizione'])
7          underscore_index = description.find(' - ')
8          # l'abbreviazione deve essere al massimo di 20 caratteri
9          abbreviation = description[:underscore_index] if
               underscore_index != -1 else description[:19]
10         name = description[underscore_index + 3:] if
               underscore_index != -1 else description
11         ET.SubElement(departments, 'department',
               externalId=row['Codice'], abbreviation=abbreviation,
               name=name, deptCode=row['Codice'])
12
13     tree = ET.ElementTree(departments)
14     tree.write('ScriptOutput\\Departments.xml')
```

Listing 6.21: Codice Generazione XML Dipartimenti

Import Insegnamenti

Le informazioni più importanti per la pianificazione dell'orario sono contenute nel file xml necessario per importare gli insegnamenti e le loro caratteristiche. Tuttavia, il csv esportato da UP2.0 non contiene al suo interno solamente le informazioni sugli insegnamenti, ma anche tutti i dati necessari per generare i file xml relativi alle preferenze, ai curriculum e alle classificazioni accademiche (vedi sezione 6.3.1). Come precedentemente accennato nella sezione sopra citata e nella sezione 6.3.2 il file xml degli insegnamenti, ovvero gli *instructional offering*, dovrà contenere al suo interno tutte le informazioni relative al corso come id, SSD (mappato con il corso di studi) e la configurazione dell'insegnamento composta dalle classi del corso e la loro rispettiva tipologia, l'istruttore di riferimento e tutte le altre informazioni correlate. Di seguito è riportata una sezione di codice che permette di generare il file xml degli insegnamenti.

```
1  def create_course_offering(course_offerings_map,
2                             subject_area_rows, instructor_rows,
3                             offerings, preferences,
4                             classifications,
5                             academic_classifications,
6                             curriculum, lecture_days_per_week,
7                             lecture_min_per_week,
8                             lab_days_per_week, lab_min_per_week):
9      class_id = 1000
10
11     for code in course_offerings_map:
12         current_course_offering_row = course_offerings_map[code][0]
13
14         subject_area = get_subject_area(subject_area_rows,
               current_course_offering_row['Corso di studio'])
15
16         offering = ET.SubElement(offerings, 'offering',
               id=current_course_offering_row['Codice'],
               offered='true', action='update|delete')
17
18         ET.SubElement(offering, 'course',
               id=current_course_offering_row['Codice'],
               subject=subject_area,
```

```
            courseNbr=current_course_offering_row['Codice'],
            controlling='true',
            title=current_course_offering_row['Evento'])
19
20          config = ET.SubElement(offering, 'config', name='1',
            limit='10')
21
22          activity_typology = current_course_offering_row['Tipo
            attivita']
23
24          typology = 'Lec' if activity_typology == 'Insegnamento'
            else 'Lab'
25
26          min_per_week = lecture_min_per_week if typology == 'Lec'
            else lab_min_per_week
27
28          class_suffix = 0
29
30          ET.SubElement(config, 'subpart', type=typology, suffix='',
31                        minPerWeek=str(min_per_week))
32
33          # creo una classe per ciascun insegnamento o modulo
34          # presente nella struttura dati
35          for course in course_offerings_map[code]:
36
37              class_suffix += 1
38
39              class_id += 1
40
41              start_date_datetime = datetime.strptime(course['data
                ini'], '%d/%m/%Y')
42
43              str_start_date = str(start_date_datetime.month) + '/' +
                str(start_date_datetime.day)
44
45              end_date_datetime = datetime.strptime(course['data
                fine'], '%d/%m/%Y')
46
47              str_end_date = str(end_date_datetime.month) + '/' +
```

```
                str(end_date_datetime.day)
48
49              class_ = ET.SubElement(config, 'class',
                    id=str(class_id), type=typology,
                    suffix=str(class_suffix), limit='10')
50
51              ET.SubElement(class_, 'date', startDate=str_start_date,
                    endDate=str_end_date)
52
53              instructor_id = get_instructor_id(instructor_rows,
                    course['Responsabili'])
54
55              if instructor_id != -1:
56                  ET.SubElement(class_, 'instructor',
                        id=instructor_id, lname='', mname='', fname='',
                        share='100', lead='true')
57
58          days_per_week = lecture_days_per_week if typology == 'Lec'
                else lab_days_per_week
59          # calcolo le preferenze dei giorni e la durata
60          # delle lezioni
61          days_preferences = []
62          for days in days_per_week:
63              days_preferences.append(str(days) + ' x ' + str(int(
64                  min_per_week / int(days))))
65
66          create_preferences(preferences, subject_area,
                current_course_offering_row['Codice'], typology,
                days_preferences, 'MyClassrooms',
                course_offerings_map[code])
67
68          create_curriculum(current_course_offering_row, curriculum,
                classifications, subject_area,
69                              academic_classifications)
```

Listing 6.22: Codice Generazione XML Insegnamenti

La funzione *create_course_offering* prende come parametri la mappa contenente gli insegnamenti di ciascun corso (un corso può essere suddiviso in più moduli), i settori scientifici disciplinari (ovvero i corsi di studio), gli

istruttori, gli oggetti di tipo *ET.Element* della libreria *xml.etree.cElementTree* quali offerings, preferences, academic_classifications e curriculum che andranno poi a strutturare i diversi file xml, l'elenco delle classificazioni ed infine le personalizzazioni effettuate dall'utente, ovvero i minuti di ciascuna lezione e il numero di giorni a settimana sia degli incontri in aula che dei laboratori. Successivamente il codice si occupa di generare l'elemento relativo al corso di interesse e le relative configurazioni. Come accennato in precedenza, per semplificare la pianificazione dell'orario è stata creata una classe per ciascun insegnamento, a discapito della situazione in cui l'insegnamento sia suddiviso in moduli: in quella determinata situazione è stata inserita una classe per ogni modulo del corso (vedi da riga 35 a riga 56). Una volta terminata questa procedura si passa alla creazione delle preferenze, dei curriculum e delle classificazioni accademiche.

```
1  def create_preferences(preferences, subject, course_number,
       typology, time_patterns, group, course_code_mapping):
2      subpart = ET.SubElement(preferences, 'subpart',
           subject=subject, course=course_number, type=typology)
3
4      for time_pattern in time_patterns:
5          ET.SubElement(subpart, 'timePref', pattern=time_pattern,
               level='R')
6
7      ET.SubElement(subpart, 'groupPref', group=group, level='R')
8
9      suffix = 0
10
11     for _ in course_code_mapping:
12         suffix += 1
13         ET.SubElement(preferences, 'class', subject=subject,
               course=course_number, type=typology, suffix=str(suffix))
```

Listing 6.23: Codice Generazione XML Preferences

Per quel che concerne la creazione delle preferenze, le informazioni per ogni classe contenute all'interno dell'elemento *subpart*, riga 2, sono: i time pattern preferiti (strutturati come visto nella sezione 6.3.1, ad esempio 2 x 100) con relativo livello di preferenza e il gruppo di stanze nel quale la lezione potrà essere assegnata. Nella casistica in esame, il gruppo di

aule prende il nome di *MyClassrooms*. UniTime non permette di importare queste informazioni tramite l'utilizzo di un file xml sfruttando la pagina di import e export dei dati, ma bisognerà creare il gruppo manualmente dalla pagina *Room Groups* della piattaforma e poi successivamente importare il file delle preferenze.

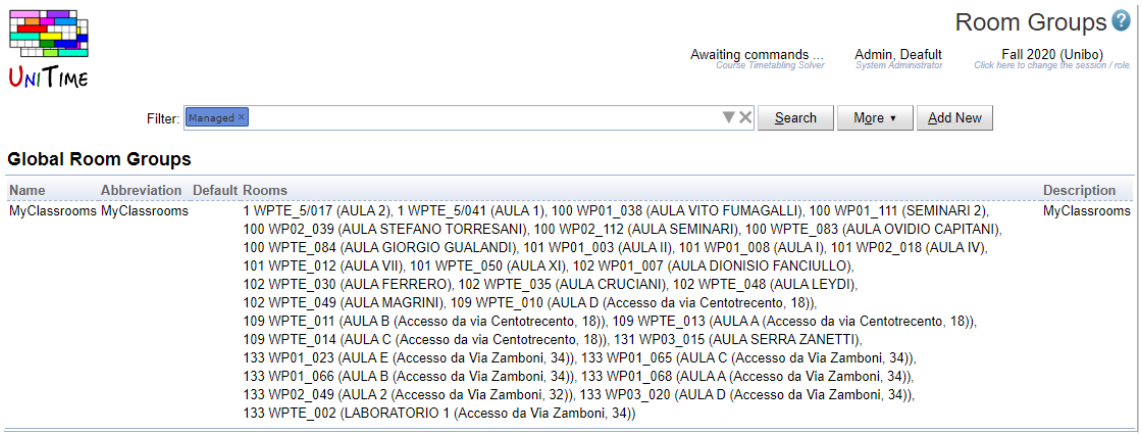

Figura 6.12: Pagina Room Groups UniTime

Infine, si procede con la creazione dei curriculum e delle classificazioni accademiche che andranno a modellare ogni singolo anno del corso di studi. Di seguito è mostrato il codice che permette di effettuare questo procedimento.

```
1 def create_curriculum(course_row, curriculum, classifications,
      subject_area, academic_classifications):
2     year = course_row['anno di corso']
3
4     if year in classifications:
5         current_classification = classifications[year]
6     else:
7         ET.SubElement(academic_classifications,
              'academicClassification', externalId=str(year),
              code=str(year), name='Antropologia' + str(year))
8         current_classification = ET.SubElement(curriculum,
              'classification', name=str(year), enrollment='10')
```

```
9         ET.SubElement(current_classification,
              'academicClassification', externalId=str(year),
              code=str(year))
10        classifications[year] = current_classification
11
12    ET.SubElement(current_classification, 'course',
          subject=subject_area, courseNbr=course_row['Codice'])
```

Listing 6.24: Codice Generazione XML Curriculum e Academic Classification

6.3.3 Esecuzione del Solver

L'import dei dati sopra descritto deve avvenire secondo un ordine logico per poter permettere al database di collegare tutti i riferimenti tra le varie risorse e rispettare tutte le dipendenze tra i dati. La figura successiva mostra le dipendenze tra le varie risorse; da essa è possibile dedurre l'ordine di import dei dati.

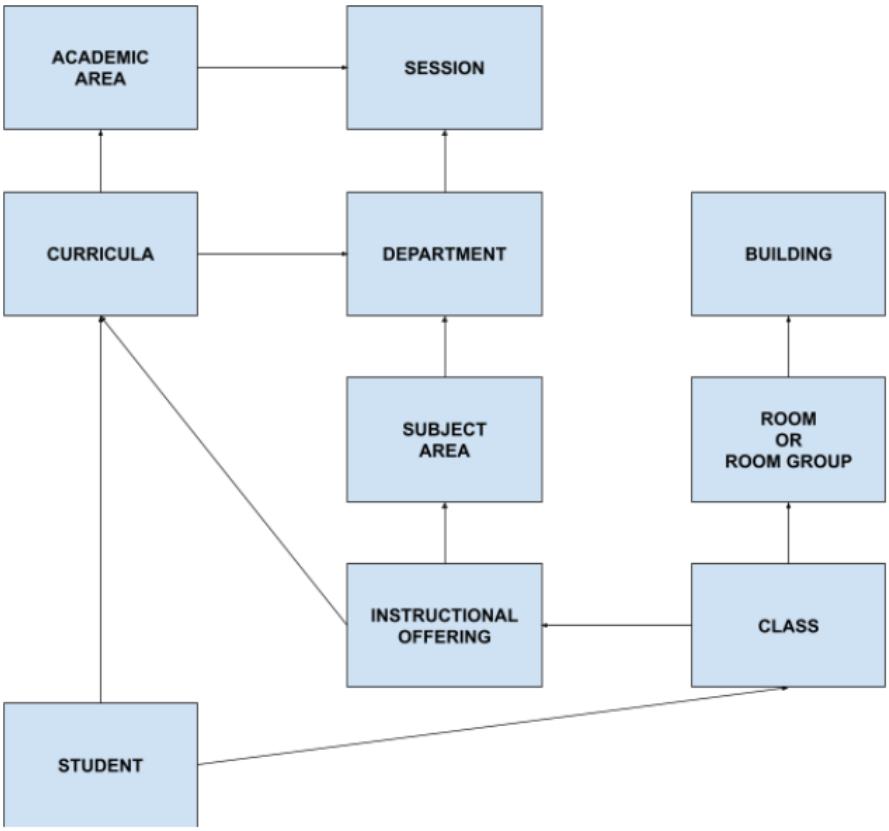

Figura 6.13: Dipendenze Dati UniTime

Una volta terminata la procedura di import e verificato che i dati siano stati importati correttamente senza errori, si potrà procedere con l'esecuzione del solver. Per effettuare questa operazione sarà sufficiente recarsi alla pagina relativa al solver, premere sul bottone *Load*, impostare i parametri di esecuzione e premere il tasto *Start*.

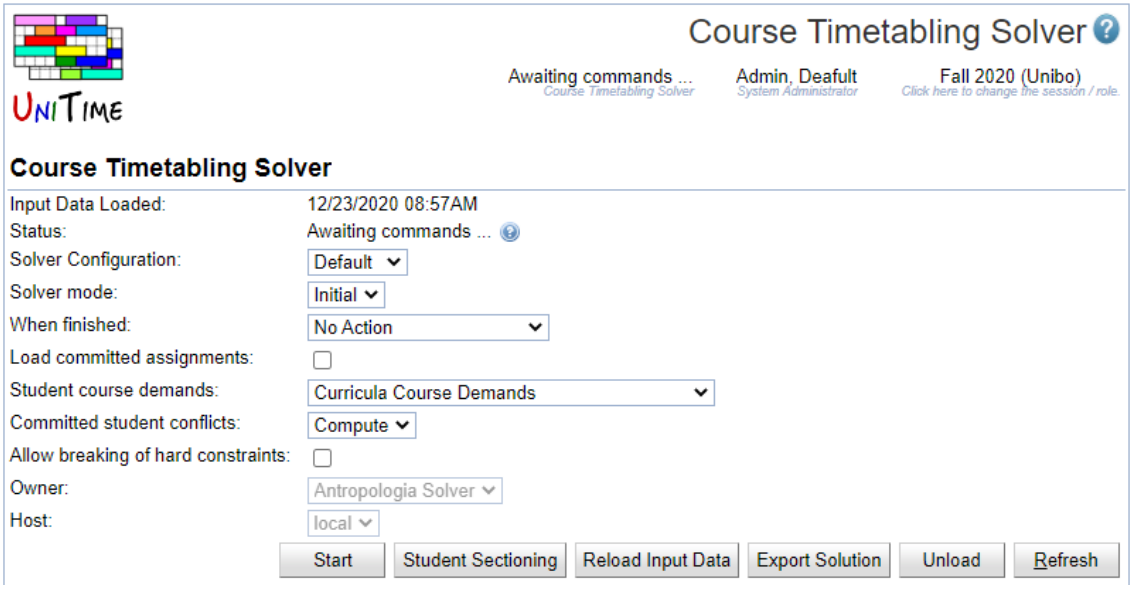

Figura 6.14: Pagina Solver Piattaforma UniTime

Come sopra accennato e come possibile vedere nella figura 6.14, il solver può essere configurato in base alle necessità dell'utente. I parametri che la piattaforma permette di configurare ad ogni singolo avvio sono [27]:

- Solver Configuration:
  - Check: verifica se i dati di input sono coerenti e se esiste una pianificazione completa (fattibile) coerente con i dati passati in input. Il solver non ottimizza la soluzione:
    - i Solution Comparator Weights sono tutti settati a 0;
    - il Placement Selection ha la maggior parte dei parametri di ottimizzazione impostata su 0;
    - il solver termina quando è stata trovata una soluzione completa;
  - Default: crea un orario ottimizzato coerente con i dati di input:
    - la maggior parte dei parametri viene mantenuta con i propri valori predefiniti;

  - Interactive: crea manualmente un orario e/o apporta modifiche a un orario esistente che non deve essere coerente con i dati di input:
    * al solver è consentito infrangere vincoli rigidi (come orari/aule obbligatori/proibiti);
    * la preferenza di distribuzione "Different Time" viene utilizzata per classi della stessa offerta didattica (come lezioni e laboratori in contemporanea) invece di "Same Students" - questo rende possibile, ad esempio, collocare una lezione in un luogo lontano dal laboratorio, cosa che non sarebbe possibile con il vincolo "Same Students";

- Solver mode:
  - Initial: crea una nuova pianificazione dell'orario;
  - MPP: continua a lavorare su una pianificazione precedentemente generata, provando a cercare una soluzione migliore con meno differenze possibili rispetto a quella caricata;

- When finished: definisce cosa fare quando il solver raggiunge il timeout;

- Allow breaking of hard constraints: consente la violazione degli slot orari e aule richiesti e/o proibiti. Ciò è utile per poter assegnare un orario o una stanza proibita a una classe utilizzando la pagina *Suggerimenti* mentre si apportano modifiche manuali;

- Student course demands: ovvero le richieste del corso da parte degli studenti che vengono utilizzate durante la pianificazione dell'orario per evitare conflitti tra i vari studenti:
  - Last Like Student Course Demands: iscrizioni reali degli studenti ai corsi nell'ultimo semestre;
  - Weighted Last Like Student Course Demands: iscrizioni reali ponderate in modo che i corsi siano riempiti di studenti (Esempio: se nell'anno precedente c'erano 15 studenti in un corso e quest'anno il corso ha un limite di 20, la media degli ultimi studenti iscritti a quel corso negli anni precedenti viene moltiplicata per 20/15)

  - Projected Student Course Demands: richieste per un determinato corso sono calcolate in base alle richieste di uno studente "tipo", utilizzando le regole di proiezione dalla schermata *Regole di proiezione del curriculum*;
  - Curricula Course Demands: richieste dei corsi calcolate in base ai curriculum;
  - Curricula Last Like Course Demands: una combinazione tra i curriculum e le richieste dei corsi degli studenti. I curriculum aiutano con le richieste dei nuovi corsi e dei corsi obbligatori, ecc.; mentre le richieste degli studenti forniscono dati riguardanti i corsi seguiti in precedenza (caso d'uso: i curriculum contengono solo informazioni sui corsi obbligatori ed opzionali, le richieste per i corsi opzionali sono tratte dai dati di immatricolazione dell'ultimo anno);
  - Student Course Requests: le richieste reali sono inserite manualmente da ciascun studente;
  - Enrolled Student Course Demands: iscrizioni reali degli studenti nel semestre di riferimento che devono essere pianificate (usato principalmente per testing);

- Student final sectioning: il risolutore effettuerà il sezionamento degli studenti come fase finale della risoluzione del problema (cioè, quando viene trovata una soluzione completa nel caso in cui sia utilizzata la configurazione *Check* o al timeout per le altre tipologie di configurazioni) per avere informazioni più accurate sui conflitti degli studenti.

Oltre ai parametri descritti, il solver può essere configurato modificando ulteriori parametri visibili nelle pagine *Solver Parameters* e *Parameter Groups* che però non stati modificati per i lanci di test.

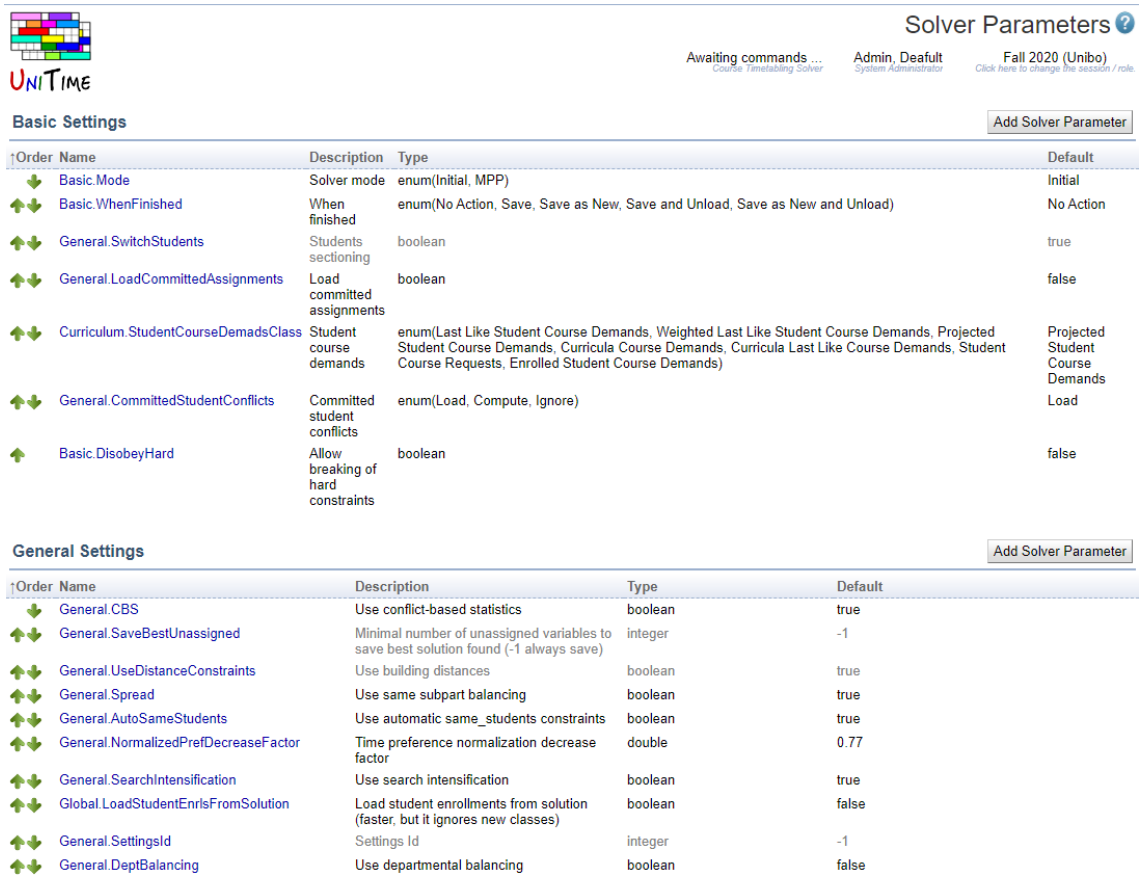

Figura 6.15: Pagina di Gestione dei Parametri del Solver

Nel capitolo successivo verranno mostrate ed analizzate le soluzioni ottenute e i parametri utilizzati per l'esecuzione del solver.

6.4 Esecuzione del Solver da Riga di Comando

In questa sezione verrà analizzato nel dettaglio l'esecuzione della libreria di UniTime vista nei capitoli 4 e 5.

6.4.1 Struttura Dati XML

Diversamente da come visto in precedenza nella sezione 6.3.1 relativa alla struttura dei file xml da importare sulla piattaforma UniTime, l'esecuzione del solver da riga di comando necessita di raggruppare tutte le informazioni all'interno di un unico file xml. Come prima cosa è necessario definire il campus e il semestre di riferimento, i giorni della settimana che si intendono pianificare (da lunedì a domenica) e gli slot che compongono ciascuna giornata [28]. Ogni slot dura 5 minuti, di conseguenza una giornata è composta da 288 slot.

```
<?xml version="1.0" encoding="UTF-8"?>
<!--University Course Timetabling-->
<timetable version="2.4"
initiative="puWestLafayetteTrdtn"
term="2007Fal"
created="Mon Mar 26 14:42:12 EDT 2007"
nrDays="7"
slotsPerDay="288">
```

Listing 6.25: Struttura Radice XML Prompt

Successivamente, è necessario inserire all'interno del file le informazioni relative alle aule utilizzabili nella pianificazione. Per ciascuna di esse bisogna inserire le informazioni relative alla posizione, alla capacità, una variabile booleana *constraint* per creare un vincolo su quella determinata aula, una variabile booleana *ignoreTooFar* se la distanza tra le aule non deve essere considerata e un valore booleano *discouraged* per identificare se quella determinata stanza è sconsigliata. La distanza tra le aule viene calcolata utilizzando la seguente formula:

```
10 * ((x2-x1)^2 + (y2-y1)^2)^1/2
```

Listing 6.26: Formula Distanza tra due Aule

che restituisce la distanza in metri tra due punti. Se la posizione di una stanza non è presente allora la distanza verrà considerata infinita. Oltre a queste informazioni, ogni aula può contenere al suo interno i dati relativi alla condivisione dell'aula tra i vari dipartimenti, il *pattern* ovvero la matrice che definisce la disponibilità dell'aula in una determinata data (F libera per tutti, X non utilizzabile oppure un valore numerico che corrisponde al dipartimento che potrà usufruirne in quel giorno) e gli elementi *freeForAll* e *notAvailable* che anche essi possono assumere i valori F libera per tutti, X non utilizzabile oppure l'id del dipartimento di interesse.

```
<rooms>
    <room id="201" constraint="true" capacity="48"
       location="489,468"/>
    <room id="202" constraint="true" capacity="40"
       location="488,473"/>
    <room id="203" constraint="true" capacity="39"
       location="488,473"/>
    <room id="54" constraint="true" capacity="51"
       location="437,411">
       <sharing>
           <pattern unit="6">FFFFFFFFFFFFFFF0000002200000000000000
           FFFFFFFFFFFFFFFFFFFFFFFFF000000000000000000000...</pattern>
           <freeForAll value="F"/>
           <notAvailable value="X"/>
           <department value="0" id="3"/>
           <department value="1" id="1"/>
           <department value="2" id="2"/>
       </sharing>
    </room>
    <room id="49" constraint="false" capacity="63"
       discouraged="true" ignoreTooFar="true"/>
    ...
</rooms>
```

Listing 6.27: Sezione XML Aule

Una volta inserite le informazioni relative alle aule si procede con l'inserimento dei docenti ovvero gli *instructors*. Per questa risorsa è sufficiente inserire l'*id* e una variabile booleana *ignDist*.

```
<instructors>
   <instructor id="307" ignDist="true"/>
   <instructor id="305" ignDist="true"/>
   <instructor id="315" ignDist="true"/>
 </instructors>
```

Listing 6.28: Sezione XML Docenti

Ciascuna classe rappresenta un gruppo di studenti che partecipa ad un corso. Ogni classe ha un limite, un offering, una configurazione, una subpart, un limite di studenti, un numero di stanze, uno scheduler (ovvero il dipartimento di appartenenza) e un pattern di date. Se il numero di stanza è definito uguale a 0 allora la stanza non verrà assegnata, in alternativa è necessario inserire le stanze che si preferiscono con il relativo attributo *pref.* Ciascuna classe oltre alle aule può avere uno o più istruttori. Esempio di classe:

```
<class id="1271" offering="715" config="716" committed="false"
    subpart="771" classLimit="30" scheduler="4"
    dates="000000000000000000000000000000000000000011111100111110111111011
11110111111011111101111110111111011111100000000111111
01111110111111011111101111110111111">
      <instructor id="537"/>
      <room id="84" pref="0"/>
      <room id="64" pref="0"/>
      <time days="1000000" start="126" length="12" pref="0.0"/>
      <time days="0100000" start="138" length="12" pref="0.0"/>
      <time days="0001000" start="138" length="12" pref="0.0"/>
      <time days="0001000" start="150" length="12" pref="-2.0"/>
      <time days="0000100" start="150" length="12" pref="-2.0"/>
      <time days="1000000" start="162" length="12" pref="-8.0"/>
      <time days="0100000" start="162" length="12" pref="-8.0"/>
      <time days="0010000" start="162" length="12" pref="-8.0"/>
      <time days="0001000" start="162" length="12" pref="-8.0"/>
      <time days="0001000" start="186" length="12" pref="2.0"/>
      <time days="0000100" start="186" length="12" pref="2.0"/>
    </class>
```

Listing 6.29: Sezione XML Classi

Altro elemento fondamentale per la creazione degli xml sono i *groupConstraints* ovvero i vincoli della pianificazione. Ciascuno di essi è definito da un id, una tipologia che può assumere differenti valori consultabili all'indirizzo `https://www.unitime.org/uct_grconstraints_v24.php` e un attributo di preferenza *pref*. Quest'ultimo può essere definito tramite i seguenti valori:

- R: required;
- P: prohibited or discouraged;
- -2: strongly preferred;
- -1: preferred;
- 1: discouraged;
- 2: strongly discouraged.

```
<groupConstraints>
    <constraint id="121" type="BTB" pref="R">
        <class id="1479"/>
        <class id="1480"/>
    </constraint>
    <constraint id="170" type="DIFF_TIME" pref="-2">
        <class id="1615"/>
        <class id="1616"/>
    </constraint>
    <constraint id="1045" type="CLASS_LIMIT" pref="R">
        <parentClass id="1609"/>
        <class id="1614"/>
        <class id="1615"/>
    </constraint>
    ...
<groupConstraints/>
```

Listing 6.30: Sezione XML Group Constraints

I vincoli utilizzati nel caso di studio di interesse sono:

- SAME_STUDENTS: vincolo che definisce che le classi presenti in questo vincolo sono frequentate dagli stessi studenti. Questo vincolo implica che le classi non possono sovrapporsi nel tempo (vincolo DIFF_TIME) e che le aule non debbano essere troppo distanti l'una dall'altra. Può assumere i valori R required, -1 preferred e -2 strongly preferred;
- SAME_INSTR: vincolo che definisce che le classi presenti all'interno di questo vincolo hanno lo stesso docente. Questo comporta l'impossibilità delle classi di sovrapporsi nel tempo e se è presente un vincolo di BTB (back-to-back), ovvero che le classi di interesse debbano avere slot orari e aule adiacenti, le aule non devono essere troppo distanti tra loro. Può assumere i valori R required, -1 preferred oppure -2 strongly preferred.

Infine, per ultimo ma non per ordine di importanza, lo studente. Lo studente è definito da un id e contiene al suo interno gli offering ovvero i corsi scelti, le classi a cui appartiene e quelle invece proibite. Nella casistica studiata è stato creato uno studente per ogni anno di corso per modellare le informazioni relative all'offerta formativa di ciascun anno di corso.

```
<student id="61">
    <offering id="429"/>
    <class id="831"/>
    <class id="539"/>
    <class id="586"/>
    <prohibited-class id="1707"/>
    <prohibited-class id="1702"/>
</student>
```

Listing 6.31: Sezione XML Studenti

6.4.2 Import dei Dati

Come precedentemente accennato, l'esecuzione del solver da riga di comando richiede in input un singolo file xml strutturato come visto nella sezione precedente.

Import Aule

Le informazioni iniziali da inserire all'interno del file xml sono quelle relative alle aule. Di seguito è riportato il codice che permette di convertire il csv di Cineca, figura 6.1, in elementi *room*. Ciascuno di essi contiene al suo interno i dati relativi alla capacità dell'aula, la rispettiva posizione geografica 6.4.1 e il valore booleano *constraint* che definisce se quella precisa aula sia da considerare un vincolo.

```
1  async def room_csv_to_command_line_xml(file_name, timetable):
2      rows = read_csv_file(file_name)
3      rooms = ET.SubElement(timetable, 'rooms')
4      current_building_full_name = ''
5
6      for row in rows:
7          location = (0, 0)
8          # presumo che le aule nello stesso edificio abbiano
9          # lo stesso indirizzo
10         if row['Edificio'] != current_building_full_name:
11             current_building_full_name = row['Edificio']
12             location = get_room_location_lat_lon(row['Via'])
13         ET.SubElement(rooms, 'room', id=row['Codice'],
               constraint='true', capacity=row['Capienza'] if
               row['Capienza'] != '' else '0',
               location=str(location[0]) + ',' + str(location[1]))
```

Listing 6.32: Codice Generazione Sezione XML Aule per Solver Prompt

A differenza dell'import dei dati all'interno della piattaforma web analizzato nella sezione 6.3.2, in questa casistica non è necessario definire gli edifici in cui le aule sono situate. Alla riga 12 è stata utilizzata la stessa funzione *get_room_location_lat_lon* vista in precedenza nella sezione 6.3.2, codice 6.18.

Import Docenti

Una volta inseriti all'interno del file xml tutte le informazioni relative alle aule, si procede con la generazione degli elementi *instructor* ovvero i docenti. Segue il codice utilizzato per trasformare il file csv dei docenti esportato da UP2.0, figura 6.4.

```
1 async def staff_csv_to_command_line_xml(file_name, timetable):
2     rows = read_csv_file(file_name)
3
4     instructors = ET.SubElement(timetable, 'instructors')
5
6     for row in rows:
7         ET.SubElement(instructors, 'instructor',
              id=row['Matricola'], ignDist='true')
8
9     return rows
```

Listing 6.33: Codice Generazione Sezione XML Docenti per Solver Prompt

Import Insegnamenti

Come nella sezione 6.3.2, anche nell'utilizzo del solver da riga di comando gli insegnamenti ricoprono il ruolo principale della pianificazione e con loro tutte le rispettive informazioni. Il file iniziale utilizzato è strutturato come visibile nella figura 6.5 ed è lo stesso utilizzato per l'import dei dati sulla piattaforma web di UniTime. Tuttavia, viste le diverse esigenze di questa casistica, il codice utilizzato per mappare il csv in elementi xml è stato sviluppato ad hoc per la situazione in esame. Di seguito è stata riportata la funzione principale che permette di effettuare queste operazioni.

```
1 async def course_to_command_line_xml(file_name, instructor_rows,
      timetable, lecture_days_per_week, lecture_min_per_week,
      lab_days_per_week, lab_min_per_week):
2     rows = read_csv_file(file_name)
3
4     # mappa contente l'id dei corsi e la riga del csv relativa
5     # a quell'insegnamento
6     course_offerings_map = {}
```

```
7
8     classes = ET.SubElement(timetable, 'classes')
9
10    # struttura dati che mantiene al suo interno
11    # le informazioni relative agli insegnamenti di ciascun
12    # docente, utilizzata nel constraint SAME_INSTR
13    instructors_map = {}
14
15    # struttura dati che mantiene i corsi e le classi
16    # di ciascun anno
17    years_map = {}
18
19    for row in rows:
20        course_offering_code = row['Codice']
21
22        activity_typology = row['Tipo attivita']
23
24        if activity_typology == 'Insegnamento' or
              activity_typology == 'Laboratorio':
25            if course_offering_code in course_offerings_map:
26                course_offerings_map[course_offering_code].append(row)
27            else:
28                course_offerings_map[course_offering_code] = [row]
29
30    class_id = 1
31
32    for code in course_offerings_map:
33        # creo una classe per ogni singolo modulo di ciascun
34        # insegnamento
35        for course in course_offerings_map[code]:
36            year = course['anno di corso']
37            if year not in years_map:
38                years_map[year] = [], []
39
40            current_year = years_map[year]
41
42            current_year[0].append(course['Codice'])
43            current_year[1].append(class_id)
44
```

```
45          # per semplificare metto l'id del corso sia come id
46          # dell'offerta, che come id della sottoparte sia come
47          # id della configurazione della classe
48          class_ = ET.SubElement(classes, 'class',
                id=str(class_id), offering=course['Codice'],
                subpart=course['Codice'], config=course['Codice'],
                classLimit='10', commited='false', scheduler='1',
                dates='000000000000111111001111101l....')
49
50          instructor_id = get_instructor_id(instructor_rows,
                course['Responsabili'])
51
52          if instructor_id != -1:
53              ET.SubElement(class_, 'instructor',
                    id=instructor_id)
54              # verifico se nella struttura dati degli istruttori
55              # e' gia' presente questo istruttore. Poi aggiungo
56              # la classe al rispettivo istruttore
57              if instructor_id not in instructors_map:
58                  instructors_map[instructor_id] = []
59
60              current_instructor = instructors_map[instructor_id]
61              current_instructor.append(class_id)
62
63          rooms = get_rooms_group(building_and_rooms_csv_path)
64
65          for room in rooms:
66              ET.SubElement(class_, 'room', pref='0',
                    id=room['Codice'])
67
68          activity_typology = course['Tipo attivita']
69
70          typology = 'Lec' if activity_typology == 'Insegnamento'
                else 'Lab'
71
72          days_per_week = lab_days_per_week if typology == 'Lab'
                else lecture_days_per_week
73          min_per_week = lab_min_per_week if typology == 'Lab'
                else lecture_min_per_week
```

```
74
75              # in base alla tipologia dell'insegnamento (lab o lec)
76              # passo a parametro il numero di giorni per
77              # settimana al metodo get_days_combination.
78              # Invece per calcolare la durata delle lezioni
79              # prendo il numero di minuti per settimana
80              # (sempre in base alla tipologia) e lo divido
81              # per il numero di giorni in modo da
82              # ottenere la durata in minuti della singola lezione;
83              # a questo punto divido i minuti per 5 (durata di
84              # ciascun slot) per trovare il numero di slot necessari.
85              # Per ogni combinazione di giorni trovata inserisco
86              # la possibilita' di iniziare ogni # mezz'ora.
87              # Ad esempio se ho una combinazione 11000 quindi
88              # il lunedi' e il martedi', la lezione potra' iniziare
89              # questi giorni alle 9, 9:30, 10, 10:30
90              # 11, 12 e cosi' via dicendo fino massimo alle 16.
91              for days in days_per_week:
92                  length = int((min_per_week / int(days)) / 5)
93                  # i giorni della settimana sono un array, per ogni
94                  # possibilita' inserita calcolo le combinazioni
95                  for days_combinations in
                        get_days_combination(int(days)):
96                      days_str = ''.join(str(day) for day in
                            days_combinations)
97                      # la prima lezione alle ore 9
98                      start = 108
99                      # l'ultima lezione alle ore 16
100                     while start <= 192:
101                         ET.SubElement(class_, 'time', days=days_str,
                                start=str(start), length=str(length),
102                                         pref='0.0')
103                         # incremento di mezz'ora il prossimo start
104                         start += 6
105
106             class_id += 1
```

Listing 6.34: Codice Generazione Sezione XML Insegnamenti per Solver Prompt

La funzione prende a parametro il percorso del file csv degli insegnamenti, le informazioni relative agli istruttori e le personalizzazioni passate in input dall'utente viste nella sezione 6.2, più precisamente nella figura 6.6. Successivamente si occupa di suddividere gli insegnamenti per anno di corso, raggruppare gli insegnamenti in moduli con lo stesso codice corso e creare una classe per ciascun modulo e/o insegnamento in base alla casistica riscontrata. La peculiarità della casistica in esame è quella di dover definire per ciascun elemento di tipo *class* tutte le aule ammissibili, restituite dalla funzione *get_rooms_group*, riga 63, oltre a tutti i *time pattern* utilizzabili con il relativo livello di preferenza (come visto in precedenza nella sezione 6.4.1). Per ciascun pattern è necessario definire la combinazione di giorni selezionabile come soluzione, ad esempio MTW ovvero Lunedì, Martedì e Mercoledì. Per automatizzare l'operazione sopra descritta, in base alla tipologia dell'insegnamento (Laboratorio o Lezione) viene passato a parametro al metodo *get_days_combination* il numero di giorni a settimana di cui si vuole avere tutte le possibili combinazioni; il risultato della chiamata al metodo viene restituito sotto forma di array, contenente al suo interno tutte le possibili combinazioni ed ogni elemento è strutturato nel formato 1110000 ovvero, per utilizzare lo stesso esempio fatto in precedenza MTW che corrisponde a Lunedì, Martedì e Mercoledì. Invece, per quel che riguarda il valore *length* del time pattern, ovvero la durata in slot da 5 minuti delle lezioni, viene diviso il numero di minuti per settimana, sempre in base alla tipologia della classe, per il numero di giorni a settimana in modo da ottenere la durata del singolo incontro. Successivamente, il risultato ottenuto viene diviso per 5 per ottenere il numero di slot necessari. Per ciascuna combinazione possibile viene data la possibilità alla lezione di iniziare ad ogni mezz'ora della fascia oraria che va dalle 9 del mattino alle 4 del pomeriggio, ovvero le 16. Ad esempio, per la combinazione 1110000, Lunedì, Martedì e Mercoledì, la lezione potrà iniziare in quei giorni alle 9, 9:30, 10, 10:30, 11, 12 e così via fino ad arrivare alle 16 incluse. L'operazione sopra descritta è implementata dalla riga 91 alla riga 104. Di seguito è riportato il codice della funzione *get_days_combination*.

```
def get_days_combination(days_num):
    combs = list(filter(lambda val: val.count(1) == days_num,
        (itertools.product([0, 1], repeat=5))))
    # aggiungo a ciascuna combinazione possibile di giorni della
    # settimana su 5 giorni (dal lun al ven) due zeri che indicano
    # rispettivamente l'impossibilita' di fare lezione il sabato
    # e la domenica
    return list(map(lambda comb: list(comb) + [0, 0], combs))
```

Listing 6.35: Codice Funzione get_days_combination

Al termine di questa operazione e sempre all'interno della funzione principale 6.34, si procede con la creazione dei *groupCostraints* visti nella sezione 6.4.1 e successivamente con l'inserimento di uno studente per ogni anno di corso; così facendo viene mantenuta l'informazione relativa all'anno di corso di ciascun insegnamento.

```
    group_constraints = ET.SubElement(timetable,
        'groupConstraints')

    students = ET.SubElement(timetable, 'students')

    constraint_id = 0

    # aggiungo un vincolo SAME_INSTR per ogni istruttore
    # che ha piu' classi
    for instructor_id in instructors_map:
        if len(instructors_map[instructor_id]) > 1:
            constraint_same_instructor =
                ET.SubElement(group_constraints, 'constraint',
                id=str(constraint_id), type='SAME_INSTR', pref='R')
            for instructor_course in instructors_map[instructor_id]:
                constraint_id += 1
                ET.SubElement(constraint_same_instructor, 'class',
                    id=str(instructor_course))

    for year in years_map:
        current_year = years_map[year]
        constraint_id += 1
```

```
19
20          # aggiungo un vincolo SAME_STUDENTS per tutte le classi di
21          # ogni anno di corso
22          constraint_same_students =
                ET.SubElement(group_constraints, 'constraint',
                id=str(constraint_id), type='SAME_STUDENTS', pref='R')
23
24          student = ET.SubElement(students, 'student', id=str(year))
25          for offering in current_year[0]:
26              ET.SubElement(student, 'offering', id=str(offering))
27
28          for class_ in current_year[1]:
29              ET.SubElement(constraint_same_students, 'class',
                    id=str(class_))
30              ET.SubElement(student, 'class', id=str(class_))
```

Listing 6.36: Sezione Funzione course_to_command_line_xml per la Generazione dei Group Constraints

6.4.3 Esecuzione del Solver

L'esecuzione del solver da riga di comando ha come prerequisiti quelli di avere installato java sulla propria macchina, aver scaricato in precedenza il jar per la pianificazione dell'orario, *coursett-version.jar*, dall'indirizzo `https://builds.unitime.org/`, aver generato il file xml da passare come parametro (strutturato come analizzato nella sezione 6.4.1) e infine il file di configurazione contenente i parametri di esecuzione del solver, visti nella sezione 6.3.3.

I parametri con cui il solver può essere eseguito sono molteplici, per questo nel caso di studio di interesse è stata utilizzata una configurazione di default. Di seguito ne è riportata solamente una sezione per mostrarne la struttura.

```
## Basic Settings
#####################################
## Solver mode
## Type: enum(Initial,MPP)
Basic.Mode=Initial
## When finished
## Type: enum(No Action,Save,Save as New,Save and Unload,Save as
    New and Unload)
Basic.WhenFinished=No Action
## Students sectioning
## Type: boolean
General.SwitchStudents=true

....

## General Settings
#####################################
## Use conflict-based statistics
## Type: boolean
General.CBS=true
## Use departmental balancing
## Type: boolean
General.DeptBalancing=false
## Automatic same student constraint
## Type: enum(SAME_STUDENTS,DIFF_TIME)
```

```
General.AutoSameStudentsConstraint=SAME_STUDENTS
## Ignore Room Sharing
## Type: boolean
General.IgnoreRoomSharing=false
## Do not load committed student conflicts (deprecated)
## Type: boolean
General.IgnoreCommittedStudentConflicts=false
## Weight last-like students (deprecated)
## Type: boolean
General.WeightStudents=true
## Student course demands
## Type: enum(Last Like Student Course Demands,Weighted Last Like
    Student Course Demands,Projected Student Course
    Demands,Curricula Course Demands,Curricula Last Like Course
    Demands,Student Course Requests,Enrolled Student Course
    Demands)
Curriculum.StudentCourseDemadsClass=Projected Student Course
    Demands

....
```

Listing 6.37: File Configurazione Solver

Una volta soddisfatti tutti i requisiti, il comando per l'esecuzione del solver è il seguente

```
java -Xmx1g -jar coursett-1.3.jar file.cfg input.xml output_folder
```

dove *file.cfg* è il file di configurazione, *input.xml* l'xml contenente i dati da passare al solver e *output_folder* la cartella in cui si vuole salvare la soluzione. La versione di java richiesta per la corretta esecuzione è java11.

Capitolo 7

Risultati Ottenuti

In questo capitolo verranno analizzate le soluzioni proposte dal solver di UniTime, sia tramite applicazione web sia da riga di comando come visto nelle sezioni precedenti. In entrambe le casistiche vengono stilate le statistiche della soluzione trovata che indicano la qualità della soluzione stessa e sono [29]:

- Created: data e ora in cui la soluzione è stata creata;
- Assigned variables: percentuale di classi che hanno aula e orario assegnati (il rapporto tra il numero di classi assegnate e il numero totale di classi da assegnare);
- Overall solution value: valore utilizzato dal solver per valutare la soluzione ottenuta. Quando si confrontano due soluzioni, la soluzione con valore minore è migliore dell'altra;
- Time preferences: soddisfazione complessiva delle preferenze di orario passate in input, espressa in percentuale (100% tutte le classi sono state assegnate negli orari migliori, 0% tutte le classi sono state assegnate negli orari peggiori). Il numero vicino alla percentuale è la somma delle preferenze di orario normalizzate di tutte le classi assegnate;
- Student conflicts: numero di conflitti tra studenti (committed, distance, hard).
- Room preferences: soddisfazione complessiva delle preferenze di aula espressa in percentuale (100% tutte le classi assegnate nelle migliori

aule possibili, 0% tutte le classi assegnate nelle peggiori aule possibili). Il numero accanto alla percentuale è la somma delle preferenze di aula normalizzate di tutte le classi assegnate;

- Distribution preferences: soddisfazione complessiva delle preferenze di distribuzione delle classi espressa in percentuale (100% tutte le preferenze di distribuzione soddisfatte, 0% nessuna preferenza di distribuzione soddisfatta). Il numero accanto alla percentuale è la somma delle preferenze di distribuzione normalizzate;

- Back-to-back instructor preferences: percentuale di lezioni consecutive tenute dallo stesso istruttore che si trovano in aule abbastanza vicine da consentirgli di camminare dall'una all'altra. Il numero accanto alla percentuale è il numero ponderato di classi back to back dello stesso istruttore (i pesi dipendono dalla distanza tra le stanze);

- Too big rooms: "Too big rooms" è una aula che ha almeno il 25% di posti in più rispetto a quanto necessario per la classe che è stata pianificata nell'aula di riferimento. Esistono due limiti per le aule troppo grandi: 25% e 50% di posti in più del dovuto. Le stanze che superano il numero di posti necessari per meno del 25% non sono penalizzate (penalità "0"), le stanze che superano i posti necessari dal 25% al 50% sono penalizzate da "1", le stanze con più del 50% più posti del necessario hanno la penalità "4". Il numero tra parentesi è la penalità totale su tutte le classi. La percentuale può essere interpretata come "0% - tutte le classi sono in aule che non hanno più del 125% di posti necessari, 100% - tutte le classi sono in aule che hanno più del 150% di posti necessari";

- Useless half-hours: percentuale di slot orari inutilizzati (intervalli di cinque minuti) in tutte le aule difficilmente utilizzabili per le lezioni a causa di uno dei seguenti motivi (ogni motivo è rappresentato da un numero accanto alla percentuale):

  - la fascia oraria è in un blocco tra le classi che dura meno di mezz'ora;
  - la fascia oraria rientra in una parte di pianificazione "interrotta", ovvero ad esempio Martedì alle 9:05 mentre Giovedì dalle 9-10:30

c'è già una lezione, quindi non può essere previsto e utilizzato un modello temporale standard TR dalle 9:00 alle 10:30

- Same subpart balancing penalty:
  - penalità se le classi di una sotto parte di ciascun insegnamento, ad esempio lezione o laboratorio, non sono distribuite uniformemente nelle diverse parti della giornata;
  - calcolato dal numero di ore utilizzate in più rispetto alla media degli slot orari disponibili;
- Department balancing penalty: penalità se le classi di un dipartimento non sono distribuite uniformemente nelle diverse parti della giornata (per essere rispettosi nei confronti dei dipartimenti e non dare ad un singolo dipartimento tutti gli slot dalle 10:00 alle 14:00);
- Perturbation variables (solo in MPP solver mode): il numero di classi che hanno un'assegnazione di orario o aula diverso da quello dell'orario caricato + il numero di nuove variabili che non erano nell'orario caricato;
- Perturbations - Total penalty (solo in MPP solver mode): penalità per modifiche apportate a un orario dopo che è stato precedentemente caricato da una soluzione salvata;
- Time: il tempo impiegato dal solver per generare questa soluzione di pianificazione;
- Iteration: numero di iterazioni necessarie per generare la pianificazione di riferimento;
- Memory usage: memoria allocata sul server del solver in cui viene caricata l'istanza;
- Speed: numero di iterazioni per secondo.

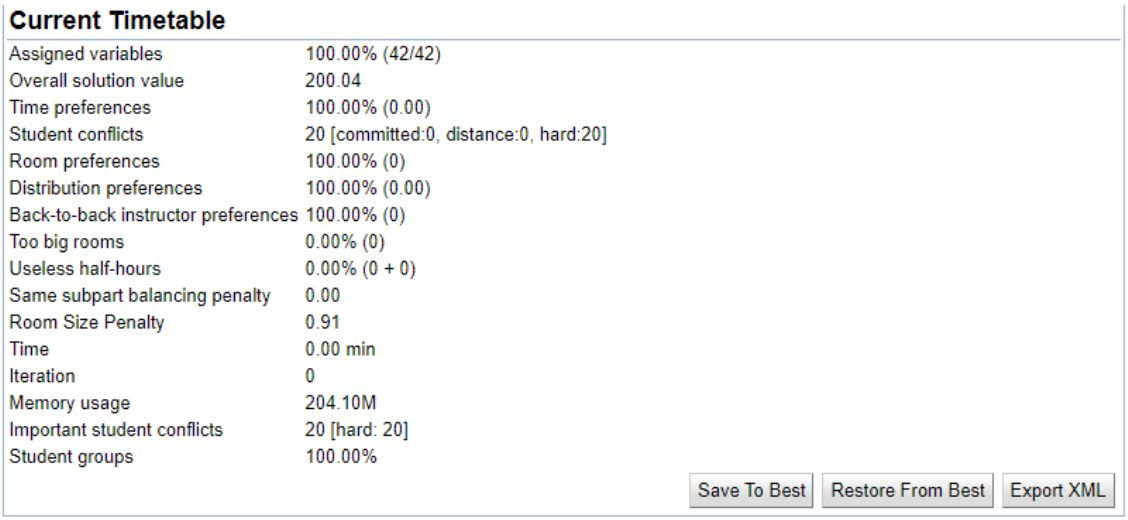

Figura 7.1: Statistiche Soluzioni UniTime

Nelle sezioni successive saranno prese in esame ed analizzate le soluzioni proposte dall'applicazione web e dal solver da riga di comando.

7.1 Risultati Ottenuti Piattaforma Web

In questa sezione verrà analizzata la soluzione ottenuta eseguendo il solver sfruttando la piattaforma web di UniTime. Tuttavia, dato che lo scopo principale dello studio è quello di eseguire l'algoritmo da riga di comando, l'esecuzione tramite interfaccia web di UniTime è stata effettuata senza definire gli studenti e senza l'inserimento di particolari vincoli se non quelli relativi alle preferenze. Si è ritenuto comunque necessario effettuare questa operazione per prendere confidenza con il sistema e imparare ad utilizzare l'applicativo web.

I parametri passati all'applicazione UniTime Wrapper, sezione 6.2, per la creazione dei file xml da importare, sono 360 minuti a settimana suddivisi in 3 incontri per le lezioni in aula e 360 minuti suddivisi in 2 incontri per le lezioni in laboratorio. I minuti per settimana sono stati impostati a 360 per verificare il comportamento del solver su slot orari di grandi dimensioni.

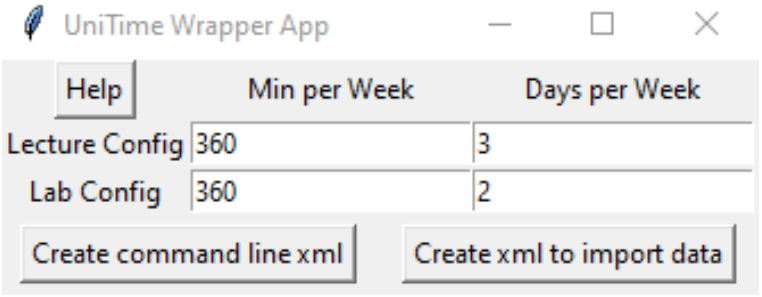

Figura 7.2: Parametri UniTime Wrapper App per UniTime Web

Il solver è stato eseguito su 39 insegnamenti suddivisi in 42 classi con i rispettivi istruttori; 3 classificazioni accademiche ciascuna definita da un curriculum e un gruppo di aule composto da 37 elementi.

I parametri di esecuzione del solver su piattaforma web sono mostrati nella seguente figura.

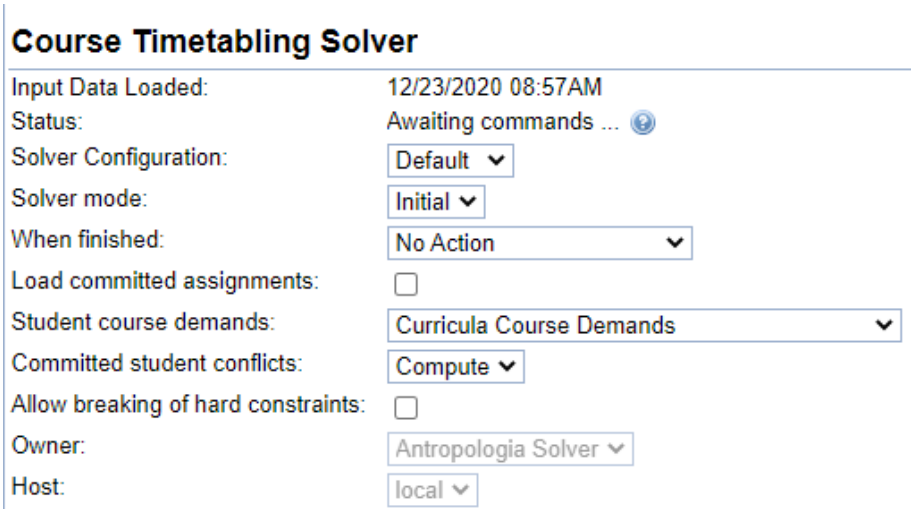

Figura 7.3: Parametri Esecuzione Solver Applicazione Web UniTime

Come è possibile notare, la configurazione del solver è stata settata su *Default* e la modalità a *Initial*. Inoltre, il parametro *Student course demands* è stato impostato a *Curricula Course Demands* invece del valore di default che è *Projected Student Course Demands*. Questa operazione è stata effettuata per poter ottenere la pianificazione orientata ai curriculum e quindi avere una pianificazione dell'orario suddivisa per ogni classificazione accademica, ovvero per ogni anno del corso di studi di riferimento.

Al termine di questa procedura, UniTime permette la visualizzazione della soluzione ricavata all'interno di una tabella suddivisa per giorno della settimana e slot orario. La figura 7.4 mostra un esempio di visualizzazione di una pianificazione relativa ad un singolo anno di corso.

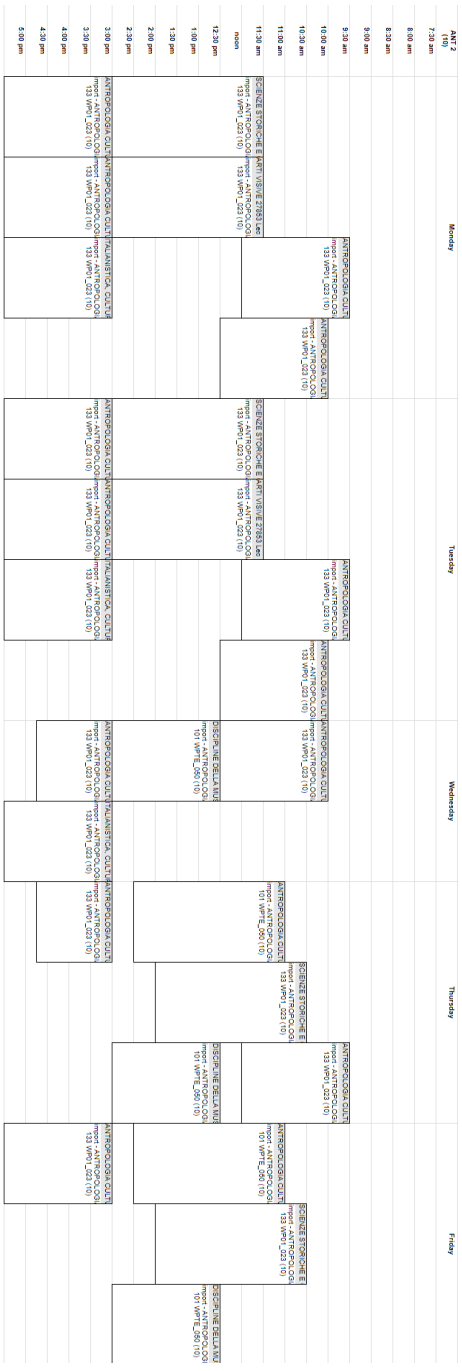

Figura 7.4: Esempio Soluzione di un Anno di Corso Pianificato

Di seguito sono riportate le statistiche della soluzione ottenuta.

Current Timetable

Assigned variables	100.00% (42/42)
Overall solution value	200.04
Time preferences	100.00% (0.00)
Student conflicts	20 [committed:0, distance:0, hard:20]
Room preferences	100.00% (0)
Distribution preferences	100.00% (0.00)
Back-to-back instructor preferences	100.00% (0)
Too big rooms	0.00% (0)
Useless half-hours	0.00% (0 + 0)
Same subpart balancing penalty	0.00
Room Size Penalty	0.91
Time	0.00 min
Iteration	0
Memory usage	204.42M
Important student conflicts	20 [hard: 20]
Student groups	100.00%

Figura 7.5: Statistiche Soluzione senza Vincoli Applicazione Web UniTime

Come è possibile vedere nella figura precedente, tutte le classi sono state assegnate ma la soluzione, nonostante non abbia vincoli, presenta 20 conflitti tra studenti e un *Overall solution value* pari a 200.04; questi due fattori indicano che la soluzione trovata non sia sufficientemente valida e che quindi ci saranno diverse modifiche da apportare.

7.2 Risultati Ottenuti Solver Prompt

In questa sezione verrà mostrata ed analizzata la soluzione proposta dal solver eseguito da riga di comando sfruttando l'eseguibile *coursett-1.3.jar* visto nel capitolo 5.

I parametri passati all'applicazione *UniTime Wrapper App*, sezione 6.2, sono 180 minuti a settimana suddivisi in 3 incontri per le lezioni in aula e 180 minuti suddivisi in 2 incontri per le lezioni in laboratorio.

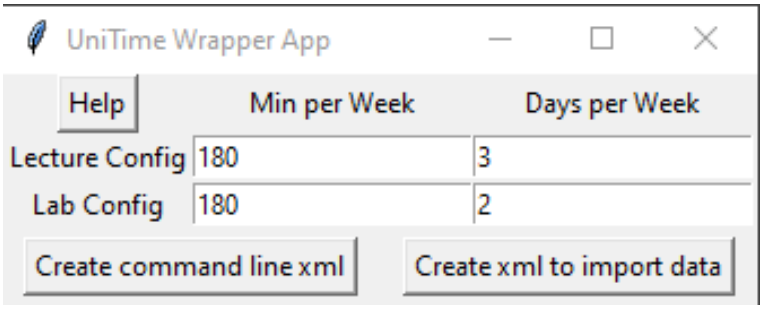

Figura 7.6: Parametri UniTime Wrapper App per Riga di Comando

Queste personalizzazioni hanno permesso all'applicativo python di generare un xml strutturato come visto nella sezione 6.4.1 contenente tutte le preferenze passate in input.

Come accennato nel capitolo precedente, la configurazione utilizzata è quella di default, xml 6.37, che ha come parametri principali il solver mode a *Initial* e il valore *Student course demands* settato a *Projected Student Course Demands*.

L'xml contenente i dati su cui effettuare la pianificazione è composto da 3 studenti, uno per ogni anno di corso, 111 aule, 39 insegnamenti suddivisi in 42 classi (con i rispettivi istruttori) divise in 3 anni di corso, più precisamente 17 classi per il primo anno, 14 classi per il secondo anno e 11 classi per il terzo anno. I vincoli utilizzati sono quelli visti nella sezione 6.4.1 ovvero *SAME_STUDENTS* e *SAME_INSTR*.

La soluzione proposta del solver è contenuta all'interno di un file xml in cui per ogni classe viene aggiunto un attributo *solution* settato a true ad ogni risorsa (istruttore, aula e time pattern) evidenziata come soluzione.

```
<class id="10" ord="9" config="75725" committed="false"
    subpart="75725" classLimit="10" name="c10" scheduler="1"
    datePatternName="not set"
    dates="000000000000000000000000000011111100111....1">
  <instructor id="37010" solution="true"/>
  ...
  <room id="1" pref="0"/>
  <room id="74" pref="0" solution="true"/>
  <room id="75" pref="0"/>
  <room id="76" pref="0"/>
  ...
  <time days="0111000" start="144" length="12" breakTime="10"
     pref="0" npref="0.0"/>
  <time days="0111000" start="150" length="12" breakTime="10"
     pref="0" npref="0.0" solution="true"/>
  <time days="0111000" start="156" length="12" breakTime="10"
     pref="0" npref="0.0"/>
  <time days="0111000" start="162" length="12" breakTime="10"
     pref="0" npref="0.0"/>
  ...
</class>
```

Listing 7.1: Struttura XML Soluzione Solver Prompt

Come precedente accennato nella sezione 6.2 l'applicativo python UniTime Wrapper App è dotato della funzionalità di trasformare l'xml (sopra mostrato) in un file csv per ogni anno di corso in modo da rendere la soluzione user friendly.

	A	B	C	D	E	F
1	course	start_time	end_time	day	instructor	room
2	28304	14.0	16.0	MTTh	39926	74
3	28346	12.0	14.0	TThF	36772	4
4	69514	16.0	18.0	MTTh	39926	110
5	81691	14.0	16.0	TWTh	38905	59
6	39426	16.0	18.0	MWF	33150	59
7	84268	16.0	18.0	MWF	36995	111
8	27867	12.0	14.0	TWTh		50
9	87346	16.0	18.0	MWF	37194	19
10	75724	12.0	14.0	MWF	37010	93
11	75725	9.0	11.0	MTTh	37010	93
12	87361	14.0	16.0	TWTh	46976	4
13	87362	12.0	14.0	TThF	46976	74
14	87348	9.0	11.0	TThF		19
15	87349	16.0	18.0	TWTh		4
16	87351	14.0	16.0	TThF	41833	19
17	87352	9.0	11.0	TThF	41833	12
18	32535	9.0	11.0	MTTh		61

Figura 7.7: Esempio di CSV Contenente un Anno di Corso Pianificato

Il solver è stato eseguito su 3 diverse casistiche: impostando i vincoli a Required, impostando i vincoli a Preferred e senza vincoli. Nelle sezioni successive verranno mostrate le statistiche di ciascuna soluzione. Nella spiegazione delle soluzioni i termini insegnamenti, variabili e classi acquisiscono lo stesso significato e sono perciò intercambiabili.

7.2.1 Soluzione con Vincoli Required

Impostando i vincoli come Required e quindi, obbligatori, il solver è riuscito ad assegnare 27 insegnamenti su 42, questo perché i vincoli inseriti sono molto restrittivi e richiedono una potenza di calcolo elevata cosa che la macchina utilizzata in fase di testing non era in grado di offrire. Di seguito sono riportate le statistiche che descrivono la soluzione trovata.

```
Solution info: [
    Assigned variables: 64.29% (27/42)
    Distribution preferences: 100.00% (0.00)
    Iteration: 12041
    Memory usage: 593.62M
    Overall solution value: 0.00
    Room preferences: 100.00% (0)
    Same subpart balancing penalty: 0.00
    Speed: 6.69 it/s
    Student conflicts: 0 [committed:0, distance:0, hard:0]
    Time: 30.01 min
    Time preferences: 100.00% (0.00)
    Too big rooms: 0.00% (0)
    Useless half-hours: 0.00% (0 + 0)
  ]
```

Figura 7.8: Statistiche Soluzione con Vincoli Required

Come evidenziato dalle statistiche sopra riportate nonostante siano stati assegnati solamente 27 insegnamenti, nella soluzione proposta sono state soddisfatte al 100% sia le preferenze di aula sia le preferenze di slot orario. Inoltre, i conflitti tra gli studenti sono 0.

7.2.2 Soluzione con Vincoli Preferred

Successivamente si è provveduto a modificare i vincoli da Required a Preferred, ovvero preferiti e non richiesti. In questa situazione, il risolutore è stato in grado di assegnare non più 27 insegnamenti ma bensì tutti gli insegnamenti. La figura seguente mostra le statistiche della soluzione trovata con vincoli settati a *Preferred*.

```
Solution info: [
    Assigned variables: 100.00% (42/42)
    Distribution preferences: 93.77% (19.00)
    Iteration: 2655
    Memory usage: 586.04M
    Overall solution value: 50.80
    Room preferences: 100.00% (0)
    Same subpart balancing penalty: 0.00
    Speed: 1.47 it/s
    Student conflicts: 16 [committed:0, distance:0, hard:12]
    Time: 30.05 min
    Time preferences: 100.00% (0.00)
    Too big rooms: 0.00% (0)
    Useless half-hours: 0.00% (0 + 0)
  ]
```

Figura 7.9: Statistiche Soluzione con Vincoli Preferred

Le statistiche mostrano chiaramente come tutte le classi siano state assegnate. Tuttavia, sono presenti 16 conflitti tra studenti di cui 12 *hard*; rimane comunque un'ottima soluzione su cui poi si potranno effettuare delle migliorie manualmente oppure eseguendo il solver in modalità *MPP*.

7.2.3 Soluzione senza Vincoli

Infine, l'eseguibile è stato lanciato senza inserire nessun vincolo ma aumentando i minuti per settimana a 360. In questo caso il solver è riuscito ad assegnare tutte le variabili di ciascun insegnamento. Di seguito sono mostrate le statistiche della soluzione generata.

```
Solution info: [
    Assigned variables: 100.00% (42/42)
    Distribution preferences: 100.00% (0.00)
    Iteration: 5503
    Memory usage: 481.59M
    Overall solution value: 43.20
    Room preferences: 100.00% (0)
    Same subpart balancing penalty: 0.00
    Speed: 3.06 it/s
    Student conflicts: 49 [committed:0, distance:0, hard:40]
    Time: 30.02 min
    Time preferences: 100.00% (0.00)
    Too big rooms: 7.14% (12)
    Useless half-hours: 0.00% (0 + 2)
  ]
```

Figura 7.10: Statistiche Soluzione senza Vincoli

Come è possibile notare sono state assegnate tutte le variabili ma la soluzione ha 49 conflitti tra studenti di cui 40 *hard* e il 7.14% (12) delle classi è svolto in aule troppo grandi: questo significa che la soluzione non è per niente buona.

Capitolo 8

Conclusioni

8.1 Resoconto

Lo studio dell'applicativo UniTime, con particolare riguardo all'algoritmo utilizzato all'interno del solver, ha fatto emergere una sostanziale diversità tra il modello dati di UP2.0 e quello di UniTime. Tuttavia, attraverso le opportune modifiche effettuate in fase di mapping (descritte nei capitoli precedenti) sono arrivato al punto di riuscire ad eseguire il solver sia tramite la piattaforma web sia da riga di comando sfruttando l'eseguibile visto nel capitolo 5.

Per poter essere eseguito il solver necessita di una struttura dati molto articolata e l'implementazione dell'applicazione python *UniTime Wrapper App*, sezione 6.2, ha facilitato notevolmente il lavoro di interpretazione e adattamento dei dati, rendendo il tutto un processo automatizzato.

I lanci di test effettuati sono stati numerosi ed ognuno di essi è stato effettuato modificando di volta in volta solamente alcuni degli innumerevoli parametri del solver di UniTime. Nonostante ciò, i risultati ottenuti mostrati nel capitolo precedente sono soddisfacenti e del tutto inaspettati. Tuttavia, rimangono evidenti alcune differenze tra i due sistemi analizzati, ovvero UP2.0 e UniTime; ad esempio, la pianificazione dell'orario in UP2.0 viene effettuata per il singolo giorno dell'anno (es. 15/02/2021), mentre in UniTime l'algoritmo restituisce la pianificazione di una settimana "tipo" che si replicherà per tutte le settimane del semestre di riferimento. Nel caso in cui ci dovessero essere feste o vincoli preimpostati, la soluzione mostrata per la settimana selezionata non mostrerà incontri in quei determinati

giorni. Inoltre, l'algoritmo di UniTime ha una scarsa flessibilità nella scelta dell'orario in cui ciascun insegnamento dovrà essere effettuato; ad esempio, se un determinato insegnamento ha come possibili scelte il Lunedì e il Martedì dalle 9:00 alle 11:00 oppure dalle 11:00 alle 13:00, l'orario scelto sarà vincolante per entrambe i giorni. Questo significa che non ci potranno essere scenari in cui la lezione inizi a due orari differenti in due giorni diversi (la soluzione Lunedì alle 9:00 e Martedì alle 11:00 non è ammissibile per il solver, dovrà perciò essere modificata manualmente all'evenienza).

Un'altra importante osservazione è che il solver durante la sua esecuzione necessita di un elevato quantitativo di risorse hardware della macchina in cui è in esecuzione. I lanci effettuati con problemi complessi e/o vincoli obbligatori hanno saturato la CPU della macchina utilizzata per tutta la durata della computazione che, nel mio caso con i parametri da me impostati, era di 30 minuti ovvero fino allo scadere del timeout.

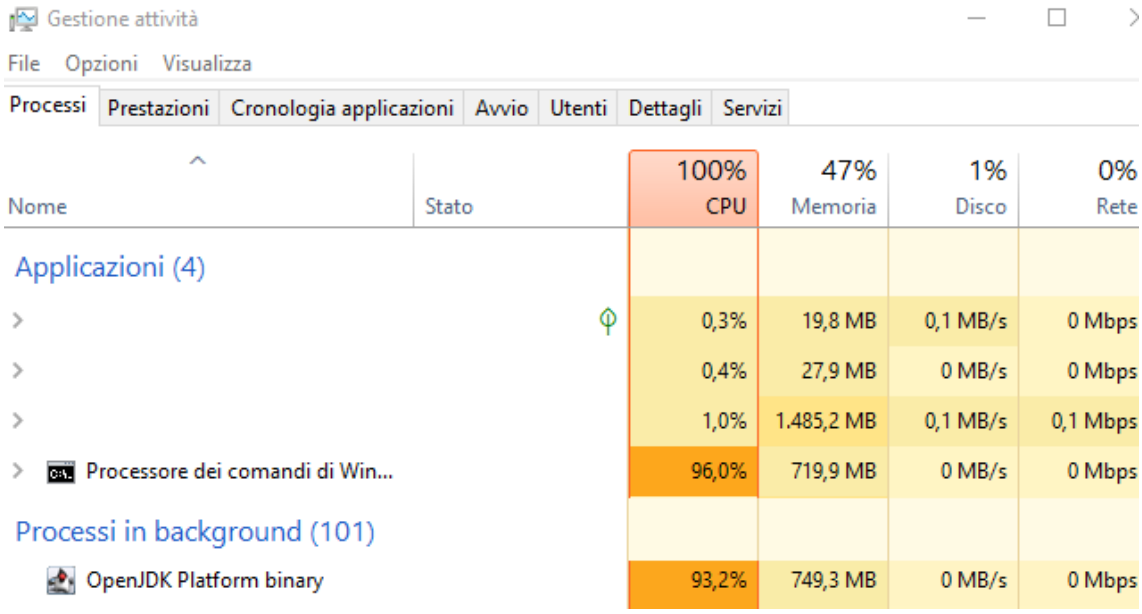

Figura 8.1: Risorse Occupate dall'Esecuzione del Solver[1]

[1]La macchina utilizzata è dotata di 16GB di memoria RAM ed un processore Intel i5 di terza generazione con 4 core (8 thread) con frequenza a 3.10GHz.

I tempi di analisi, sviluppo e progettazione preventivati si sono dimostrati pressoché veritieri, permettendomi di agire ed operare in maniera elastica in base anche ai problemi e alle difficoltà riscontrate.

I feedback ricevuti dalla società Cineca riguardanti il lavoro svolto sono positivi e per questo mi ritengo altamente soddisfatto del mio operato e dello studio effettuato al termine di questo primo passo del progetto.

8.2 Sviluppi Futuri

Lo studio svolto continuerà a pieno regime, approfondendo con maggior dettaglio tutte le potenzialità del sistema UniTime e del suo modello dati e tutte le personalizzazioni effettuabili tramite la modifica di ogni singolo parametro di esecuzione del solver.

Il prossimo step sarà quello di implementare un'interfaccia che, tramite l'ausilio dell'applicativo sviluppato *UniTime Wrapper App*, si occuperà di automatizzare in toto la procedura di export dei dati da UP2.0, il mapping dei dati e successivamente l'esecuzione del solver. Questo provvedimento offrirà all'utente finale maggiore elasticità nella modifica dei parametri del risolutore e velocizzerà tutto il procedimento volto a produrre una soluzione di pianificazione dell'orario.

Infinite sono ancora le possibilità che l'informatica e la tecnologia possono offrire all'uomo, perciò è mia intenzione sfruttarle al meglio per far sì che questo progetto arrivi al massimo delle sue potenzialità.

> "I computer sono incredibilmente veloci, accurati e stupidi. Gli uomini sono incredibilmente lenti, inaccurati e intelligenti. L'insieme dei due costituisce una forza incalcolabile."
>
> *Albert Einstein*

Ringraziamenti

Volevo ringraziare in primis tutti coloro che mi sono stati vicini e mi hanno permesso di raggiungere questo traguardo, il professor Boschetti per la sua disponibilità e competenza, la mia famiglia e per ultimi, ma non meno importanti, Furlati e tutto lo staff di Cineca che mi hanno permesso di realizzare questo progetto.

Bibliografia

[1] MIUR. Portale dei dati dell'istruzione superiore - i numeri della formazione. http://ustat.miur.it/. [Online; accessed 07/03/2021].

[2] MIUR. Portale dei dati dell'istruzione superiore - didattica. http://ustat.miur.it/dati/didattica/italia/atenei. [Online; accessed 07/03/2021].

[3] Cineca. Il consorzio. https://www.cineca.it/chi-siamo/il-consorzio. [Online; accessed 21/01/2021].

[4] MIUR. Il sistema universitario italiano. https://www.miur.gov.it/il-sistema-universitario. [Online; accessed 15/01/2021].

[5] Atti ministeriali MIUR. Il sistema universitario italiano. http://attiministeriali.miur.it/media/211291/il_sistema_universitario_italiano.pdf. [Online; accessed 15/01/2021].

[6] MIUR. Processo di bologna. https://www.miur.gov.it/processo-di-bologna. [Online; accessed 18/01/2021].

[7] European Higher Education Area ehea. Ministerial conference bologna 1999. http://www.ehea.info/page-ministerial-conference-bologna-1999. [Online; accessed 18/01/2021].

[8] European Higher Education Area ehea. Sorbonne declaration 1998. http://www.ehea.info/page-sorbonne-declaration-1998. [Online; accessed 18/01/2021].

[9] Europa.eu. Sistema europeo di trasferimento e accumulazione dei crediti (ects). https:

//ec.europa.eu/education/resources-and-tools/european-credit-transfer-and-accumulation-system-ects_it. [Online; accessed 18/01/2021].

[10] Wikipedia. Istruzione negli stati uniti d'america - università. https://it.wikipedia.org/wiki/Istruzione_negli_Stati_Uniti_d%27America#Universit%C3%A0. [Online; accessed 15/01/2021].

[11] Wikipedia. College. https://it.wikipedia.org/wiki/College. [Online; accessed 19/01/2021].

[12] Sara Nosenzo. Università e college americani. https://www.wep.it/take-the-leap/come-funzionano-i-college-in-america/. [Online; accessed 19/01/2021].

[13] Universando. Italia vs stati uniti: Sistemi universitari a confronto. https://universando.com/italia-vs-stati-uniti-sistemi-universitari-a-confronto/. [Online; accessed 18/02/2021].

[14] Cineca. University planner 2.0. https://www.cineca.it. [Online; accessed 13/01/2021].

[15] Unitime Authors. Unitime. https://www.unitime.org. [Online; accessed 18/01/2021].

[16] Apereo. Apereo foundation. https://www.apereo.org/. [Online; accessed 10/03/2021].

[17] Wikipedia. Apache tomcat. https://it.wikipedia.org/wiki/Apache_Tomcat. [Online; accessed 27/02/2021].

[18] Wikipedia. Mysql. https://it.wikipedia.org/wiki/MySQL. [Online; accessed 18/02/2021].

[19] Unitime Authors. Unitime. https://www.unitime.org/index.php?tab=1. [Online; accessed 20/02/2021].

[20] Tomáš Müller. Constraint-based timetabling. https://www.unitime.org/papers/phd05.pdf. [Online; accessed 27/02/2021].

[21] Okpedia. Ricerca locale. `https://www.okpedia.it/ricerca_locale`. [Online; accessed 27/02/2021].

[22] Wikipedia. Backtracking. `https://it.wikipedia.org/wiki/Backtracking`. [Online; accessed 01/03/2021].

[23] Tomáš Müller. University course timetabling - solver evolution. `https://www.unitime.org/papers/patat2016.pdf`. [Online; accessed 01/03/2021].

[24] Hana Rudová Tomáš Müller, Roman Barták. Conflict-based statistics. `https://www.unitime.org/papers/eume04.pdf`. [Online; accessed 01/03/2021].

[25] Wikipedia. Dipartimento universitario. `https://it.wikipedia.org/wiki/Dipartimento_universitario`. [Online; accessed 21/02/2021].

[26] UniTime Authors. Unitime xml interfaces. `https://www.unitime.org/uct_interfaces.php`. [Online; accessed 18/02/2021].

[27] UniTime Authors. Unitime 4.5 online documentation - solver. `http://help.unitime.org/Solver`. [Online; accessed 18/02/2021].

[28] UniTime Authors. Unitime university course timetabling data format (v2.4). `https://www.unitime.org/uct_dataformat_v24.php`. [Online; accessed 18/02/2021].

[29] UniTime Authors. Unitime solution properties. `http://help.unitime.org/Solution_Properties`. [Online; accessed 18/02/2021].